Dept. Mathematik
Naturwissenschaftlich - Technische Fakultät

The category of representations of a concrete category as a functor category

Bachelorarbeit
vorgelegt von
Tibor Grün
am October 30, 2020

Gutachter

Prof. Dr. Mohamed Barakat	1. GA
Prof. Dr. Jörg Jahnel	2. GA

Thanks

I would like to thank all the people who helped and supported me while working on this thesis.

I express my deep gratitude to my advisor Mohamed Barakat who always had time for questions and problems and who helped me with the structure of the thesis.

I want to thank Peter Webb for our interesting discussions and for introducing me to the topic of category representations.

Many thanks go to Sebastian Posur and Sebastian Gutsche for their software project CAP as well as to all of the contributors to the CAP project.

I also like to thank the many unnamed people answering questions online for `tikz` and LaTeX. Among them I especially want to thank Yichuan Shen for his `tikzcd-editor` which let me easily create every commutative diagram you see in this thesis.

Last, I would like to thank my family for providing me with a home and work atmosphere at a time where the university was closed.

Contents

1 Introduction

In this thesis we study the functor category of a finite concrete category with values in a matrix category over a field.

We give an introduction to quivers, categories, functors and natural transformations in Section 2 allowing a student with no background in category theory to understand this thesis. We are also introducing finite concrete categories which we model as finite subcategories of FinSets, the category of finite sets. The main source for this section is [1, Chapter 1, Sections 1.1 - 1.4].

Then we define (co-)limits in Section 3 and use them to build up a hierarchy of categorical doctrines leading up to three equivalent definitions for Abelian categories. As an example for an Abelian category we define the matrix category $\Bbbk$-**mat** and follow the steps from Ab-category, category with zero object, additive category, pre-Abelian category and finally Abelian category, each time giving constructive algorithms for that doctrine in $\Bbbk$-**mat**. This section is in part a citation from [2, I.1.38-I.1.47 and I.2] and from the lecture notes [3].

In Section 4 we revisit the relation between quivers and categories from Section 2 and give an example for an adjunction, the free-forgetful adjunction. The rest of Section 4 is concerned with rewriting our finite concrete category as a finitely presented category. The endomorphism monoid at each object plays an important role. We formulate and prove the σ-lemma which holds in general finite monoids. For a finite concrete category with explicitly cyclic endomorphism monoids we can compute a finite set of relations among the generating morphisms. The first part of the algorithm is an application of the orbit algorithm and terminates due to the σ lemma. This allows us to write the finite category $\mathcal{C}$ as a quotient modulo finitely many relations of a free category Fq generated by a finite quiver q. Since we consider functors from $\mathcal{C}$ into the $\Bbbk$-linear category $\Bbbk$-**mat**, we can equally consider $\Bbbk$-linear functors from the $\Bbbk$-linear closure $\Bbbk\mathcal{C}$ of $\mathcal{C}$. This closure is a finite-dimensional $\Bbbk$-algebroid. The $\Bbbk$-linear closure $\Bbbk\mathcal{C}$ can be modeled as the $\Bbbk$-algebra of paths of the quiver q modulo finitely many relations. This is a finite-dimensional algebra with a complete set of orthogonal idempotents.

Section 5 is the synthesis of Sections 3 and 4, in that we define the category of representations of our finite concrete category, which is the category $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ of $\Bbbk$-linear functors from the $\Bbbk$-algebroid $\mathcal{A}$ to $\Bbbk$-**mat**. Using only categorical constructions provided by the axioms of the abelian category $\Bbbk$-**mat**, we prove that the functor category with values in an abelian category is an abelian category.

A thesis using the language of category theory would be incomplete without mentioning Yoneda's lemma. In Section 6 we formulate Yoneda's lemma, citing a proof in [1, 2.2], and define the Yoneda embedding of a category $\mathcal{C}$ into the functor category from $\mathcal{C}^{\mathrm{op}}$ to **Set**. For our purposes we define our $\Bbbk$-linear version of Yoneda's embedding of the opposite $\Bbbk$-algebroid $\mathcal{A}^{\mathrm{op}}$ into the functor category $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$. In the rest of Section 6 we define projective objects and proof that the so-called Yoneda projectives are in fact projective objects, closing with a statement that $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ has enough projectives.

The last two sections deal with the Hom-structure of $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$. Since the morphisms between two functors are natural transformations, they must fulfill certain naturality constraints leading to Sylvester equations. In Section 7 we also give a probabilistic algorithm for the direct sum decomposition of functors. The proof of the decomposition is a linear analogue of the σ lemma.

In Section 8 we give Hom-based invariants in order to distinguish between non-isomorphic objects in our functor category.

The result of this thesis is the GAP package `CatReps` [12] which was originally meant to wrap Peter Webb's pre-package `catreps` [15] into the categorical framework

offered by CAP [11]. Since the category of representations is just an application of the package `FunctorCategories` [9], most of the functionality used by `CatReps` is provided through `FunctorCategories`. Currently `CatReps` consists of the methods

- `ConvertToMapOfFinSets`,
- `ConcreteCategoryForCAP`,
- `RightQuiverFromConcreteCategory`,
- `RelationsOfEndomorphisms`,
- `AsFpCategory`,
- `Algebroid`.

The content of this package might migrate to `FunctorCategories` in the future. All the algorithms developed and implemented while working on this thesis are included as code listings in the appendix.

2 Quivers and categories

This section serves two purposes: On the one hand, it is an introduction to quivers and category theory. On the other hand it introduces concrete categories which we want to represent, and all the additional constructions that are needed to that goal.

2.1 Quivers

In this section, we first define the notion of a quiver. Very much like a monoid is generated by a set, a category is generated by a quiver. [1]

Definition 2.1.1. (Quiver)
A directed graph or quiver q consists of a class of objects (or vertices) $q_0 = \operatorname{Obj} q$ and a class of morphisms (or arrows) $q_1 = \operatorname{Mor} q$ together with two defining maps

$$s, t \colon q_1 \rightrightarrows q_0$$

s called source and t called target.

In the next definition we are giving a new characterization for q_1 by looking at all arrows between two fixed objects.

Definition 2.1.2. (Hom-set of a (locally) small quiver)

(1) Given two objects $M, N \in q_0$ we write $\operatorname{Hom}_q(M, N)$ or $q(M, N)$ for the fiber $(s,t)^{-1}(\{(M,N)\})$ of the product map $(s,t) : q_1 \longrightarrow q_0 \times q_0$ over the pair $(M, N) \in q_0 \times q_0$. This is the class of all morphisms with source $= M$ and target $= N$. We indicate this by writing $\varphi : M \longrightarrow N$ or $M \xrightarrow{\varphi} N$. Hence q_1 is the disjoint union

$$\dot{\bigcup_{M,N \in q_0}} \operatorname{Hom}_q(M, N) = q_1.$$

As usual we define $\operatorname{End}_q(M) := \operatorname{Hom}_q(M, M)$.

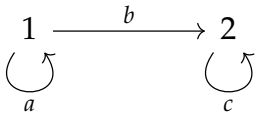
(2) If the class $\operatorname{Hom}_q(M, N)$ is a set for all pairs (M, N) then we call the quiver locally small. We therefore talk about hom-sets. If additionally, q_0 is a set, then the quiver is called small.

(3) A quiver with a finite set of objects and a finite set of morphisms is called a finite quiver.

(4) When we don't assume the category to be locally small, but still talk about its hom-sets, we mean the class of morphisms, if we don't explicitly use the fact that it's a set of morphisms.

(5) If the hom-set itself is an object in our quiver, as is the case e.g. in **Set**, where $\operatorname{Hom}_{\mathbf{Set}}(M, N) = N^M$ is again a set, we are talking about the internal hom. Otherwise the hom-set exists outside of the category and we call it the external hom.

Example 2.1.3 (Quiver with 2 objects and 3 morphisms)**.**



The objects of this quiver q are $q_0 = \{1,2\}$, and the morphisms are $q_1 = \{a,b,c\}$ with
$s(a) = 1 = t(a)$, $s(c) = 2 = t(c)$ and $s(b) = 1, t(b) = 2$.
Thus $\mathrm{End}_q(1) = \{a\}, \mathrm{End}_q(2) = \{c\}$ and $\mathrm{Hom}_q(1,2) = \{b\}$ whereas $\mathrm{Hom}_q(2,1) = \emptyset$.

In `QPA` this quiver is encoded as `q(2)[a:1->1,b:1->2,c:2->2]` where the first `(2)` in parentheses stands for the total number of objects.

Definition 2.1.4 (Opposite quiver)**.** For a quiver q we can define the opposite quiver q^{op} which has all objects of the original quiver q, i.e.

$$q_0^{\mathrm{op}} = q_0$$

But all arrows in q^{op} are facing in the other direction, i.e. have their source and target swapped:

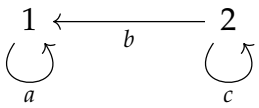
$$\forall (a : s(a) \to t(a)) \in q_1 \; \exists (a' : t(a) \to s(a)) \in q_1^{\mathrm{op}},$$

i.e. $s(a') = t(a)$ and $t(a') = s(a)$.

The opposite quiver (and especially later the opposite category) is a very helpful concept in that we can prove one statement for a quiver q and argument by duality that there exists a dual or co statement in the opposite quiver, without needing to prove the same thing twice. Especially in section 5 the opposite category becomes important.

The opposite quiver of 2.1.3 is

Example 2.1.5 (Opposite quiver)**.**



Definition 2.1.6 (Composable arrows; path in a quiver)**.** [2] Let q be a quiver.

(1) We say two arrows $a, b \in q_1$ are composable if $t(a) = s(b)$ or $t(b) = s(a)$. In this case we can write a sequence of composable arrows $p = a_1 a_2 \cdots a_n$ where $t(a_i) = s(a_{i+1})$ for $i = 1, \ldots, n-1$. We call this sequence a path from $s(a_1)$ to $t(a_n)$ and the integer $n \in \mathbb{Z}_{\geq 0}$ the length $l(p)$ of the path p. Although it may not be an arrow, we can define the source and target of a path $p = a_1 \cdots a_n$ as $s(p) := s(a_1)$ and $t(p) := t(a_n)$. Then again we define two paths p and p' as composable, if $t(p) = s(p')$ (or $t(p') = s(p)$) and we call p' (or $p'p$) the concatenation or composition of the two paths. We can identify each arrow again as a path of length 1. A path $p = a_1 \cdots a_n$ with $s(a_1) = t(a_n)$, i.e. $s(p) = t(p)$, is called cyclic.

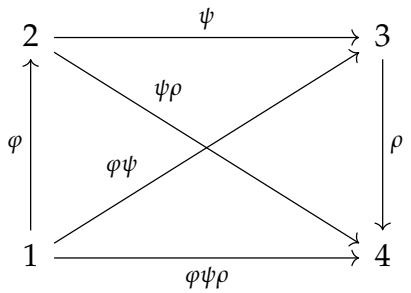
(2) For an endomorphism $a \in \mathrm{End}_q(M)$ we write a^n for $aa \cdots a$ (n times).

(3) In the case of $n = 0$ an empty path whose source and target are the vertex $i \in q_0$ is called the trivial path at i and is denoted e_i. Note that the composition of paths $e_i e_i$ has length zero starting at i therefore $e_i^2 = e_i$, in other words, each e_i is an idempotent.

Lemma 2.1.7. Let q be a quiver. If there is a path of length at least $|q_0|$, then there are cyclic paths, and thus infinitely many paths.[5]

Proof. Assume that there exists a path of length greater or equal to $|q_0|$. Then there exists a path of length $n = |q_0|$, say $\alpha_1 \cdots \alpha_n$. Consider the vertices $x_i = s(\alpha_i)$ for $1 \leq i \leq n$ and $x_{n+1} = t(\alpha_n)$. Then these are $n+1$ vertices, thus there has to exist $i < j$ with $x_i = x_j$. Let $\omega = \alpha_i \cdots \alpha_{j-1}$, this is a path with source and target $x_i = x_j$, thus a cyclic path. But then ω^m is a path for any natural number m. The path ω has length $j - i \geq 1$, thus ω^m has length $m(j-i)$. This shows that these paths are pairwise different. □

Example 2.1.8. (A quiver with no cycles)



The longest path $1 \to 2 \to 3 \to 4$ has length 3. If after the object 4 another arrow would go to either $1, 2, 3$ or 4 itself, we would have a cyclic path and thus infinitely many paths.

2.2 Categories

Definition 2.2.1. (Category)
A <u>category</u> $\mathcal{C}$ is a quiver with two further maps:

(id) The <u>identity map</u> $1_{()}$ mapping every object $X \in \mathcal{C}_0$ to its <u>identity morphism</u> 1_X:

$$\mathcal{C}_0 \xrightarrow{1} \mathcal{C}_1$$

(μ) And for any two <u>composable</u> morphisms φ and $\psi \in \mathcal{C}_1$, i.e. with $t(\varphi) = s(\psi)$, the <u>composition map</u> μ, which maps $\varphi, \psi \in \mathcal{C}_1 \times \mathcal{C}_1$ to $\mu(\varphi, \psi) \in \mathcal{C}_1$ which we also write as $\varphi\psi$.

$$\mathcal{C}_1 \times \mathcal{C}_1 \xrightarrow{\mu} \mathcal{C}_1$$

The defining properties for 1 and μ are:

(1) $s(1_M) = M = t(1_M)$, i.e.
$1_M \in \mathrm{End}_{\mathcal{C}} \forall M \in \mathcal{C}$.

(2) $s(\varphi\psi) = s(\varphi)$ and
$t(\varphi\psi) = t(\psi)$
for all composable morphisms $\varphi, \psi \in \mathcal{C}$.

$$\mu : \mathrm{Hom}_{\mathcal{C}}(M, L) \times \mathrm{Hom}_{\mathcal{C}}(L, N) \longrightarrow \mathrm{Hom}_{\mathcal{C}}(M, N)$$

(3) $(\varphi\psi)\rho = \varphi(\psi\rho)$ [associativity of composition]

(4) $1_{s(\varphi)}\varphi = \varphi = \varphi 1_{t(\varphi)}$ [unit property]
The identity is a left and right unit of the composition.

So with categories you always answer the four questions

- What are the objects?
- What are the morphisms? (which includes the question What are the identity morphisms?)
- How do you compose morphisms?
- Why is the composition associative?

Example 2.2.2 (The category FinSets)**.**

- The objects are finite sets.
- The morphisms are functions between sets, with the identity morphism on the set X defined as $1_X := \mathrm{id}_X : X \to X; x \mapsto x$.
- Composition of morphisms is the usual composition of functions: $f : X \to Y; x \mapsto f(x)$,
  $g : Y \to Z; y \mapsto g(y)$. Then $f \circ g : X \to Z; x \mapsto g(f(x))$.[3]
- The composition is associative: With f, g as above, and $h : Z \to W$, $(f \circ (g \circ h))(x) = (g \circ h)(f(x)) = h(g(f(x))) = h((f \circ g)(x)) = ((f \circ g) \circ h)(x), \forall x \in X$.

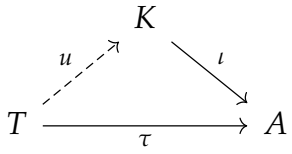
Definition 2.2.3 (Monomorphism and epimorphism)**.** A morphism $f : A \to B$ in a category is

- a <u>monomorphism</u> if for any parallel morphisms $h, k : Z \rightrightarrows A, hf = kf$ implies that $h = k$
- an <u>epimorphism</u> if for any parallel morphisms $h, k : B \rightrightarrows C, fh = fk$ implies that $h = k$.

We write $f : A \hookrightarrow B$ for a monomorphism and $f : A \twoheadrightarrow B$ for an epimorphism.

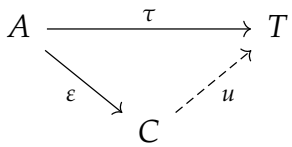
Definition 2.2.4 (Lift and colift, dominate and codominate)**.**
Let $A, K, C, T \in \mathcal{C}_0$ be objects in a category $\mathcal{C}$.

(1) For two morphisms with the same target $\iota : K \to A, \tau : T \to A$ we call a morphism $u : T \to K$ a <u>lift of τ along ι</u> if $u\iota = \tau$.



In this case we say that ι <u>dominates</u> τ

(2) For two morphisms with the same source $\varepsilon : A \to C, \tau : A \to T$ we call a morphism $u : C \to T$ a <u>colift of τ along ε</u> if $\varepsilon u = \tau$.



In this case we say that ε <u>codominates</u> τ.

Remark 2.2.5 (Lift along mono, colift along epi)**.** If they exist, lifts along monos and colifts along epis are necessarily unique.

Proof. In the above diagram for ι a monomorphism and as lifts of τ along ι we have the two competing $u, v : T \rightrightarrows K$, both with $u\iota = \tau = v\iota$, the definition of ι being a monomorphism implies $u = v$. Dually for colifts along epis. □

Definition 2.2.6. (Subobject; Subcategory)

(1) A subobject $K \leq M$ is an equivalence class of monos with M as target under the equivalence relation of mutual dominance.

(2) A subcategory $\mathcal{C} \leq \mathcal{D}$ is a subobject in the category **Cat** of categories.

Definition 2.2.7. (Concrete category) A concrete category is a category $\mathcal{C}$ together with a faithful functor $U : \mathcal{C} \to \mathbf{Set}$.
This functor U typically carries an object of $\mathcal{C}$ to its *underlying set* and is called *forgetful functor*. For more on functors see the next subsection, for more on the forgetful functor see 4.1.

Definition 2.2.8 (Finite concrete category)**.** The category FinSets is a finite subcategory of **Set**. Both are examples of concrete categories. A finite concrete category $\mathcal{C}$ is a finite subcategory of FinSets.

Remark 2.2.9. (Endomorphism monoid; explicitly cyclic monoid) Let $\mathcal{C}$ be a category and $M \in \mathcal{C}_0$ an object. Then the endomorphism set $\mathrm{End}_{\mathcal{C}}(M)$ together with the composition of morphisms as operation, and 1_M as identity element, is a monoid, called the endomorphism monoid. We call the endomorphism monoid explicitly cyclic if it is generated by one object, i.e. of the form

$$\{1_M = \alpha^0, \alpha^1, \alpha^2, \dots\}$$

for an $\alpha \in \mathrm{End}_{\mathcal{C}}(M)$. If the endomorphism monoid is finite and explicitly cyclic, it is of the form $\{1_M, \alpha^1, \dots, \alpha^n\}$ for an $n \in \mathbb{N}$ and an $\alpha \in \mathrm{End}_{\mathcal{C}}(M)$.

Proof. The properties of a monoid are precisely the associativity of composition and the unit property from 3 and 4. □

Definition 2.2.10 (Isomorphism)**.** An isomorphism in a category $\mathcal{C}$ is a morphism $f : X \to Y$ for which there exists a morphism $g : Y \to X$ so that $fg = 1_X$ and $gf = 1_Y$. The objects X and Y are isomorphic whenever there exists an isomorphism between X and Y, in which case one writes $X \cong Y$.

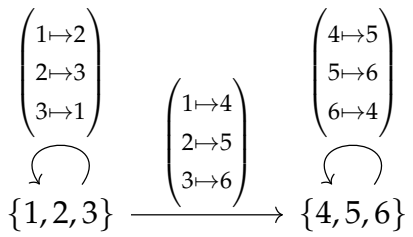
Definition 2.2.11 (The automorphism set $\mathrm{Aut}_{\mathcal{C}}(M)$)**.** Let $\mathcal{C}$ be a category and $M \in \mathcal{C}_0$ an object. The subset $\mathrm{Aut}_{\mathcal{C}}(M) \subset \mathrm{End}_{\mathcal{C}}(M)$ consisting of all endomorphisms on M that are isomorphisms is called the automorphism set. It is a group and therefore also called the automorphism group on M.

Definition 2.2.12 (Skeletal category, skeleton)**.** A category $\mathcal{C}$ is skeletal if $A \cong A'$ implies $A = A'$ for all objects $A, A' \in \mathcal{C}_0$. We define a skeleton $S\mathcal{C}$ to be a skeletal subcategory of $\mathcal{C}$ whose inclusion functor exhibits it as equivalent to $\mathcal{C}$.

Example 2.2.13 ($\Bbbk$-$\mathbf{vec}^{\mathrm{fd}}$ is not skeletal)**.** The category $\Bbbk$-$\mathbf{vec}^{\mathrm{fd}}$ of finite-dimensional vector spaces over a field $\Bbbk$ is obviously not skeletal. For example, the standard vector space $\Bbbk^3$ and the quotient vector space $\Bbbk[x]/(x^3)$ are isomorphic but not equal.

Example 2.2.14 (The finite concrete category C_3C_3)**.** The following category has as morphisms the two identities $1_{\{1,2,3\}}, 1_{\{4,5,6\}}$, the other four isomorphisms $(2,3,1)$, $(3,1,2)$, $(5,6,4)$, $(6,4,5)$ and their pre- and post-compositions with the non-endomorphism

$(1 \mapsto 4, 2 \mapsto 5, 3 \mapsto 6)$. We are only displaying the three morphisms $(2,3,1), (5,6,4)$ and $(1 \mapsto 4, 2 \mapsto 5, 3 \mapsto 6)$, the other morphisms are implied.



2.3 Functors

Categories are themselves objects in the category of categories, which leads to a question: What is a morphism between categories?

Definition 2.3.1. (Functor)
A functor $F : \mathcal{C} \to \mathcal{D}$, between categories $\mathcal{C}$ and $\mathcal{D}$, consists of the following data:

- An object $Fc \in \mathcal{D}_0$, for each object $c \in \mathcal{C}_0$.
- A function $Ff : Fc \to Fc' \in \mathcal{D}_1$, for each morphism $f : c \to c' \in \mathcal{C}_1$, so that the source and target of Ff are, respectively, equal to F applied to the source or target of f, in other words, $s(Ff) = Fs(f)$ and $t(Ff) = Ft(f)$.

The assignments are required to satisfy the following two functoriality axioms:

- For any composable pair $f : M \to N, g : N \to L \in \mathcal{C}_1, F f \cdot F g = F(f \cdot g)$.
- For each object $c \in \mathcal{C}_0, F(1_c) = 1_{Fc}$.

Put concisely, a functor consists of a mapping on objects and a mapping on morphisms that preserves all of the structure of a category, namely domains and codomains, composition, and identities.

So with functors you always answer the four questions

- How does it work on objects?
- How does it work on morphisms?
- Why does it respect composition?
- Why does it respect identity morphisms?

We have already seen an example for a functor in definition 2.1.2 where we defined the hom-set $\mathrm{Hom}(M, N)$ between two objects M and N. There are two ways to leave blank one of the objects and thus define the

Example 2.3.2. (partial Hom-functor) Let $\mathcal{C}$ be a category and $P \in \mathcal{C}_0$ any object. The Hom-functor, also called partial Hom-functor,

(1) $\mathrm{Hom}(P, -)$ is a functor from $\mathcal{C}$ to $\mathcal{C}_1$ where objects in $\mathcal{C}_1$ are the hom-sets $\mathrm{Hom}(P, N)$, and morphisms are maps from one hom-set to another.

- $\mathrm{Hom}(P, -)$ works on objects by mapping the object $N \in \mathcal{C}_0$ to the hom-set $\mathrm{Hom}(P, N) \in \mathcal{C}_1$.

- $\mathrm{Hom}(P,-)$ works on morphisms by mapping the morphism $(f : M \to N) \in \mathcal{C}_1$ to the transformation

$$f_* := \mathrm{Hom}(P,f) : \mathrm{Hom}(P,M) \to \mathrm{Hom}(P,N); \varphi \mapsto \varphi f, \tag{2.3.1}$$

  so for every morphism $\varphi \in \mathrm{Hom}(P,M)$, you post-compose $f \in \mathrm{Hom}(M,N)$ to get a new morphism $\varphi f \in \mathrm{Hom}(P,N)$.

(2) $\mathrm{Hom}(-,P)$ is a functor from $\mathcal{C}$ to $\mathcal{C}_1$ where objects in $\mathcal{C}_1$ are the hom-sets $\mathrm{Hom}(N,P)$, and morphisms are maps from one hom-set to another.

- $\mathrm{Hom}(-,P)$ works on objects by mapping the object $N \in \mathcal{C}_0$ to the hom-set $\mathrm{Hom}(N,P) \in \mathcal{C}_1$.
- $\mathrm{Hom}(-,P)$ works on morphisms by mapping the morphism $(f : M \to N) \in \mathcal{C}_1$ to the transformation

$$f^* := \mathrm{Hom}(f,P) : \mathrm{Hom}(N,P) \to \mathrm{Hom}(M,P); \varphi \mapsto f\varphi, \tag{2.3.2}$$

  so for every morphism $\varphi \in \mathrm{Hom}(N,P)$, you pre-compose $f \in \mathrm{Hom}(M,N)$ to get a new morphism $f\varphi \in \mathrm{Hom}(M,P)$.

The important difference between these two functors was how they worked on morphisms. If in both cases we take a morphism $f : M \to N$ as given, then we have to arrange the source and target for $\mathrm{Hom}(P,f)$ and $\mathrm{Hom}(f,P)$ according to the post-composition and pre-composition. Thus if we wanted $\mathrm{Hom}(f,P)$ to be defined by pre-composition $\varphi \mapsto f\varphi$, then we were forced to invert M and N as source and target to get $\mathrm{Hom}(f,P) : \mathrm{Hom}(N,P) \to \mathrm{Hom}(M,P)$. This process of inverting source and target is caught in the following definition.

Definition 2.3.3. (covariant / contravariant functor)[4]
The way we defined a functor in definition 2.3.1 was in the covariant way.
A contravariant functor $F : \mathcal{C} \to \mathcal{D}$ works on objects the same way as a covariant one, i.e. an object $Fc \in \mathcal{D}_0$ for each object $c \in \mathcal{C}_0$. For morphisms on the other hand, we have a morphism $Ff : Fc' \to Fc \in \mathcal{D}_1$ for each morphism $f : c \to c' \in \mathcal{C}_1$, so that $s(Ff) = Ft(f)$ and $t(Ff) = Fs(f)$. The functoriality axioms are also inverted for a contravariant functor: For any composable pair, $f : M \to N, g : N \to L \in \mathcal{C}_1$, $Fg \cdot Ff = F(f \cdot g)$. For the identity morphisms, it is again the same as in the covariant case: For each object $c \in \mathcal{C}_0$, $F(1_c) = 1_{Fc}$.

Definition 2.3.4. (Full functor)[5]
A functor $F : \mathcal{C} \to \mathcal{D}$ is full if $\forall x, y \in \mathcal{C}_0$, the map $\mathcal{C}(x,y) \to \mathcal{D}(Fx,Fy)$ is surjective.

Definition 2.3.5. (Faithful functor)
A functor F as in 2.3.4 is faithful if $\forall x, y \in \mathcal{C}_0$, the map $\mathcal{C}(x,y) \to \mathcal{D}(Fx,Fy)$ is injective.

Definition 2.3.6. (Essentially surjective on objects)
A functor F as in 2.3.4 is essentially surjective on objects if for every object $d \in \mathcal{D}_0$ there is some $c \in \mathcal{C}_0$ such that d is isomorphic to Fc.

Definition 2.3.7. (Embedding)
A faithful functor that is injective on objects is called an embedding and identifies the source category as a subcategory of the target. In this case, faithfulness implies that the functor is (globally) injective on arrows.

Definition 2.3.8. (Full embedding / full subcategory)
A full and faithful functor, called fully faithful for short, that is injective on objects defines a full embedding of the source category into the target category. The source then defines a full subcategory of the target category.

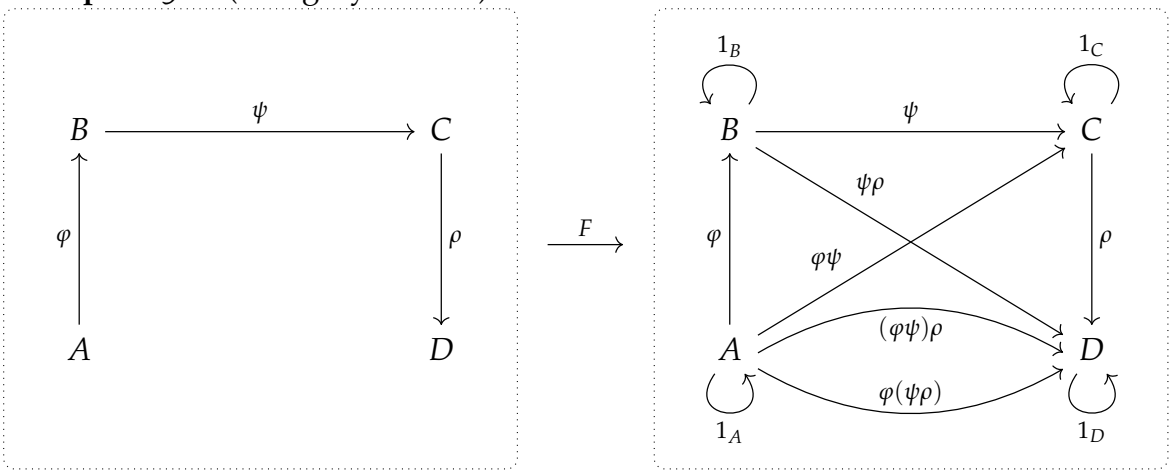
Theorem 2.3.9. *(characterizing equivalences of categories)*[6] *A functor defining an equivalence of categories is full, faithful, and essentially surjective on objects. Assuming the axiom of choice, any functor with these properties defines an equivalence of categories.*

As we have seen, every category is a quiver, but in general, to become a category, a quiver is lacking identity morphisms and the composition of morphisms. To be more precise, there is a functor U from the category of categories CAT to the category of quivers Quiv, called the underlying quiver or forgetful functor.

$$\mathrm{Cat} \xrightarrow{\quad U \quad} \mathrm{Quiv}$$

mapping every object $M \in \mathcal{C}_0$ to the same objects in q_0, mapping every arrow $\varphi \in \mathcal{C}_1$ to an arrow $a \in q_1$, respecting source and target, but forgetting the special role of the identity morphisms and of the composition morphisms.

Example 2.3.10 (Category closure)**.**



We can think of a quiver as a prototype for a category. That means we can construct the missing data for a category from a quiver by adding the identity morphisms and the composed arrows.

2.4 Natural transformations

With fixed categories $\mathcal{C}$ and $\mathcal{D}$ we can consider functors $F, G \in \mathrm{Hom}(\mathcal{C}, \mathcal{D})$ themselves as objects in the category $\mathrm{Hom}(\mathcal{C}, \mathcal{D})$ of functors between $\mathcal{C}$ and $\mathcal{D}$. In this functor category, the morphisms between two functors are called natural transformations.

Definition 2.4.1. (Natural transformations)
Given categories $\mathcal{C}$ and $\mathcal{D}$ and functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$, a natural transformation $\alpha : F \Rightarrow G$ consists of:

- a morphism $\alpha_c : Fc \to Gc \in \mathcal{D}_1$ for each object $c \in \mathcal{C}_0$, the collection of which define the components of the natural transformation, so that, for any morphism $f : c \to c' \in \mathcal{C}_1$, the following square of morphisms in $\mathcal{D}$

$$\begin{array}{ccc} Fc & \xrightarrow{\alpha_c} & Gc \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle Gf} \\ Fc' & \xrightarrow{\alpha_{c'}} & Gc' \end{array}$$

  commutes, i.e., has a a common composite $Fc \to Gc' \in \mathcal{D}_1$. This means explicitly that

$$Ff\alpha_{c'} = \alpha_c Gf, \forall f : c \to c' \in \mathcal{C}_1. \tag{2.4.1}$$

Definition 2.4.2 (Monic, epic and iso natural transformations)**.** When each component α_C is a...

- monomorphism, we call α a natural monomorphism,
- epimorphism, we call α a natural epimorphism,
- isomorphism, we call α a natural isomorphism.

2.5 The functor category

Definition 2.5.1. (The functor category)[7]
Given categories $\mathcal{C}$ and $\mathcal{D}$, the functor category - written $\mathcal{D}^{\mathcal{C}}$, $\mathrm{Hom}(\mathcal{C}, \mathcal{D})$ or $[\mathcal{C}, \mathcal{D}]$ - is the category whose

- objects are functors $F : \mathcal{C} \to \mathcal{D}$
- morphisms are natural transformations between these functors.

3 Limits and colimits

3.1 Limit and colimit of a functor

Definition 3.1.1. (Source of a functor) Let $D : \mathbf{I} \to \mathcal{C}$ be a functor. A source of D consists of the following data:

(1) An object $S \in \mathcal{C}$.

(2) A dependent function s mapping an object $i \in \mathbf{I}_0$ to a morphism $s(i) : S \to D(i)$ such that for all $i, j \in \mathbf{I}, \iota : i \to j$, we have $D(\iota) \cdot s(i) = s(j)$.

