A constructive approach to Freyd categories

Sebastian Posur

University of Siegen

April 26, 2018

Finitely presented functors

Finitely presented functors

Let R be a ring.

Finitely presented functors

Let R be a ring.

Q: Is it possible to model the category of finitely presented functors

$$R\text{-}\mathrm{Mod} \longrightarrow \mathbf{Ab}$$

on the computer?

Let R be a ring.

Q: Is it possible to model the category of finitely presented functors

$$R\text{-Mod} \longrightarrow \mathbf{Ab}$$

on the computer?

What are finitely presented functors?

Finitely presented functors

Let R be a ring.

Finitely presented functors

Let R be a ring.

Definition

A functor $F : R\text{-}\mathrm{Mod} \to \mathbf{Ab}$ is called **finitely presented** if there exist $A, B \in R\text{-}\mathrm{Mod}$ and an exact sequence of functors

$$0 \longleftarrow F \longleftarrow \mathrm{Hom}(B, -) \longleftarrow \mathrm{Hom}(A, -)$$

Finitely presented functors

Let R be a ring.

Q: Is it possible to model the category of finitely presented functors

$$R\text{-}\mathrm{Mod} \longrightarrow \mathbf{Ab}$$

on the computer?

Finitely presented functors

Let R be a ring.

Q: Is it possible to model the category of finitely presented functors

$$R\text{-fpmod} \longrightarrow \mathbf{Ab}$$

on the computer?

CAP

CAP

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

CAP

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,

CAP

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

CAP - Categories, Algorithms, Programming
(Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

Category of finite dim. vector spaces

Let k be a field.

Let k be a field.

Category of finite dim. vector spaces

- $\mathrm{Obj} :=$ finite dim. k-vector spaces

Category of finite dim. vector spaces

Let k be a field.

Category of finite dim. vector spaces

- $\mathrm{Obj} :=$ finite dim. k-vector spaces
- $\mathrm{Hom}(V, W) := k$-linear maps $V \to W$

Let k be a field.

Category of finite dim. vector spaces

- $\mathrm{Obj} :=$ finite dim. k-vector spaces
- $\mathrm{Hom}(V, W) := k$-linear maps $V \to W$

Category of matrices

- $\mathrm{Obj} := \mathbb{N}_0$

Category of finite dim. vector spaces

Let k be a field.

Category of finite dim. vector spaces

- $\mathrm{Obj} :=$ finite dim. k-vector spaces
- $\mathrm{Hom}(V, W) := k$-linear maps $V \to W$

Category of matrices

- $\mathrm{Obj} := \mathbb{N}_0$
- $\mathrm{Hom}(m, n) := k^{m \times n}$

Let k be a field.

Category of finite dim. vector spaces

- $\mathrm{Obj} :=$ finite dim. k-vector spaces
- $\mathrm{Hom}(V, W) :=$ k-linear maps $V \to W$

Category of matrices

- $\mathrm{Obj} := \mathbb{N}_0$
- $\mathrm{Hom}(m, n) := k^{m \times n}$

Category of finite dim. vector spaces

Let k be a field.

Category of finite dim. vector spaces

- $\mathrm{Obj} :=$ finite dim. k-vector spaces
- $\mathrm{Hom}(V, W) :=$ k-linear maps $V \to W$

$$\simeq$$

Category of matrices

- $\mathrm{Obj} := \mathbb{N}_0$
- $\mathrm{Hom}(m, n) := k^{m \times n}$

Let k be a field.

Category of finite dim. vector spaces

- $\mathrm{Obj} :=$ finite dim. k-vector spaces
- $\mathrm{Hom}(V, W) :=$ k-linear maps $V \to W$

$$\simeq$$

Category of matrices (computerfriendly model)

- $\mathrm{Obj} := \mathbb{N}_0$
- $\mathrm{Hom}(m, n) := k^{m \times n}$

Category of finite dim. vector spaces

Let k be a field.