Definition 3.1.2. (Limit and colimit of a functor) Let $D : \mathbf{I} \to \mathcal{C}$ be a functor. A limit of D consists of the following data:

(1) A source of D given by the data $(\lim D, (\lambda(i) : \lim D \to D(i))_{i \in \mathbf{I}_0})$.

(2) A dependent function u, called the lift, mapping every source $\tau = (T, (\tau(i) : T \to D(i))_{i \in \mathbf{I}})$ to a morphism $u(\tau) : T \to \lim D$ such that $\lambda(i) \cdot u(\tau) = \tau(i)$ for all $i \in \mathbf{I}$. This dependent function u is unique with this property.

A colimit in $\mathcal{C}$ is a limit in $\mathcal{C}^{\mathrm{op}}$.

Definition 3.1.3 (Limits of type **I**)**.** Let **I** be a category. We say a category $\mathcal{C}$ has limits of type **I** if it is equipped with a dependent function λ mapping a functor $D : \mathbf{I} \to \mathcal{C}$ to a limit $(\lim D, \lambda_D, u_D)$ of D. We say $\mathcal{C}$ has colimits of type **I** if $\mathcal{C}^{\mathrm{op}}$ has limits of that type.

Example 3.1.4. Depending on **I** some limits and colimits have special names:

generating quiver of **I**	limit	colimit
$\emptyset$	terminal object	initial object
$\bullet \quad \bullet$	binary product	binary coproduct
$\bullet \to \bullet \leftarrow \bullet$	binary pullback	-
$\bullet \leftarrow \bullet \to \bullet$	-	binary pushout
$\bullet \rightrightarrows \bullet$	binary equalizer	binary coequalizer

3.2 Preserving limits and colimits

Lemma 3.2.1.

(1) A morphism $f : a \to b$ is a monomorphism if and only if the pullback of f and f exists and is a, together with the identity maps $1_a : a \to a$. In other words, $f : a \to b$ is a monomorphism if and only if the commutative square

$$\begin{array}{ccc} a & \xrightarrow{1_a} & a \\ {\scriptstyle 1_a}\downarrow & & \downarrow{\scriptstyle f} \\ a & \xrightarrow{f} & b \end{array}$$

is a pullback square.[8]

(2) A morphism $f : a \to b$ is an epimorphism if and only if the pushout of f and f exists and is b, together with the identity maps $1_b : b \to b$. In other words, $f : a \to b$ is a monomorphism if and only if the commutative square

$$\begin{array}{ccc} a & \xrightarrow{f} & b \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle 1_b} \\ b & \xrightarrow{1_b} & b \end{array}$$

is a pushout square.[9]

Corollary 3.2.2.

(1) Being a monomorphism is a "limit property": more precisely, any functor which preserves pullbacks (in particular any functor which preserves finite limits, in particular any functor which preserves all limits) preserves monomorphisms.

(2) Being an epimorphism is a "colimit property": more precisely, any functor which preserves pushouts (in particular any functor which preserves finite colimits, in particular any functor which preserves all colimits) preserves epimorphisms.[10]

3.3 Left-exact, right-exact and exact functors

In the following definitions, we define different subclasses of functors. These adjectives often come in opposite pairs, so that you may be tempted to think, duality lets you just swap all the adjectives for the opposite ones, but be careful there. E.g. when $\mathrm{Hom}(P,-)$ is a covariant, left-exact functor, the opposite $\mathrm{Hom}(-,P)$ is a contravariant, but still left-exact functor. But their respective right-exactedness is equivalent to dual concepts concerning projective and injective objects.

Definition 3.3.1. (Exact functor)[11]
A functor between finitely complete categories is called right exact or finitely co-continuous if it preserves finite colimits, and left exact or finitely continuous if it preserves finite limits.

Lemma 3.3.2. The hom functors $\mathrm{Hom}(P,-)$ and $\mathrm{Hom}(-,P)$ from 2.3.2 are left exact, i.e. respect monos.

Proof. Let $f : M \to N \in \mathcal{C}_1$ be a monomorphism.

(1) $\mathrm{Hom}(P,-)$ is left exact: Let $h,k : L \rightrightarrows M$ such that

$$\begin{aligned}
\mathrm{Hom}(P,h)\cdot\mathrm{Hom}(P,f) &= \mathrm{Hom}(P,k)\cdot\mathrm{Hom}(P,f)\\
\Rightarrow \mathrm{Hom}(P,hf) &= \mathrm{Hom}(P,kf)\\
\Rightarrow \varphi hf &= \varphi kf \quad \forall \varphi \in \mathrm{Hom}(P,M)\\
\Rightarrow \varphi h &= \varphi k \quad \forall \varphi \in \mathrm{Hom}(P,M) \text{ since } f \text{ is a monomorphism}\\
\Rightarrow \mathrm{Hom}(P,h) &= \mathrm{Hom}(P,k)
\end{aligned}$$

Therefore $\mathrm{Hom}(P,f)$ is also a monomorphism, i.e. $\mathrm{Hom}(P,-)$ is left exact.

(2) The dual statement is that $\mathrm{Hom}(-,P)$ is left exact.

□

3.4 A hierarchy of categorical doctrines

A category having or lacking limits and colimits of a certain type formalizes the notion of what one can or cannot *do* in a category. Since we are *doing* constructive category theory, this all boils down to which limits and colimits we can compute, i.e. for which we have algorithms, and what needs to be true for the category in order for those algorithms to terminate with a correct output. Just as in algebra words like "field" or "ring" or "abelian group" are established names and adjectives for sets with additional structure, in category theory we give special names for categories

which have certain limits. This is summarized under the vaguely defined notion of "doctrine".

We say a category is of a certain categorical doctrine, if it has all categorical operations appearing in the defining axioms of the doctrine, in particular if it has all limits and colimits of a required set of types. In this thesis, we will define a categorical doctrine by specifying a set of algorithms for all existential quantifyers and disjunctions appearing in the defining axioms of the doctrine, in particular by specifying all algorithms needed to compute the required (co-)limits.

Being a skeletal category (`IsSkeletalCategory`) is not a categorical doctrine. Still, it is of high computational benefit if we can represent a category by a skeletal model.

In this subsection, we give explicit definitions for the (co-)limits in 3.1.4 and define the doctrines for our categories with such (co-)limits. The doctrine we are interested in is that of an Abelian category with enough projectives and enough injectives. For now we will describe the doctrines up to and including Abelian categories, and leave projective and injective objects for section 5. In section 3 we will work on the doctrine of $\Bbbk$-algebroids or $\Bbbk$-linear categories.

Doctrine 3.4.1 (Category)**.** The doctrine `IsCategory` involves the two algorithms

- `Source`[12],
- `Range`,
- `PreCompose`,
- `IdentityMorphism`.

3.4.1 Ab-categories

We are starting simple with Ab-categories. All we need is an abelian group structure on the hom-set between two objects.

Definition 3.4.2 (Zero morphism)**.**
A zero morphism from M to N is the neutral element of the abelian group $\mathrm{Hom}_{\mathcal{C}}(M,N)$.

Note that every hom-set has its own unique zero morphism. E.g. in $\Bbbk$-**mat** the 2×3 zero-matrix $0_{2,3}\in \mathrm{Hom}_{\Bbbk\text{-}\mathbf{mat}}(2,3)$ is different from the 4×4 zero-matrix $0_{4,4}\in \mathrm{Hom}_{\Bbbk\text{-}\mathbf{mat}}(4,4)$.

Definition 3.4.3 (Ab-category)**.** An Ab-category (also called pre-additive category) is a category in which all homomorphism sets are abelian groups, and composition distributes over addition.
In other words, a category $\mathcal{C}$ is an Ab-category if for every pair of objects $M,N\in\mathcal{C}_0$, $(\mathrm{Hom}_{\mathcal{C}}(M,N),+)$ is an abelian group (with the zero morphism $0_{M,N}$ as the neutral element), and for all morphisms $\gamma,\delta\in\mathrm{Hom}_{\mathcal{C}}(M,N), \alpha,\beta\in\mathrm{Hom}_{\mathcal{C}}(N,L)$

$$(\gamma+\delta)\alpha=\gamma\alpha+\delta\alpha \quad \text{and} \tag{3.4.1}$$
$$\gamma(\alpha+\beta)=\gamma\alpha+\gamma\beta. \tag{3.4.2}$$

Doctrine 3.4.4 (Ab-category)**.** The doctrine `IsAbCategory` therefore involves algorithms for `IsCategory` and

- `AdditionForMorphisms`,
- `AdditiveInverseForMorphisms`,
- (`SubtractionForMorphisms`),

- `ZeroMorphism`,
- `IsZeroForMorphisms`.

Definition 3.4.5 (Ab-functor)**.** A functor between Ab-categories is called an Ab-functor if in the functor definition 2.3.1 the function Ff for each morphism f is a homomorphism of abelian groups, i.e. $F(f+g) = Ff + Fg$.

3.4.2 Categories with a zero object

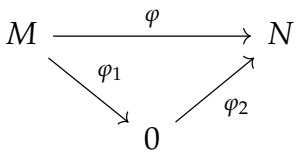
Remark 3.4.6 (Terminal object, initial object, zero object)**.** The limit / colimit of $\emptyset$.

(1) A terminal object T in a category $\mathcal{C}$ is an object such that $\mathrm{Hom}_{\mathcal{C}}(-, T)$ is a singleton.

(2) An initial object I in a category $\mathcal{C}$ is an object such that $\mathrm{Hom}_{\mathcal{C}}(I, -)$ is a singleton.

(3) An object Z or 0 is a zero object if it is both initial and terminal. (Bilimit of $\emptyset$).

Definition 3.4.7 (Zero morphism)**.**
A zero morphism in a category with a zero object 0 is a morphism factoring over 0, i.e. $\varphi : M \to N$ is called a zero morphism, if



$\exists \varphi_1 : M \to 0, \varphi_2 : 0 \to N$
such that $\varphi = \varphi_1 \varphi_2$.

Since the zero object 0 is both initial and terminal, a zero morphism is uniquely defined by its source and target, thus we can talk about *the* zero morphism from M to N, which we denote by $0_{M,N}$.

Doctrine 3.4.8 (Category with zero object)**.** The doctrine `IsCategoryWithZeroObject` therefore involves algorithms for

- `ZeroObject`,
- `UniversalMorphismFromZeroObject`,
- `UniversalMorphismIntoZeroObject`.

If an Ab-category has a zero object, then both notions of a zero morphism coincide.

3.4.3 Additive categories

The definition of the binary operation $+$ in a pre-additive category came as arbitrary outside data and could be defined in multiple ways. A category with the following limits is additive in at most one way, warrenting the name additive category.

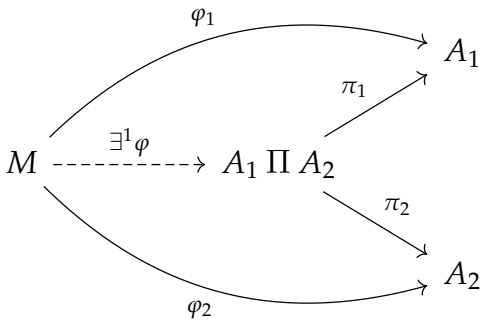
Definition 3.4.9 (Product, coproduct)**.** The product / coproduct is the limit / colimit of a functor $F : \mathbf{I} \to \mathcal{C}$, where $\mathbf{I}$ is generated by a quiver with no arrows, i.e. $\mathbf{I}$ is a category where the only morphisms are the identities (a so-called set). Another way to express this:
Let I be an index set and $\{A_i\}_{i \in I}$ a family of objects in a category $\mathcal{C}$.

(prod) The <u>product</u> of the family $\{A_i\}_{i\in I}$ is an object ΠA_i together with a family of morphisms

$$\{\pi_i : \Pi A_i \to A_i\}$$

called <u>projections</u>, such that the following universal property is satisfied:
For any object $M \in \mathcal{C}_0$ and any family $\{\varphi_i : M \to A_i\}_{i\in I}$ of morphisms, there exists a unique morphism $\varphi : M \to \Pi A_i$ called the <u>product morphism</u> such that

$$\varphi\pi_i = \varphi_i \,\forall i \in I.$$

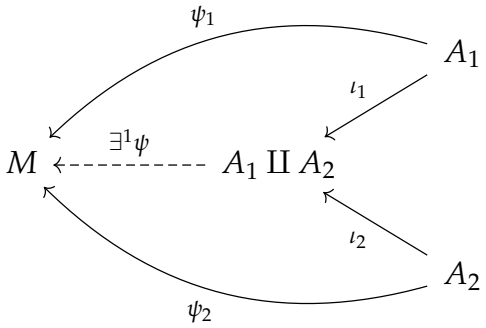


(coprod) The dual notion to product is the <u>coproduct</u> of the family $\{A_i\}_{i\in I}$, that is an object $\amalg A_i$ together with a family of morphisms

$$\{\iota_i : A_i \to \amalg A_i\}$$

called <u>coprojections</u> or sometimes <u>injections</u>, <u>inclusions</u> or <u>embeddings</u>, such that the following universal property is satisfied:
For any object $M \in \mathcal{C}$ and any family $\{\psi_i : A_i \to M\}$ of morphisms, there exists a unique morphism $\psi : \amalg A_i \to M$ called the <u>coproduct morphism</u> such that

$$\iota_i\psi = \psi_i \,\forall i \in I.$$



Definition 3.4.10 (Biproduct)**.** Let I be an index set and $\{S_i\}_{i\in I}$ a family of objects in a category $\mathcal{C}$. A <u>biproduct</u> is a product and a coproduct simultaneously, i.e. consists of the following data:

- an object $S \in \mathcal{C}_0$,
- a family of morphisms $\pi = \{\pi_i : S \to S_i\}_{i\in I}$,
- a family of morphisms $\iota = \{\iota_i : S_i \to S\}_{i\in I}$,
- a dependent function u_{in} mapping every family $\tau = \{\tau_i : T \to S_i\}_{i\in I}$ of morphisms with the same source T to a morphism $u_{\text{in}}(\tau) : T \to S$ such that $u_{\text{in}}(\tau)\pi_i \sim \tau_i$ for all $i \in I$,
- a dependent function u_{out} mapping every family $\rho = \{\rho_i : S_i \to R\}_{i\in I}$ of morphisms with the same target R to a morphism $u_{\text{out}}(\rho) : S \to R$ such that $\iota_i u_{\text{out}}(\rho) \sim \rho_i$ for all $i \in I$,

Definition 3.4.11. (Direct sum) A direct sum is a biproduct of objects in an Ab-category such that

- $\sum_{i \in I} \pi_i \iota_i \sim 1_S$,
- $\iota_i \pi_j \sim \delta_{i,j} = \begin{cases} 1_{S_i} & \text{if } i = j \\ 0_{ij} & \text{if } i \neq j \end{cases}$,

where $\delta_{i,j} \in \mathrm{Hom}(S_i, S_j)$ is the identity if $i = j$, and the zero morphism $0_{ij} := 0_{S_i,S_j}$ otherwise.

Example 3.4.12 (The direct sum of $\emptyset$)**.** For the index set $I = \emptyset$ we get an empty family of objects in a category $\mathcal{C}$. Its direct sum is

- an object $Z \in \mathcal{C}_0$,
- empty morphism sets π and ι,
- a dependent function u_{in} maps an empty collection τ of morphisms with same source T to a unique morphism $u_{\text{in}}(\tau) : T \to Z$.
- a dependent function u_{out} maps an empty collection ρ of morphisms with same target R to a unique morphism $u_{\text{out}}(\rho) : Z \to R$.

The unique morphisms $T \to Z$ and $Z \to T$ for any object T (the empty family τ imposing no further condition) suggests that our object $Z = \mathrm{DirectSum}(\emptyset)$ is in fact the zero object from 3.4.6, and the unique morphisms are the unique zero morphisms in $\mathrm{Hom}(Z, T)$ and $\mathrm{Hom}(T, Z)$ from 3.4.7.

Definition 3.4.13. An additive category is a pre-additive category $\mathcal{C}$ together with a dependent function $\oplus^{\mathcal{C}}$ mapping a finite set I and a family $(A_i)_{i \in I}$ of objects in $\mathcal{C}$ to a corresponding direct sum $(\oplus^{\mathcal{C}}_{i \in I} A_i, (\pi_i)_{i \in I}, (\iota_i)_{i \in I})$.

Remark 3.4.14 (Addition of morphisms)**.** In an additive category the abelian group structure on $\mathrm{Hom}_{\mathcal{C}}(M, N)$ can be derived from the direct sum:
For $\rho_1, \rho_2 \in \mathrm{Hom}_{\mathcal{C}}(M, N)$

$$\rho_1 + \rho_2 = u_{\text{in}}(1_M, 1_M) u_{\text{out}}(\rho_1, \rho_2) = u_{\text{in}}(\rho_1, \rho_2) u_{\text{out}}(1_N, 1_N)$$

The above equation illustrates that an additive category is pre-additive in at most one way, i.e. we don't have a choice how we define the abelian group structure on the hom-sets. [13]

Doctrine 3.4.15 (Additive category)**.** The doctrine `IsAdditiveCategory` therefore involves algorithms of `IsAbCategory` and `IsCategoryWithZeroObject` together with algorithms for

- `DirectSum`,
- `ProjectionInFactorOfDirectSum`,
- `InjectionOfCofactorOfDirectSum`,
- `UniversalMorphismIntoDirectSum`,
- `UniversalMorphismFromDirectSum`.

3.4.4 Pre-abelian categories

A pre-abelian category is an additive category with kernels and cokernels, and hence images and coimages. To define kernels and cokernels we need the following definition.

Definition 3.4.16 (Equalizer and coequalizer)**.** The equalizer / coequalizer is the limit / colimit of a functor $F : \mathbf{I} \to \mathcal{C}$, where $\mathbf{I}$ is generated by a quiver with two vertices and parallel arrows.

For example the <u>equalizer</u> of two morphisms $f, g : A \rightrightarrows B \in \mathcal{C}_1$ consists of the data

- an object $E := \mathrm{Eq}(f, g) \in \mathcal{C}_0$
- a morphism $\iota := E \hookrightarrow A$ such that pulled back to E, both morphisms are equal $\iota f = \iota g$:

$$\begin{aligned} & E \xrightarrow{\iota} A \xrightarrow{f} B \\ = E & \xrightarrow{\iota} A \xrightarrow{g} B \end{aligned}$$

- a dependent function u such that for any other morphism $\tau : T \to A$ with

$$\begin{aligned} & T \xrightarrow{\tau} A \xrightarrow{f} B \\ = T & \xrightarrow{\tau} A \xrightarrow{g} B \end{aligned}$$

  we have a unique morphism $u(\tau) : T \to E$ such that $u(\tau)\iota = \tau$.

$$\begin{array}{ccccc} E & \xhookrightarrow{\iota} & A & \overset{f}{\underset{g}{\rightrightarrows}} & B \\ {\scriptstyle u(\tau)}\uparrow & \nearrow_{\tau} & & & \\ T & & & & \end{array}$$

The following definition is a special case of an equalizer where $g = 0_{A,B}$. We will write it all out explicitly with their own names for objects, morphisms and (co-)lifts.

Definition 3.4.17 (Kernel)**.**
In an additive category $\mathcal{C}$, the <u>kernel</u> of a morphism $f : A \to B \in \mathcal{C}_1$ is the equalizer of f and $0_{A,B}$, i.e. consists of the data

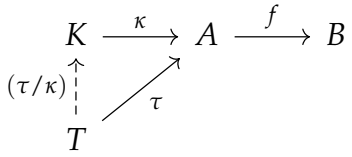
(1) An object $K = \mathrm{Ker}(f)$

(2) A morphism $\mathrm{KernelEmbedding}(f) := \kappa : K \to A$ such that $\kappa f = \kappa\, 0_{A,B} = 0_{K,B}$:

$$\begin{aligned} K & \xrightarrow{\kappa} A \xrightarrow{f} B \\ = K & \xrightarrow{\kappa} A \xrightarrow{0_{A,B}} B \\ = K & \xrightarrow{\quad 0_{K,B} \quad} B \end{aligned}$$

(3) A unique dependent function $\mathrm{KernelLift}(f, -) := (-/\kappa)$ mapping a morphism $\tau : T \to A$ with $\tau f = 0_{T,B}$ to a morphism $\mathrm{KernelLift}(f, \tau) = (\tau/\kappa) : T \to K$ such that

$$\tau = (\tau/\kappa)\,\kappa.$$



The last property means that the kernel embedding κ dominates every such τ, and that the kernel lift (τ/κ) is the unique lift of τ along κ in the sense of 2.2.4(1).

Definition 3.4.18 (Cokernel)**.** In an additive category $\mathcal{C}$, the cokernel of a morphism $f : A \to B \in \mathcal{C}_1$ is the coequalizer of f and $0_{A,B}$, i.e. consists of the data

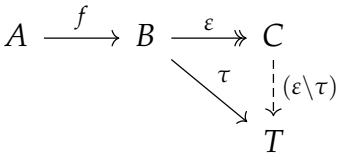
(1) An object $C = \mathrm{Coker}(f)$

(2) A morphism $\mathrm{CokernelProjection}(f) := \varepsilon : B \to C$ such that $f\,\varepsilon = 0_{A,B}\,\varepsilon = 0_{A,C}$:

$$\begin{aligned} & A \xrightarrow{f} B \xrightarrow{\varepsilon} C \\ = \; & A \xrightarrow{0_{A,B}} B \xrightarrow{\varepsilon} C \\ = \; & A \xrightarrow{\quad 0_{A,C} \quad} C \end{aligned}$$

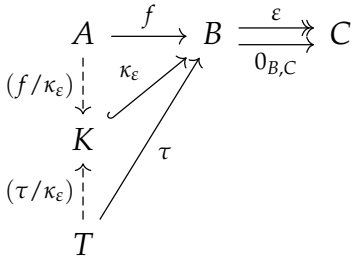
(3) A unique dependent function $\mathrm{CokernelColift}(f, -) := (\varepsilon\backslash -)$ mapping a morphism $\tau : B \to T$ with $f\tau = 0_{A,T}$ to a morphism $\mathrm{CokernelColift}(f, \tau) = (\varepsilon\backslash\tau) : C \to T$ such that

$$\tau = \varepsilon\,(\varepsilon\backslash\tau).$$



The last property means that the cokernel projection ε codominates every such τ, and that the cokernel colift $(\varepsilon\backslash\tau)$ is the unique colift of τ along ε in the sense of 2.2.4(2).

Definition 3.4.19 (Image)**.** In an additive category $\mathcal{C}$ we define the image of a morphism
$f : A \to B \in \mathcal{C}_1$ as the kernel of its cokernel:



Since ε is the cokernel projection of f, the morphism $f : A \to B$ with $f\,\varepsilon = 0_{A,C}$ plays the same role as any $\tau : T \to B$ with $\tau\,\varepsilon = 0_{T,C}$ in that it factors over the kernel object K in a unique way:

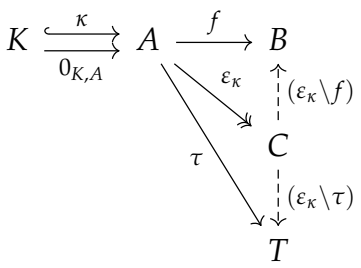
$$f = (f/\kappa_\varepsilon)\,\kappa_\varepsilon$$

Thus, the image is

- An object $\mathrm{Im}(f) := \mathrm{Ker}(\varepsilon)$,
- A morphism $\kappa_\varepsilon := \mathrm{ImageEmbedding}(f) := \mathrm{KernelEmbedding}(\varepsilon)$,

where $\varepsilon = \mathrm{CokernelProjection}(f)$. To differentiate the image of f (which is a kernel, but not the kernel of f) from the kernel of f, we are using I for $\mathrm{Im}(f)$ and ι for $\mathrm{ImageEmbedding}(f)$.

Definition 3.4.20 (Coimage)**.** In an additive category $\mathcal{C}$ we define the coimage of a morphism $f : A \to B \in \mathcal{C}_1$ as the cokernel of its kernel:



Since κ is the kernel embedding of f, the morphism $f : A \to B$ with $\kappa f = 0_{K,B}$ plays the same role as any $\tau : A \to T$ with $\kappa \tau = 0_{K,T}$ in that it factors over the cokernel object C in a unique way:

$$f = \varepsilon_\kappa (\varepsilon_\kappa \backslash f)$$

Thus, the coimage is

- An object $\mathrm{Coim}(f) := \mathrm{Coker}(\kappa)$,
- A morphism $\varepsilon_\kappa := \mathrm{CoimageProjection}(f) := \mathrm{CokernelProjection}(\kappa)$,

where $\kappa = \mathrm{KernelEmbedding}(f)$. Since it was reserved for cokernel, we are not using C to denote the coimage object.

Definition 3.4.21. (Pre-Abelian category) A pre-abelian category consists of the following data:

(1) An additive category $\mathcal{C}$.

(2) A dependent function mapping every morphism $f : A \to B$ for $A, B \in \mathcal{C}_0$ to a kernel of f.

(3) A dependent function mapping every morphism $f : A \to B$ for $A, B \in \mathcal{C}_0$ to a cokernel of f.

Doctrine 3.4.22 (Pre-abelian category)**.** The doctrine `IsPreAbelianCategory` therefore involves algorithms of `IsAdditiveCategory` together with algorithms for

- `KernelObject`,
- `KernelEmbedding`,
- `KernelLift`,
- `CokernelObject`,
- `CokernelProjection`,
- `CokernelColift`

Remark 3.4.23 (Kernel lift of cokernel colift = cokernel colift of kernel lift)**.**
In categories with kernels and cokernels we can compute images and coimages together with an induced morphism

$$\overline{\varphi} : \mathrm{Coim}(\varphi) \to \mathrm{Im}(\varphi). \tag{3.4.3}$$

This morphism being an isomorphism is one of the defining axioms of Abelian categories.

Proof. Given a morphism $\varphi : M \to N$ in a pre-Abelian category $\mathcal{C}$, we can compute its kernel embedding $\kappa : \mathrm{Ker}(\varphi) \hookrightarrow M$ and its cokernel projection $\varepsilon : N \twoheadrightarrow \mathrm{Coker}(\varphi)$. They are the equalizer and coequalizer of φ and $0_{M,N}$, in particular we have $\kappa \varphi = 0_{\mathrm{Ker}(\varphi),N}$ and $\varphi \varepsilon = 0_{M,\mathrm{Coker}(\varphi)}$ (the two 0-arrows on the top).

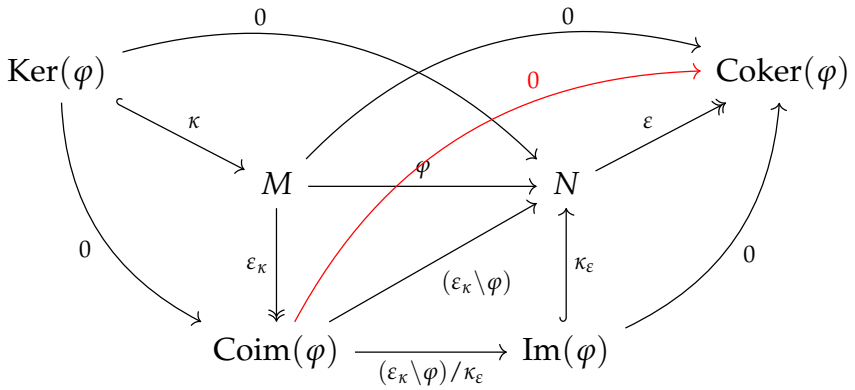
For the kernel embedding κ we can compute its cokernel projection $\varepsilon_\kappa : M \twoheadrightarrow \mathrm{Coim}(\varphi)$ which we called the coimage projection of φ. Keeping in mind, that it is the cokernel of κ, i.e. the coequalizer of κ and $0_{\mathrm{Ker}(\varphi),M}$, we get the zero morphism $0_{\mathrm{Ker}(\varphi),\mathrm{Coim}(\varphi)}$ (the bottom left 0-arrow).

Dually for the cokernel projection ε we can compute its kernel embedding $\kappa_\varepsilon : \mathrm{Im} \hookrightarrow N$ which we called the image embedding of φ. Again we see that we have the zero morphism $0_{\mathrm{Im}(\varphi),\mathrm{Coker}(\varphi)}$ (the bottom right 0-arrow).

The two morphisms φ and its coimage projection ε_κ have the same source M and composed with κ, both yield zero morphisms. Since ε_κ codominates φ, we have a unique cokernel colift
$(\varepsilon_\kappa \backslash \varphi) : \mathrm{Coim}(\varphi) \to N$, which is one of the two diagonals in the square diagram, and for which
$\varepsilon_\kappa \cdot (\varepsilon_\kappa \backslash \varphi) = \varphi$.

Now κ_ε and the above colift $(\varepsilon_\kappa \backslash \varphi)$ both have the same target N. For κ_ε we already know that composed with ε we get the zero morphism. If we also proved that $(\varepsilon_\kappa \backslash \varphi) \cdot \varepsilon = 0_{\mathrm{Coim}(\varphi),\mathrm{Coker}(\varphi)}$, i.e. the red 0-arrow in the picture, then since κ_ε dominates $(\varepsilon_\kappa \backslash \varphi)$ we get the existence of the kernel lift of the cokernel colift

$$(\varepsilon_\kappa \backslash \varphi) / \kappa_\varepsilon : \mathrm{Coim}(\varphi) \to \mathrm{Im}(\varphi)$$



Proof that $(\varepsilon_\kappa \backslash \varphi) \cdot \varepsilon = 0_{\mathrm{Coim}(\varphi),\mathrm{Coker}(\varphi)}$.
We have $\varepsilon_\kappa \cdot (\varepsilon_\kappa \backslash \varphi) = \varphi$, i.e. the top left triangle of the square commutes. But then since composition with a zero morphism gives a zero morphism we have

$$\begin{aligned} & \varepsilon_\kappa \cdot 0_{\mathrm{Coim}(\varphi),\mathrm{Coker}(\varphi)} : M \to \mathrm{Coker}(\varphi) \\ = & \, 0_{M,\mathrm{Coker}(\varphi)} \\ = & \, \varphi \cdot \varepsilon \\ = & \, \varepsilon_\kappa \cdot (\varepsilon_\kappa \backslash \varphi) \cdot \varepsilon \end{aligned}$$

And pre-cancellation of the epimorphism ε_κ gives

$$0_{\mathrm{Coim}(\varphi),\mathrm{Coker}(\varphi)} = (\varepsilon_\kappa \backslash \varphi) \cdot \varepsilon.$$

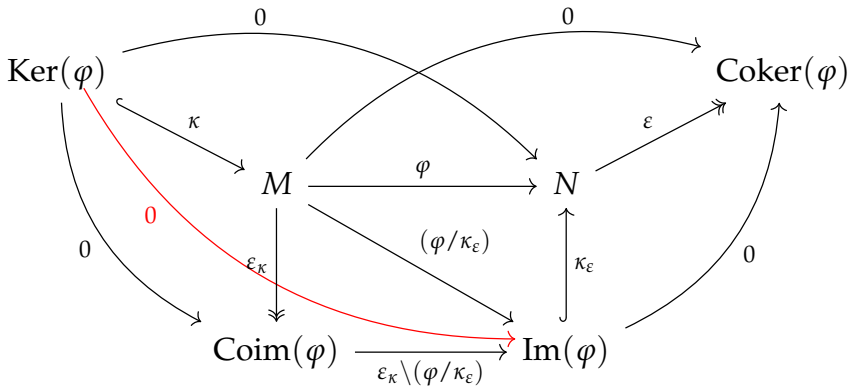
//

Dual to the above picture, the two morphisms φ and its image embedding κ_ε both have the same target N and composed with ε, both yield zero morphisms. Since κ_ε dominates φ, we have a unique kernel lift $(\varphi / \kappa_\varepsilon) : M \to \mathrm{Im}(\varphi)$, which is the other diagonal in the square diagram, and for which $(\varphi / \kappa_\varepsilon) \cdot \kappa_\varepsilon = \varphi$.

Now ε_κ and $(\varphi / \kappa_\varepsilon)$ both have the same source M. For ε_κ we already know that composed with κ we get the zero morphism. If we also proved that $\kappa \cdot (\varphi / \kappa_\varepsilon) =$

$0_{\mathrm{Ker}(\varphi),\mathrm{Im}(\varphi)}$, i.e. the red 0-arrow in the picture, then, since ε_κ codominates $(\varphi/\kappa_\varepsilon)$ we get the existence of the cokernel colift of the kernel lift

$$\varepsilon_\kappa \backslash (\varphi/\kappa_\varepsilon) : \mathrm{Coim}(\varphi) \to \mathrm{Im}(\varphi)$$



Proof that $\kappa \cdot (\varphi/\kappa_\varepsilon) = 0_{\mathrm{Ker}(\varphi),\mathrm{Im}(\varphi)}$.
We have $(\varphi/\kappa_\varepsilon) \cdot \kappa_\varepsilon = \varphi$, i.e. the top right triangle of the square commutes. But then since composition with a zero morphism gives a zero morphism we have

$$\begin{aligned} & 0_{\mathrm{Ker}(\varphi),\mathrm{Im}(\varphi)} \cdot \kappa_\varepsilon : \mathrm{Ker}(\varphi) \to N \\ = {} & 0_{\mathrm{Ker}(\varphi),N} \\ = {} & \kappa \cdot \varphi \\ = {} & \kappa \cdot (\varphi/\kappa_\varepsilon) \cdot \kappa_\varepsilon \end{aligned}$$

And post-cancellation of the monomorphism κ_ε gives

$$0_{\mathrm{Ker}(\varphi),\mathrm{Im}(\varphi)} = \kappa \cdot (\varphi/\kappa_\varepsilon).$$

//

So we have two morphisms from $\mathrm{Coim}(\varphi)$ to $\mathrm{Im}(\varphi)$:

$$\underbrace{(\varepsilon_\kappa \backslash \varphi)/\kappa_\varepsilon}_{\alpha} \quad \text{and} \quad \underbrace{\varepsilon_\kappa \backslash (\varphi/\kappa_\varepsilon)}_{\beta}.$$

In fact they are equal since the two commutative traingles give commutative squares.

$$\varepsilon_\kappa \, \alpha \, \kappa_\varepsilon = \varphi = \varepsilon_\kappa \, \beta \, \kappa_\varepsilon$$

Since the epi ε_κ is pre-cancelable and the mono κ_ε is post-cancelable, we get $\alpha = \beta$, i.e.

$$(\varepsilon_\kappa \backslash \varphi)/\kappa_\varepsilon = \varepsilon_\kappa \backslash (\varphi/\kappa_\varepsilon)$$

This justifies the notation

$$\overline{\varphi} := \varepsilon_\kappa \backslash \varphi / \kappa_\varepsilon : \mathrm{Coim}(\varphi) \to \mathrm{Im}(\varphi). \tag{3.4.4}$$

In this context associativity holds, i.e. we can leave out the parentheses of the lift and colift. □

3.4.5 Abelian categories

In this section we give three equivalent definitions for Abelian categories, starting abstractly from the above diagrams and ending with the existence of two algorithms needed for the doctrine `IsAbelianCategory`.

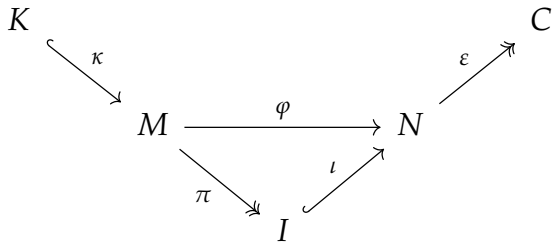
Definition 3.4.24 (Abelian category)**.** A pre-Abelian category $\mathcal{C}$ is Abelian if for every morphism $\varphi \in \mathcal{C}_1$, the natural morphism in (3.4.4)

$$\overline{\varphi} := \varepsilon_\kappa \backslash \varphi / \kappa_\varepsilon : \mathrm{Coim}(\varphi) \xrightarrow{\sim} \mathrm{Im}(\varphi)$$

is an isomorphism.

Therefore in the doctrine of Abelian categories we will not need to distinguish between coimages and images[14], and the diagrams from 3.4.23 simplify into an epi-mono factorization diagram.

Corollary 3.4.25 (Epi-mono-factorization)**.** Every morphism $\varphi : M \to N$ in an abelian category $\mathcal{C}$ can be factored as the composition of an epimorphism $\pi : M \twoheadrightarrow I$ and a monomorphism $\iota : I \hookrightarrow N$
where $I \cong \mathrm{Im}(\varphi) \cong \mathrm{Coim}(\varphi)$.



The factorization is unique up to unique isomorphism.

Definition 3.4.26 (Abelian category)**.** An abelian category is

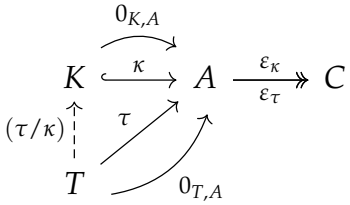
- a pre-abelian category $\mathcal{C}$ where
- every monomorphism κ is a kernel-embedding of its cokernel-projection and
- every epimorphism ε is a cokernel-projection of its kernel-embedding.

Remark 3.4.27. The above definition, that we can regard

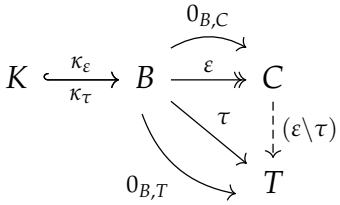
- every monomorphism $\kappa : K \hookrightarrow A$ as a kernel-embedding of its cokernel-projection $\varepsilon_\kappa : A \twoheadrightarrow C$, with K being the kernel object $\mathrm{Ker}(\varepsilon_\kappa) = s(\kappa)$,
- and every epimorphism $\varepsilon : B \twoheadrightarrow C$ as a cokernel-projection of its kernel-embedding $\kappa_\varepsilon : K \hookrightarrow B$, with C being the cokernel object $\mathrm{Coker}(\kappa_\varepsilon) = t(\varepsilon)$

implies that we also have the third ingredient for kernels and cokernels, namely the dependent functions

- kernel lift $(-/\kappa) : \_ \to K$ mapping a morphism $\tau : T \to A$ with same target $t(\tau) = t(\kappa) = A$ and same cokernel $\varepsilon_\kappa = \varepsilon_\tau$ (i.e. $\tau\varepsilon_\kappa = 0_{T,C}$) as κ to a unique kernel lift $(\tau/\kappa) : T \to K$



- cokernel colift $(\varepsilon\backslash -) : C \to \_$ mapping a morphism $\tau : B \to T$ with same source $s(\tau) = s(\varepsilon) = B$ and same kernel $\kappa_\varepsilon = \kappa_\tau$ (i.e. $\kappa_\varepsilon \tau = 0_{K,T}$) as ε to a unique cokernel colift $(\varepsilon\backslash\tau) : C \to T$.



The existence of lifts along monos and colifts along epis (in the sense of the above remark) in a pre-abelian category is therefore an equivalent definition of an abelian category.

Definition 3.4.28 (Abelian category)**.** An Abelian category consists of the following data:

(1) A pre-abelian category $\mathcal{C}$.

(2) A dependent function $(-/-)$ mapping a monomorphism $\kappa : K \hookrightarrow A$ and a morphism $\tau : T \to A$ with the same target $t(\tau) = t(\kappa)$ and the same cokernel projection $\varepsilon_\tau = \varepsilon_\kappa$ to a lift (τ/κ) of τ along κ.

(3) A dependent function $(-\backslash -)$ mapping an epimorphism $\varepsilon : B \twoheadrightarrow C$ and a morphism $\tau : B \to T$ with same source $s(\tau) = s(\varepsilon)$ and the same kernel embedding $\kappa_\tau = \kappa_\varepsilon$ to a colift $(\varepsilon\backslash\tau)$ of τ along ε.

Doctrine 3.4.29 (Abelian category)**.** The doctrine `IsAbelianCategory` therefore involves algorithms of `IsPreAbelianCategory` together with two additional algorithms

- `LiftAlongMonomorphism`,
- `ColiftAlongEpimorphism`,

fulfilling the specification of definition 3.4.28

3.5 The matrix category $\Bbbk$-mat is an abelian category

The following is an example of a category which has all the limits and colimits mentioned so far, and has them implemented constructively. We will check the four doctrines `IsAbCategory`, `IsAdditiveCategory`, `IsPreAbelianCategory` and `IsAbelianCategory` by providing the needed algorithms.

Example 3.5.1. (The matrix category $\Bbbk$-**mat** over a commutative ring $\Bbbk$)

- Objects are natural numbers $\Bbbk\text{-}\mathbf{mat}_0 = \mathbb{N} = \mathbb{N}_0 = \{0, 1, 2, \dots\}$ for wich we use small latin letters $(m, n, k, \dots)$.
- Morphisms $(m \to n) \in \Bbbk\text{-}\mathbf{mat}_1$ are $m \times n$ matrices over $\Bbbk$. We write the set of morphisms between m and n, as $\Bbbk^{m\times n} := \mathrm{Hom}_{\Bbbk\text{-}\mathbf{mat}}(m, n)$. For variables that are Matrices we use small greek letters $(\varphi, \psi, \dots)$ or capital latin letters $(A, B, C, \dots)$. When only source and target are relevant, we write $(m \times n)$.
- $s(\varphi) = \texttt{Source}(\varphi) := \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) := \texttt{NrColumns}(\varphi)$

- Identity morphisms are the identity matrices.

$$1_m = \texttt{IdentityMorphism}(m) := \texttt{IdentityMat}(m, \Bbbk).$$

- Composition is matrix multiplication which is associative.

$$\varphi\psi = \texttt{PreCompose}(\varphi, \psi) := \texttt{MatMul}(\varphi, \psi).$$

- It is a skeletal category, i.e. $m \cong n \Rightarrow m = n$. Only quadratic matrices ($m = n$) can be isomorphisms.

Example 3.5.2 ($\Bbbk$-**mat** is an Ab-category)**.** In $\Bbbk$-**mat**, the number 0 is the zero object.
$\texttt{ZeroObject} := 0$
A zero matrix (zero morphism) is a matrix factoring through the zero object 0.

$$\Bbbk^{m\times n} \ni A = 0_{m,n} \iff \begin{array}{ccc} m & \xrightarrow{\quad A \quad} & n \\ & {\scriptstyle (m\times 0)}\searrow \quad \nearrow{\scriptstyle (0\times n)} & \\ & 0 & \end{array}$$
$$\Rightarrow A = (m \times 0) \cdot (0 \times n).$$

The matrices $(m \times 0)$ and $(0 \times n)$ have zero columns or zero rows respectively, but it is important to note that for each $m \in \Bbbk$-$\mathbf{mat}_0$ there is exactly one such matrix $(m \times 0)$ and $(0 \times m)$ (that's what initial and terminal object means), and for different m, these morphisms are different.[15]

- For the matrix $(m \times 0)$ we have

$$\begin{gathered}\texttt{UniversalMorphismIntoZeroObject}(m) \\ := \texttt{ZeroMorphism}(m, 0) := \texttt{ZeroMatrix}(m, 0),\end{gathered}$$

- For the matrix $(0 \times n)$ we have
$\texttt{UniversalMorphismFromZeroObject}(\texttt{n}) := \texttt{ZeroMorphism}(0, \texttt{n}) := \texttt{ZeroMatrix}(0, \texttt{n})$,

- For zero matrices $(m \times n)$ we have $\texttt{ZeroMorphism}(\texttt{m}, \texttt{n}) := \texttt{ZeroMatrix}(\texttt{m}, \texttt{n})$.

For two natural numbers $m, n \in \Bbbk$-$\mathbf{mat}_0 = \mathbb{N} = \mathbb{N}_0$, the set of morphisms with source m and target n is $\Bbbk^{m\times n}$, the set of $m \times n$-matrices. This is an abelian group:

- The neutral element of the addition is the $m \times n$ zero matrix $0_{m,n}$.

- Addition of matrices $\texttt{AdditionForMorphisms}(\texttt{phi}, \texttt{psi}) := \texttt{phi} + \texttt{psi}$ is associative and commutative.

- For every matrix $A \in \Bbbk^{m\times n}$ there is a negative matrix $-A \in \Bbbk^{m\times n}$ such that $A + (-A) = 0_{m,n}$.

Composition of matrices is defined as matrix multiplication, which is bilinear, i.e. satisfies the distributive laws (3.4.1) and (3.4.2).
It is an easy exercise to provide the algorithms for an Ab-category mentioned in doctrine 3.4.4.

Example 3.5.3 ($\Bbbk$-**mat** is an additive category)**.** Let for $I = \{1, \ldots, N\}$, the set $\{n_1, \ldots, n_N\}$ be a family of objects in $\Bbbk$-$\mathbf{mat}_0$. Their direct sum is

- the object $n := \bigoplus_{i=1}^{N} n_i := \sum_{i=1}^{N} n_i = \texttt{Sum}$

- For $i \in I$ we have as identity morphism 1_{n_i} of the object n_i the $n_i \times n_i$ identity matrix. Define
$$n_{<i} := \sum_{j=1}^{i-1} n_j \quad \text{and} \quad n_{>i} := \sum_{j=i+1}^{N} n_j.$$
Then we have

- The projection $\pi_i : n \to n_i$ is an $n \times n_i$ matrix that is a stacked matrix of the $n_j \times n_i$ zero matrices not including $0_{n_i,n_i}$ and the identity matrix 1_{n_i}.
$$\pi_i = \begin{pmatrix} 0_{n_1,n_i} \\ 0_{n_2,n_i} \\ \vdots \\ 0_{n_{i-1},n_i} \\ 1_{n_i} \\ 0_{n_{i+1},n_i} \\ \vdots \\ 0_{n_N,n_i} \end{pmatrix} = \begin{pmatrix} 0_{n_{<i},n_i} \\ 1_{n_i} \\ 0_{n_{>i},n_i} \end{pmatrix} \tag{3.5.1}$$

- The coprojection $\iota_i : n_i \to n$ is the transposed matrix $\iota_i = \pi_i^T$, i.e. an $n_i \times n$ matrix of the $n_i \times n_j$ zero matrices not including $0_{n_i,n_i}$ and the identity matrix 1_{n_i} lined up next to each other.
$$\iota_i = \begin{pmatrix} 0_{n_i,n_1} & 0_{n_i,n_2} & \dots & 0_{n_i,n_{i-1}} & 1_{n_i} & 0_{n_i,n_{i+1}} & \dots & 0_{n_i,n_N} \end{pmatrix} = \begin{pmatrix} 0_{n_i,n_{<i}} & 1_{n_i} & 0_{n_i,n_{>i}} \end{pmatrix} \tag{3.5.2}$$

- For a family $\tau = (\tau_i : t \to n_i)_{i \in I}$ we have the morphism $u_{\text{in}}(\tau)$ which is a $t \times n$ block matrix of the τ_i:
$$u_{\text{in}}(\tau) = \begin{pmatrix} \tau_1 & \cdots & \tau_N \end{pmatrix} \tag{3.5.3}$$
with $u_{\text{in}}(\tau)\pi_i = \tau_i$.

- For a family $\tau = (\tau_i : n_i \to t)_{i \in I}$ we have the morphism $u_{\text{out}}(\tau)$ which is an $n \times t$ block matrix of the τ_i:
$$u_{\text{out}}(\tau) = \begin{pmatrix} \tau_1 \\ \vdots \\ \tau_N \end{pmatrix} \tag{3.5.4}$$
with $\iota_i u_{\text{out}}(\tau) = \tau_i$.

One can easily verify the conditions for π, ι, u_{in} and u_{out} in Definitions 3.4.10 and 3.4.11.

Computation 3.5.4. If we assume algorithms for adding natural numbers
$$\texttt{Sum}([m,n]) = m + n,$$
$$\texttt{Sum}([\quad]) = 0,$$
for stacking two matrices with the same number of columns (same target) on top of each other
$$\varphi : k \to n$$
$$\psi : m \to n$$
$$\texttt{StackMatrix}(\varphi,\psi) = \begin{pmatrix} \varphi \\ \psi \end{pmatrix} : k + m \to n,$$

and for aligning two matrices with the same number of rows (same source) next to each other

$$\begin{aligned} &\varphi : m \to k \\ &\psi : m \to n \\ \texttt{AugmentMatrix}(\varphi, \psi) = &\begin{pmatrix} \varphi & \psi \end{pmatrix} : m \to k + n. \end{aligned}$$

Together with the algorithms `IdentityMorphism(n)` and `ZeroMorphism(m,n)` from `IsAbCategory` we can then calculate all the algorithms in the doctrine `IsAdditiveCategory`:

The direct sum of the objects `D` $:= [$`V1, V2, ..., VN`$]$ in $\Bbbk$-**mat** is defined as

- `DirectSum ( D ) := VectorSpaceObject( Sum( List( D, V ↦ Dimension( V ) ) ), k )`
- `ProjectionInFactorOfDirectSum( D, i ) := StackMatrix( [ List( D[1, ..., (i-1)], V ↦ ZeroMorphism( V, D[i] ) ), IdentityMorphism( D[i] ), List( D[(i+1), ..., N], V ↦ ZeroMorphism( V, D[i] ) ) ] )`
- `InjectionOfCofactorOfDirectSum( D, i ) := AugmentMatrix( [ List( D[1, ..., (i-1)], V ↦ ZeroMorphism( D[i], V ) ), IdentityMorphism( D[i] ), List( D[(i+1), ..., N], V ↦ ZeroMorphism( D[i], V ) ) ] )`
- `UniversalMorphismIntoDirectSum( [ phi, psi ] ) := StackMatrix( [ phi, psi ] )`
- `UniversalMorphismFromDirectSum( [ phi, psi ] ) := AugmentMatrix( [ phi, psi ] )`

In the following GAP session, we demonstrate the algorithms of `IsAdditiveCategory` and verify in an example the two axioms of the direct sum in 3.4.11. To verify two these properties for a general case of a direct sum of objects in the matrix category is left as an exercise.

The implementation of objects in the matrix category $\Bbbk$-**mat** in CAP is slightly different than simply natural numbers, since we always have to mention the commutative ring $\Bbbk$ for the objects. We are using the finite field $\Bbbk := \mathbb{F}_3$ for our calculations below.

Example

```
gap> LoadPackage("LinearAlgebraForCAP");
true
gap> GF3 := HomalgRingOfIntegers( 3 );
GF(3)
gap> V3 := VectorSpaceObject( 3, GF3 );
<A vector space object over GF(3) of dimension 3>
gap> V5 := VectorSpaceObject( 5, GF3 );
<A vector space object over GF(3) of dimension 5>
gap> V2 := VectorSpaceObject( 2, GF3 );
<A vector space object over GF(3) of dimension 2>
gap> D := [ V3, V5, V2 ];
[ <A vector space object over GF(3) of dimension 3>,
  <A vector space object over GF(3) of dimension 5>,
  <A vector space object over GF(3) of dimension 2> ]
gap> S := DirectSum( D );
<A vector space object over GF(3) of dimension 10>
gap> zero35 := ZeroMorphism( V3, V5 );
```

```
<A zero morphism in Category of matrices over GF(3)>
gap> Display( zero35 );
 . . . . .
 . . . . .
 . . . . .

 A zero morphism in Category of matrices over GF(3)
gap> one5 := IdentityMorphism( V5 );
<An identity morphism in Category of matrices over GF(3)>
gap> Display( one5 );
 1 . . . .
 . 1 . . .
 . . 1 . .
 . . . 1 .
 . . . . 1

 An identity morphism in Category of matrices over GF(3)
gap> zero25 := ZeroMorphism( V2, V5 );
<A zero morphism in Category of matrices over GF(3)>
gap> Display( zero25 );
 . . . . .
 . . . . .

 A zero morphism in Category of matrices over GF(3)
gap> pi2 := ProjectionInFactorOfDirectSum( D, 2 );
<A morphism in Category of matrices over GF(3)>
gap> Display( pi2 );
 . . . . .
 . . . . .
 . . . . .
 1 . . . .
 . 1 . . .
 . . 1 . .
 . . . 1 .
 . . . . 1
 . . . . .
 . . . . .

A morphism in Category of matrices over GF(3)
gap> iota1 := InjectionOfCofactorOfDirectSum( D, 1 );
<A morphism in Category of matrices over GF(3)>
gap> Display( iota1 );
 1 . . . . . . . . .
 . 1 . . . . . . . .
 . . 1 . . . . . . .

 A morphism in Category of matrices over GF(3)
gap> Display( PreCompose( iota1, pi2 ) );
 . . . . .
 . . . . .
 . . . . .

 A morphism in Category of matrices over GF(3)
gap> IsEqualForMorphisms( PreCompose( iota1, pi2 ),
>      ZeroMorphism( V3, V5 ) );
true
gap> iota2 := InjectionOfCofactorOfDirectSum( D, 2 );;
```

```
gap> Display( iota2 );
 . . . 1 . . . . . .
 . . . . 1 . . . . .
 . . . . . 1 . . . .
 . . . . . . 1 . . .
 . . . . . . . 1 . .

 A morphism in Category of matrices over GF(3)
gap> Display( PreCompose( iota2, pi2 ) );
 1 . . . .
 . 1 . . .
 . . 1 . .
 . . . 1 .
 . . . . 1

 A morphism in Category of matrices over GF(3)
gap> IsEqualForMorphisms( PreCompose( iota2, pi2 ),
>      IdentityMorphism( V5 ) );
true
gap> Display( PreCompose( pi2, iota2 ) );
 . . . . . . . . . .
 . . . . . . . . . .
 . . . . . . . . . .
 . . . 1 . . . . . .
 . . . . 1 . . . . .
 . . . . . 1 . . . .
 . . . . . . 1 . . .
 . . . . . . . 1 . .
 . . . . . . . . . .
 . . . . . . . . . .

 A morphism in Category of matrices over GF(3)
gap> iota3 := InjectionOfCofactorOfDirectSum( D, 3 );;
gap> pi1 := ProjectionInFactorOfDirectSum( D, 1 );;
gap> pi3 := ProjectionInFactorOfDirectSum( D, 3 );;
gap> Display( PreCompose( pi1, iota1 ) + PreCompose( pi2, iota2 )
>    + PreCompose( pi3, iota3 ) );
 1 . . . . . .
 . 1 . . . . . .
 . . 1 . . . . . .
 . . . 1 . . . . . .
 . . . . 1 . . . . .
 . . . . . 1 . . . .
 . . . . . . 1 . . .
 . . . . . . . 1 . .
 . . . . . . . . 1 .
 . . . . . . . . . 1

 A morphism in Category of matrices over GF(3)
gap> IsEqualForMorphisms(
>      PreCompose( pi1, iota1 )
>    + PreCompose( pi2, iota2 )
>    + PreCompose( pi3, iota3 ),
>    IdentityMorphism( S ) );
true
```

We can also verify the result in 3.4.14 that the abelian group operation

AdditionForMorphisms

can be derived from

UniversalMorphismIntoDirectSum,
UniversalMorphismFromDirectSum,
IdentityMorphism and
PreCompose.

As an example we add two GF_3-matrices from V2 to V3 in three different ways.

Example

```
gap> mat1 := HomalgMatrix( [ 0, 1, 2, 1, 1, 2 ], 2, 3, GF3 );
<A 2 x 3 matrix over an internal ring>
gap> mat2 := HomalgMatrix( [ 1, 1, 1, 1, 1, 1 ], 2, 3, GF3 );
<A 2 x 3 matrix over an internal ring>
> mor1 := VectorSpaceMorphism( V2, mat1, V3 );
<A morphism in Category of matrices over GF(3)>
gap> Display( mor1 );
 . 1 2
 1 1 2

 A morphism in Category of matrices over GF(3)
> mor2 := VectorSpaceMorphism( V2, mat2, V3 );
<A morphism in Category of matrices over GF(3)>
gap> Display( mor2 );
 1 1 1
 1 1 1

 A morphism in Category of matrices over GF(3)
> result1 := mor1 + mor2;
<A morphism in Category of matrices over GF(3)>
gap> Display( result1 );
 1 2 .
 2 2 .

 A morphism in Category of matrices over GF(3)
gap> one2 := IdentityMorphism( V2 );
<An identity morphism in Category of matrices over GF(3)>
gap> result2 := PreCompose(
>    UniversalMorphismIntoDirectSum( [ one2, one2 ] ),
>    UniversalMorphismFromDirectSum( [ mor1, mor2 ] ) );
<A morphism in Category of matrices over GF(3)>
gap> Display( result2 );
 1 2 .
 2 2 .

 A morphism in Category of matrices over GF(3)
gap> result1 = result2;
true
gap> one3 := IdentityMorphism( V3 );
<An identity morphism in Category of matrices over GF(3)>
gap> result3 := PreCompose(
>   UniversalMorphismIntoDirectSum( [ mor1, mor2 ] ),
>   UniversalMorphismFromDirectSum( [ one3, one3 ] ) );
```

```
<A morphism in Category of matrices over GF(3)>
gap> Display( result3 );
 1 2 .
 2 2 .

 A morphism in Category of matrices over GF(3)
gap> result3 = result2;
true
```

Next we provide the algorithms from 3.4.22 that make $\Bbbk$-**mat** into a pre-abelian category. They are all based on the well-known `Gauss` algorithm that gives us the row echolon form (REF) and the column echolon form (CEF) of a matrix.

Computation 3.5.5. Let $\varphi : m \to n \in \Bbbk\text{-}\mathbf{mat}_1$ be a matrix. We assume algorithms for

- The rank of a matrix, $r := \texttt{Rank}(\texttt{phi})$, defined as $\Bbbk$-dimension of the column space $\dim_{\text{Col}}(\varphi) = r$ which is the same[16] as the $\Bbbk$-dimension of the row space $\dim_{\text{Row}}(\varphi) = r$,
- The left nullspace of a matrix φ is a matrix $x = \texttt{LeftNullSpace}(\texttt{phi})$ satisfying $x\,\varphi = 0$ and for each matrix y with $y\,\varphi = 0$ there exists a matrix z with $zx = y$.
- The right nullspace of a matrix φ is a matrix $x = \texttt{RightNullSpace}(\texttt{phi})$ satisfying $\varphi\,x = 0$ and for each matrix y with $\varphi\,y = 0$ there exists a matrix z with $xz = y$.
- As the standardized form to represent these subspaces, the `Gauss`-algorithm can calculate the row-echolon-form $\texttt{REF}(\texttt{LeftNullSpace}(\texttt{phi}))$ and the column-echolon-form $\texttt{CEF}(\texttt{RightNullSpace}(\texttt{phi}))$.

Example 3.5.6 ($\Bbbk$-**mat** is a pre-abelian category)**.** With the algorithms in 3.5.5 taken as given, we now give all the algorithms needed for the doctrine `IsPreAbelianCategory` in 3.4.22:

- $\texttt{KernelObject}(\texttt{phi}) := \texttt{NrRows}(\texttt{phi}) - \texttt{Rank}(\texttt{phi})$
- $\texttt{KernelEmbedding}(\texttt{phi}) := \texttt{REF}(\texttt{LeftNullSpace}(\texttt{phi}))$
- $\texttt{KernelLift}(\texttt{phi},\texttt{tau}) := \texttt{Solve}(\texttt{x} \cdot \texttt{REF}(\texttt{LeftNullSpace}(\texttt{phi})) = \texttt{tau})$
- $\texttt{CokernelObject}(\texttt{phi}) := \texttt{NrColumns}(\texttt{phi}) - \texttt{Rank}(\texttt{phi})$
- $\texttt{CokernelProjection}(\texttt{phi}) := \texttt{CEF}(\texttt{RightNullSpace}(\texttt{phi}))$
- $\texttt{CokernelColift}(\texttt{phi},\texttt{tau}) := \texttt{Solve}(\texttt{CEF}(\texttt{RightNullSpace}(\texttt{phi})) \cdot \texttt{x} = \texttt{tau})$

With these constructions, $\Bbbk$-**mat** becomes a pre-abelian category.

Note that the right-hand side B in the equation

$$A \cdot x = B$$

can be more than a single column vector, but a matrix with multiple columns, as long as they have the same number of rows as A. This corresponds to solving the system of equations simultaneously for different right-hand sides. In case that it is solvable, we get a particular solution as a matrix $x = \texttt{LeftDivide}(\texttt{A},\texttt{B})$.

Dually for B and A matrices having the same number of columns, for the equation

$$x \cdot A = B$$

we get a particular solution as a matrix $x = \texttt{RightDivide}(\texttt{B},\texttt{A})$, if it exists.

Example 3.5.7 ($\Bbbk$-**mat** is an Abelian category)**.**

(1) Let $\kappa : K \hookrightarrow A \in \Bbbk\text{-}\mathbf{mat}_1$ and $\tau : T \to A \in \Bbbk\text{-}\mathbf{mat}_1$ be as in 3.4.28(2). Then with the algorithms from 3.5.5 we define

- `LiftAlongMonomorphism(kappa, tau)` := `Solve(x · kappa = tau)` := `RightDivide(tau, kappa)`.

(2) Let $\varepsilon : B \twoheadrightarrow C \in \Bbbk\text{-}\mathbf{mat}_1$ and $\tau : B \to T \in \Bbbk\text{-}\mathbf{mat}_1$ be as in 3.4.28(3). Then with the algorithms from 3.5.5 we define

- `ColiftAlongEpimorphism(epsilon, tau)` := `Solve(epsilon · x = tau)` := `LeftDivide(epsilon, tau)`

Since a matrix $\kappa : m \to n$ is a monomorphism iff its kernel is 0 iff it has full row rank `Rank(kappa) = m`, the lift along monomorphism `RightDivide(tau, kappa)` always exists.

Since a matrix $\varepsilon : m \to n$ is an epimorphism iff its image is n iff it has full column rank `Rank(kappa) = n`, the colift along epimorphism `LeftDivide(epsilon, tau)` always exists.

With these algorithms, $\Bbbk$-**mat** becomes an abelian category. In particular we have

$$\begin{aligned} & \operatorname{Coim}(\varphi) \cong \operatorname{Im}(\varphi) \quad \text{and since } \Bbbk\text{-}\mathbf{mat} \text{ is skeletal} \\ \Rightarrow\ & \operatorname{Coim}(\varphi) = \operatorname{Im}(\varphi). \end{aligned}$$

The following situation where we have a family of matrices $\{a_i : m_i \to n_i\}_{i\in I}$, i.e. a family $\{m_i\}_{i\in I}$ of sources and $\{n_i\}_{i\in I}$ of targets, is useful to understand. There are two different direct sums involved, one of the m_i's and one of the n_i's. We will need this construction in section 4 for the direct sum of functors, and in section 6 for the Sylvester equations.

Example 3.5.8 (Block-Diagonal matrices)**.**
In a situation with an index set $I = \{1, \dots, N\}$, two families of objects $\{m_i\}_{i\in I}, \{n_i\}_{i\in I}$ and a family of morphisms $\{a_i : m_i \to n_i\}_{i\in I}$ in $\Bbbk$-**mat**, we have the two direct sums

$$m := \bigoplus_{i\in I} m_i, \quad (\pi_i)_m : m \to m_i, \quad (\iota_i)_m : m_i \to m, \tag{3.5.5}$$

$$n := \bigoplus_{i\in I} n_i, \quad (\pi_i)_n : n \to n_i, \quad (\iota_i)_n : n_i \to n. \tag{3.5.6}$$

This situation is displayed in the following diagram

$$\begin{array}{ccc} m & \xrightarrow{\ a\ } & n \\ {\scriptstyle (\iota_i)_m}\uparrow \ \downarrow{\scriptstyle (\pi_i)_m} & & {\scriptstyle (\iota_i)_n}\uparrow \ \downarrow{\scriptstyle (\pi_i)_n} \\ m_i & \xrightarrow{\ a_i\ } & n_i \end{array}$$

The morphism $a : m \to n$ defined as

$$a = \sum_{i\in I} (\pi_i)_m a_i (\iota_i)_n \tag{3.5.7}$$

satisfies

$$(\iota_i)_m\, a = a_i\, (\iota_i)_n, \tag{3.5.8}$$

$$a\, (\pi_i)_n = (\pi_i)_m\, a_i \ \text{ and} \tag{3.5.9}$$

$$(\iota_i)_m\, a\, (\pi_i)_n = a_i. \tag{3.5.10}$$

This can be interpreted in two ways:

(1) For the family $\iota_m a := \{(\iota_i)_m a : m_i \to n\} := \{a_i\,(\iota_i)_n : m_i \to n\}$ of morphisms with same target n, we have the morphism $u_{\text{out}}(\iota_m a) : m \to n$ such that $(\iota_i)_m u_{\text{out}}(\iota_m a) = (\iota_i)_m a = a_i\,(\iota_i)_n$, and

(2) For the family $a\pi_n := \{a(\pi_i)_n : m \to n_i\} := \{(\pi_i)_m\, a_i : m \to n_i\}$ of morphisms with same source m, we have the morphism $u_{\text{in}}(a\pi_n) : m \to n$ such that $u_{\text{in}}(a\pi_n)(\pi_i)_n = a(\pi_i)_n = (\pi_i)_m\, a_i$.

So we have

$$(\iota_i)_m u_{\text{out}}(\iota_m a)(\pi_i)_n = a_i = (\iota_i)_m u_{\text{in}}(a\pi_n)(\pi_i)_n \text{ and} \tag{3.5.11}$$

$$u_{\text{out}}(\iota_m a) = a = u_{\text{in}}(a\pi_n) \tag{3.5.12}$$

Theorem 3.5.9. *The functor category has all limits, colimits and bilimits which exist in the target category.*

Instead of proving this in general, we prove this as part of 5.1.2 for the direct sum, from which the procedure of the general proof becomes apparent.