Category of finite dim. vector spaces

- $\mathrm{Obj} :=$ finite dim. k-vector spaces
- $\mathrm{Hom}(V, W) :=$ k-linear maps $V \to W$

$$\simeq$$

Category of matrices (computerfriendly model)

$$\left.\begin{array}{l} \bullet\ \mathrm{Obj} := \mathbb{N}_0 \\ \bullet\ \mathrm{Hom}(m, n) := k^{m \times n} \end{array}\right\} \mathrm{Rows}_k$$

The language of category theory

Some categorical operations in abelian categories

Some categorical operations in abelian categories

- $\oplus : \mathrm{Obj} \times \mathrm{Obj} \to \mathrm{Obj}$

The language of category theory

Some categorical operations in abelian categories

- $\oplus : \mathrm{Obj} \times \mathrm{Obj} \to \mathrm{Obj}$
- $+, - : \mathrm{Hom}(A, B) \times \mathrm{Hom}(A, B) \to \mathrm{Hom}(A, B)$

The language of category theory

Some categorical operations in abelian categories

- $\oplus : \mathrm{Obj} \times \mathrm{Obj} \to \mathrm{Obj}$
- $+, - : \mathrm{Hom}(A, B) \times \mathrm{Hom}(A, B) \to \mathrm{Hom}(A, B)$
- $\ker : \mathrm{Hom}(A, B) \to \mathrm{Obj}$

The language of category theory

Some categorical operations in abelian categories

- $\oplus : \mathrm{Obj} \times \mathrm{Obj} \to \mathrm{Obj}$
- $+, - : \mathrm{Hom}(A, B) \times \mathrm{Hom}(A, B) \to \mathrm{Hom}(A, B)$
- $\mathrm{ker} : \mathrm{Hom}(A, B) \to \mathrm{Obj}$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$.

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$.

$$A \xrightarrow{\varphi} B$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

$$A \xrightarrow{\varphi} B$$

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker \varphi$,

$$\begin{tikzcd} \ker \varphi & & \\ & A \arrow[r, "\varphi"] & B \end{tikzcd}$$

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of $\varphi \ldots$

$\ldots$ one needs an object $\ker\varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,

$$\begin{tikzcd}
\ker\varphi \arrow[dr, hook, "\kappa"] & & \\
& A \arrow[r, "\varphi"] & B
\end{tikzcd}$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of $\varphi \ldots$

… one needs an object $\ker\varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ

$$\begin{tikzcd}
\ker\varphi \arrow[dr, hook, "\kappa"] \arrow[drr, bend left, "0"] & & \\
& A \arrow[r, "\varphi"] & B \\
T \arrow[ur, "\tau"] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker\varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a *unique* morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$

$$\begin{tikzcd}
\ker\varphi \arrow[dr, hook, "\kappa"] \arrow[drr, bend left, "0"] & & \\
& A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker \varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a *unique* morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$, such that

$$\begin{tikzcd}
\ker\varphi \arrow[dr, hook, "\kappa"] \arrow[drr, bend left, "0"] & & \\
\arrow[r, phantom, "\circlearrowright"] & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"'] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker \varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a *unique* morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$, such that

$$
\begin{tikzcd}
\ker \varphi \arrow[dr, hook, "\kappa"] \arrow[drr, bend left, "0"] & & \\
 & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"'] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}
$$

3 algorithms needed for the kernel

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker \varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a *unique* morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$, such that

$$\begin{tikzcd}
\ker \varphi \arrow[dr, hook, "\kappa"] \arrow[drr, bend left, "0"] & & \\
 & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"'] \arrow[urr, bend right, "0"'] \arrow[uu, phantom, shift right=4ex, "\circlearrowright"] & &
\end{tikzcd}$$

3 algorithms needed for the kernel/cokernel.

CAP - Categories, Algorithms, Programming
(Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

Category constructors

$$\text{category} \longrightarrow \boxed{\text{category constructor}} \longrightarrow \text{another category}$$

Category constructors

$$\text{category} \longrightarrow \boxed{\text{category constructor}} \longrightarrow \text{another category}$$

Example

Category constructors

category $\longrightarrow$ $\boxed{\text{category constructor}}$ $\longrightarrow$ another category

Example

$\mathbf{A}$ additive $\longrightarrow$ $\boxed{\text{Freyd category}}$ $\longrightarrow$ $\mathcal{A}(\mathbf{A})$

Category constructors

$$\text{category} \longrightarrow \boxed{\text{category constructor}} \longrightarrow \text{another category}$$

Example

$$\mathbf{A} \text{ additive} \longrightarrow \boxed{\text{Freyd category}} \longrightarrow \mathcal{A}(\mathbf{A})$$

Freyd category

Freyd category

Freyd category: implicit characterization

The Freyd category of an additive category $\mathbf{A}$ is its universal cokernel completion

Freyd category

Freyd category: implicit characterization

The Freyd category of an additive category **A** is its universal cokernel completion, i.e., it consists of

- a category $\mathcal{A}(\mathbf{A})$ having cokernels
- a functor $\mathbf{A} \to \mathcal{A}(\mathbf{A})$

Freyd category

Freyd category: implicit characterization

The Freyd category of an additive category **A** is its universal cokernel completion, i.e., it consists of

- a category $\mathcal{A}(\mathbf{A})$ having cokernels
- a functor $\mathbf{A} \to \mathcal{A}(\mathbf{A})$

such that for all additive **T** with cokernels and functors $\mathbf{A} \to \mathbf{T}$, we have:

Freyd category

Freyd category: implicit characterization

The Freyd category of an additive category **A** is its universal cokernel completion, i.e., it consists of

- a category $\mathcal{A}(\mathbf{A})$ having cokernels
- a functor $\mathbf{A} \to \mathcal{A}(\mathbf{A})$

such that for all additive **T** with cokernels and functors $\mathbf{A} \to \mathbf{T}$, we have:

$$\begin{tikzcd} \mathbf{A} \arrow[r] \arrow[dr] & \mathcal{A}(\mathbf{A}) \arrow[d, dashed, "\exists! \text{ (up to nat. iso.) resp. cokernels}"] \\ & \mathbf{T} \end{tikzcd}$$

Freyd category

Examples of Freyd categories

Freyd category

Examples of Freyd categories

- $\mathcal{A}(\mathrm{Rows}_R) \simeq R\text{-}\mathrm{fpmod}$

Freyd category

Examples of Freyd categories

- $\mathcal{A}(\mathrm{Rows}_R) \simeq R\text{-fpmod}$
- $\mathcal{A}(\{\, \mathrm{Hom}(A, -) \mid A \in R\text{-fpmod} \}) \simeq$ finitely presented functors over $R\text{-fpmod}$

Freyd category

Examples of Freyd categories

- $\mathcal{A}(\mathrm{Rows}_R) \simeq R\text{-fpmod}$
- $\mathcal{A}(\{\, \mathrm{Hom}(A, -) \mid A \in R\text{-fpmod} \,\}) \simeq$ finitely presented functors over $R\text{-fpmod}$

Yoneda's lemma

$$\{\, \mathrm{Hom}(A, -) \mid A \in R\text{-fpmod} \,\} \simeq R\text{-fpmod}^{\mathrm{op}}$$

Freyd category

Examples of Freyd categories

- $\mathcal{A}(\mathrm{Rows}_R) \simeq R\text{-}\mathrm{fpmod}$
- $\mathcal{A}(\{\, \mathrm{Hom}(A, -) \mid A \in R\text{-}\mathrm{fpmod} \}) \simeq$ finitely presented functors over $R\text{-}\mathrm{fpmod}$

Yoneda's lemma

$$\{\, \mathrm{Hom}(A, -) \mid A \in R\text{-}\mathrm{fpmod} \} \simeq R\text{-}\mathrm{fpmod}^{\mathrm{op}}$$