4 $\Bbbk$-linear closure of a finite concrete category

4.1 Finite concrete categories and the free/forgetful adjunction

Our model for a finite concrete category $\mathcal{C}$ is that of a finite subcategory of FinSets. In particular we restrict ourselves to finite concrete categories that are generated by a finite set of morphisms `SetOfGeneratingMorphisms` and whose endomorphism monoids are explicitly cyclic.

The `SetOfObjects` $= \{c_1, \ldots, c_N\}$ and the `SetOfGeneratingMorphisms` $= \{a_1, a_2, \ldots, a_{N'}\}$ together define a finite quiver:

Definition 4.1.1 (Finite quivers)**.** A quiver is finite if it consists of finitely many objects and morphisms.

The fact that every category is also a quiver can be expressed in the following as the existence of a certain forgetful functor:

Example 4.1.2. (Forgetful functor $U : \mathcal{C} \to \mathcal{D}$)

(1) We denote by the letter U (for *underlying*) a forgetful functor between two categories $U : \mathcal{C} \to \mathcal{D}$ if we can identify every object $c \in \mathcal{C}$ as an object $Uc \in \mathcal{D}$ by *forgetting* some additional structure that c had in $\mathcal{C}$ but that is not defined for objects in $\mathcal{D}$. The object Uc is called the underlying object of c (e.g. the underlying set of a group).

(2) For a morphism $a : c \to c' \in \mathcal{C}$ that was some structure-preserving map between c and c', if that structure doesn't exist in the category $\mathcal{D}$ then $Ua : Uc \to Uc'$ *forgets* the structure-preserving property of a.

(3) There are other conceivable functors that even map morphisms between two objects (e.g. functors between two categories are morphisms in **Cat**) to objects (e.g. functors in the functor category). If you now want to get back the morphism from the object, you again are using a forgetful functor (e.g. to get the underlying functor of the functor object).

Some authors define a forgetful functor in the strict sense that its target category $\mathcal{D} = \mathbf{Set}$, i.e. it forgets all structure; and functors that only forget some but not all of the algebraic structure are called intermediate forgetful functors.

We are interested in the forgetful functor with $\mathcal{C} = \mathbf{Cat}$ and $\mathcal{D} = \mathbf{Quiv}$:

Example 4.1.3. (Forgetful functor $U : \mathbf{Cat} \hookrightarrow \mathbf{Quiv}$) Let $\mathcal{C} \in \mathbf{Cat}$ be a category. The quiver $q = U\mathcal{C}$ is defined by

$$q_0 = \mathcal{C}_0 \tag{4.1.1}$$
$$q_1 = \mathcal{C}_1 \tag{4.1.2}$$

In particular every identity morphism $1_c \in \mathcal{C}_1$ for an object $c \in \mathcal{C}$ now is just any other endomorphism on that object (but it is still true that $s(1_c) = t(1_c) = c$). And every morphism $\varphi\psi \in \mathcal{C}_1$ that was the composition of φ with ψ is now just any morphism without much deeper connection to φ and ψ apart from

$$s(\varphi\psi) = s(\varphi) \text{ and} \tag{4.1.3}$$
$$t(\varphi\psi) = t(\psi), \tag{4.1.4}$$

which is still true in **Quiv**. Of course, associativity and unital property of the composition μ doesn't exist in **Quiv** since there is no composition of arrows.

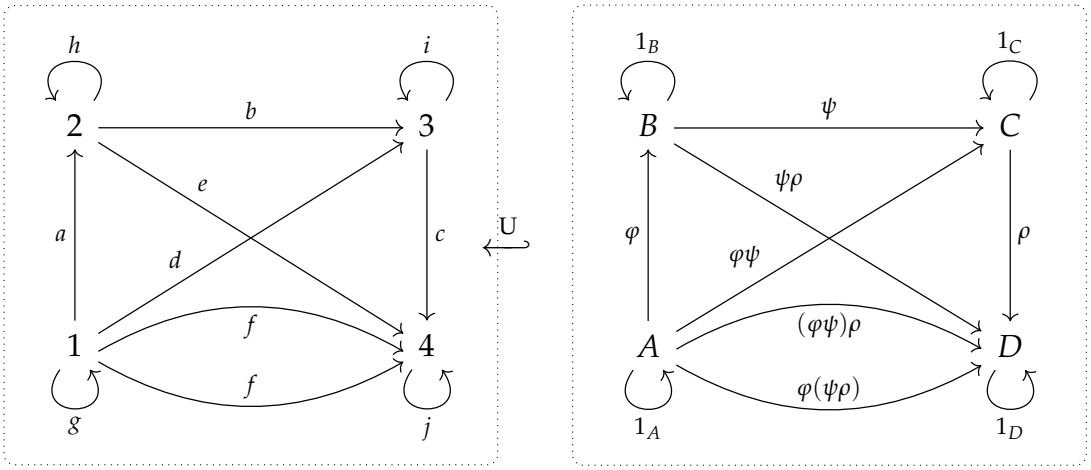
The following algorithm maps the objects and generating morphisms of a finite concrete category $\mathcal{C}$ to a right quiver with the same number of objects and with an arrow for each generating morphism.

Algorithm 1: `RightQuiverFromConcreteCategory`

Input : a finite concrete category C with n objects
Output : the right quiver q with the same number of objects and an arrow for each generating morphism

```
1  Obj := SetOfObjects(C);
2  n := Length(Obj);
3  gMor := SetOfGeneratingMorphisms(C);
4  A := ∅;        // this will be the set of arrows as pairs of natural
     numbers
5  i := 1;
6  foreach morphism mor ∈ gMor do
7  |   A_{i,1} := the position of Source(mor) in Obj;
8  |   A_{i,2} := the position of Range(mor) in Obj;
9  |   i := i + 1;
10 end
11 q := RightQuiver with vertices {1, ..., n} and arrows A.
12 return q;
```

Example 4.1.4. (Underlying quiver)



In the category on the right, associativity of composition guaranteed that $(\varphi\psi)\rho = \varphi(\psi\rho)$, so those two arrows were already the same, so they are mapped to the same arrow $f = U((\varphi\psi)\rho) = U(\varphi(\psi\rho))$ in the quiver on the left. We didn't have to draw both arrows for f, but since they are equal, there is still only one arrow in the hom-set $\mathrm{Hom}_q(1,4) = \{f, f\} = \{f\}$.
All the other identities are not preserved under the forgetful functor, e.g. d doesn't know what it has to do with a and b apart from $s(d) = s(a)$ and $t(d) = t(b)$. Especially the former identity arrows are now just endomorphisms with no defining property.
The paths g^2f, gf and fj^3 are all different, while in the category, they all simplify to $1_A 1_A(\varphi\psi)\rho = 1_A(\varphi\psi)\rho = (\varphi\psi)\rho 1_D 1_D 1_D = (\varphi\psi)\rho$ due to the unit property and associativity.

The category $\mathcal{C}$ in the last example has the set of morphisms

$$\mathcal{C}_1 = \{1_A, 1_B, 1_C, 1_D, \varphi, \psi, \rho, \varphi\psi, \psi\rho, \varphi\psi\rho\},$$

i.e. 10 morphisms. But once the three morphisms φ, ψ and ρ were defined, the other seven morphisms were forced from the unit and composition axioms of a category.

Example 4.1.5. (Category generated by one endomorphism) As another example, take a category $\mathcal{M}$ with one object $*$ and apart from 1_* one other endomorphism $\alpha : * \to *$. It already has a priori countably infinitely many morphisms $\mathcal{M}_1 = \{1_*, \alpha, \alpha^2, \alpha^3, \dots\}$. But the information to generate that category is all encoded in the one morphism α.

What we are looking for is a construction of a finite concrete category from a finite set of generating morphisms. For this we can take the generated quiver from 4.1.1 and from it the free category.

Definition 4.1.6. (The free category F : **Quiv** $\to$ **Cat**)[17] The free category Fq of a quiver q has q_0 as its set of objects. The set $(Fq)_1$ of morphisms consists of all finite paths of arrows in q_1. The identity morphism 1_c of an object $c \in q_0$ is defined as the empty path from c to itself. Composition is defined by concatenation of paths.

Definition 4.1.7. (Free $\dashv$ forgetful adjunction) The functor pair F : **Quiv** $\leftrightarrows$ **Cat** : U is an example for an adjunction, i.e. for the functors F : **Quiv** $\to$ **Cat** and U : **Cat** $\to$ **Quiv** there is an isomorphism

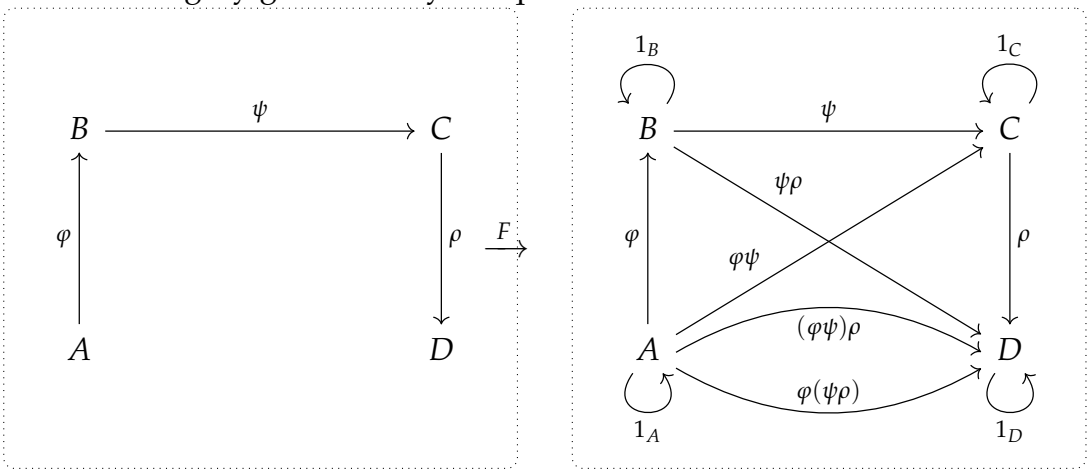
$$\mathrm{Hom}_{\mathbf{Cat}}(Fq, \mathcal{C}) \cong \mathrm{Hom}_{\mathbf{Quiv}}(q, U\mathcal{C}) \tag{4.1.5}$$

for each $q \in$ **Quiv** and $\mathcal{C} \in$ **Cat**, that is natural in both q and $\mathcal{C}$. Here U is right adjoint to F and the forgetful functor U admits a left adjoint, free construction F.

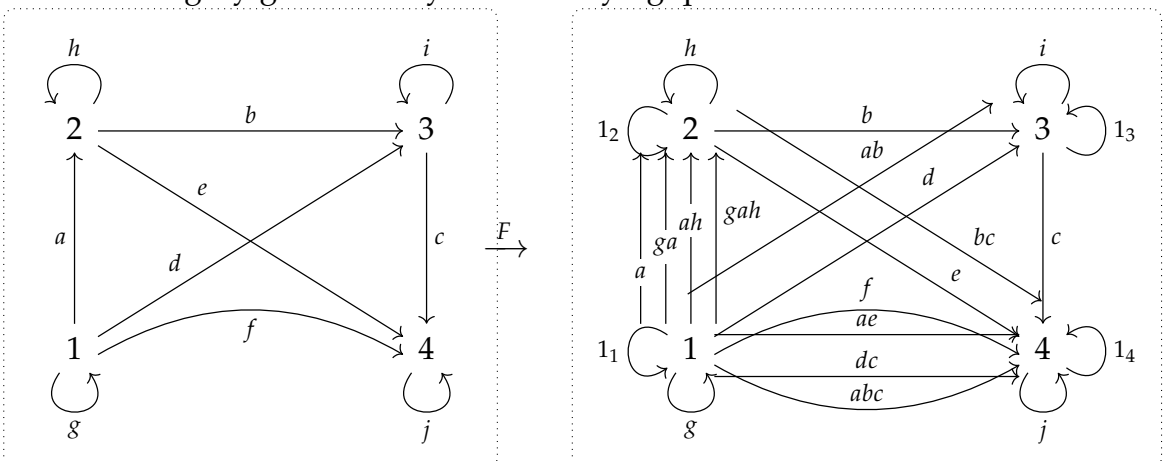
We can view the free category functor in action in two different ways: Where does the category $\mathcal{C}$ in example 4.1.4 come from, i.e. what is the quiver q such that $\mathcal{C} = Fq$? And where does it go after we forget the category structure, i.e. what is the category $F(U(\mathcal{C}))$? We will illustrate the answers to both questions in the next example:

Example 4.1.8. (Generating quiver $\xrightarrow{F}$ category $\xrightarrow{U}$ underlying quiver $\xrightarrow{F}$ category)

(1) The free category generated by the quiver:

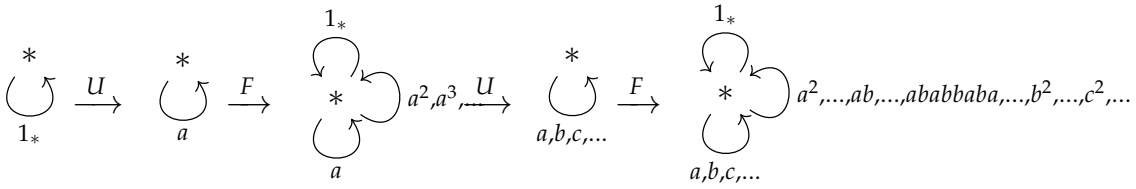


(2) The free category generated by the underlying quiver:



As you can see, this figure gets cluttered very fast (not all morphisms were drawn in the figure). The reason for this is the existence of non-identity endomorphisms. As we have shown in lemma 2.1.7, one non-identity endomorphism is enough for a quiver to have infinitely many paths. Here it is even worse than in the example 4.1.5 where we had countably many endomorphisms $\alpha^n, n \in \mathbb{N}$. Through the arrow $a : 1 \to 2$ we can concatenate countably many morphisms $g^m a h^n, (m, n) \in \mathbb{N} \times \mathbb{N}$ and even $g^{n_1} a h^{n_2} b i^{n_3} c j^{n_4}, (n_1, n_2, n_3, n_4) \in \mathbb{N}^4$. If we were to construct the path algebra (see 4.3.8) on the quiver, it already had an infinite basis.

Example 4.1.9 (Example 4.1.5 continued)**.** As a last example to see how bad it can get from seemingly innocent quivers, take the category with 1 object and its identity morphism:



After the first forgetful functor, we are in the situation of 4.1.5 where then the first free functor gives us countably many morphisms. The following forgetful functor only renames them to countably many distinct morphisms. In the last step, we are constructing the free monoid on countably many generators. Among other things it contains the free monoid on two generators.[18]

We can learn two lessons from these examples:

1. An adjunction is more general than an equivalence of categories. The free functor F doesn't just *undo* the forgetful functor U. You will end up with much more than you started with.

2. If we want to still work with finite categories, we really need to control the size of our hom-sets, especially regarding the endomorphisms. This is the topic of the next section.

4.2 Relations of endomorphisms

Lemma 4.2.1. (σ-Lemma)

(1) Let $\mathcal{C}$ be a finite concrete category. In remark 2.2.9 we showed that for each object $M \in \mathcal{C}_0$ the set $\mathrm{End}_{\mathcal{C}}(M)$ is a monoid. For each endomorphism $f \in \mathrm{End}_{\mathcal{C}}(M)$ there exist $m, n \in \mathbb{N}, n \geq 1$, such that $f^{m+n} = f^m$.

(2) When we restrict both the source and target of f to $\mathrm{Im}(f^m)$, then $f|_{\mathrm{Im}(f^m)}$ is an isomorphism, i.e. $f|_{\mathrm{Im}(f^m)} \in \mathrm{Aut}_{\mathcal{C}}(\mathrm{Im}(f^m))$ with $f|^n_{\mathrm{Im}(f^m)} = 1_{\mathrm{Im}(f^m)}$.

(3) If $m = 0$ and $n \geq 1$ then $f \in \mathrm{Aut}_{\mathcal{C}}(M)$ is an automorphism with $f^{-1} = f^{n-1}$ and has order n.

(4) An upper bound for m is $|M|$ and for n is $g(N)$ with $N = |M|$ and Landau's function[19] $g(N)$ defined as the largest order of an element of the symmetric group S_N.

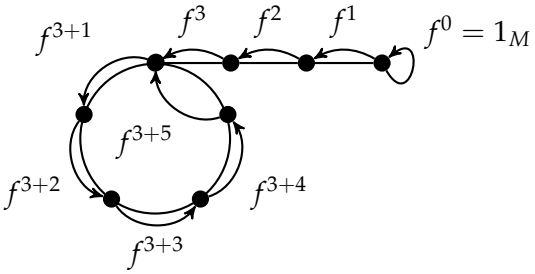


Figure 3: σ lemma with $m = 3$ and $n = 5$.

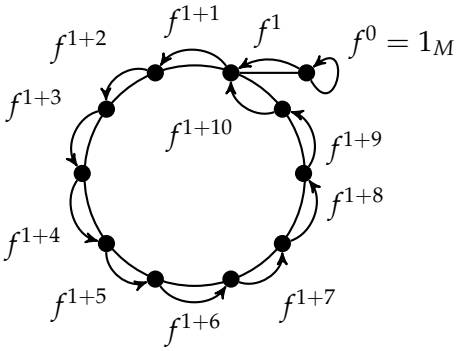


Figure 4: σ lemma with $m = 1$ and $n = 10$.

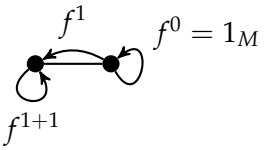


Figure 1: f is idempotent. $m = 1$ and $n = 1$.

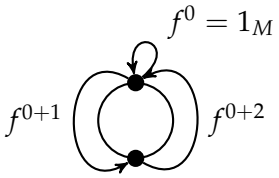


Figure 2: f is bijective with order 2. $m = 0$ and $n = 2$.

Proof of (1). The set of all endomorphisms $\mathrm{End}_{\mathcal{C}}(M)$ of the finite set M with $|M| < \infty$ is also finite:

$$\left|\mathrm{End}_{\mathcal{C}}(M)\right| = \left|M^M\right| = |M|^{|M|} < \infty. \tag{4.2.1}$$

Let $f \in \mathrm{End}_{\mathcal{C}}(M)$. Then $\{f^k : k \in \mathbb{N}_0\} \subseteq \mathrm{End}_{\mathcal{C}}(M)$ is a finite set $\{f^k : k \in \mathbb{N}_0\} = \{f^0, f^1, \ldots, f^N\}$. There is a minimal $l \in \mathbb{N}_{\geq 1}$ such that $\{f^0, f^1, \ldots, f^{l-1}\} = \{f^0, f^1, \ldots, f^l\}$, i.e. the first repetition $f^l = f^j$ for $j \in \{0, \ldots, l-1\}$. Since l is the first such number, the j is also uniquely determined. With $m := j$ and $n := l - j \geq l \geq 1$ we have found m, n satisfying

$$f^{m+n} = f^{j+l-j} = f^l = f^j = f^m. \tag{4.2.2}$$

//

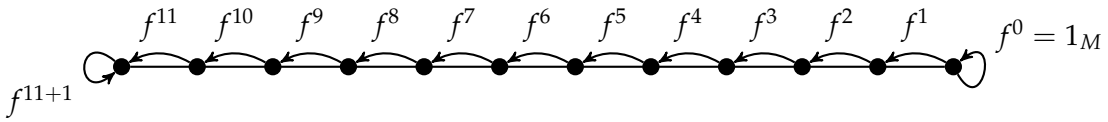
Proof of (2).

$$\begin{aligned} & f^{m+n} && = f^m \\ \Rightarrow & f^n f^m && = f^m \\ \Rightarrow & f^n(y) && = y, \text{ if } y = f^m(x), x \in M \\ \Rightarrow & f^n(y) && = y, \text{ if } y \in \mathrm{Im}(f^m) \\ \Rightarrow & f^n && = 1_{\mathrm{Im}(f^m)} \end{aligned}$$

The identity on $\mathrm{Im}(f^m)$ is $1_{\mathrm{Im}(f^m)} : \mathrm{Im}(f^m) \to \mathrm{Im}(f^m)$ and maps $\mathrm{Im}(f^m) \ni y \mapsto y \in \mathrm{Im}(f^m)$. Let $x \in M$ and $y := f^m(x)$. Then we can identify as the identity on $\mathrm{Im}(f^m)$ the function that maps $y \mapsto y$ iff it maps $f^m(x) \mapsto f^m(x)$.
For $f : \mathrm{Im}(f^m) \to \mathrm{Im}(f^m)$ we have $f^{n-1} : \mathrm{Im}(f^m) \to \mathrm{Im}(f^m)$ such that for all $x \in M$ we have $(f\, f^{n-1})(f^m(x)) = f^{m+n}(x) = f^m(x)$ and $(f^{n-1}\, f)(f^m(x)) = f^{m+n}(x) = f^m(x)$. Therefore $f : \mathrm{Im}(f^m) \to \mathrm{Im}(f^m)$ is an iso with inverse f^{n-1}. //

Proof of (3). Since $f^{m+n} = 1_{\mathrm{Im}(f^m)}$, for $m = 0$ we have $\mathrm{Im}(f^0) = \mathrm{Im}(1_M) = M$ and therefore $f^n = 1_M$, i.e. f is bijective with $f^{-1} = f^{n-1}$. //

Proof of (4). In the case of $m = 0$, the morphism f is a bijective function, i.e. an element of S_M. With the obvious bijection $M \cong \{1, \ldots, N\}$ we also have a bijection of groups $\varphi : S_M \cong S_N$. By the definition of Landau's function g, there is a permutation $a \in S_N$ of order at most $g(N)$ with $a = \varphi(f)$, i.e. $\mathrm{ord}(a) \leq g(N)$, in other words, $\exists n \leq g(N) : a^n = 1_{\{1,\ldots,N\}}$. For the same n we have $f^n = (\varphi^{-1}a)^n = \varphi^{-1}(a^n) = \varphi^{-1}(1_{\{1,\ldots,N\}}) = 1_M$.
In *(2)* we proved that $f|_{\mathrm{Im}(f^m)}$ is bijective on $\mathrm{Im}(f^m)$, therefore an element of $S_{\mathrm{Im}(f^m)}$ and we can find with the same argument as above an upper bound $g(N') \leq g(N), N' := \left|\mathrm{Im}(f^m)\right|$ since $\mathrm{Im}(f^m) \subseteq M$. This gives us an upper bound for n in the general case.
For an upper bound for m let's imagine an example with $m = 11$ and $n = 1$:



If by f^{10} we had already nine distinct functions $f^1, \ldots, f^9$ with no repetitions, and also the inclusions $M \supsetneq \mathrm{Im}(f) \supsetneq \mathrm{Im}(f^2) \supsetneq \cdots \supsetneq \mathrm{Im}(f^9)$, we can only have so many proper subsets of M. If in the chain of images, $|\mathrm{Im}(f^{k+1})| = |\mathrm{Im}(f^k)| - 1$ then after $m = |M|$ steps the set $\mathrm{Im}(f^m)$ is a singleton and the next function $f : \mathrm{Im}(f^m) \to \mathrm{Im}(f^m)$ is forced to be bijective. //

□

Example 4.2.2. (Endomorphism f satisfying the relation $f^{m+n} = f^m$)

(1) For any $m, n \in \mathbb{N}, n \geq 1$ we can find a finite set M and an endomorphism $f \in \mathrm{End}_{\mathrm{FinSet}}(M)$ with $f^{m+n} = f^m$:
Let $\Bbbk$ be a field, e.g. $\Bbbk := \{\cdot, 1\}$, then on the vector space $V := \Bbbk^{m+n}$ with standard basis $B = \{e_1, \ldots, e_{m+n}\}$ we can define the endomorphism f via the companion matrix $\mathcal{M}_p$ to the polynomial $p(x) = x^{m+n} - x^m \in \Bbbk[x]$. This matrix defines an endomorphism
$f : \{1, \ldots, m+n\} \to \{1, \ldots, m+n\}; 1 \mapsto 2, 2 \mapsto 3, \ldots, m+n-1 \mapsto m+n, m+n \mapsto m+1$ on the finite set $M := \{1, \ldots, m+n\}$ satisfying $f^{m+n} = f^m$.

(2) For $m = 3$ and $n = 5$ that is the matrix

$$\mathcal{M}_{x^8-x^3} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \end{pmatrix}$$

which has $p(x) = x^{3+5} - x^3$ as its minimal polynomial, i.e. $\mathcal{M}_p^{3+5} = \mathcal{M}_p^3$. The endomorphism of vector spaces can be restricted to an endomorphism of the set of standard basis elements $\{e_1, \ldots, e_8\}$,

$$f : \{e_1, \ldots, e_8\} \to \{e_1, \ldots, e_8\}; e_1 \mapsto e_2, e_2 \mapsto e_3, \ldots, e_7 \mapsto e_8, e_8 \mapsto e_6$$

and satisfies $f^{3+5} = f^3$.

With the σ lemma 4.2.1 as a tool in our toolbox, we can tackle the problem of the free category generated by a finite quiver of FinSets with endomorphisms.
Recall definition 4.3.8 of the path algebra $\Bbbk q$ of a quiver q, which is the $\Bbbk$-vector space with basis set of all paths in q and concatenation of composable paths as multiplication. By lemma 2.1.7 one cyclic path, i.e. an endomorphism, in q is enough for there to be infinitely many paths, i.e. $\Bbbk q$ is infinite-dimensional.

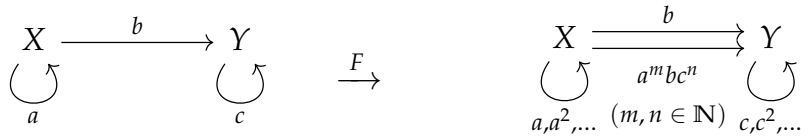
Our algorithm can deal with endomorphism monoids that are explicitly cyclic.

Definition 4.2.3 (Cyclic quiver)**.** We call a quiver q with relations cyclic, if it has at most one loop at each vertex and the endomorphism monoid of each vertex in the category generated by q is generated by the corresponding loop in the quiver. We call the category generated by q a category with explicitly cyclic endomorphism monoids.

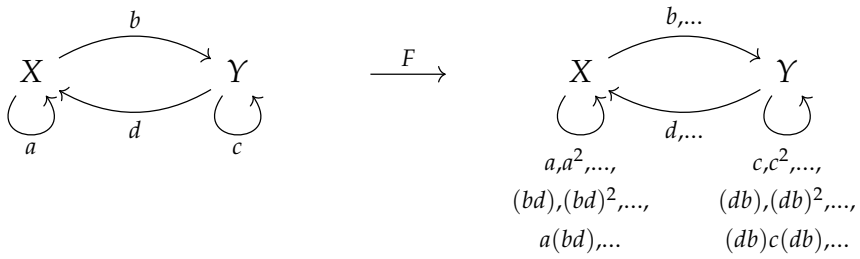
We will give some examples and counterexamples of cyclic quivers and which arrow's existence violates the above condition.

Example 4.2.4. (Examples and counterexamples for cyclic quivers)

(1) A cyclic quiver

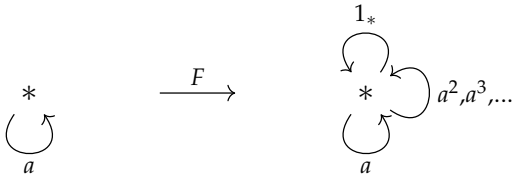


(2) This quiver is not cyclic.

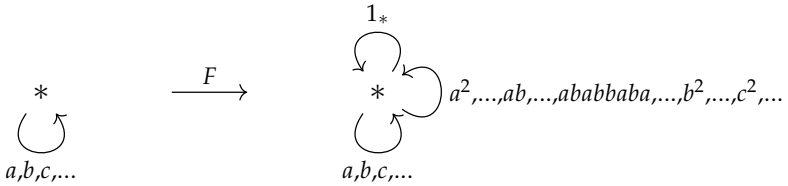


With the path $bd : X \to X$ we have two paths $a, bd \in \mathrm{End}_{Fq}(X)$ and thus an endomorphism monoid that is generated by two generators a and bd, and thus is not explicitly cyclic.[20]

(3) The first quiver from example 4.1.9 is cyclic:



(4) But after another forgetful functor U the underlying quiver is not cyclic:



Definition 4.2.5. (Ideal of an algebra) Let $\mathcal{A}$ be a $\Bbbk$-algebra over a field $\Bbbk$. A subset $L \subset \mathcal{A}$ is a <u>left ideal</u> of $\mathcal{A}$, denoted $L \trianglelefteq \mathcal{A}$, if for every $x, y \in L$, $z \in \mathcal{A}$ and $c \in \Bbbk$ we have the following three statements:

$$x + y \in L \ (L \text{ is closed under addition}) , \tag{4.2.3}$$
$$cx \in L \ (L \text{ is closed under scalar multiplication}) , \tag{4.2.4}$$
$$z \cdot x \in L \ (L \text{ is closed under left multiplication by arbitrary elements}) . \tag{4.2.5}$$

If (4.2.5) were replaced with

$$x \cdot z \in L \ (L \text{ is closed under right multiplication by arbitrary elements}) , \tag{4.2.6}$$

then this would define a <u>right ideal</u>. A <u>two-sided ideal</u> is a subset that is both a left and a right ideal.

Remark 4.2.6. (Ideal of a unital algebra) If in 4.2.5 $\mathcal{A}$ is a unital associative algebra with unit e, then we only need statements (1) and (3). The statement (2) follows from (3).

Proof. For $c \in \Bbbk$ and $x \in L$, $ce \in \mathcal{A}$ and with (3) we have $L \ni (ce) \cdot x = c(e \cdot x) = cx$, i.e. (2). □

Remark 4.2.7. Let $\mathcal{A}$ be a unital $\Bbbk$-algebra, and $\mathcal{I} \trianglelefteq \mathcal{A}$ a two-sided ideal. The <u>quotient algebra</u> of$\mathcal{A}$ modulo $\mathcal{I}$ is the set

$$\mathcal{A} \,/\, \mathcal{I} := \{z + I : z \in \mathcal{A}\}$$

of cosets $z + I$ together with the operations of addition, scalar multiplication and multiplication defined respectively for all $z + I, w + I \in \mathcal{A} \,/\, \mathcal{I}$ by:

$$+ : (z + I) + (w + I) := (z + w) + I \tag{4.2.7}$$
$$\cdot : \lambda \cdot (z + I) := (\lambda z) + I \tag{4.2.8}$$
$$\cdot : (z + I) \cdot (w + I) := (zw) + I \tag{4.2.9}$$

These operations are independend of the representatives.

Proof. Two cosets $(z + I), (z' + I)$ are equal iff $z - z' \in I$. Let $z, z' \in z + I$ and $w, w' \in w + I$ be two representatives for $z + I$ and $w + I$ respectively, and let $\lambda \in \Bbbk$. Therefore $z - z' \in I$ and $w - w' \in I$.

$$(z + w) - (z' + w') = (z - z') + (w - w') \in I \text{ by (4.2.3)} \ \Rightarrow (z + w) + I = (z' + w') + I. \tag{4.2.10}$$

$$\lambda z - \lambda z' = \lambda(z - z') \in I \text{ by (4.2.4)} \ \Rightarrow (\lambda z) + I = (\lambda z') + I. \tag{4.2.11}$$

For well-definedness of multiplication, let $z' = z + i_1$ and $w' = w + i_2$ with $i_1, i_2 \in I$. Then

$$zw - z'w' = zw - (z + i_1)(w + i_2) \tag{4.2.12}$$
$$= zw - zw - zi_2 - i_1 w - i_1 i_2 \tag{4.2.13}$$
$$= -zi_2 - i_1 w - i_1 i_2 \in I, \tag{4.2.14}$$
$$\Rightarrow zw + I = z'w' + I \tag{4.2.15}$$

because of (4.2.3), (4.2.4) and the closure under left and right multiplication (4.2.5) and (4.2.6). □

Remark 4.2.8 (Relations of Endomorphisms)**.** Let the generating quiver q of a finite concrete category $\mathcal{C}$ be explicitly cyclic with one generating endomorphism $\alpha_i : M_i \to M_i \in \mathcal{C}_1$ for each finite set $M_i \in \mathcal{C}_0, i = 1, \dots, N$ with $N := |\mathcal{C}_0|$.
The free category Fq of the quiver q is infinite, since q_1 contains loops that get mapped by the functor F to non-identity endomorphisms in Fq_1. Using the σ-lemma 4.2.1 we can map each endomorphism $\alpha_i \in \mathcal{C}_1$ to a pair of natural numbers $m, n \in \mathbb{N}$ such that $\alpha^{m+n} = \alpha^m$, i.e. there are only finitely many endomorphisms in $\mathcal{C}$. We want to accomplish the same for our abstract category Fq.