Corollary

finitely presented functors over $R\text{-}\mathrm{fpmod} \simeq \mathcal{A}\big(\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}\big)$

Freyd category

Examples of Freyd categories

- $\mathcal{A}(\mathrm{Rows}_R) \simeq R\text{-}\mathrm{fpmod}$
- $\mathcal{A}(\{\,\mathrm{Hom}(A,-) \mid A \in R\text{-}\mathrm{fpmod}\}) \simeq$ finitely presented functors over $R\text{-}\mathrm{fpmod}$

Yoneda's lemma

$$\{\,\mathrm{Hom}(A,-) \mid A \in R\text{-}\mathrm{fpmod}\} \simeq R\text{-}\mathrm{fpmod}^{\mathrm{op}}$$

Corollary

finitely presented functors over $R\text{-}\mathrm{fpmod} \simeq \mathcal{A}\left(\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}\right)$

Study the constructiveness of $\mathcal{A}(-)$.

Freyd category

Let $\mathbf{A}$ be additive.

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

Freyd category

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

- An object in $\mathcal{A}(\mathbf{A})$ is simply a morphism $(A \xleftarrow{\rho_A} R_A)$ in **A**.

Freyd category

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

- An object in $\mathcal{A}(\mathbf{A})$ is simply a morphism $(A \xleftarrow{\rho_A} R_A)$ in **A**.
- Morphisms:

Freyd category

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

- An object in $\mathcal{A}(\mathbf{A})$ is simply a morphism $(A \xleftarrow{\rho_A} R_A)$ in **A**.

$$A \xleftarrow{\quad \rho_A \quad} R_A$$

- Morphisms:

Freyd category

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

- An object in $\mathcal{A}(\mathbf{A})$ is simply a morphism $(A \xleftarrow{\rho_A} R_A)$ in **A**.

$$A \xleftarrow{\quad \rho_A \quad} R_A$$

- Morphisms:

$$B \xleftarrow{\quad \rho_B \quad} R_B$$

Freyd category

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

- An object in $\mathcal{A}(\mathbf{A})$ is simply a morphism $(A \xleftarrow{\rho_A} R_A)$ in **A**.
- Morphisms:

$$\begin{tikzcd} A \arrow[d, "\alpha"'] & R_A \arrow[l, "\rho_A"'] \\ B & R_B \arrow[l, "\rho_B"'] \end{tikzcd}$$

Freyd category

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

- An object in $\mathcal{A}(\mathbf{A})$ is simply a morphism $(A \xleftarrow{\rho_A} R_A)$ in **A**.
- Morphisms:

$$\begin{tikzcd} A \arrow[d, "\alpha"'] & R_A \arrow[l, "\rho_A"'] \arrow[d, dashed, "\exists \rho_\alpha"] \\ B & R_B \arrow[l, "\rho_B"'] \end{tikzcd}$$

Freyd category

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

- An object in $\mathcal{A}(\mathbf{A})$ is simply a morphism $(A \xleftarrow{\rho_A} R_A)$ in **A**.
- Morphisms:

$$\begin{tikzcd}
A \arrow[d, "\alpha"'] \arrow[drr, dashed] & & R_A \arrow[ll, "\rho_A"'] \arrow[d, dashed, "\exists \rho_\alpha"] \\
B \arrow[ur, phantom, "\circlearrowleft"] & & R_B \arrow[ll, "\rho_B"]
\end{tikzcd} \qquad = \qquad 0$$

Freyd category

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

- An object in $\mathcal{A}(\mathbf{A})$ is simply a morphism $(A \xleftarrow{\rho_A} R_A)$ in **A**.
- Morphisms:

$$
\begin{tikzcd}
A \arrow[d, "\alpha"'] \arrow[dr, dashed] & R_A \arrow[l, "\rho_A"'] \arrow[d, dashed, "\exists \rho_\alpha"] \\
B \arrow[ur, phantom, "\circlearrowright" near start] & R_B \arrow[l, "\rho_B"']
\end{tikzcd}
\qquad = \qquad 0
$$

We also write

$$(A \xleftarrow{\rho_A} R_A) \xrightarrow{\alpha} (B \xleftarrow{\rho_B} R_B)$$

Freyd category

Let **A** be additive.

Freyd category: data structures

The **Freyd category** $\mathcal{A}(\mathbf{A})$ is given by the following data:

- An object in $\mathcal{A}(\mathbf{A})$ is simply a morphism $(A \xleftarrow{\rho_A} R_A)$ in **A**.
- Morphisms:

$$\begin{tikzcd} A \arrow[d, "\alpha"'] \arrow[drr, dashed] & & R_A \arrow[ll, "\rho_A"'] \arrow[d, dashed, "\exists \rho_\alpha"] \\ B \arrow[r, phantom, "\circlearrowleft"] & & R_B \arrow[ll, "\rho_B"] \end{tikzcd} \qquad = \qquad 0$$

We also write

$$(A \xleftarrow{\rho_A} R_A) \xrightarrow{\alpha} (B \xleftarrow{\rho_B} R_B)$$

Algorithms in Freyd categories

Freyd category: algorithms

Freyd category: algorithms

- **composition** in **A** $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$

Algorithms in Freyd categories

Freyd category: algorithms

- **composition** in $\mathbf{A}$ $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$
- **identities** in $\mathbf{A}$ $\rightsquigarrow$ **identities** in $\mathcal{A}(\mathbf{A})$

Algorithms in Freyd categories

Freyd category: algorithms

- **composition** in $\mathbf{A}$ $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$
- **identities** in $\mathbf{A}$ $\rightsquigarrow$ **identities** in $\mathcal{A}(\mathbf{A})$
- **Cokernels** in $\mathcal{A}(\mathbf{A})$

Algorithms in Freyd categories

Freyd category: algorithms

- **composition** in $\mathbf{A}$ $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$
- **identities** in $\mathbf{A}$ $\rightsquigarrow$ **identities** in $\mathcal{A}(\mathbf{A})$
- **Cokernels** in $\mathcal{A}(\mathbf{A})$

Freyd category: algorithms

- **composition** in $\mathbf{A}$ $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$
- **identities** in $\mathbf{A}$ $\rightsquigarrow$ **identities** in $\mathcal{A}(\mathbf{A})$
- **Cokernels** in $\mathcal{A}(\mathbf{A})$

$$(A \xleftarrow{\rho_A} R_A) \xrightarrow{\alpha} (B \xleftarrow{\rho_B} R_B)$$

Algorithms in Freyd categories

Freyd category: algorithms

- **composition** in $\mathbf{A}$ $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$
- **identities** in $\mathbf{A}$ $\rightsquigarrow$ **identities** in $\mathcal{A}(\mathbf{A})$
- **Cokernels** in $\mathcal{A}(\mathbf{A})$

$$(B \xleftarrow{\begin{pmatrix}\rho_B \\ \alpha\end{pmatrix}} R_B \oplus A)$$

$$(A \xleftarrow{\rho_A} R_A) \xrightarrow{\alpha} (B \xleftarrow{\rho_B} R_B)$$

Freyd category: algorithms

- **composition** in **A** $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$
- **identities** in **A** $\rightsquigarrow$ **identities** in $\mathcal{A}(\mathbf{A})$
- **Cokernels** in $\mathcal{A}(\mathbf{A})$

$$\begin{tikzcd}
 & & \left(B \xleftarrow{\begin{pmatrix}\rho_B \\ \alpha\end{pmatrix}} R_B \oplus A\right) \\
\left(A \xleftarrow{\rho_A} R_A\right) \arrow[r, "\alpha"] & \left(B \xleftarrow{\rho_B} R_B\right) \arrow[ur, "\mathrm{id}_B"] &
\end{tikzcd}$$