The algorithm Algorithm 2 below calculates a list `relsEndo` of pairs of paths $[a^{m+n}, a^m]$ in Fq for each relation $\alpha^{m+n} = \alpha^m$ in $\mathcal{C}$. Let the functor $E : Fq \to \mathcal{C}$ be the unique extension of the obvious embedding of the quiver q in our concrete category $\mathcal{C}$. Then for each endomorphism $a \in q$ with $\alpha = E(a)$ for an endomorphism α with $\alpha^{m+n} = \alpha^m$ we add the relation $[a^{m+n}, a^m]$ as a pair of paths to our list `relsEndo`:

$$\texttt{relsEndo} := [[a_1^{m_1+n_1}, a_1^{m_1}], \dots, [a_N^{m_N+n_N}, a_N^{m_N}]].$$

We can now define an auxiliary finitely presented category $\mathcal{C}'$ from the quiver q modulo the endomorphism relations `relsEndo`,

$$\mathcal{C}' := Fq/\texttt{relsEndo}.$$

This category has finitely many morphisms.

In a second step[21] for all different vertices $s, t \in \mathcal{C}'_0, s \neq t$ we are mapping each non-endomorphism path $p : s \to t \in \mathrm{Hom}_{\mathcal{C}'}(s,t)$ to a list $\texttt{list}(p) := [p, p', \dots]$ containing all the paths $p, p' \in \mathrm{Hom}_{\mathcal{C}'}(s,t)$ that are equal as morphisms $G(p) = G(p') \in \mathcal{C}$, where $G : \mathcal{C}' \to \mathcal{C}$ is the factorization of the functor E over $\mathcal{C}'$. Each list $\texttt{list}(p)$ of length greater than 1 defines a list of pairs of paths

$$\texttt{relations}(p) := [[p, p'], [p, p''], \dots].$$

Taking the union of these lists of pairs of paths over all paths $p : s \to t \in \mathrm{Hom}_{\mathcal{C}'}(s,t)$,

$$\texttt{relations}(s,t) := \bigcup_{p \in \mathrm{Hom}_{\mathcal{C}'}(s,t)} \texttt{relations}(p),$$

and finally over all vertices $s, t \in \mathcal{C}'_0$

$$\texttt{relations} := \bigcup_{s,t \in \mathcal{C}'_0, s \neq t} \texttt{relations}(s,t),$$

we end up with a list of relations such that

$$\begin{aligned} \mathrm{fp}\mathcal{C} :&= \mathcal{C}'/\texttt{relations} \\ &= (Fq/\texttt{relsEndo})/\texttt{relations} \\ &= Fq/(\texttt{relsEndo} \cup \texttt{relations}) \\ &\cong \mathcal{C} \end{aligned}$$

which is a finitely presented category isomorphic to $\mathcal{C}$.[22]

Algorithm 2: `RelationsOfEndomorphisms`

Input: a finite concrete category $\mathcal{C}$

Output: the endomorphism relations of the category C given as list `relsEndo` of pairs of paths

```
1  q := RightQuiverFromConcreteCategory(C);
2  Fq := Categoryclosure(q);
3  gMor := SetOfGeneratingMorphisms(C);
4  A := Arrows(q);
5  relsEndo := ∅;
6  for i = 1,...,Length(gMor) do
7  |  let mor := gMor_i
8  |  if not IsEndomorphism(mor) then
9  |  |  continue;
10 |  end
11 |  mPowers := ∅;
12 |  m := 0;
13 |  foundEqual := false;
14 |  while mor^m ∉ mPowers do
15 |  |  n := 1;
16 |  |  nPowers := ∅;
17 |  |  while not foundEqual and mor^(m+n) ∉ nPowers do
18 |  |  |  if IsCongruentForMorphisms(mor^(m+n), mor^m) then
19 |  |  |  |  Add the relation [Fq.(A_i)^(m+n), Fq.(A_i)^m] to relsEndo;
20 |  |  |  |  foundEqual := true;
21 |  |  |  end
22 |  |  |  Add mor^(m+n) to mpowers;
23 |  |  |  n := n + 1;
24 |  |  end
25 |  |  Add mor^m to mpowers;
26 |  |  m := m + 1;
27 |  end
28 end
29 return relsEndo;
```

4.3 $\Bbbk$-linear categories, $\Bbbk$-algebroids and $\Bbbk$-algebras with orthogonal idempotents

Definition 4.3.1 ($\Bbbk$-linear category, $\Bbbk$-algebroid)**.** Let $\Bbbk$ be a commutative unital ring. A $\Bbbk$-linear category, also called $\Bbbk$-algebroid, $\mathcal{A}$ is a category where every hom-set is a $\Bbbk$-module, and where for $x, y, z \in \mathcal{A}_0$ composition of morphisms

$$\mu : \mathrm{Hom}_{\mathcal{A}}(x, y) \times \mathrm{Hom}_{\mathcal{A}}(y, z) \to \mathrm{Hom}_{\mathcal{A}}(x, z)$$

is $\Bbbk$-bilinear.

Note that this does imply that a $\Bbbk$-linear category is an Ab-category, but it need not be additive nor does it need to have a zero object.

Doctrine 4.3.2 ($\Bbbk$-algebroid)**.** The doctrine `IsAlgebroid` also known as `IsLinearCategoryOverCommutativeRing` involves algorithms for `IsAbCategory` and

- `MultiplyWithElementOfCommutativeRingForMorphisms`.

Definition 4.3.3. ($\Bbbk$-linear functor) Let $\Bbbk$ be a commutative ring. A functor of $\Bbbk$-linear categories or a $\Bbbk$-linear functor is a functor $F : \mathcal{A} \to \mathcal{B}$ between $\Bbbk$-linear categories $\mathcal{A}$ and $\mathcal{B}$, where for all objects $x, y \in \mathcal{A}_0$, the map

$$F : \mathrm{Hom}_{\mathcal{A}}(x, y) \to \mathrm{Hom}_{\mathcal{B}}(F(x), F(y))$$

is a homomorphism of $\Bbbk$-modules.

We can associate to any category a $\Bbbk$-linear category called its $\Bbbk$-linear closure:

Definition 4.3.4. ($\Bbbk$-linear closure of a category) Let $\mathcal{C}$ be a category, and $\Bbbk$ a commutative unital ring. We define $\Bbbk\mathcal{C}$ to be the $\Bbbk$-algebroid with the same object set as $\mathcal{C}$ and with

$$\mathrm{Hom}_{\Bbbk\mathcal{C}}(a, b) = \bigoplus_{\varphi \in \mathrm{Hom}_{\mathcal{C}}(a,b)} \Bbbk \cdot \varphi,$$

the free $\Bbbk$-module on the set of free generators $\mathrm{Hom}_{\mathcal{C}}(a, b)$. We call $\Bbbk\mathcal{C}$ the $\Bbbk$-linear closure of $\mathcal{C}$.

Definition 4.3.5. (Idempotent) Let $\mathbf{A}$ be a unital algebra over the commutative ring $\Bbbk$.

(1) An element $e \in \mathbf{A}$ is an idempotent if $e^2 = e$.

(2) If $e_1, e_2 \in \mathbf{A}$ are idempotents, then we will say that they are orthogonal iff $e_1 e_2 = e_2 e_1 = 0$.

(3) An idempotent is called trivial if it is either 0 or 1.

(4) An idempotent $e \in \mathbf{A}$ is called primitive if $e = e_1 + e_2$ with idempotents e_1, e_2, then e_1 or e_2 is trivial.

(5) A finite set $\{e_1, \ldots, e_n\}$ of orthogonal idempotents is called complete if $\sum_{i=1}^{n} e_i = 1$.

Proposition 4.3.6. Let $\mathbf{A}$ be a unital algebra over the commutative ring $\Bbbk$, and $M_1, \ldots, M_n$ $\mathbf{A}$-submodules of $\mathbf{A}$ such that

$$\mathbf{A} = \bigoplus_{i=1}^{n} M_i, \tag{4.3.1}$$

i.e.

$$\mathbf{A} = \{m_1 + \cdots + m_n : m_i \in M_i\} \text{ and}$$
$$M_i \cap M_j = \{0\} \text{ if } i \neq j$$

Write $1 = \sum_{i=1}^{n} e_i$ for some $e_i \in M_i$. Then the e_i are orthogonal idempotents in $\mathbf{A}$, and

$$M_i = \mathbf{A}e_i,\ 1 \leq i \leq n.$$

Furthermore, M_i is indecomposable as a module if and only if e_i is primitive.

Proof that the e_i are orthogonal idempotents.

$$1 = \sum_{i=1}^{n} e_i$$
$$e_j = e_j \sum_{i=1}^{n} e_i = e_j^2 + \sum_{i \neq j} e_j e_i$$

Since we can write $e_j = m_1 + \cdots + m_n$ in a unique way with $m_j \in M_j$, it follows that

$$e_j = m_j = e_j^2 \text{ and}$$
$$m_i = e_j e_i = 0 \,\forall i \neq j,$$

i.e. the e_j are orthogonal idempotents in $\mathbf{A}$. //

Proof that $M_i = \mathbf{A}e_i$.
Since the M_i are $\mathbf{A}$-submodules, they are already closed under multiplication by elements in $\mathbf{A}$. So with $e_i \in M_i$ we have $ze_i \in M_i \,\forall z \in \mathbf{A}$, i.e. $\mathbf{A}e_i \subseteq M_i$. We only have to show the other inclusion.

"$\subseteq$:" Let $x \in M_i$. We are done, if we show that $x = xe_i$. With $1 = \sum_{i=1}^{n} e_i$ we have

$$x = x1 = x \sum_{j=1}^{n} e_j = xe_i + \sum_{j \neq i} xe_j$$

By the above, we have $xe_j \in M_j$ for $j \neq i$, and since $x \in M_i$, we again have the unique sum $x = m_1 + \cdots + m_n$ with $m_i = xe_i$ and $m_j = xe_j = 0$ for $j \neq i$. Therefore $x = xe_i$, i.e. $M_i \subseteq \mathbf{A}e_i$.

//

Proof of equivalence M_i indecomposable $\Leftrightarrow$ m_i primitive.

"$\Rightarrow$:" Let M_i be indecomposable. Let $m_i = n + p$ for some idempotent $n, p \in \mathbf{A}$. Then $M_i = m_i\mathbf{A} = (n+p)\mathbf{A} = \{nx + py : x, y \in \mathbf{A}\} = n\mathbf{A} \oplus p\mathbf{A}$. Since M_i was indecomposable, it follows that $n\mathbf{A} = \{0\} \vee p\mathbf{A} = \{0\}$. Therefore $n = 0 \vee p = 0$, i.e. m_i is primitive.

"$\Leftarrow$:" Let m_i be primitive. Let $m_i\mathbf{A} = M_i = N \oplus P$. Since

$$\mathbf{A} = \bigoplus_{j=1}^{n} M_j = \bigoplus_{\substack{j=1,\\ j \neq i}}^{n} M_j \oplus (N \oplus P)$$

there exist orthogonal idempotent $n, p \in \mathbf{A}$ such that $N = n\mathbf{A}$ and $P = p\mathbf{A}$ and also $1 = \sum_{j=1}^{n} m_j = \sum_{j \neq i} m_j + n + p$. Therefore $m_i = n + p$. With m_i primitive, we have $n = 0 \vee p = 0$, and therefore $M_i = \{0\} \oplus (p\mathbf{A}) \vee M_i = (n\mathbf{A}) \oplus \{0\}$, i.e. M_i is indecomposable.

//

□

A unital associative $\Bbbk$-algebra is a $\Bbbk$-algebroid with one object. Let $\Bbbk$ be a commutative unital ring and $\mathbf{A}$ a unital associative algebra over $\Bbbk$. This defines a category $\mathcal{A}$ with a single object $*$ and the morphisms being the elements of the algebra, which are all endomorphisms since there is only one object. Composition of morphisms is defined by the multiplication in $\mathbf{A}$, which is assumed to be associative. The unit e of the algebra acts as the identity morphism. The set $\{e\}$ is trivially a set of orthogonal idempotents. The hom-set $\mathrm{Hom}_{\mathcal{A}}(*,*) = \mathrm{End}_{\mathcal{A}}(*)$ is the whole algebra $\mathbf{A}$, which is a $\Bbbk$ module with bilinear multiplication as composition, i.e. $\mathcal{A}$ is a $\Bbbk$-algebroid.

Proposition 4.3.7. Each $\Bbbk$-algebra $\mathbf{A}$ with $\{e_1, \dots, e_n\}$ a finite complete system of (not necessarily primitive) orthogonal idempotents defines a $\Bbbk$-algebroid $\mathcal{A}$ with object set $\mathcal{A}_0 := \{1, \dots, n\}$. More precisely:

(1) The set of morphisms between two objects i, j is defined as $\mathrm{Hom}_{\mathcal{A}}(i,j) := e_i \mathbf{A} e_j$, i.e. a morphism is an element $e_i \mathbf{A} e_j \ni \alpha = e_i a e_j, a \in \mathbf{A}$. It follows that the identity morphism $1_i = e_i$ for $1 \leq i \leq n$.

(2) $\mathbf{A} := \mathrm{Algebra}(\mathcal{A}) := \bigoplus_{i,j} \mathrm{Hom}_{\mathcal{A}}(i,j)$ is again a unital algebra with multiplication

$$\varphi_{i,j} \cdot \psi_{k,l} := \begin{cases} \varphi_{i,j}\psi_{k,l} & \text{if } j = k \\ 0 \in \mathbf{A} & \text{if } j \neq k, \end{cases}$$

where the first case is the composition in the algebroid. It follows that $1 = \sum_i 1_i$ is the multiplicative identity of $\mathbf{A}$.

These two constructions are mutually inverse.

Proof. (1) We can write $\mathbf{A}$ as

$$\mathbf{A} = 1\mathbf{A}1 = \left(\sum_i e_i\right) \mathbf{A} \left(\sum_j e_j\right) = \bigoplus_{i,j} e_i \mathbf{A} e_j,$$

which is a direct sum decomposition of $\mathbf{A}$ in $\Bbbk$-submodules. The last sum is a direct sum:

$$e_i \mathbf{A} e_j \cap e_{i'} \mathbf{A} e_{j'} = \delta_{i,i'} \cdot \delta_{j,j'}, \text{ since} \tag{4.3.2}$$

$$x = e_i \cdot a \cdot e_j = e_{i'} \cdot b \cdot e_{j'} \tag{4.3.3}$$

$$x = \underbrace{e_i e_i}_{e_i} \cdot a \cdot e_j = \underbrace{e_i e_{i'}}_{=0} \cdot b \cdot e_{j'}, \qquad e_i \neq e_{i'} \tag{4.3.4}$$

$$x = e_i \cdot a \cdot \underbrace{e_j e_j}_{e_j} = e_{i'} \cdot b \cdot \underbrace{e_{j'} e_j}_{=0}, \qquad e_j \neq e_{j'} \tag{4.3.5}$$

(2) Composition of morphisms $i \xrightarrow{\alpha} j \xrightarrow{\beta} k$ is defined by the multiplication in $\mathbf{A}$. Let $\alpha = e_i a e_j, \beta = e_j b e_k$ with $a, b \in \mathbf{A}$. Then

$$\alpha\beta = (e_i a e_j)(e_j b e_k) \tag{4.3.6}$$

$$= e_i (a e_j b) e_k \tag{4.3.7}$$

$$= e_i c e_k \text{ with } c := a e_j b \in \mathbf{A}. \tag{4.3.8}$$

Associativity of composition follows from associativity of multiplication in $\mathbf{A}$.

(3) It's an easy exercise to see that the above defined 1 is indeed a multiplicative unit of the algebra $\mathbf{A}$. We define $M_i := \bigoplus_{j=1}^{n} \mathrm{Hom}_{\mathcal{A}}(j,i)$ an $\mathbf{A}$-submodule of $\mathbf{A}$. Then $\mathbf{A} = \bigoplus_{i=1}^{n} M_i$ as in Proposition 4.3.6 and it follows that the idempotent $e_i = 1_i$.

□

Definition 4.3.8. (Path algebra of a quiver) We define the path algebra of a quiver q over a commutative ring $\Bbbk$ by

$$\Bbbk q := \mathrm{Algebra}(\Bbbk Fq),$$

where Fq is the free category of the quiver q from definition 4.1.6.

Lemma 4.3.9. For a quiver q and a field $\Bbbk$, the path algebra $\Bbbk q$ is an associative $\Bbbk$-algebra.[23]

Proof. Let w, w', w'' be paths. Then both $(ww')w''$ and $w(w'w'')$ are the concatenation of w on the left, w' in the middle and w'' on the right, in case both conditions $t(w) = s(w')$ and $t(w') = s(w'')$ are satisfied, and otherwise the zero element (since $(ww')0 = 0, 0(w'w'') = 0$, according to bilinearity).
Since the multiplication was defined on a basis and extended bilinearly, the axioms of an algebra are clearly satisfied. □

Lemma 4.3.10. If the set of vertices of a quiver q_0 is finite, then $\Bbbk q$ has a unit element $\sum_{x \in q_0} e_x$. In this case, $\Bbbk q$ is a unital associative algebra, i.e. a unital ring that is also a vector space.[24]

Proof. Let $e := \sum_{x \in q_0} e_x$. Let w be a path with $s(w) = x$ and $t(w) = y$, then $e_x w = w$ and $e_z w = 0$ for all $z \neq x$, thus $ew = e_x w + \sum_{z \neq x} e_z w = w + 0 = w$. Similarly, $we_y = w$ and $we_z = 0$ for $z \neq x$. □

Computation 4.3.11 (A finitely presented category isomorphic to the finite concrete category $C_3C_3C_3$)**.**

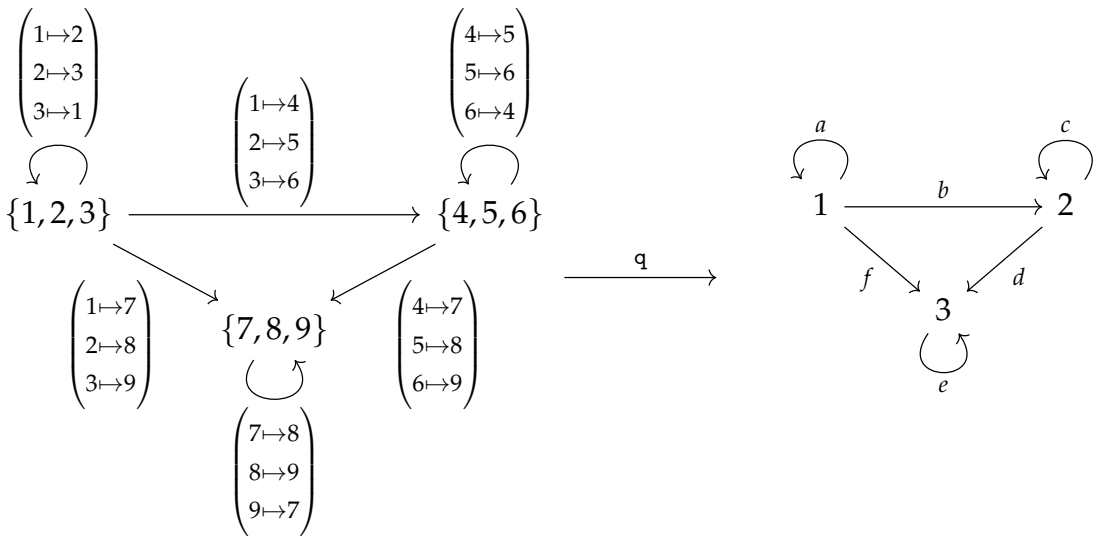
Example

```
gap> c3c3c3 := ConcreteCategoryForCAP(
>                  [ [2,3,1], [4,5,6], [,,,5,6,4],
>                    [,,,7,8,9], [,,,,,,8,9,7], [7,8,9] ] );
A finite concrete category
gap> objects := SetOfObjects( c3c3c3 );
[ An object in subcategory given by: <An object in FinSets>,
  An object in subcategory given by: <An object in FinSets>,
  An object in subcategory given by: <An object in FinSets> ]
gap> gmorphisms := SetOfGeneratingMorphisms( c3c3c3 );
[ A morphism in subcategory given by: <A morphism in FinSets>,
  A morphism in subcategory given by: <A morphism in FinSets>,
  A morphism in subcategory given by: <A morphism in FinSets>,
  A morphism in subcategory given by: <A morphism in FinSets>,
  A morphism in subcategory given by: <A morphism in FinSets>,
  A morphism in subcategory given by: <A morphism in FinSets> ]
gap> q := RightQuiverFromConcreteCategory( c3c3c3 );
q(3)[a:1->1,b:1->2,c:2->2,d:2->3,e:3->3,f:1->3]
gap> relEndo := RelationsOfEndomorphisms( c3c3c3 );
[ [ (a*a*a), (1) ], [ (c*c*c), (2) ], [ (e*e*e), (3) ] ]
gap> C := AsFpCategory( c3c3c3 );
Category generated by the right quiver
q(3)[a:1->1,b:1->2,c:2->2,d:2->3,e:3->3,f:1->3] with relations
```

The underlying quiver algebra of $\mathcal{C}$ is a quotient algebra by the following relations. It also has a finite set of vertices, thus it has a unit.

Example

```
gap> A := UnderlyingQuiverAlgebra( C );
(Q * q) / [ 1*(a*a*a) - 1*(1), 1*(c*c*c) - 1*(2), 1*(e*e*e) - 1*(3),
1*(b*c) - 1*(a*b), 1*(b*d) - 1*(f), 1*(f*e) - 1*(a*f), 1*(d*e) - 1*(c*d) ]
gap> unit := A.1 + A.2 + A.3;
{ 1*(3) + 1*(2) + 1*(1) }
gap> unit * A.a = A.a;
true
gap> A.f * unit = A.f;
true
```



5 The functor category with values in an abelian category is an abelian category

From now on instead of dealing with a finite concrete category $\mathcal{C}$ directly, we can instead take its finite-dimensional $\Bbbk$-linear closure, i.e. the algebroid $\mathcal{A}$. This is the source of our functors in the functor category $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$. The target of our functors is the abelian category $\Bbbk$-**mat**.

In the following sections we prove that the functor category with values in $\Bbbk$-**mat** is an Abelian category with enough projectives (constructively) and direct sum decomposition (constructively).

5.1 $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ is a $\Bbbk$-linear abelian category

Definition 5.1.1 (The functor category $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$)**.** Let $\Bbbk$ be a commutative ring, and let $\mathcal{A}$ be a finite-dimensional algebroid over $\Bbbk$.
The functor category $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ has

- as objects $\Bbbk$-linear functors $F : \mathcal{A} \to \Bbbk\text{-}\mathbf{mat} \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0$, where each object $i \in \mathcal{A}$ gets mapped to a natural number $F(i) \in \Bbbk\text{-}\mathbf{mat}_0 = \mathbb{N}_0$, and each morphism $b : i \to j \in \mathcal{A}_1$ gets mapped to an $F(i) \times F(j)$ matrix $F(b) : F(i) \to F(j) \in \Bbbk\text{-}\mathbf{mat}_1$,
- as morphisms $\alpha : F \to G \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_1$ natural transformations $\alpha : F \Rightarrow G$ with each component $\alpha_i : F(i) \to G(i) \in \Bbbk\text{-}\mathbf{mat}_1$ for an object $i \in \mathcal{A}$ being an $F(i) \times G(i)$ matrix satisfying the naturality conditions in (2.4.1), i.e.

$$\begin{array}{ccc} F(i) & \xrightarrow{\alpha_i} & G(i) \\ \downarrow{\scriptstyle Fb} & & \downarrow{\scriptstyle Gb} \\ F(j) & \xrightarrow{\alpha_j} & G(j) \end{array}$$

commutes for all $b : i \to j \in \mathcal{A}_1$, i.e.

$$\alpha_i \, Gb = Fb \, \alpha_j \tag{5.1.1}$$

Theorem 5.1.2. *Let $\mathcal{A}$ be a finite-dimensional algebroid over some field $\Bbbk$. The functor category $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ is a $\Bbbk$-linear abelian category.*

Proof. We prove that $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ is a $\Bbbk$-linear category, then that it is also a $\Bbbk$-linear additive category and a $\Bbbk$-linear pre-abelian category and finally that it is a $\Bbbk$-linear abelian category.

(1) In order to show that $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ is a $\Bbbk$-linear category, we must show that

(i) for any two objects $F, G \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0$ the hom-set

$$\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})}(F, G)$$

between them is a $\Bbbk$-module, and

(ii) that the composition of two morphisms

$$\mu : \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})}(F, G) \times \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})}(G, H) \to \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})}(F, H)$$

is a $\Bbbk$-bilinear map, i.e. for

$$\begin{aligned}\eta, \varepsilon &\in \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G),\\ \varphi, \psi &\in \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(G, H) \text{ and for}\\ x &\in \Bbbk\end{aligned}$$

$$\begin{aligned}(\eta+\varepsilon)\varphi &= \eta\varphi + \varepsilon\varphi\\ \eta(\varphi+\psi) &= \eta\varphi + \eta\psi\\ (x\eta)\varphi &= \eta(x\varphi) = x(\eta\varphi).\end{aligned}$$

Proof of (i). For any object $c \in \mathcal{A}$, the set of components $\mathrm{Hom}_{\Bbbk\text{-}\mathbf{mat}}(Fc, Gc)$ is a $\Bbbk$-module.

We define the addition and scalar multiplication

$$\begin{aligned}+ :\ & \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G) \times \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G) \to \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G)\\ \cdot :\ & \Bbbk \times \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G) \to \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G)\end{aligned}$$

component-wise: $\forall \eta, \varepsilon \in \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G), \forall x \in \Bbbk, \forall c \in \mathcal{A}$

$$(\eta+\varepsilon)_c := \eta_c + \varepsilon_c \tag{5.1.2}$$
$$(x\eta)_c := x\,\eta_c \tag{5.1.3}$$

where the right-hand side is the usual addition and scalar multiplication of matrices. This includes the additive inverse $-\eta$ for $x = -1$ and the zero morphism $0_{F,G}$ for $x = 0$:

We identify as the neutral element $0_{F,G} \in \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G)$ the natural transformation $0_{F,G}$ with each component $(0_{F,G})_c = 0_{Fc,Gc}$ being the $Fc \times Gc$ zero matrix. For each natural transformation η the additive inverse $-\eta$ is defined component-wise as

$$(-\eta)_c := -\eta_c. \tag{5.1.4}$$

We also confirm that the addition is commutative:

$$(\eta+\varepsilon)_c = \eta_c + \varepsilon_c \tag{5.1.5}$$
$$= \varepsilon_c + \eta_c \tag{5.1.6}$$
$$= (\varepsilon+\eta)_c \tag{5.1.7}$$

This concludes that for each $F, G \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$, $\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G)$ is a $\Bbbk$-module. //

Proof of (ii). Let $F, G, H \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ and let $\eta, \varepsilon \in \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G)$, $\varphi, \psi \in \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(G, H)$ and $x \in \Bbbk$.
The composition $\eta\varphi \in \mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, H)$ is defined by component-wise matrix-multiplication, i.e. $\forall c \in \mathcal{A}$

$$(\eta\varphi)_c := \eta_c\varphi_c$$

and from this follows the $\Bbbk$-bilinearity of the composition, since the matrix multiplication is bilinear.
This concludes the first part of the proof, i.e. $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ is a $\Bbbk$-linear category. //

(2) Next we show that $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ is a $\Bbbk$-linear additive category, i.e. it is

(i) A $\Bbbk$-linear category with

(ii) A dependent function $\oplus$ mapping a finite set I and a collection $(F_i)_{i\in I}$ of objects in $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ to a corresponding direct sum

$$(\oplus_{i\in I} F_i, (\pi_i)_{i\in I}, (\iota_i)_{i\in I}, u_{\text{in}}, u_{\text{out}}).$$

Proof of (ii). We will make extensive use of the direct sum in $\Bbbk$-**mat** as described in 3.5.3 and 3.5.8 to define the direct sum in $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$. Let $I = \{1, \ldots, N\}$ be a finite set, and $\{F_i\}_{i\in I}$ a family of objects in $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0$.

- The object $F := \oplus_{i\in I} F_i$ is a functor defined by its image on objects $c \in \mathcal{A}_0$ and its image on morphisms $a : c \to c' \in \mathcal{A}_1$ in the following way:
  - For an object $c \in \mathcal{A}_0$ we have a family $\{F_i\, c\}_{i\in I}$ of objects in $\Bbbk\text{-}\mathbf{mat}_0$ for which we can define their direct sum

$$F\, c := \left(\bigoplus_{i=1}^{N} F_i\right) c := \bigoplus_{i=1}^{N} (F_i\, c) := \sum_{i=1}^{N} (F_i\, c). \tag{5.1.8}$$

  - The projections $\pi_i : F \to F_i$ and coprojections $\iota_i : F_i \to F$ are defined component-wise exactly as in (3.5.1) and (3.5.2), i.e.

$$(\pi_i)_c := \begin{pmatrix} 0_{F_{<i}(c), F_i(c)} \\ 1_{F_i(c)} \\ 0_{F_{>i}(c), F_i(c)} \end{pmatrix} \tag{5.1.9}$$

$$(\iota_i)_c := \begin{pmatrix} 0_{F_i(c), F_{<i}(c)} & 1_{F_i(c)} & 0_{F_i(c), F_{>i}(c)} \end{pmatrix} \tag{5.1.10}$$

  We verify the property of a direct sum, that

$$\sum_{i\in I} (\pi_i)(\iota_i) = 1_F \text{ and}$$
$$(\iota_i)(\pi_j) = (\delta_{i,j}) = \begin{cases} 1_{F_i} & \text{if } i = j \\ 0_{F_i, F_j} & \text{if } i \neq j \end{cases}$$

  which again can be checked component-wise

$$\sum_{i\in I} (\pi_i)_c (\iota_i)_c = 1_{Fc} \text{ and}$$
$$(\iota_i)_c (\pi_j)_c = (\delta_{i,j})_c = \begin{cases} 1_{F_i c} & \text{if } i = j \\ 0_{F_i c, F_j c} & \text{if } i \neq j \end{cases}$$

  - For a morphism $a : c \to c' \in \mathcal{A}_1$ we have a family of morphisms $\{F_i a : F_i c \to F_i c'\}_{i\in I}$. Analogous to 3.5.8 we define

$$Fa := \sum_{i\in I} (\pi_i)_c F_i a (\iota_i)_{c'} : Fc \to Fc'$$

which satisfies

$$\begin{aligned}(\iota_i)_c F a (\pi_i)_{c'} &= (\iota_i)_c \sum_{j\in I} (\pi_j)_c F_j a (\iota_j)_{c'} (\pi_i)_{c'} \\ &= \sum_{j\in I} (\iota_i)_c (\pi_j)_c F_j a (\iota_j)_{c'} (\pi_i)_{c'} \\ &= \sum_{j\in I} (\delta_{i,j})_c F_j a (\delta_{j,i})_{c'} \\ &= 1_{F_i c} F_i a 1_{F_i c'} \\ &= F_i a\end{aligned}$$

This defines how F works on objects and on morphisms.

- For a family $\tau = \{\tau_i : G \to F_i\}_{i\in I}$ of natural transformations with the same source $G \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0$ we have for each object $c \in \mathcal{A}_0$ a family $\tau_c = \{(\tau_i)_c : Gc \to F_i c\}_{i\in I}$ of morphisms in $\Bbbk\text{-}\mathbf{mat}_1$ with same source $Gc \in \Bbbk\text{-}\mathbf{mat}_0$. For these we have by (3.5.3) the morphism $u_{\mathrm{in}}(\tau_c) : Gc \to Fc$ such that

  $$u_{\mathrm{in}}(\tau_c)(\pi_i)_c = (\tau_i)_c$$

  We can thus define the natural transformation

  $$u_{\mathrm{in}}(\tau) : G \to F \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_1$$

  by the components

  $$(u_{\mathrm{in}}(\tau))_c := u_{\mathrm{in}}(\tau_c)$$

  and we can verify that $u_{\mathrm{in}}(\tau)(\pi_i)$ is the natural transformation with components

  $$\begin{aligned}(u_{\mathrm{in}}(\tau)(\pi_i))_c &= (u_{\mathrm{in}}(\tau))_c (\pi_i)_c \\ &= u_{\mathrm{in}}(\tau_c)(\pi_i)_c \\ &= (\tau_i)_c\end{aligned}$$

  per definition, i.e. $u_{\mathrm{in}}(\tau)$ is the natural transformation fulfilling $u_{\mathrm{in}}(\tau)(\pi_i) = \tau_i$.