Freyd category: algorithms

- **composition** in **A** $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$
- **identities** in **A** $\rightsquigarrow$ **identities** in $\mathcal{A}(\mathbf{A})$
- **Cokernels** in $\mathcal{A}(\mathbf{A})$

$$\begin{tikzcd}
 & & (B \xleftarrow{\begin{pmatrix}\rho_B \\ \alpha\end{pmatrix}} R_B \oplus A) \\
(A \xleftarrow{\rho_A} R_A) \arrow[r, "\alpha"] \arrow[rrd, bend right, "0"] & (B \xleftarrow{\rho_B} R_B) \arrow[ru, "\mathrm{id}_B"] \arrow[rd, "\tau"] & \\
 & & (T \xleftarrow{\rho_T} R_T).
\end{tikzcd}$$

Freyd category: algorithms

- **composition** in **A** $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$
- **identities** in **A** $\rightsquigarrow$ **identities** in $\mathcal{A}(\mathbf{A})$
- **Cokernels** in $\mathcal{A}(\mathbf{A})$

$$\begin{tikzcd}
 & & (B \xleftarrow{\begin{pmatrix}\rho_B \\ \alpha\end{pmatrix}} R_B \oplus A) \arrow[dd, dashed, "\tau"] \\
(A \xleftarrow{\rho_A} R_A) \arrow[r, "\alpha"] \arrow[drr, bend right, "0"'] & (B \xleftarrow{\rho_B} R_B) \arrow[ur, "\mathrm{id}_B"] \arrow[dr, "\tau"'] & \\
 & & (T \xleftarrow{\rho_T} R_T).
\end{tikzcd}$$

Algorithms in Freyd categories

Freyd category: algorithms

- **composition** in **A** $\rightsquigarrow$ **composition** in $\mathcal{A}(\mathbf{A})$
- **identities** in **A** $\rightsquigarrow$ **identities** in $\mathcal{A}(\mathbf{A})$
- **Cokernels** in $\mathcal{A}(\mathbf{A})$ are constructive if **A** is:

$$\begin{tikzcd}
 & & \left(B \xleftarrow{\begin{pmatrix}\rho_B \\ \alpha\end{pmatrix}} R_B \oplus A\right) \arrow[d, dashed, "\tau"] \\
\left(A \xleftarrow{\rho_A} R_A\right) \arrow[r, "\alpha"] \arrow[rr, bend right, "0"'] & \left(B \xleftarrow{\rho_B} R_B\right) \arrow[ur, "\mathrm{id}_B"] \arrow[r, "\tau"] & \left(T \xleftarrow{\rho_T} R_T\right).
\end{tikzcd}$$

Algorithms in Freyd categories

More delicate algorithmic issues:

Algorithms in Freyd categories

More delicate algorithmic issues:

- **Kernels** in Freyd categories.

Algorithms in Freyd categories

More delicate algorithmic issues:

- **Kernels** in Freyd categories.
- **Equality** of morphisms in Freyd categories.

Algorithms in Freyd categories

More delicate algorithmic issues:

- **Kernels** in Freyd categories.
- **Equality** of morphisms in Freyd categories.

Kernels in Freyd categories

Theorem (Freyd)

Let $\mathbf{A}$ be an additive category. Then $\mathcal{A}(\mathbf{A})$ has kernels if and only if $\mathbf{A}$ has weak kernels.

Kernels in Freyd categories

Theorem (Freyd)

Let $\mathbf{A}$ be an additive category. Then $\mathcal{A}(\mathbf{A})$ has kernels if and only if $\mathbf{A}$ has weak kernels.

This theorem can be proven constructively. In particular, an algorithm for weak kernels in $\mathbf{A}$ gives an algorithm for kernels in $\mathcal{A}(\mathbf{A})$.

Weak kernels

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker\varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a *unique* morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$, such that

$$\begin{tikzcd}
\ker\varphi \arrow[dr, "\kappa"] \arrow[drr, bend left, "0"] & & \\
\circlearrowright & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"'] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}$$

Weak kernels

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker\varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a *unique* morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$, such that

$$\begin{tikzcd}
\ker\varphi \arrow[dr, "\kappa"] \arrow[drr, bend left, "0"] & & \\
 & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"] \arrow[urr, bend right, "0"'] \arrow[u, phantom, "\circlearrowleft"] & &
\end{tikzcd}$$

Weak kernels

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker \varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a *unique* morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$, such that

$$\begin{tikzcd}
\ker \varphi \arrow[dr, "\kappa"] \arrow[drr, bend left, "0"] & & \\
\arrow[r, phantom, "\circlearrowleft"] & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}$$

Weak kernels

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker \varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a ~~unique~~ morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$, such that

$$\begin{tikzcd}
\ker \varphi \arrow[dr, "\kappa"] \arrow[drr, bend left, "0"] & & \\
\arrow[r, phantom, "\circlearrowright"] & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"'] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}$$

Weak kernels

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the weak kernel of φ ...

... one needs an object $\mathrm{weakker}\varphi$,
its embedding $\kappa = \mathrm{WeakKernelEmbedding}(\varphi)$,
and for every test morphism τ
a morphism $\lambda = \mathrm{WeakKernelLift}(\varphi, \tau)$, such that

$$\begin{tikzcd}
\mathrm{weakker}\varphi \arrow[dr, "\kappa"] \arrow[drr, bend left, "0"] & & \\
\arrow[r, phantom, "\circlearrowleft"] & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}$$

Weak kernels in Rows_R

Weak kernels in Rows_R

Rows_R has weak kernels iff the (row) kernel of every $M \in R^{m \times n}$ is finitely generated.

Weak kernels in Rows_R

Weak kernels in Rows_R

Rows_R has weak kernels iff the (row) kernel of every $M \in R^{m \times n}$ is finitely generated. In this case R is called (left) **coherent.**

Weak kernels in Rows_R

Weak kernels in Rows_R

Rows_R has weak kernels iff the (row) kernel of every $M \in R^{m \times n}$ is finitely generated. In this case R is called (left) **coherent.**

Remark

Algorithms computing such **syzygies** usually rely on Gröbner bases.

Weak kernels in Rows_R

Weak kernels in Rows_R

Rows_R has weak kernels iff the (row) kernel of every $M \in R^{m \times n}$ is finitely generated. In this case R is called (left) **coherent**.

Remark

Algorithms computing such **syzygies** usually rely on Gröbner bases.

Examples

- $k[x_1, \ldots, x_n]$

Weak kernels in Rows_R

Weak kernels in Rows_R

Rows_R has weak kernels iff the (row) kernel of every $M \in R^{m \times n}$ is finitely generated. In this case R is called (left) **coherent.**

Remark

Algorithms computing such **syzygies** usually rely on Gröbner bases.

Examples

- $k[x_1, \ldots, x_n]$ (coherent)

Weak kernels in Rows_R

Weak kernels in Rows_R

Rows_R has weak kernels iff the (row) kernel of every $M \in R^{m \times n}$ is finitely generated. In this case R is called (left) **coherent.**

Remark

Algorithms computing such **syzygies** usually rely on Gröbner bases.

Examples

- $k[x_1, \ldots, x_n]$ (coherent)
- $k[x_i \mid i \in \mathbb{N}]$

Weak kernels in Rows_R

Weak kernels in Rows_R

Rows_R has weak kernels iff the (row) kernel of every $M \in R^{m \times n}$ is finitely generated. In this case R is called (left) **coherent**.

Remark

Algorithms computing such **syzygies** usually rely on Gröbner bases.

Examples

- $k[x_1, \dots, x_n]$ (coherent)
- $k[x_i \mid i \in \mathbb{N}]$ (coherent)

Weak kernels in Rows_R

Weak kernels in Rows_R

Rows_R has weak kernels iff the (row) kernel of every $M \in R^{m\times n}$ is finitely generated. In this case R is called (left) **coherent**.

Remark

Algorithms computing such **syzygies** usually rely on Gröbner bases.