- Analogous for a family $\rho = \{\rho_i : F_i \to H\}$ of natural transformations with the same target $H \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0$ we have for each object $c \in \mathcal{A}_0$ a family $\rho_c = \{(\rho_i)_c : F_i c \to Hc\}_{i\in I}$ and get the natural transformation

  $$u_{\mathrm{out}}(\rho) : F \to H \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_1$$

  with components

  $$(u_{\mathrm{out}}(\rho))_c := u_{\mathrm{out}}(\rho_c)$$

  fulfilling $\iota_i u_{\mathrm{out}}(\rho) = \rho_i$.

All in all we have

$$\begin{array}{lll}
\forall c \in \mathcal{A}_0, & & (\oplus_{i\in I} F_i)c = \oplus_{i\in I}(F_i c) \\
\forall c \in \mathcal{A}_0, & \forall i \in I, & (\pi_i : F \to F_i)_c = (\pi_i)_c : Fc \to F_i c \\
\forall c \in \mathcal{A}_0, & \forall i \in I, & (\iota_i : F_i \to F)_c = (\iota_i)_c : F_i c \to Fc \\
\forall c \in \mathcal{A}_0, & \forall \tau = \{\tau_i : G \to F_i\}_{i=1,\dots,n}, & (u_{\mathrm{in}}(\tau))_c = u_{\mathrm{in}}(\tau_c) \\
\forall c \in \mathcal{A}_0, & \forall \rho = \{\rho_i : F_i \to H\}_{i=1,\dots,n}, & (u_{\mathrm{out}}(\rho))_c = u_{\mathrm{out}}(\rho_c)
\end{array}$$

where $\tau_c = \{(\tau_i)_c : Gc \to F_i c\}_{i\in I}$ and $\rho_c = \{(\rho_i)_c : F_i c \to Hc\}_{i\in I}$.

And thus we proved that $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ is a $\Bbbk$-linear additive category. //

(3) Next we show that $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ is a $\Bbbk$-linear pre-abelian category, i.e.

(i) a $\Bbbk$-linear additive category with

(ii) a dependent function mapping a morphism $\eta : F \to G \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_1$ to a kernel of η and

(iii) a dependent function mapping a morphism $\eta : F \to G \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_1$ to a cokernel of η.

Proof of (ii). For a morphism $\eta : F \to G \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_1$ we want to define its kernel, i.e. the triple

- $K := \mathrm{KernelObject}(\eta)$
- $\kappa := \mathrm{KernelEmbedding}(\eta) : K \to F$ such that $\kappa\eta = 0_{K,G}$.
- For any other $\tau : T \to F$ such that $\tau\eta = 0_{T,G}$ we have a unique morphism $(\tau/\eta) := \mathrm{KernelLift}(\eta, \tau)$ such that $\tau = (\tau/\eta)\kappa$

For the components $\eta_c : Fc \to Gc \in \Bbbk\text{-}\mathbf{mat}_1$ we have in $\Bbbk$-**mat** the kernel object

$$Kc := \mathrm{KernelObject}(\eta_c)$$

and the kernel embedding

$$\kappa_c := \mathrm{KernelEmbedding}(\eta_c) : Kc \to Fc$$

such that $\kappa_c\eta_c = 0_{Kc,Fc}$. We define the kernel embedding of η as

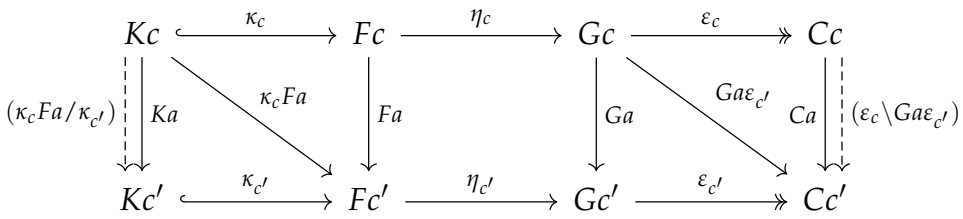
$$\kappa : K \hookrightarrow F$$

by its components

$$\kappa_c = \mathrm{KernelEmbedding}(\eta)_c := \mathrm{KernelEmbedding}(\eta_c) : Kc \to Fc$$

with source $K = \mathrm{KernelObject}(\eta)$ defined on objects as $Kc := \mathrm{KernelObject}(\eta_c)$ which is the source of the kernel embedding of η_c.
Rather than proving that the so defined $\kappa : K \to F$ is a natural transformation, we assume it is and fill in the blanks, i.e. how K acts on morphisms.

We have for each morphism $a : c \to c' \in \mathcal{A}_1$ the morphisms $Fa : Fc \to Fc'$ and $Ga : Gc \to Gc'$. Then the kernel embedding κ of a natural transformation $\eta : F \to G$ has components $\kappa_c : Kc \to Fc$ with $\kappa_c = \mathrm{KernelEmbedding}(\eta_c)$ and the kernel object K acts on objects as $\mathrm{KernelObject}(\eta)c = \mathrm{KernelObject}(\eta_c) = Kc$. In the following we show that K acts on morphisms via the kernel lift $Ka := (\kappa_c Fa/\kappa_{c'}) := \mathrm{KernelLift}(\eta_c, \kappa_c Fa)$:

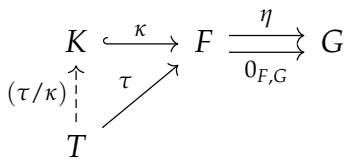


The composition morphism $\kappa_c Fa : Kc \to Fc'$ is a competing morphism to $\kappa_{c'} : Kc' \hookrightarrow Fc'$ in that for both morphisms we have $\kappa_c Fa\, \eta_{c'} = 0_{Kc,Gc'}$ and $\kappa_{c'}\, \eta_{c'} = 0_{Kc',Gc'}$. The second equation is just the definition of kernel embedding $\kappa_{c'}$, while the first equation comes from the commutative center square $Fa\, \eta_{c'} =$

$\eta_c\, Ga$ and thus $\kappa_c\, Fa\, \eta_{c'} = \kappa_c\, \eta_c\, Ga$ which again with the definition of the kernel embedding κ_c gives $0_{Kc,Gc}\, Ga = 0_{Kc,Gc'}$.

As we have seen in 3.4.17, the kernel embedding $\kappa_{c'}$ dominates $\kappa_c\, Fa$, i.e. there exists a unique lift $(\kappa_c Fa/\kappa_{c'}) : Kc \to Kc'$ with $(\kappa_c Fa/\kappa_{c'})\, \kappa_{c'} = \kappa_c\, Fa$. Naturality of the left square, i.e. $\kappa_c\, Fa = Ka\, \kappa_{c'}$ leads to $Ka\, \kappa_{c'}\, \eta_{c'} = \kappa_c\, Fa\, \eta_{c'} = 0_{Kc,Gc'}$ and thus by the above argument, $Ka = (\kappa_c Fa/\kappa_{c'})$ what we wanted to show.

Now for the kernel lift $(\tau/\kappa) := \mathrm{KernelLift}(\eta, \tau)$:



Let $\kappa : K \hookrightarrow F$ be the kernel embedding of the morphism $\eta : F \to G$, and let $\tau : T \to F$ with $\tau\eta = 0_{T,G}$. For all $c \in \mathcal{A}_0$ we have $\tau_c : Tc \to Fc$ with $\tau_c\eta_c = 0_{Tc,Gc}$. Therefore in $\Bbbk$-**mat** we have $(\tau_c/\kappa_c) := \mathrm{KernelLift}(\eta_c, \tau_c)$ such that $\tau_c = (\tau_c/\kappa_c)\kappa_c$.

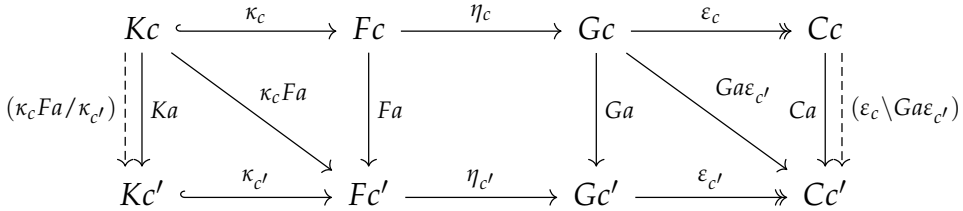
The natural transformation $\lambda : T \to K$ defined by its components
$\lambda_c := (\tau_c/\kappa_c) := \mathrm{KernelLift}(\eta_c, \tau_c)$ satisfies $\tau_c = \lambda_c\kappa_c = (\tau_c/\kappa_c)\kappa_c$ and therefore $\tau = \lambda\kappa$, i.e. $\lambda = (\tau/\kappa)$ is the kernel lift $\mathrm{KernelLift}(\eta, \tau)$.

This fully describes the kernel of η. //

Proof of (iii). For the same morphism $\eta : F \to G \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_1$ we want to define its cokernel, i.e. the triple

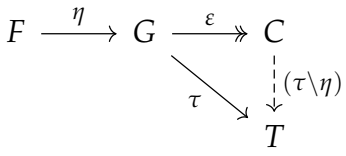
- $C := \mathrm{CokernelObject}(\eta)$
- $\varepsilon := \mathrm{CokernelProjection}(\eta) : G \twoheadrightarrow C$ such that $\eta\,\varepsilon = 0_{F,C}$.
- For any other $\tau : G \to T$ such that $\eta\,\tau = 0_{G,T}$ we have a unique morphism $(\varepsilon\backslash\tau) := \mathrm{CokernelColift}(\eta, \tau)$ such that $\tau = \varepsilon(\varepsilon\backslash\tau)$

We have for each morphism $a : c \to c' \in \mathcal{A}_1$ the morphisms $Fa : Fc \to Fc'$ and $Ga : Gc \to Gc'$. Then the cokernel projection $\varepsilon : G \twoheadrightarrow C$ of a natural transformation $\eta : F \to G$ has components $\varepsilon_c : Gc \to Cc$ with $\varepsilon_c = \mathrm{CokernelProjection}(\eta_c)$ and the Cokernel object C acts on objects as $\mathrm{CokernelObject}(\eta)c = \mathrm{CokernelObject}(\eta_c) = Cc$. And on morphisms $Ca := \mathrm{CokernelColift}(\eta_c, Ga\,\varepsilon_{c'})$.



Dually to the above, and by focusing on the naturality square on the right, we have the cokernel object C with $Cc = \mathrm{CokernelObject}(\eta_c)$ and $Ca = \mathrm{CokernelColift}(\eta_c, Ga\varepsilon_{c'})$.

Again we get the cokernel colift $(\varepsilon\backslash\tau) = \mathrm{CokernelColift}(\eta, \tau)$



component-wise, i.e. for each $c \in \mathcal{A}_0$ we get

$$(\varepsilon \backslash \tau)_c := (\varepsilon_c \backslash \tau_c).$$

This fully describes the cokernel of η. //

(4) Finally to show that $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ is an abelian category, we need to show that it is

(i) a pre-abelian category with

(ii) a dependent function $(-/-)$ mapping a pair $\iota : K \to A, \tau : T \to A$ to a lift τ/ι of τ along ι, where $A, K, T \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0$, ι is a monomorphism and $\tau \mathrm{CokernelProjection}(\iota) = 0$.

(iii) a dependent function $(-\backslash-)$ mapping a pair $\varepsilon : A \to C, \tau : A \to T$ to a colift $\epsilon\backslash\tau$ of τ along ε, where $A, K, T \in \mathcal{A}_0$, ε is an epimorphism and $\mathrm{KernelEmbedding}(\varepsilon)\, \tau = 0$.

Proof. Let $A, K, T \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0$ and $\iota, \tau \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_1$ such that

- $\iota : K \to A$ is a monomorphism,
- $\tau : T \to A$ is a morphism with $\tau\, \mathrm{CokernelProjection}(\iota) = 0$.

Then for each $c \in \mathcal{A}_0$ we have $\iota_c, \tau_c \in \Bbbk\text{-}\mathbf{mat}_1$ such that

- $\iota_c : Kc \to Ac$ is a monomorphism, and
- $\tau_c\, \mathrm{CokernelProjection}(\iota_c) = 0$.

Since $\Bbbk$-**mat** is an abelian category, we have a lift τ_c/ι_c of τ_c along ι_c, i.e. $(\tau_c/\iota_c)\iota_c = \tau_c$. This lift $(\tau_c/\iota_c) \in \Bbbk\text{-}\mathbf{mat}_1$ defines the components for $(\tau/\iota) \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_1$. And since ι is a monomorphism, the lift along ι is necessarily unique.
Thus we have a dependent function $(-/-)$ mapping a a pair $\iota : K \to A, \tau : T \to A$ to a lift (τ/ι) of τ along ι, where $A, K, T \in \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0$, ι is a monomorphism and $\tau\, \mathrm{CokernelProjection}(\iota) = 0$ with (τ/ι) defined as

$$(\tau/\iota)_c := (\tau_c/\iota_c). \tag{5.1.11}$$

Analogous for (iii) we get the colift along epimorphisms $(\varepsilon\backslash\tau)$ defined by its components

$$(\varepsilon\backslash\tau)_c := (\varepsilon_c\backslash\tau_c) \tag{5.1.12}$$

//

□

6 Yoneda's Lemma and projective objects

6.1 The category of presheaves

Definition 6.1.1. (The category of presheaves)[25]
For $\mathcal{C}$ a small category, its category of presheaves is the functor category

$$\mathrm{PSh}(\mathcal{C}) := \mathrm{Hom}(\mathcal{C}^{\mathrm{op}}, \mathbf{Set})$$

from the opposite category of $\mathcal{C}$ to **Set**. An object in this category is a presheaf. Taking $\mathcal{C}^{\mathrm{op}}$ instead of $\mathcal{C}$ (and with $(\mathcal{C}^{\mathrm{op}})^{\mathrm{op}} = \mathcal{C}$) we get the functor category as in 2.5.1

$$\mathrm{Hom}(\mathcal{C}, \mathbf{Set}) = \mathrm{PSh}(\mathcal{C}^{\mathrm{op}})$$

Remark 6.1.2. (General properties of presheaves)
The category of presheaves $\mathrm{PSh}(\mathcal{C})$ is called the free cocompletion of $\mathcal{C}$.

Definition 6.1.3. (Representable functor)[26]

(1) A functor from a locally small category $\mathcal{C}$ to **Set** is representable if there is an object $c \in \mathcal{C}$ and a natural isomorphism between F and $\mathrm{Hom}(c, -)$ (for a covariant F, otherwise $\mathrm{Hom}(-, c)$ for a contravariant F), in which case one says thet the functor F is represented by the object c.

(2) A representation for a functor F is a choice of object $c \in \mathcal{C}$ together with a specified natural isomorphism $\mathrm{Hom}(c, -) \cong F$ (for a covariant F, or $\mathrm{Hom}(-, c) \cong F$ for a contravariant F).

Lemma 6.1.4. (Yoneda's Lemma)[27] For any functor $F : \mathcal{C} \to \mathrm{Set}$, whose source $\mathcal{C}$ is locally small and any object $c \in \mathcal{C}_0$, there is a bijection

$$\mathrm{Hom}(\mathrm{Hom}_{\mathcal{C}}(c, -), F) \cong Fc$$

that associates a natural transformation $\mathrm{Hom}(\mathrm{Hom}_{\mathcal{C}}(c, -), F) \ni \alpha : \mathrm{Hom}_{\mathcal{C}}(c, -) \Rightarrow F$ to the element $\alpha_c(1_c) \in Fc$. Moreover, this correspondence is natural in both c and F.

As $\mathcal{C}$ is locally small but not necessarily small, a priori the collection of natural transformations $\mathrm{Hom}(\mathrm{Hom}_{\mathcal{C}}(c, -), F)$ might be large. However, the bijection in the Yoneda lemma proves that this particular collection of natural transformations indeed forms a set.

Proof. A proof of Yoneda's lemma can be found in many books on category theory, e.g. [1], Lemma 2.2.4, pages 57-59. □

Definition 6.1.5. (Yoneda embedding)[28]
The Yoneda embedding for a locally small category $\mathcal{C}$ is the functor

$$Y : \mathcal{C} \hookrightarrow \mathrm{Hom}(\mathcal{C}^{\mathrm{op}}, \mathrm{Set})$$

from $\mathcal{C}$ to the category of presheaves over $\mathcal{C}$ which is the image of the hom-functor

$$\mathrm{Hom} : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathrm{Set}$$

under the Hom adjunction

$$\mathrm{Hom}(\mathcal{C}^{\mathrm{op}} \times \mathcal{C}, \mathrm{Set}) \simeq \mathrm{Hom}(\mathcal{C}, \mathrm{Hom}(\mathcal{C}^{\mathrm{op}}, \mathrm{Set}))$$

in the closed symmetric monoidal category Cat. If instead we have the opposite category $\mathcal{C}^{\mathrm{op}}$, then we get the embedding into the functor category:

$$Y^{\mathrm{op}} : \mathcal{C}^{\mathrm{op}} \hookrightarrow \mathrm{Hom}(\mathcal{C}, \mathrm{Set})$$

Remark 6.1.6 (Our $\Bbbk$-linear version of Yoneda's embedding)**.** Let $\mathcal{A}$ be a $\Bbbk$-linear category with finite-dimensional $\Bbbk$-vector spaces as hom-sets. Then we get $\Bbbk$-linear versions of Yoneda's lemma and Yondeda's embedding:

For any $\Bbbk$-linear functor $F : \mathcal{A} \to \Bbbk\text{-}\mathbf{mat}$ and any object $i \in \mathcal{A}_0$, there is a bijection

$$\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(\mathrm{Hom}_{\mathcal{A}}(-,i),F) \cong F(i)$$

$$Y : \mathcal{A} \hookrightarrow \mathrm{Hom}_{\Bbbk}(\mathcal{A}^{\mathrm{op}},\Bbbk\text{-}\mathbf{mat})$$

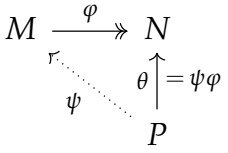
$$Y^{\mathrm{op}} : \mathcal{A}^{\mathrm{op}} \hookrightarrow \mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})$$

$\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})$ is Abelian and therefore finitely complete and finitely cocomplete.

6.2 Projective objects and the Yoneda projective

Lemma 6.2.1. Let $\mathcal{C}$ be a locally small category. For an object $P \in \mathcal{C}_0$ the following are equivalent:

- The covariant functor $\mathrm{Hom}(P,-)$ is exact.
- For all epimorphisms $\varphi : M \twoheadrightarrow N$ and morphisms $\theta : P \to N$, there exists a projective lift $\psi : P \dashrightarrow M$ such that $\theta = \psi\varphi$.

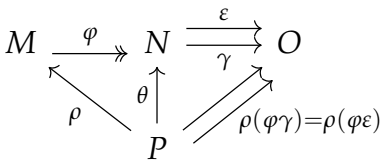


Proof. For an object $L \in \mathcal{C}_0$, $\mathrm{Hom}(L,-)$ is always a covariant left-exact functor, i.e. respects monos.

"$\Leftarrow$:" Prove that $\mathrm{Hom}(P,-)$ is right exact, i.e. respects epis.
For this, let $M, N \in \mathcal{C}_0$ and $\varphi : M \twoheadrightarrow N$ be an epi. The Hom-functor works on morphisms by mapping the Hom-sets of the source and target objects of the morphism, i.e. $\mathrm{Hom}(P,\varphi) : \mathrm{Hom}(P,M) \to \mathrm{Hom}(P,N)$, given by $\rho \mapsto \rho\varphi \; \forall \rho \in \mathrm{Hom}(P,M)$. Now given that φ is an epi, we want to show that $\mathrm{Hom}(P,\varphi)$ is also an epi.
Let $O \in \mathcal{C}_0$, $\gamma : N \to O$ and $\varepsilon : N \to O$ such that $\mathrm{Hom}(P,\gamma) : \mathrm{Hom}(P,N) \to \mathrm{Hom}(P,O)$; $\theta \mapsto \theta\gamma$ and $\mathrm{Hom}(P,\varepsilon) : \mathrm{Hom}(P,N) \to \mathrm{Hom}(P,O)$; $\theta \mapsto \theta\varepsilon$ and $\mathrm{Hom}(P,\varphi)\mathrm{Hom}(P,\gamma) = \mathrm{Hom}(P,\varphi)\mathrm{Hom}(P,\varepsilon)$. From the functoriality axioms (ref. definition 2.3.1 of a functor) it follows that $\mathrm{Hom}(P,\varphi\gamma) = \mathrm{Hom}(P,\varphi\varepsilon)$. This implies

$$\rho(\varphi\gamma) = \rho(\varphi\varepsilon) \; \forall \rho \in \mathrm{Hom}(P,M) \tag{6.2.1}$$

.



We want to show that the parallel morphisms $\mathrm{Hom}(P,\gamma)$ and $\mathrm{Hom}(P,\varepsilon)$ are the same, i.e. for all $\theta \in \mathrm{Hom}(P,N), \theta\gamma = \theta\varepsilon$. Our assumtion that there exists a projective lift helps us in this situation: $\forall \theta \in \mathrm{Hom}(P,N) \; \exists \rho \in \mathrm{Hom}(P,M)$

such that $\theta = \rho\varphi$ and therefore with the above equation (6.2.1), $\theta\gamma = (\rho\varphi)\gamma = \rho(\varphi\gamma) = \rho(\varphi\varepsilon) = (\rho\varphi)\varepsilon = \theta\varepsilon$ and therefore $\mathrm{Hom}(P,\gamma) = \mathrm{Hom}(P,\varepsilon)$, i.e. $\mathrm{Hom}(P,\varphi)$ is epi.

"$\Rightarrow$:" Let $\mathrm{Hom}(P,-)$ be right exact. Let $M, N \in \mathcal{C}_0$, the morphism $\varphi : M \twoheadrightarrow N$ be an epi and $\theta : P \to N$ any morphism. We want to show the existence of a morphism $\psi : P \dashrightarrow M$ such that $\theta = \psi\varphi$. With $\mathrm{Hom}(P,-)$ being exact, we have that $\mathrm{Hom}(P,\varphi) : \mathrm{Hom}(P,M) \twoheadrightarrow \mathrm{Hom}(P,N)$ is an epi, and is given by $\mathrm{Hom}(P,M) \ni \rho \mapsto \rho\varphi \in \mathrm{Hom}(P,N)$.
$\mathcal{C}$ is locally small, i.e. for the two objects $P, N \in \mathcal{C}_0$, there is a <u>set</u> $\mathrm{Hom}(P,N)$ of morphisms between them. The Hom-functor moves the morphisms from a general categorical context in $\mathcal{C}$ into the category of sets, i.e. $\mathrm{Hom}(P,\varphi)$ is a function in the category of sets. And for those it's true that every epimorphism is surjective. Thus $\forall\theta \in \mathrm{Hom}(P,N)\ \exists\rho \in \mathrm{Hom}(P,M)$ such that $\theta = (\mathrm{Hom}(P,\varphi))(\rho) = \rho\varphi$. This ρ is the projective lift $\psi := \rho$ we were looking for.

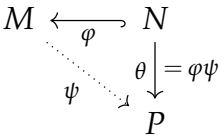
□

Definition 6.2.2. (Projective object)
An object P in a category $\mathcal{C}$ that satisfies one (and thus both) of the equivalent properties in Lemma 6.2.1 is called a <u>projective object</u>.

The dual statement to Lemma 6.2.1 is

Lemma 6.2.3. Let $\mathcal{C}$ be a category. For an object $P \in \mathcal{C}_0$ the following are equivalent:

- The contravariant functor $\mathrm{Hom}(-,P)$ is exact.
- For all monomorphisms $\varphi : M \hookleftarrow N$ and morphisms $\theta : P \leftarrow N$, there exists a projective colift $\psi : P \dashleftarrow M$ such that $\theta = \varphi\psi$.



Definition 6.2.4. (Injective object)
An object P in a category $\mathcal{C}$ that satisfies one (and thus both) of the equivalent properties in Lemma 6.2.3 is called an <u>injective object</u>.

Definition 6.2.5. (Yoneda projective) Let $\mathcal{A}$ be a $\Bbbk$-algebroid with finitely many objects and finite-dimensional hom-sets over $\Bbbk$. With the Yoneda embedding

$$Y^{\mathrm{op}} : \begin{cases} \mathcal{A}^{\mathrm{op}} \hookrightarrow \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat}) \\ i \mapsto \mathrm{Hom}_{\mathcal{A}}(-,i) \end{cases}$$

we see that the image $Y^{\mathrm{op}}(i)$ of the object $i \in \mathcal{A}^{\mathrm{op}}$ is the representable functor $\mathrm{Hom}_{\mathcal{A}}(-,i)$ in $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$. It is called the <u>$i$-th Yoneda projective</u>.

Analogous to Prop. 4.3.7 one can easily prove that a $\Bbbk$-representation F of $\mathcal{A}$ corresponds to a module $\bigoplus_{i\in\mathcal{A}_0} F(i)$ over the algebra $\mathbf{A} = \mathrm{Algebra}(\mathcal{A})$. In particular the Yoneda projective $Y^{\mathrm{op}}(i) = \mathrm{Hom}_{\mathcal{A}}(-,i)$ corresponds to the $\mathbf{A}$-module $M_i = \bigoplus_{j\in\mathcal{A}_0} \mathrm{Hom}_{\mathcal{A}}(j,i)$.

Lemma 6.2.6. Yoneda projectives are projective objects in $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$.

Proof.
We want to show that $\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(Y^{\mathrm{op}}(i), -)$ is right exact, i.e. finitely cocontinuous.
For this, let

$$0 \longrightarrow F_1 \xrightarrow{\eta} F_2 \xrightarrow{\rho} F_3 \longrightarrow 0$$

be a short exact sequence in $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$, i.e.

1. η is a monomorphism,

2. ρ is an epimorphism and

3. the image of η equals the kernel of ρ.

Then applying the functor $\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(Y^{\mathrm{op}}(i), -)$ on the sequence and simplifying with Yoneda's lemma

$$\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(\mathrm{Hom}_{\mathcal{A}}(-, i), F) \simeq F(i)$$
$$\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(\mathrm{Hom}_{\mathcal{A}}(-, i), \rho) \simeq \rho_i$$

We only have to measure exactness on the components

$$0 \longrightarrow F_1(i) \xrightarrow{\eta_i} F_2(i) \xrightarrow{\rho_i} F_3(i) \longrightarrow 0,$$

but this coincides with the definition of the exactness of functors. So the proof is a tautology by Yoneda's lemma. □

6.2.1 Abelian categories with enough projective objects (constructively)

Definition 6.2.7 (Enough projective objects)**.** A category $\mathcal{C}$ is said to have enough projective objects if every object admits an epimorphism from a projective object, i.e. for any object $A \in \mathcal{C}_0$ there exists an epimorphism $P \twoheadrightarrow A$, where P is projective.

We state without a proof the following fact about our functor category:

Theorem 6.2.8 ($\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ has enough projectives)**.**
Let $\mathcal{A}$ be a finite-dimensional algebroid over some field $\Bbbk$. The functor category $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ *has sufficiently many projectives.*

Doctrine 6.2.9 (Abelian category with enough projective objects)**.**
The doctrine `IsAbelianCategoryWithEnoughProjectives` therefore involves algorithms of
`IsAbelianCategory` together with algorithms for

- `EpimorphismFromSomeProjectiveObject`,

- `ProjectiveLift`,

6.2.2 Abelian categories with enough injective objects (constructively)

Dually to 6.2.7 we define

Definition 6.2.10 (Enough injective objects)**.** A category $\mathcal{C}$ is said to have enough injective objects if every object admits a monomorphism into an injective object, i.e. for any object $A \in \mathcal{C}_0$ there exists a monomorphism $A \hookrightarrow J$, where J is injective.

We state without a proof the following fact about our functor category:

Theorem 6.2.11 ($\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ has enough injectives)**.**
Let $\mathcal{A}$ be a finite-dimensional algebroid over some field $\Bbbk$. The functor category $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ has sufficiently many injectives.

Doctrine 6.2.12 (Abelian category with enough injective objects)**.**
The doctrine `IsAbelianCategoryWithEnoughInjectives` therefore involves algorithms of
`IsAbelianCategory` together with algorithms for

- `MonomorphismIntoSomeInjectiveObject`,
- `InjectiveColift`,

7 Direct sum decomposition (constructively)

We have shown in section 5 that for a family $\{F_i\}_{i\in I}$ of functors in $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$, there is the direct sum $F := \bigoplus_{i\in I} F_i$ with embeddings $\iota_i : F_i \to F$ (and projections $\pi_i : F \to F_i$). In this section we show constructively the other direction, i.e. that for each F there is a direct sum decomposition $\{F_i\}_{i\in I}$ and $\iota_i : F_i \to F$ such that

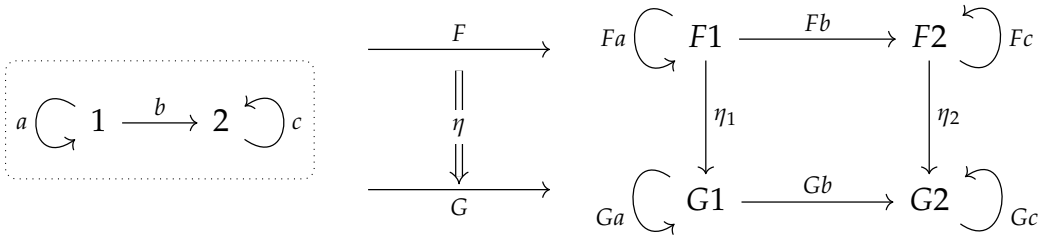
$$F = \bigoplus_{i\in I} F_i$$

7.1 Hom-structure of $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$

The following algorithms assume that we already have an algorithm to compute a $\Bbbk$-basis of the external hom $\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, F)$.
That algorithm `BasisOfExternalHom`(F, G) for two functors F and G works by solving systems of matrix equations, also known as Sylvester equations, which can be solved using the Kronecker product.

As already noticed by Sylvester, one can bring a two-sided homogeneous system of linear equations (of morphisms in $\Bbbk$-**mat**) into a classical one-sided homogeneous system of linear equations (of morphisms in $\Bbbk$-**mat**).

Example 7.1.1 (Sylvester equations for a natural transformation $\eta : F \Rightarrow G$ between two representations of C_3C_3)**.** The naturality constraints, that the diagram on the right



commutes for all three morphisms $a, b, c \in \mathcal{A}_1$, with $\mathcal{A}$ being (the $\Bbbk$-linear closure of) our finite concrete category C_3C_3, are the following:

$$Fa\,\eta_1 = \eta_1 Ga \tag{7.1.1a}$$
$$Fb\,\eta_2 = \eta_1 Gb \tag{7.1.1b}$$
$$Fc\,\eta_2 = \eta_2 Gc \tag{7.1.1c}$$

We bring the terms with the matrix variables η_j all on one side:

$$Fa\,\eta_1 - \eta_1 Ga = 0 \tag{7.1.2a}$$
$$Fb\,\eta_2 - \eta_1 Gb = 0 \tag{7.1.2b}$$
$$Fc\,\eta_2 - \eta_2 Gc = 0 \tag{7.1.2c}$$

This system of equations can be obtained from the generalized system of Sylvester equations[29]

$$\sum_{1\le j\le k} \Phi_{ij}\,\Theta_j\,\Psi_{ij} = P_i, \quad 1 \le i \le f \tag{7.1.3}$$

by setting $f = 3$, $k = 2$, and defining the constant matrices Φ, Ψ and the right-hand side P as follows:

For the left matrices Φ we have

$$\begin{aligned}\Phi_{1,1} &= \mathrm{Diag}(Fa, 0_{F2,F2}), & \Phi_{1,2} &= \mathrm{Diag}(-1_{F1}, 0_{F2,F2}),\\ \Phi_{2,1} &= \mathrm{Diag}(0_{F1,F1}, Fb), & \Phi_{2,2} &= \mathrm{Diag}(-1_{F1}, 0_{F2,F2}),\\ \Phi_{3,1} &= \mathrm{Diag}(0_{F1,F1}, Fc), & \Phi_{3,2} &= \mathrm{Diag}(0_{F1,F1}, -1_{F2}).\end{aligned}$$

For the right matrices Ψ we have

$$\begin{aligned}\Psi_{1,1} &= \mathrm{Diag}(1_{G1}, 0_{G2,G2}), & \Psi_{1,2} &= \mathrm{Diag}(Ga, 0_{F2,F2}),\\ \Psi_{2,1} &= \mathrm{Diag}(0_{G1,G1}, 1_{G2}), & \Psi_{2,2} &= \mathrm{Diag}(Gb, 0_{F2,F2}),\\ \Psi_{3,1} &= \mathrm{Diag}(0_{G1,G1}, 1_{G2}), & \Psi_{3,2} &= \mathrm{Diag}(0_{G1,G1}, Gc).\end{aligned}$$

For the right-hand side we have $P_1 = P_2 = P_3 = 0_{F1+F2,\, G1+G2}$.

Then the matrix variable Θ is

$$\Theta_1 = \Theta_2 = \mathrm{Diag}(\eta_1, \eta_2).$$

With these definitions, our system of matrix equations (7.1.1) is equivalent to the following system of block matrix equations:

$$\begin{aligned}
&\begin{pmatrix} Fa & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} \eta_1 & 0 \\ 0 & \eta_2 \end{pmatrix}\begin{pmatrix} 1_{G1} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} -1_{F1} & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} \eta_1 & 0 \\ 0 & \eta_2 \end{pmatrix}\begin{pmatrix} Ga & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}\\
&\begin{pmatrix} 0 & 0 \\ 0 & Fb \end{pmatrix}\begin{pmatrix} \eta_1 & 0 \\ 0 & \eta_2 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ 0 & 1_{G2} \end{pmatrix} + \begin{pmatrix} -1_{F1} & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} \eta_1 & 0 \\ 0 & \eta_2 \end{pmatrix}\begin{pmatrix} Gb & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}\\
&\begin{pmatrix} 0 & 0 \\ 0 & Fc \end{pmatrix}\begin{pmatrix} \eta_1 & 0 \\ 0 & \eta_2 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ 0 & 1_{G2} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & -1_{F2} \end{pmatrix}\begin{pmatrix} \eta_1 & 0 \\ 0 & \eta_2 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ 0 & Gc \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}
\end{aligned}$$

And can be solved with the method suitable for the general system of Sylvester equations (7.1.3). With the constants $F1, F2, G1, G2, Fa, Fb, Fc, Ga, Gb, Gc$ given as coefficients for the equations, we can thus solve the system and get a finite basis of the external hom, i.e. a basis for $\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F, G)$.

7.2 The algorithm `DecomposeOnceByRandomEndomorphism`

Algorithm 3: `DecomposeOnceByRandomEndomorphism`

Input : a functor F in a functor category
Output : a pair $[\iota : I \to F, \kappa : K \to F]$ of morphisms such that $I \oplus K = F$ with $I \neq 0$ and $K \neq 0$ or `fail` if it was unable to further decompose F;

$d := \max\{\dim_{\Bbbk} Fc\}_{c \in \mathcal{A}_0}$;
if $d = 0$ **then**
  return `fail`; `// the zero representation is indecomposable`
end
$\mathcal{B} = [\beta_1, \ldots, \beta_h]$ is a $\Bbbk$-basis of $\mathrm{Hom}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})}(F, F)$;
add $0_{F,F}$ to $\mathcal{B}$;
$n := \lfloor \log_2(d) \rfloor + 1$;
for $b \in [h+1, h, \ldots, 2]$ **do**
  $\alpha := \beta_b + \mathrm{random}(\Bbbk) \cdot \beta_{b-1}$; `// a heuristic ansatz for a random endomorphism`
  for $i \in [1, \ldots, n]$ **do**
    $\alpha_2 := \alpha^2$;
    `/* We do not expect the exponentiation to produce an idempotent, still this is a very cheap test: */`
    if $\alpha = \alpha_2$ **then**
      break;
    end
    $\alpha := \alpha_2$;
  end
  if $\alpha = 0$ **then**
    continue ; `// try another endomorphism`
  end
  $\kappa := \mathrm{KernelEmbedding}(\alpha)$;
  if $\kappa = 0$ **then**
    continue ; `// try another endomorphism`
  end
  $\iota := \mathrm{ImageEmbedding}(\alpha)$;
  return $[\iota, \kappa]$;
end
return `fail`; `// The input functor F is indecomposable with a high probability.`

To justify proposition 7.2.2 below we need the following lemma which is a linear analogue of the σ-lemma 4.2.1.

Lemma 7.2.1. If α is an endomorphism of a d-dimensional vector space, then $\alpha^e|_{\mathrm{Im}(\alpha^d)}$ is an automorphism for all $e \geq d$.

Proof. This follows from the inclusion $\mathrm{Im}(\alpha^2) \subseteq \mathrm{Im}(\alpha)$ and that the dimension of the vector space is equal to d. □

Proposition 7.2.2. Algorithm 3 terminates with the correct output.

Proof. For the input $F = 0$ we have $Fc = 0 \, \forall c \in \mathcal{A}$ and thus $d = 0$ which returns `fail`, i.e. there is no decomposition $0 = I \oplus K$ with $I \neq 0$ and $K \neq 0$.

For any other input $F \neq 0$ there is a $c \in \mathcal{A}_0$ with $Fc > 0$, thus in line 1 we have $d > 0$.
Since $F \neq 0$ the vector space $\mathrm{End}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F)$ is at least 1-dimensional, so our basis $\mathcal{B}$ has length $h \geq 1$ and doesn't contain $0_{F,F}$. Thus after line 6 we can assume $\mathrm{Length}(\mathcal{B}) = h + 1 \geq 2$ and the list $[h+1, h, \ldots, 2]$ in line 8 to be nonempty, thus we will enter the for loop at least once.

In the for-loop in lines 10-17, we are squaring α at most n times with

$$2^n = 2^{\lfloor \log_2(d) \rfloor + 1} \geq d.$$

Hence $\alpha_c^{2^n}$ is an automorphism on its image $\mathrm{Im}(\alpha_c^{2^n}) = \mathrm{Im}(\alpha_c^d)$ by the above lemma and with the definition of d. Let $I = \mathrm{Im}(\alpha^{2^n})$ and $K = \mathrm{Ker}(\alpha^{2^n})$ with image embedding $\iota : I \to F$ and kernel embedding $\kappa : K \to F$. Then

$$F = I \oplus K. \tag{7.2.1}$$

□

Remark 7.2.3. The set of endomorphisms which yield a nontrivial decomposition of a decomposable functor is Zariski-open and hence Zariski-dense in the vector space $\mathrm{End}_{\mathrm{Hom}_{\Bbbk}(\mathcal{A},\Bbbk\text{-}\mathbf{mat})}(F)$.

Computation 7.2.4. In this GAP session we will apply Algorithm 3 first on an indecomposable representation, which will `fail` as expected, and then on a decomposable one, resulting in the two embeddings `iota` and `kappa`. The former with indecomposable source, and the latter `kappa` with a source that can be further decomposed.

Example

```
gap> LoadPackage( "CatReps" );
true
gap> c3c3 := ConcreteCategoryForCAP( [ [2,3,1], [4,5,6], [,,,5,6,4] ] );
A finite concrete category
gap> GF3 := HomalgRingOfIntegers( 3 );
GF(3)
gap> kq := Algebroid( GF3, c3c3 );
Algebroid generated by the right quiver q(2)[a:1->1,b:1->2,c:2->2]
gap> SetIsLinearClosureOfACategory( kq, true );
gap> CatReps := Hom( kq, GF3 );
The category of functors: Algebroid generated by the right quiver
q(2)[a:1->1,b:1->2,c:2->2] -> Category of matrices over GF(3)
gap> d := [[1,1,0,0,0],[0,1,1,0,0],[0,0,1,0,0],[0,0,0,1,1],[0,0,0,0,1]];;
gap> e := [[0,1,0,0],[0,0,1,0],[0,0,0,0],[0,1,0,1],[0,0,1,0]];;
gap> f := [[1,1,0,0],[0,1,1,0],[0,0,1,0],[0,0,0,1]];;
gap> nine := AsObjectInHomCategory( kq, [ 5, 4 ], [ d, e, f ] );
<(1)->5, (2)->4; (a)->5x5, (b)->5x4, (c)->4x4>
gap> DecomposeOnceByRandomEndomorphism( nine );
fail
```

The above shows that our representation `nine` is indecomposable (with a high probability). We use the tensor product to generate another representation `fortyone`, that is hopefully decomposable, and inspect the two resulting embeddings `iota` and `kappa`.

Example

```
gap> fortyone := TensorProductOnObjects( nine, nine );
<(1)->25, (2)->16; (a)->25x25, (b)->25x16, (c)->16x16>
gap> result := DecomposeOnceByRandomEndomorphism( fortyone );
[ <(1)->3x25, (2)->1x16>, <(1)->22x25, (2)->15x16> ]
gap> iota := result[1];
<(1)->3x25, (2)->1x16>
gap> kappa := result[2];
<(1)->22x25, (2)->15x16>
gap> Display( fortyone );
An object in The category of functors: Algebroid generated by the
right quiver q(2)[a:1->1,b:1->2,c:2->2] -> Category of matrices
over GF(3) defined by the following data:


Image of <(1)>:
A vector space object over GF(3) of dimension 25

Image of <(2)>:
A vector space object over GF(3) of dimension 16

Image of (1)-[{ Z(3)^0*(a) }]->(1):
 1 1 . . . 1 1 . . . . . . . . . . . . . . . . . . . . . . . . .
 . 1 1 . . . 1 1 . . . . . . . . . . . . . . . . . . . . . . . .
 . . 1 . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . .
 . . . 1 1 . . . 1 1 . . . . . . . . . . . . . . . . . . . . . .
 . . . . 1 . . . . 1 . . . . . . . . . . . . . . . . . . . . . .
 . . . . . 1 1 . . . 1 1 . . . . . . . . . . . . . . . . . . . .
 . . . . . . 1 1 . . . 1 1 . . . . . . . . . . . . . . . . . . .
 . . . . . . . 1 . . . . 1 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . 1 1 . . . 1 1 . . . . . . . . . . . . . . . . .
 . . . . . . . . 1 . . . . 1 . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . 1 1 . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . 1 1 . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . 1 1 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . 1 1 . . . 1 1 . . . . . . . . . . .
 . . . . . . . . . . . . . . 1 1 . . . 1 1 . . . . . . . . . . .
 . . . . . . . . . . . . . . . 1 . . . . 1 . . . . . . . . . . .
 . . . . . . . . . . . . . . . . 1 1 . . . 1 1 . . . . . . . . .
 . . . . . . . . . . . . . . . . 1 1 . . . 1 1 . . . . . . . . .
 . . . . . . . . . . . . . . . . . 1 . . . . 1 . . . . . . . . .
 . . . . . . . . . . . . . . . . . . 1 1 . . . 1 1 . . . . . . .
 . . . . . . . . . . . . . . . . . . 1 1 . . . 1 1 . . . . . . .
 . . . . . . . . . . . . . . . . . . . 1 . . . . 1 . . . . . . .
 . . . . . . . . . . . . . . . . . . . . 1 1 . . . 1 1 . . . . .
 . . . . . . . . . . . . . . . . . . . . 1 . . . . 1 . . . . . .
 . . . . . . . . . . . . . . . . . . . . . 1 1 . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . 1 1 . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . 1 1 . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . .

A morphism in Category of matrices over GF(3)


Image of (1)-[{ Z(3)^0*(b) }]->(2):
 . . . . . 1 . . . . . . . . . .
 . . . . . . 1 . . . . . . . . .
 . . . . . . . . . . . . . . . .
 . . . . . 1 . 1 . . . . . . . .
 . . . . . . 1 . . . . . . . . .
```

```
 . . . . . 1 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . . . . . . . .
 . . . . . 1 . 1 . . . .
 . . . . . 1 . . . . . .
 . . . . . . . . . . . .
 . . . . . . . . . . . .
 . . . . . . . . . . . .
 . . . . . . . . . . . .
 . . . . . 1 . . . . . 1 . .
 . . . . . 1 . . . . . 1 .
 . . . . . . . . . . . .
 . . . . . 1 . 1 . . . . 1 . 1
 . . . . . 1 . . . . . 1 .
 . . . . . 1 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . . . . . . . .
 . . . . . 1 . 1 . . . .
 . . . . . 1 . . . . . .

A morphism in Category of matrices over GF(3)


Image of (2)-[{ Z(3)^0*(c) }]->(2):
 1 1 . . 1 1 . . . . . .
 . 1 1 . . 1 1 . . . . . .
 . . 1 . . . 1 . . . . . .
 . . . 1 . . . 1 . . . . . .
 . . . . 1 1 . . 1 1 . . . . .
 . . . . . 1 1 . . 1 1 . . . . .
 . . . . . 1 . . . 1 . . . . .
 . . . . . 1 . . . 1 . . . .
 . . . . . 1 1 . . . . . .
 . . . . . 1 1 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 1 . . . . .
 . . . . . 1 1 . .
 . . . . . 1 1 .
 . . . . . 1 .
 . . . . . 1

A morphism in Category of matrices over GF(3)
gap> S := DirectSum( [ Source( iota ), Source( kappa ) ] );
<(1)->25, (2)->16; (a)->25x25, (b)->25x16, (c)->16x16>
gap> Display( S );
An object in The category of functors: Algebroid generated by the
right quiver q(2)[a:1->1,b:1->2,c:2->2] -> Category of matrices
over GF(3) defined by the following data:


Image of <(1)>:
A vector space object over GF(3) of dimension 25

Image of <(2)>:
A vector space object over GF(3) of dimension 16
```

```
Image of (1)-[{ Z(3)^0*(a) }]->(1):
 . 2 . . . . . . . . . . . . . . . . . . . . . . .
 1 2 2 . . . . . . . . . . . . . . . . . . . . . .
 . . 1 . . . . . . . . . . . . . . . . . . . . . .
 . . . 1 1 . . . 1 1 . . . . . . . . . . . . . . .
 . . . . 1 1 . . . 1 1 . . . . . . . . . . . . . .
 . . . . . 1 . . . . 1 . . . . . . . . . . . . . .
 . . . . . . 1 1 . . . 1 1 . . . . . . . . . . . .
 . . . . . . . 1 . . . . 1 . . . . . . . . . . . .
 . . . . . . . . 1 1 . . . 1 1 . . . . . . . . . .
. . . . . . . . . 1 1 . . . 1 1 . . . . . . . . .
. . . . . . . . . . 1 . . . . 1 . . . . . . . . .
. . . . . . . . . . . 1 1 . . . 1 1 . . . . . . .
. . . . . . . . . . . . 1 . . . . 1 . . . . . . .
. . . . . . . . . . . . . 1 1 . . . . . . . . . .
. . . . . . . . . . . . . . 1 1 . . . . . . . . .
. . . . . . . . . . . . . . . 1 . . . . . . . . .
. . . . . . . . . . . . . . . . 1 1 . . . . . . .
. . . . . . . . . . . . . . . . . 1 . . . . . . .
. . . . . . . . . . . . . . . . . . 1 1 . 1 1 . .
. . . . . . . . . . . . . . . . . . . 1 1 . 1 1 .
. . . . . . . . . . . . . . . . . . . . 1 . . 1 .
. . . . . . . . . . . . . . . . . . . . . 1 1 . .
. . . . . . . . . . . . . . . . . . . . . . 1 1 .
. . . . . . . . . . . . . . . . . . . . . . . 1 .
. . . . . . . . . . . . . . . . . . . . . . . . 1

A morphism in Category of matrices over GF(3)


Image of (1)-[{ Z(3)^0*(b) }]->(2):
 2 . . . . . . . . . . . . . . .
 1 . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . .
 . . . . . . 1 . . . . . . . . .
 . . . . . . . 1 . . . . . . . .
 . . . . . . . . . . . . . . . .
 . . . . . . 1 . 1 . . . . . . .
 . . . . . . . 1 . . . . . . . .
 . . . . . . . . . . 1 . . . . .
 . . . . . . . . . . . 1 . . . .
 . . . . . . . . . . . . . . . .
 . . . . . . . . . . 1 . 1 . . .
 . . . . . . . . . . . 1 . . . .
 . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . .
 . . . . . . 1 . . . . . . . 1 .
 . . . . . . . 1 . . . . . . . 1
 . . . . . . . . . . . . . . . .
 . . . . . . . . . . 1 . . . . .
 . . . . . . . . . . . 1 . . . .
 . . . . . . . . . . . . . . . .
 . . . . . . . 2 . . 1 . 1 . . 2
```

```
A morphism in Category of matrices over GF(3)


Image of (2)-[{ Z(3)^0*(c) }]->(2):
 1 . . . . . . . . . . . . . . . .
 . 1 1 . . 1 1 . . . . . . . . . .
 . . 1 1 . . 1 1 . . . . . . . . .
 . . . 1 . . . 1 . . . . . . . . .
 . . . . 1 . . . 1 . . . . . . . .
 . . . . . 1 1 . . 1 1 . . . . . .
 . . . . . . 1 1 . . 1 1 . . . . .
 . . . . . . . 1 . . . 1 . . . . .
 . . . . . . . . 1 . . . 1 . . . .
 . . . . . . . . . 1 1 . . . . . .
 . . . . . . . . . . 1 1 . . . . .
 . . . . . . . . . . . 1 . . . . .
 . . . . . . . . . . . . 1 . . . .
 . . . . . . . . . . . . . 1 1 . .
 . . . . . . . . . . . . . . 1 1 .
 . . . . . . . . . . . . . . . 1 1
 . . . . . . . . . . . . . . . . 1

A morphism in Category of matrices over GF(3)
```

Comparing the matrices of `fortyone` with those of `S`, we see that after decomposing once, we have separated one small matrix on the diagonal: A 3×3-matrix from `S(kq.a)`, a 3×1-matrix from `S(kq.b)` and a 1×1-matrix from `S(kq.c)`. This matches with the source of the embedding `iota`.

Example

```
gap> Display( iota );
A morphism in The category of functors: Algebroid generated by the
right quiver q(2)[a:1->1,b:1->2,c:2->2] -> Category of matrices
over GF(3) defined by the following data:


Image of <(1)>:
 2 2 . 1 1 . . . . . . . . . . . 1 2 1 2 1 . . . . .
 1 2 1 2 1 1 2 1 2 1 . . . . . 2 . . 1 . 2 . . 1 .
 . . 2 . . . 1 2 . 2 2 . . 1 . . . 1 . . . 2 . . 1

A split monomorphism in Category of matrices over GF(3)


Image of <(2)>:
 . . . . . . . . . . . . . . . . 1

A split monomorphism in Category of matrices over GF(3)
gap> Source( iota );
<(1)->3, (2)->1; (a)->3x3, (b)->3x1, (c)->1x1>
gap> Display( Source( iota ) );
An object in The category of functors: Algebroid generated by the
right quiver q(2)[a:1->1,b:1->2,c:2->2] -> Category of matrices
over GF(3) defined by the following data:

Image of <(1)>:
A vector space object over GF(3) of dimension 3

Image of <(2)>:
```

```
A vector space object over GF(3) of dimension 1

Image of (1)-[{ Z(3)^0*(a) }]->(1):
 . 2 .
 1 2 2
 . . 1

A morphism in Category of matrices over GF(3)


Image of (1)-[{ Z(3)^0*(b) }]->(2):
 2
 1
 .

A morphism in Category of matrices over GF(3)


Image of (2)-[{ Z(3)^0*(c) }]->(2):
 1

A morphism in Category of matrices over GF(3)
```

We can then look at the other embedding of the direct sum decomposition, `kappa`. The iteration of `WeakDirectSumDecomposition` will continue then with `Source( kappa )`. Each time the random endomorphism might decompose the representation resulting in matrices of smaller and smaller dimensions.

Example

```
gap> Source( kappa );
<(1)->22, (2)->15; (a)->22x22, (b)->22x15, (c)->15x15>
gap> Display( Source( kappa ) );
An object in The category of functors: Algebroid generated by the
right quiver q(2)[a:1->1,b:1->2,c:2->2] -> Category of matrices
over GF(3) defined by the following data:

Image of <(1)>:
A vector space object over GF(3) of dimension 22

Image of <(2)>:
A vector space object over GF(3) of dimension 15

Image of (1)-[{ Z(3)^0*(a) }]->(1):
 1 1 . . . 1 1 . . . . . . . . . . . . . . .
 . 1 1 . . . 1 1 . . . . . . . . . . . . . .
 . . 1 . . . . 1 . . . . . . . . . . . . . .
 . . . 1 1 . . . 1 1 . . . . . . . . . . . .
 . . . . 1 . . . . 1 . . . . . . . . . . . .
 . . . . . 1 1 . . . 1 1 . . . . . . . . . .
 . . . . . 1 1 . . . 1 1 . . . . . . . . . .
 . . . . . . . 1 . . . . 1 . . . . . . . . .
 . . . . . . . . 1 1 . . . 1 1 . . . . . . .
 . . . . . . . . 1 . . . . 1 . . . . . . . .
 . . . . . . . . . . 1 1 . . . . . . . . . .
 . . . . . . . . . . 1 1 . . . . . . . . . .
 . . . . . . . . . . . 1 . . . . . . . . . .
 . . . . . . . . . . . . 1 1 . . . . . . . .
 . . . . . . . . . . . . . 1 . . . . . . . .
```

```
. . . . . . . . . . . . . . . . . . 1 1 . 1 1 . .
. . . . . . . . . . . . . . . . . . . 1 1 . 1 1 .
. . . . . . . . . . . . . . . . . . . . 1 . . 1 .
. . . . . . . . . . . . . . . . . . . . . 1 1 . .
. . . . . . . . . . . . . . . . . . . . . 1 1 .
. . . . . . . . . . . . . . . . . . . . . . 1 .
. . . . . . . . . . . . . . . . . . . . . . . 1

A morphism in Category of matrices over GF(3)


Image of (1)-[{ Z(3)^0*(b) }]->(2):
. . . . . 1 . . . . . .
. . . . . 1 . . . . . .
. . . . . .
. . . . . 1 . 1 . . . . .
. . . . . 1 . . . . . .
. . . . . 1 . . . . .
. . . . . 1 . . . .
. . . . . .
. . . . . 1 . 1 . . .
. . . . . 1 . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . 1 . . . . . 1 .
. . . . . 1 . . . . . 1
. . . . . .
. . . . . 1 . . . . .
. . . . . 1 . . . .
. . . . . .
. . . . . 2 . . 1 . 1 . . 2

A morphism in Category of matrices over GF(3)


Image of (2)-[{ Z(3)^0*(c) }]->(2):
 1 1 . . 1 1 . . . . . .
 . 1 1 . . 1 1 . . . . . .
 . . 1 . . . 1 . . . . . .
 . . . 1 . . . 1 . . . . . .
 . . . . 1 1 . . 1 1 . . . . .
 . . . . . 1 1 . . 1 1 . . . .
 . . . . . 1 . . . 1 . . . .
 . . . . . 1 . . . 1 . . .
 . . . . . 1 1 . . . . .
 . . . . . 1 1 . . . .
 . . . . . 1 . . . .
 . . . . . 1 . . .
 . . . . . 1 1 .
 . . . . . 1 1
 . . . . . 1

A morphism in Category of matrices over GF(3)
gap> result2 := DecomposeOnceByRandomEndomorphism( Source( kappa ) );
```

```
[ <(1)->3x22, (2)->3x15>, <(1)->19x22, (2)->12x15> ]
```

7.3 The algorithm `WeakDirectSumDecomposition`

Algorithm 4: `WeakDirectSumDecomposition`

Input : a functor F in a functor category
Output : a list $[\eta_i : F_i \to F]$ of embeddings such that $\oplus_i F_i = F$ and each F_i is indecomposable (with a high probability).

```
1  queue := [1_F];
2  summands := ∅;
3  while queue ≠ ∅ do
4  |  let η be the last element in queue and delete η from queue;
5  |  result := DecomposeOnceByRandomEndomorphism(s(η));
6  |  if result = fail then      // s(η) was indecomposable (with a high
   |   probability)
7  |  |  add η to summands;
8  |  else
9  |  |  [ι, κ] = result;
10 |  |  append [ιη, κη] to queue;
11 |  end
12 end
13 return summands;
```

Proposition 7.3.1. Algorithm 4 terminates with the correct output.

Proof. We assert certain truths about the algorithm by formulating invariants, and how they stay constant in each line of the algorithm.

Proof that the output is correct.
In line 1, the morphism $1_F : F \to F$, which is initially the only morphism in `queue`, satisfies $t(1_F) = F$.
In line 10, since $\iota : I \to s(\eta)$ and $\kappa : K \to s(\eta)$ are each composable with η, then we are appending the list $[\iota\eta, \kappa\eta]$ of morphisms with target $t(\iota\eta) = t(\kappa\eta) = F$ to the `queue`.
Thus in each step of the algorithm the `queue` only contains morphisms η with target $t(\eta) = F$.

In line 2, the list `summands` is initially empty.
In line 7, we add a morphism η from the `queue` to the list `summands` only if in line 6 we checked that $s(\eta)$ is indecomposable.
Thus in each step of the algorithm the list `summands` only contains indecomposable morphisms with target F.

Initially with `queue` $= [1_F]$ and `summands` $= \emptyset$ we have

$$F = \bigoplus_{\eta \in \texttt{queue}} s(\eta) \oplus \bigoplus_{\eta \in \texttt{summands}} s(\eta) \tag{7.3.1}$$

For the first run of the while loop we take $\eta_1 := 1_F$ from the `queue`. Then there are two possibilities:
If F was indecomposable, we now add η_1 to `summands`, and have `queue` $= \emptyset$ and `summands` $= [1_F]$ which also satisfies (7.3.1).

Otherwise we get a decomposition of F with $\iota_1 : I_1 \to F$ and $\kappa_1 : K_1 \to F$. In this case `summands` stays empty, and instead we have `queue` $= [\iota_1\eta_1, \kappa_1\eta_1]$. For (7.3.1) to hold, we need to prove that

$$I_1 \oplus K_1 = F$$

which is exactly what we proved above for the output of

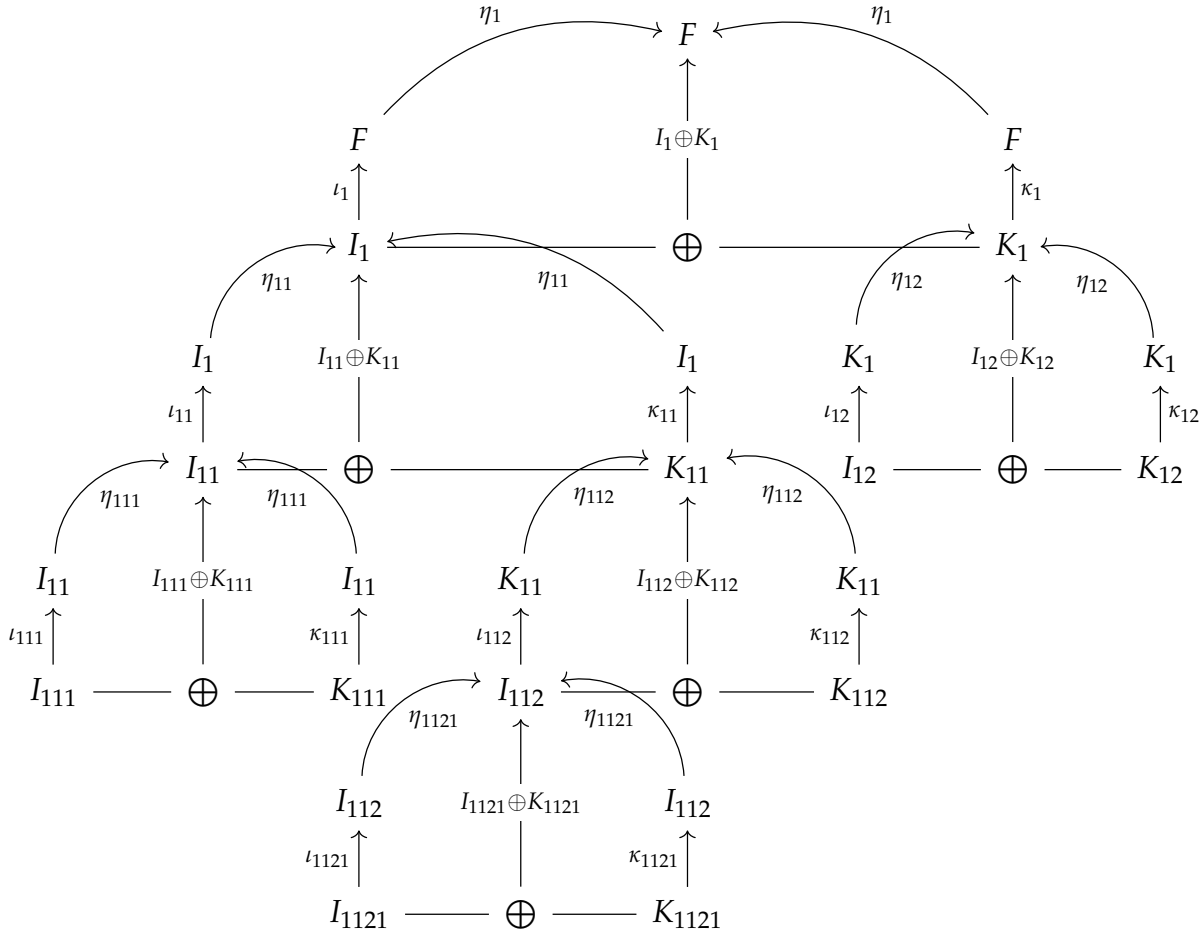
`DecomposeOnceByRandomEndomorphism`.

Thus after the first while loop, equation (7.3.1) holds.

In each run of the while loop, we are replacing I_j with $I_{j,1}$ and $K_{j,1}$ with $I_j = I_{j,1} \oplus K_{j,1}$ and K_j with $I_{j,2}$ and $K_{j,2}$ with $K_j = I_{j,2} \oplus K_{j,2}$, if possible. That is we have

$$\begin{aligned} F &= \quad I_1 \quad \oplus \quad K_1 \\ &= (I_{11} \oplus K_{11}) \oplus (I_{12} \oplus K_{12}) \\ &= \dots \end{aligned}$$

This decomposition tree can end earlier for some K_j and $I_{j'}$, which are already decomposed into indecomposables after two steps, while for other K_i and $I_{i'}$ the decomposition can go on for three or more steps, as illustrated in the following figure:



Assuming that the bottom-most nodes in the tree represent indecomposable functors (for which the algorithm stopped in line 6), in this example we have a decomposition of F as the direct sum

$$\begin{aligned} F &= I_1 \oplus K_1 \\ &= I_{11} \oplus K_{11} \oplus I_{12} \oplus K_{12} \\ &= I_{111} \oplus K_{111} \oplus I_{112} \oplus K_{112} \oplus I_{12} \oplus K_{12} \\ &= I_{111} \oplus K_{111} \oplus I_{1121} \oplus K_{1121} \oplus K_{112} \oplus I_{12} \oplus K_{12} \end{aligned}$$

and with $I_{111}, K_{111}, I_{1121}, K_{1121}, K_{112}, I_{12}, K_{12}$ indecomposable, we are finished. Tracing the morphisms from the bottom to the top gives us the list of embeddings, i.e. morphisms with source an indecomposable representation and target F, which are thus added to `summands`.

Proof that the algorithm terminates, i.e. that `queue` *will be empty.*
In each run of the while loop, a morphism η from the `queue` gets either deleted and not replaced, since $s(\eta)$ was indecomposable, or it gets deleted and replaced by two morphisms $[\iota\eta, \kappa\eta]$.
If at some point, $s(\eta)$ is indecomposable for every $\eta \in$ `queue`, then the η will be deleted from the `queue` (and added to `summands`) until the `queue` is empty. Then the while loop will end.

The case that each η gets replaced by $[\iota\eta, \kappa\eta]$ where again the $s(\iota\eta)$ and $s(\kappa\eta)$ are decomposable must come to an end after a finite number of steps:

Whenever $G := s(\eta)$ is decomposable with $I \oplus K = G$ and $I, K \neq 0$, the variable $d_G := \max_{c \in \mathcal{A}_0} Gc$ also gets decomposed into $d_I := \max_{c \in \mathcal{A}_0} Ic$ and $d_K := \max_{c \in \mathcal{A}_0} Kc$ with

$$\begin{aligned} 0 < d_I < d_G \\ 0 < d_K < d_G \end{aligned}$$

and both have the lower bound of 0. Thus after a finite number of steps there are only η with indecomposable $s(\eta)$ in the `queue` so that by the above result, `queue` will eventually become empty and the loop will end. //

With the `queue` $= \emptyset$ and `summands` containing only η with indecomposable $s(\eta)$, equation (7.3.1) becomes

$$F = \bigoplus_{\eta \in \texttt{summands}} s(\eta) \tag{7.3.2}$$

//

This proves that the algorithm terminates with the correct output. □

Computation 7.3.2. Continuing with the computation 7.2.4, we want to see the result of our direct sum decomposition of `fortyone`:

Example

```
gap> etas := WeakDirectSumDecomposition( fortyone : random := false );;
gap> iso := UniversalMorphismFromDirectSum( etas );
<(1)->25x25, (2)->16x16>
gap> Display( Source( iso ) );
An object in The category of functors: Algebroid generated by the
right quiver q(2)[a:1->1,b:1->2,c:2->2] -> Category of matrices
over GF(3) defined by the following data:
```

```
Image of <(1)>:
A vector space object over GF(3) of dimension 25

Image of <(2)>:
A vector space object over GF(3) of dimension 16

Image of (1)-[{ Z(3)^0*(a) }]->(1):
 1 1 . . . . . .
 . 1 . . . . . .
 1 1 1 . . . . . .
 . . . 1 2 . . . . . .
 . . . . 1 . . . . . .
 . . . 1 1 1 . . . . . .
 . . . . . 1 2 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 1 1 1 . . . . . .
 . . . . . 1 2 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 1 1 1 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 1 2 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 2 . 1 . . . . . .
 . . . . . 2 2 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 2 . 1 . . . . . .
 . . . . . 1 2 . . . .
 . . . . . 1 . . . .
 . . . . . 2 . 1 . . .
 . . . . . 2 .
 . . . . . 1 2 2
 . . . . . 1

A morphism in Category of matrices over GF(3)


Image of (1)-[{ Z(3)^0*(b) }]->(2):
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . 2 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 1 2 . . . . . .
 . . . . . 2 . . . .
 . . . . . 2 . . . .
 . . . . . 1 . . . . .
 . . . . . 1 .
```

```
  . . . . . .
  . . . . . 2 . .
  . . . . . . 2
  . . . . . . 1
  . . . . . .

A morphism in Category of matrices over GF(3)


Image of (2)-[{ Z(3)^0*(c) }]->(2):
 1 . . . . . .
 . 1 1 . . . . . .
 1 . 1 . . . . . .
 . . . 1 1 . . . . . .
 . . . . 1 1 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 1 . . . . . .
 . . . . . 1 1 . . . . . .
 . . . . . 2 . 1 . . . . . .
 . . . . . 1 1 . . . . .
 . . . . . 1 1 . . . .
 . . . . . 1 . . . .
 . . . . . 1 1 . .
 . . . . . 1 1 .
 . . . . . 1 .
 . . . . . 1

A morphism in Category of matrices over GF(3)
```

You can clearly see the diagonal block structure of the matrices in comparison with `fortyone`.

8 Hom-based invariants

We want to distinguish between different objects F, G in our functor category $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$ up to isomorphism. A good method for this is by comparing the output of binary categorical operations

$$\varphi(-,-) : \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0 \times \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0 \to \mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})_0$$

up to isomorphism, with the first input being the same functor, e.g. the tensor unit in $\mathrm{Hom}_{\Bbbk}(\mathcal{A}, \Bbbk\text{-}\mathbf{mat})$

$$\texttt{unit} : \mathcal{A}_0 \to \{1\}; \quad \mathcal{A}_1 \to \{(1)^{1\times 1}\}.$$

The idea is that the outputs of φ are easier to compare up to isomorphism than F and G directly. If $\varphi(\texttt{unit}, F)$ is not isomorphic to $\varphi(\texttt{unit}, G)$, then F and G are not isomorphic.

We are defining such a morphism φ in three steps.

8.1 The algorithm `MorphismOntoSumOfImagesOfAllMorphisms`

In an Abelian category the hom-set $\texttt{hom} := \texttt{BasisOfExternalHom}(M, N)$ of morphisms from M to N is a family of morphisms with same source M and same target N.

$$\begin{aligned} M &\xrightarrow{b_1} N \\ M &\xrightarrow{b_2} N \\ M &\xrightarrow{\cdots} N \end{aligned}$$

If we take this basis `hom` as an index set, we can make a direct sum out of that many copies of M,

$$S := \bigoplus_{i \in \texttt{hom}} M.$$

Then for these morphisms `hom` we can calculate their universal morphism out of the direct sum as in 3.4.10, i.e.

$$\texttt{UniversalMorphismFromDirectSum}(\texttt{hom}) := u_{\text{out}}(\texttt{hom}) : S \to N.$$

In the case where $\texttt{hom} = \emptyset$, i.e. there are no (non-zero) morphisms from M to N, we still get a universal morphism with target N, namely the unique morphism from the zero object

$$\texttt{UniversalMorphismFromZeroObject}(N) := 0_{0,N} : 0 \to N$$

as in 3.4.12.

In both cases we get one morphism that encodes all the information of the whole family of morphisms, especially their images.

$$\bigoplus_{i \in \texttt{hom}} M \xrightarrow[u_{\text{out}}(\texttt{hom})]{} N$$

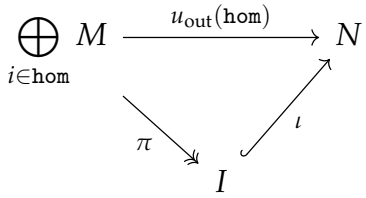
Algorithm 5: `MorphismOntoSumOfImagesOfAllMorphisms`

Input : two objects M and N in an abelian category
Output : a morphism with source $\oplus_{i \in \mathtt{hom}} M$ and target N

```
1 hom := BasisOfExternalHom(M, N);
2 if hom = ∅ then        // the direct sum of the empty set is the zero
    object
3 |  return UniversalMorphismFromZeroObject(N);
4 end
5 return UniversalMorphismFromDirectSum(hom);
```

8.2 The algorithm `EmbeddingOfSumOfImagesOfAllMorphisms`

Since we are in the setting of an Abelian category, the morphism $u_{\text{out}}(\mathtt{hom})$ from above can be factored with the Corollary 3.4.25 into an epimorphism π and a monomorphism ι.



The monomorphism ι is the image embedding of $u_{\text{out}}(\mathtt{hom})$. What can be said about I, M and N if the morphism ι is also an epimorphism?

Algorithm 6: `EmbeddingOfSumOfImagesOfAllMorphisms`

Input : two objects M and N in an abelian category
Output : an embedding morphism with source I and target N

```
1 return ImageEmbedding(MorphismOntoSumOfImagesOfAllMorphisms(M, N));
```

8.3 The algorithm `SumOfImagesOfAllMorphisms`

As a last step, we are now looking at the source I of the above embedding $\iota : I \hookrightarrow N$. The image embedding has all the information from $u_{\text{out}}(\mathtt{hom})$, but its source I is considerably "smaller" than the direct sum $\bigoplus_{i \in I} M$ that we started with.

Algorithm 7: `SumOfImagesOfAllMorphisms`

Input : two objects M and N in an abelian category
Output : the image I over which the direct sum factors

```
1 return Source(EmbeddingOfSumOfImagesOfAllMorphisms(M, N));
```

Since the objects and morphisms we calculated,

- $\mathtt{hom} := \mathtt{BasisOfExternalHom}(\mathtt{unit}, F)$,
- $u_{\text{out}}(\mathtt{hom}) := \mathtt{UniversalMorphismFromDirectSum}(\mathtt{hom})$,
- $\iota := \mathtt{ImageEmbedding}(u_{\text{out}}(\mathtt{hom}))$ and
- the image object $I := \mathtt{Source}(\iota)$

are all unique up to unique isomorphism, we get as output an object that is unique up to unique isomorphism, and thus a good candidate to distinguish between two categorically different objects F and G as we wanted.

Computation 8.3.1. Continuing with the computation from 7.3.2, we are now taking a closer look at the embeddings resulting from our direct sum decomposition.

Example

```
gap> etas := WeakDirectSumDecomposition( fortyone : random := false );;
gap> dec := List( etas, eta -> List( SetOfObjects( A ),
>       o -> Dimension( Source(
>         UnderlyingCapTwoCategoryCell( eta )( o ) ) ) ) );
[ [ 3, 0 ], [ 3, 0 ], [ 3, 0 ], [ 3, 0 ], [ 0, 3 ],
  [ 1, 3 ], [ 3, 3 ], [ 3, 3 ], [ 3, 3 ], [ 3, 1 ] ]
```

The list `etas` we get as result of the direct sum decomposition is a list of ten embeddings $\eta_i : F_i \to F$. The sources F_i of the `etas` have different images on the objects (1) and (2) of the algebroid $\mathcal{A}$, which can be seen by their dimensions in the list `dec`, ranging between 0 and 3. There are three different embeddings `etas`$[7]$, `etas`$[8]$ and `etas`$[9]$ in the list with F_7, F_8 and F_9 mapping to the full three dimensions $[3,3]$ at both objects. The last one we call `six`:

Example

```
gap> eta := etas[9];
<(1)->3x25, (2)->3x16>
gap> six := Source( eta );
<(1)->3, (2)->3; (a)->3x3, (b)->3x3, (c)->3x3>
```

We get another representation with dimensions $[3,3]$ at both objects by the first Yoneda projective `proj1`:

Example

```
gap> Aop := AlgebroidOverOppositeAlgebra( A );
Algebroid generated by the right quiver q_op(2)[a:1->1,b:2->1,c:2->2]
gap> Yop := YonedaEmbedding( Aop );
Yoneda embedding functor
gap> proj1 := Yop( Aop.1 );
<(1)->3, (2)->3; (a)->3x3, (b)->3x3, (c)->3x3>
```

We don't know if `proj1` is the source of one of the three embeddings

$$\texttt{etas}[7], \texttt{etas}[8], \texttt{etas}[9]$$

above, so we are looking for ways to compare it for example to the representation `six`.

In the following we give two proofs that `six` and `proj1` are nonisomorphic:

For the first proof as described above we compare the outputs of our binary categorical operator `SumOfImagesOfAllMorphisms` with the same first input being the tensor unit:

Example

```
gap> unit := TensorUnit( CatReps );
<(1)->1, (2)->1; (a)->1x1, (b)->1x1, (c)->1x1>
gap> emb := EmbeddingOfSumOfImagesOfAllMorphisms( unit, six );
<(1)->1x3, (2)->0x3>
gap> s1 := Source( emb );
<(1)->1, (2)->0; (a)->1x1, (b)->1x0, (c)->0x0>
gap> e1 := EmbeddingOfSumOfImagesOfAllMorphisms( unit, proj1 );
<(1)->1x3, (2)->1x3>
gap> Source( e1 );
<(1)->1, (2)->1; (a)->1x1, (b)->1x1, (c)->1x1>
```

Comparing how they map object 2 in $\mathcal{A}$ to different dimensions, we can see that `six` and `proj1` are nonisomorphic:

$$\begin{aligned}\texttt{SumOfImagesOfAllMorphisms}(\texttt{unit},\texttt{six})(2) &= 0\\ \texttt{SumOfImagesOfAllMorphisms}(\texttt{unit},\texttt{proj1})(2) &= 1\end{aligned}$$

As a second proof, we are comparing them directly using only the embedding from Algorithm 6:

Example

```
gap> IsEpimorphism( EmbeddingOfSumOfImagesOfAllMorphisms( proj1, six ) );
false
```

If `proj` and `six` were isomorphic, then the monomorphism ι would also be an epimorphism.

Annotations

1 For a definition of the quiver category in the language of category theory, see [7].

2 Ref. [5] 4.1.

3 The relation "f, then g" can be expressed by the left-to-right application of first f, and then g, if you write x^f for $f(x)$. Then the contravariantly ordered $g(f(x))$ becomes $(x^f)^g$.

4 Def 1.3.5. in [1], p. 17 (35/258).

5 The following five definitions are from [1], p. 30-31 (48/258).

6 Theorem 1.5.9. in [1].

7 Cited from [19].

8 Cited from [6].

9 Cited from [6].

10 Cited from [6], after pullback square and pushout square respectively.

11 Def 4.5.9. in [1], p. 139 (157/258).

12 The two algorithms `Source` and `Range` are implicit since each morphism in CAP has to be defined with source and target.

13 This result is already implemented in CAP as a derivation of `AdditionForMorphisms` from the four morphisms `UniversalMorphismIntoDirectSum`, `IdentityMorphism`, `UniversalMorphismFromDirectSum` and `PreCompose`. See https://github.com/homalg-project/CAP_project/blob/v2019.06.06/CAP/gap/DerivedMethods.gi#L1024

14 Especially in a skeletal category such as $\mathbb{k}$-**mat**.

15 It is challenging to decide between different types of zero matrices with zero rows or zero columns, if they are represented by lists of lists with their entries. How many rows does the (3×0) matrix have, if you represent it by the empty list `[ ]`? That's why the implementation in `homalg_project` uses a special function `HomalgZeroMatrix`, and why the morphisms in CAP are always implemented with their source and target defined.

16 The result "row-rank = column-rank" whose importance a first-year student might not understand right away, is a very nice property of matrices.

17 Ref. [1] example 4.1.13.

18 As my father rightly remarked when I showed this to him, "You can hide the whole world inside there!"

19 My thanks to Felix Potthast and Michael Figelius who both sent me in the direction of Landau's function

20 The difference between the two quivers (1) and (2) is that the first one has some sense of order. Object Y comes *after* X in the sense that you have an arrow from X to Y, but not the other way around. If a and c were the identity morphisms, then (1) would be an example for a poset or partially ordered set, i.e. a category such that for any pair of objects x, y there is at most one morphism from x to y, and if there is a morphism from x to y and a morphism from y to x, then $x = y$, i.e. the objects are equal. An exercise for the reader could be to formulate the exact relations between posets and cyclic quivers.

21 Refer to `AsFpCategory`.

22 Using Groebner Bases some relations can be identified in order to present the finitely presented category fp$\mathcal{C}$ in a shorter form. This is done in `Category`.

23 From [5] 4.1

24 From [5] 4.1

25 Cited from [17].

26 Cited from [20].

27 Statement of Yoneda's lemma from [1], Lemma 2.2.4.

28 Cited from ncatlab [18].

29 This is equation (A.17) in [21].

References

[1] Emily Riehl, *Category Theory in Context*, (http://www.math.jhu.edu/~eriehl/context.pdf)

[2] Sebastian Posur, *Constructive Category Theory and Applications to Equivariant Sheaves*, Siegen 2017

[3] Mohamed Barakat, *Algorithmische Algebra*, Siegen, WiSe 2018 / 2019, (https://algebra.mathematik.uni-siegen.de/barakat/Lehre/WS18/AA/Skript/AA.pdf)

[4] Alberto Facchini, *Pretorsion theories in general categories*, (https://web.northeastern.edu/martsinkovsky/p/Parnu2019/slides-facchini.pdf)

[5] Claus Michael Ringel, 2012, *The path algebra of a quiver*, (https://www.math.uni-bielefeld.de/~sek/kau/leit4.pdf)

[6] Qiaochu Yuan, *Annoying Precision*, (https://qchu.wordpress.com/2012/09/29/monomorphisms-and-epimorphisms/

[7] Jan Geuenich, (https://hss.ulb.uni-bonn.de/2017/4681/4681.pdf)

[8] Mohamed Barakat, Julia Mickisch and Fabian Zickgraf, `FinSetsForCAP`, (https://homalg-project.github.io/pkg/FinSetsForCAP)

[9] Mohamed Barakat, `FunctorCategories`, (https://homalg-project.github.io/pkg/FunctorCategories)

[10] Sebastian Posur, *Abelian Categories*, (https://homalg-project.github.io/nb/AbelianCategories)

[11] Gutsche, Sebastian and Skartsæterhagen, Øystein and Posur, Sebastian and Barakat, Mohamed, *The* `CAP` *project – Categories, Algorithms, Programming*, (http://homalg-project.github.io/CAP_project)

[12] Barakat, Mohamed and Grün, Tibor, *Representations and cohomology of finite categories*, (https://homalg-project.github.io/pkg/CatReps)

[13] Barakat, Mohamed and Grün, Tibor, *The category of GF(3)-representations of the finite concrete category encoding a homomorphism between two copies of the cyclic group of order* 3, (https://homalg-project.github.io/nb/C3C3)

[14] Øystein Skartsæterhagen, Øyvind Solberg, `QPA2`, (https://homalg-project.github.io/pkg/QPA2)

[15] Peter Webb, Dan Christensen, Fan Zhang, and Moriah Elkin, *catrepstutorialMarch11*, (http://www-users.math.umn.edu/~webb/GAPfiles/catrepstutorial.html)

[16] Peter Webb *An introduction to the representations and cohomology of categories*, (https://www-users.math.umn.edu/~webb/Publications/CategoryAlgebras.pdf)

[17] The nLab wiki An "open lab-book" wiki devoted to Mathematics, Physics and Philosophy, *category of presheaves*, (https://ncatlab.org/nlab/show/category+of+presheaves)

[18] The nLab wiki, *Yoneda embedding*, (https://ncatlab.org/nlab/show/Yoneda+embedding)

[19] The nLab wiki, *functor category*, (https://ncatlab.org/nlab/show/functor+category)

[20] The nLab wiki, *representable functor*, (https://ncatlab.org/nlab/show/representable+functor)

[21] Yu Feng and Mohamed Yagoubi, *Robust Control of Linear Descriptor Systems*, Appendix A *Generalized Sylvester Equation*, (https://link.springer.com/content/pdf/bbm:978-981-10-3677-4/1.pdf)

[22] Henri Bourlès, Categories and Functors, in Fundamentals of Advanced Mathematics, 2017, cited from sciencedirect, (https://www.sciencedirect.com/topics/mathematics/projectives)

A Implementation in Cap

The code listings below display the actual code lines from the gap packages. For this the file `Pfad.tex.sample` is included in the git repository of this thesis. The file should be edited to point towards the GAP directory where the packages are installed, and then renamed to `Pfad.tex`. We use the following versions of the GAP packages (which is important, since change in the codebase changes the first and last line read in):

Package name	Version	Sha
CatReps	2020.10 – 07	ac2124f392067d30
Algebroids	2020.10 – 12	4b47253d82f557c6
FunctorCategories	2020.10 – 04	2a57bc4cf1c81784
CategoryConstructor	2020.10 – 02	c0352cf43691044c

Listings

Procedure 1: ConvertToMapOfFinSets

```
InstallGlobalFunction( ConvertToMapOfFinSets,

  function( objects, gen )
    local O, T, fl, S, G, i;

    T := First( objects, O -> Length(
        Intersection( gen, AsList( O ) ) ) > 0 );

    if T = fail then
        Error( "unable to find target set\n" );
    fi;

    fl := Flat( List( objects, O -> AsList( O ) ));
    S := fl{PositionsProperty( fl , i -> IsBound( gen[i] ))};

    S := First( objects, O -> AsList( O ) = S );

    if S = fail then
        Error( "unable to find source set\n" );
    fi;

    G := List( S, i -> [ i, gen[i] ] ); # gen[i] is sure to be bound
```

```
    return MapOfFinSets( S, G, T );

end );
```

From the package CatReps (d & i by Tibor Grün and improved by Mohamed Barakat on 12 May 2020)
Back to Index

Procedure 2: ConcreteCategoryForCAP

```
InstallMethod( ConcreteCategoryForCAP,
       "for a list",
       [ IsList ],

  function( L )
    local C, c, objects;

    DeactivateCachingOfCategory( FinSets );
    CapCategorySwitchLogicOff( FinSets );
    DisableSanityChecks( FinSets );

    C := Subcategory( FinSets, "A finite concrete category" :
        overhead := false, FinalizeCategory := false );

    DeactivateCachingOfCategory( C );
    CapCategorySwitchLogicOff( C );
    DisableSanityChecks( C );

    SetFilterObj( C, IsFiniteConcreteCategory );

    AddIsAutomorphism( C,
      function( alpha )
        return IsAutomorphism( UnderlyingCell( alpha ) );
    end );

    AddInverseForMorphisms( C,
      function( alpha )
        return Inverse( UnderlyingCell( alpha ) ) / CapCategory( alpha );
    end );

    c := ConcreteCategory( L );

    C!.ConcreteCategoryRecord := c;

    objects := List( c.objects, FinSet );

    SetSetOfObjects( C, List( objects, o -> o / C ) );

    SetSetOfGeneratingMorphisms( C, List(
        c.generators, g -> ConvertToMapOfFinSets( objects, g ) / C ) );

    Finalize( C );

    return C;

end );
```

From the package CatReps (d & i by Mohamed Barakat on 20 Feb 2020)
Back to Index

Procedure 3: RightQuiverFromConcreteCategory

```
InstallMethod( RightQuiverFromConcreteCategory,
       "for a finite category",
       [ IsFiniteConcreteCategory ],

  function( C )
    local objects, gmorphisms, arrows, i, mor, q;

    objects := SetOfObjects( C );
    gmorphisms := SetOfGeneratingMorphisms( C );
    arrows := [];

    i := 1;

    for mor in gmorphisms do
        arrows[i] :=[
                    PositionProperty( objects,
                           o -> IsEqualForObjects( Source( mor ), o ) ),
                    PositionProperty( objects,
                           o -> IsEqualForObjects( Range( mor ), o ) )
                    ];
        i := i+1;
    od;

    q := RightQuiver( "q(1)[a]", Length( objects ), arrows );

    return q;

end );
```

From the package CatReps (d & i by Tibor Grün on 7 May 2020)
Back to Index

Procedure 4: RelationsOfEndomorphisms

```
InstallMethod( RelationsOfEndomorphisms,
       "for a finite category",
       [ IsFiniteConcreteCategory ],

  function( C )
    local gmorphisms, q, relation_of_endomorphism,
          arrows, endos, vertices, i, mor, mpowers, m, npowers, n, foundEqual,
          relsEndo;

    gmorphisms := SetOfGeneratingMorphisms( C );
    q := RightQuiverFromConcreteCategory( C );

    relation_of_endomorphism := function( a, m, n )
        if m = 0 then
            return [ a^n, Source( a ) ];
        fi;
        return [ a^(m+n), a^m ];
    end;

    arrows := Arrows( q );
```

```
    endos := Filtered( arrows, a -> Source( a ) = Target( a ) );

    vertices := Collected( List( endos, Source ) );

    if ForAny( vertices, l -> l[2] > 1 ) then
        Error( "we assume at most 1 generating endomorphism per vertex\n" );
    fi;

    relsEndo := [ ];

    for i in [ 1 .. Length( gmorphisms ) ] do
        mor := gmorphisms[i];
        if not IsEndomorphism( mor ) then
            continue;
        fi;
        mpowers := [];
        m := 0;

        # sigma lemma
        foundEqual := false;
        while not mor^m in mpowers do
            n := 1;
            npowers := [];
            while not mor^(m+n) in npowers and
              not foundEqual do
                if IsCongruentForMorphisms( mor^(m+n), mor^m ) then
                    Add( relsEndo,
                         relation_of_endomorphism( arrows[i], m, n ) );
                    foundEqual := true;
                fi;
                Add( npowers, mor^(m+n) );
                n := n+1;
            od;
            Add( mpowers, mor^m );
            m := m+1;
        od;
    od;

    return relsEndo;

end );
```

From the package CatReps (d & i by Tibor Grün on 7 May 2020)
Back to Index

Procedure 5: AsFpCategory

```
        "for a finite category",
        [ IsFiniteConcreteCategory ],

  function( C )
    local objects, gmorphisms, q, Qq, relEndo, fpC, A, F, vertices, relations,
          func, st, s, t, homST, list, p, pos;

    objects := SetOfObjects( C );
    gmorphisms := SetOfGeneratingMorphisms( C );

    q := RightQuiverFromConcreteCategory( C );
```

```
relEndo := RelationsOfEndomorphisms( C );

fpC := Category( q, relEndo );

A := UnderlyingQuiverAlgebra( fpC );

F := CapFunctor( fpC, objects, gmorphisms, C );

vertices := List( SetOfObjects( fpC ), UnderlyingVertex );

relations := [ ];

func :=
  function( p, l )
    return ForAny( l, p1->
                  IsCongruentForMorphisms(
                         ApplyToQuiverAlgebraElement( F, p ),
                         ApplyToQuiverAlgebraElement( F, p1 ) )
                  );
end;

for st in Cartesian( vertices, vertices ) do
    s := st[1];
    t := st[2];
    if s = t then
        continue;
    fi;
    homST := BasisPathsBetweenVertices( A, s, t );
    homST := List( homST, p -> PathAsAlgebraElement( A, p ) );

    list := [ ];

    for p in homST do
        pos := PositionProperty( list, l -> func( p, l ) );
        if IsInt(pos) then
            Add( list[pos], p );
        else
            Add( list, [ p ] );
        fi;
    od;
    list := List( list, l-> List( l, p -> Paths( p!.representative )[1] ) )
        ;
    Append( relations, list );
od;

relations := Filtered( relations, l -> Length( l ) > 1 );
relations := List( relations, l -> List( l{[ 2 .. Length( l ) ]}, p -> [ p,
     l[1] ] ) );
relations := Concatenation( relations );

relations := Concatenation( relEndo, relations );

fpC := Category( q, relations );

SetUnderlyingCategory( fpC, C );

return fpC;
```

```
end );
```

From the package CatReps (d & i by Tibor Grün on 26 Oct. 2020)
Back to Index

Procedure 6: Algebroid

```
        "for a homalg ring and a finite category",
        [ IsHomalgRing and IsCommutative, IsFiniteConcreteCategory ],

  function( k, C )

    return Algebroid( k, AsFpCategory( C ) );

end );
```

From the package CatReps (By Mohamed Barakat and Tibor Grün, most code moved to AsFpCategory on 26 Okt. 2020)
Back to Index

Procedure 7: Category

```
    relations := List( L, a -> PathAsAlgebraElement( Qq, a[1] ) -
        PathAsAlgebraElement( Qq, a[2] ) );

    A := Qq / Ideal( Qq, relations );

    A := Qq / GroebnerBasis( IdealOfQuotient( A ) );

    C := Category( A : relations := L );

    SetRelationsOfFpCategory( C, L );

    return C;

end );

##
InstallMethod( Category,
        "for a QPA quiver",
        [ IsQuiver, IsList ],

  function( quiver, L )

    return Category( PathAlgebra( ALGEBRIODS.ring, quiver ), L );

end );
```

From the package Algebroids (d & i by Tibor Grün on 26 Oct. 2020)
Back to Index

Procedure 8: Algebroid

```
        SetCommutativeRingOfLinearCategory( A, LeftActingDomain( Rq ) );
    fi;

    SetUnderlyingQuiverAlgebra( A, Rq );
    SetFilterObj( A, IsAlgebroid );
    if Length( Vertices( quiver ) ) = 1 then
```

```
        SetFilterObj( A, IsAlgebraAsCategory );
    fi;

    A!.Vertices := rec( );
    A!.Arrows := rec( );

    return ADD_FUNCTIONS_FOR_ALGEBROID( A, over_Z );

end );

##
InstallMethod( Algebroid,
        "for a QPA quiver algebra",
        [ IsQuiverAlgebra ],

  function( A )

    return Algebroid( A, false );

end );

##
InstallMethod( Algebroid,
        "for a QPA path algebra and a list",
```

From the package `Algebroids` (d & i by Mohamed Barakat on 26 Okt. 2020)
Back to Index

Procedure 9: `YonedaEmbedding`

```
##
InstallMethod( YonedaEmbedding,
        [ IsAlgebroid ],

  function ( algebroid )
    local k, matrix_cat, algebroid_op, objs, mors, functors_cat, name, Yoneda;

    k := CommutativeRingOfLinearCategory( algebroid );

    matrix_cat := MatrixCategory( k );

    algebroid_op := OppositeAlgebroidOverOppositeQuiverAlgebra( algebroid );

    objs := SetOfObjects( algebroid_op );

    mors := SetOfGeneratingMorphisms( algebroid_op );

    functors_cat := Hom( algebroid_op, matrix_cat );

    name := "Yoneda embedding functor";

    Yoneda := CapFunctor( name, algebroid, functors_cat );

    AddObjectFunction( Yoneda,
      function ( o )
        local m, Yo;
```

```
      o := OppositePath( UnderlyingVertex( o ) ) / algebroid_op;

      m := List( mors, mor -> HomStructure( o, mor ) );

      o := List( objs, obj -> HomStructure( o, obj ) );

      Yo := AsObjectInHomCategory( algebroid_op, o, m );

      SetIsProjective( Yo, true );

      return Yo;

  end );

  AddMorphismFunction( Yoneda,
    function ( s, m, r )

      m := MorphismInAlgebroid(
             OppositePath( UnderlyingVertex( Range( m ) ) ) / algebroid_op,
             OppositeAlgebraElement( UnderlyingQuiverAlgebraElement( m ) ),
             OppositePath( UnderlyingVertex( Source( m ) ) ) / algebroid_op
           );

      m := List( objs, o -> HomStructure( m, o ) );
```

From the package FunctorCategories (d & i by Kamal Saleh on 18 May 2020)
Back to Index

Procedure 10: DecomposeOnceByRandomEndomorphism

```
InstallMethod( DecomposeOnceByRandomEndomorphism,
       "for an object in a Hom-category",
       [ IsCapCategoryObjectInHomCategory ],

 function ( F )
   local d, endbas, k, n, random, b, alpha, i, alpha2, keremb;

   d := Maximum( List( ValuesOnAllObjects( F ), Dimension ) );

   if d = 0 then
       return fail;
   fi;

   endbas := ShallowCopy( BasisOfExternalHom( F, F ) );

   Add( endbas, ZeroMorphism( F, F ) );

   k := CommutativeRingOfLinearCategory( CapCategory( F ) );

   n := Int( Log2( Float( d ) ) ) + 1;

   ## the default is true
   random := not IsIdenticalObj( ValueOption( "random" ), false );

   for b in Reversed( [ 2 .. Length( endbas ) ] ) do

       if random then
           alpha := endbas[b] + Random( k ) * endbas[b-1];
```

```
        else
            alpha := endbas[b] + endbas[b-1];
        fi;

        SetFilterObj( alpha, IsMultiplicativeElementWithInverse );

        for i in [ 1 .. n ] do
            alpha2 := PreCompose( alpha, alpha );
            if IsCongruentForMorphisms( alpha, alpha2 ) then
                break;
            fi;
            alpha := alpha2;
        od;

        if IsZero( alpha ) then
            continue;
        fi;

        keremb := KernelEmbedding( alpha );

        if IsZero( keremb ) then
            continue;
        fi;

        return [ ImageEmbedding( alpha ), keremb ];

    od;

    return fail;

end );
```

From the package FunctorCategories
Back to Index

Procedure 11: WeakDirectSumDecomposition

```
InstallMethod( WeakDirectSumDecomposition,
        "for an object in a Hom-category",
        [ IsCapCategoryObjectInHomCategory ],

  function ( F )
    local queue, summands, eta, result;

    queue := [ IdentityMorphism( F ) ];

    summands := [ ];

    while not IsEmpty( queue ) do

        eta := Remove( queue );

        result := DecomposeOnceByRandomEndomorphism( Source( eta ) );

        if result = fail then
            Add( summands, eta );
        else
            Append( queue, List( result, emb -> PreCompose( emb, eta ) ) );
```

```
        fi;

    od;

    return summands;

end );
```

From the package FunctorCategories
Back to Index

Procedure 12: MorphismOntoSumOfImagesOfAllMorphisms

```
    return MereExistenceOfUniqueSolutionOfLinearSystemInAbCategory(
        CapCategory( right_coeffs[1,1] ), left_coeffs, right_coeffs );

end );

####################################
#
# categorical methods derivations:
#
####################################

##
AddDerivationToCAP( PreInverse,
        [ [ IdentityMorphism, 1 ],
          [ Lift, 1 ],
          ],

  function( cat, alpha )

    return Lift( IdentityMorphism( Range( alpha ) ), alpha );
```

From the package CategoryConstructor (d & i by Mohamed Barakat on 10 April 2020)
Back to Index

Procedure 13: EmbeddingOfSumOfImagesOfAllMorphisms

```

##
AddDerivationToCAP( PostInverse,
        [ [ IdentityMorphism, 1 ],
          [ Colift, 1 ],
          ],

  function( cat, alpha )

    return Colift( alpha, IdentityMorphism( Source( alpha ) ) );
```

From the package CategoryConstructor (d & i by Mohamed Barakat on 10 April 2020)
Back to Index

Procedure 14: SumOfImagesOfAllMorphisms

```
##
AddDerivationToCAP( MorphismOntoSumOfImagesOfAllMorphisms,
        [ [ BasisOfExternalHom, 1 ],
```

```
        [ UniversalMorphismFromZeroObject, 1 ],
        [ UniversalMorphismFromDirectSum, 1 ]
        ],

function( cat, a, b )
  local hom;
```

From the package `CategoryConstructor` (d & i by Mohamed Barakat on 10 April 2020)
Back to Index

B Selbstständigkeitserklärung

Erklärung

Hiermit versichere ich, dass ich die vorliegende schriftliche Bachelorarbeit selbstständig verfasst und keine anderen als die von mir angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinne nach entnommen sind, wurden in jedem Fall unter Angabe der Quellen (einschliesslich des World Wide Web und anderer elektronischer Text- und Datensammlungen) kenntlich gemacht. Dies gilt auch für die beigegebenen Zeichnungen, bildlichen Darstellungen, Skizzen und dergleichen. Mir ist bewusst, dass jedes Zuwiderhandeln als Täuschungsversuch zu gelten hat, der die Anerkennung der Bachelor ausschliesst und weitere angemessene Sanktionen zur Folge haben kann.
Siegen, (October 30, 2020)

_____

(Tibor Grün)