Examples

- $k[x_1, \dots, x_n]$ (coherent)
- $k[x_i \mid i \in \mathbb{N}]$ (coherent)
- $k[z, x_i \mid i \in \mathbb{N}]/\langle zx_i \mid i \in \mathbb{N}\rangle$

Weak kernels in Rows_R

Weak kernels in Rows_R

Rows_R has weak kernels iff the (row) kernel of every $M \in R^{m \times n}$ is finitely generated. In this case R is called (left) **coherent**.

Remark

Algorithms computing such **syzygies** usually rely on Gröbner bases.

Examples

- $k[x_1, \ldots, x_n]$ (coherent)
- $k[x_i \mid i \in \mathbb{N}]$ (coherent)
- $k[z, x_i \mid i \in \mathbb{N}]/\langle zx_i \mid i \in \mathbb{N} \rangle$ (not coherent)

Weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$

Weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$

We want to compute kernels in $\mathcal{A}(\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}})$.

Weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$

We want to compute kernels in $\mathcal{A}(\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}})$.

- By Freyd's theorem, it suffices to have an algorithm for weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$.

Weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$

We want to compute kernels in $\mathcal{A}(\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}})$.

- By Freyd's theorem, it suffices to have an algorithm for weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$.
- Cokernels in $\mathcal{A}(\mathrm{Rows}_R)$ yield weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$.

Weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$

We want to compute kernels in $\mathcal{A}(\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}})$.

- By Freyd's theorem, it suffices to have an algorithm for weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$.
- Cokernels in $\mathcal{A}(\mathrm{Rows}_R)$ yield weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$.
- Cokernels in $\mathcal{A}(\mathrm{Rows}_R)$ are algorithmic.

Weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$

We want to compute kernels in $\mathcal{A}(\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}})$.

- By Freyd's theorem, it suffices to have an algorithm for weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$.
- Cokernels in $\mathcal{A}(\mathrm{Rows}_R)$ yield weak kernels in $\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$.
- Cokernels in $\mathcal{A}(\mathrm{Rows}_R)$ are algorithmic.

$\rightsquigarrow$ Kernels in $\mathcal{A}(\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}})$ are algorithmic.

Algorithms in Freyd categories

More delicate algorithmic issues:

- **Kernels** in Freyd categories.
- **Equality** of morphisms in Freyd categories.

More delicate algorithmic issues:

- **Kernels** in Freyd categories.
- **Equality** of morphisms in Freyd categories.

Algorithms in Freyd categories

More delicate algorithmic issues:

- **Kernels** in Freyd categories. ✓
- **Equality** of morphisms in Freyd categories.

Algorithms in Freyd categories

More delicate algorithmic issues:

- **Kernels** in Freyd categories. ✓
- **Equality** of morphisms in Freyd categories.

Let $\mathbf{A}$ be an additive category.

Equality of morphisms in Freyd categories

Let **A** be an additive category.

Freyd category: equality

Morphisms: $\begin{tikzcd} A \arrow[d, "\alpha"'] \arrow[dr, dashed, blue] & R_A \arrow[l, "\rho_A"'] \arrow[d, dashed, "\exists \rho_\alpha"] \\ B \arrow[ur, phantom, "\circlearrowright"] & R_B \arrow[l, "\rho_B"] \end{tikzcd}$ $=$ 0

Equality of morphisms in Freyd categories

Let **A** be an additive category.

Freyd category: equality

Morphisms:
$$
\begin{tikzcd}
A \arrow[d, "\alpha"'] \arrow[drr, dashed, blue] & & R_A \arrow[ll, "\rho_A"'] \arrow[d, dashed, "\exists \rho_\alpha"] \\
B & \circlearrowleft & R_B \arrow[ll, "\rho_B"]
\end{tikzcd}
\quad = \quad 0
$$

- **A** $= \mathrm{Rows}_R$: linear system $X \cdot D = E$ for matrices D, E in R.

Equality of morphisms in Freyd categories

Let **A** be an additive category.

Freyd category: equality

Morphisms:
$$\begin{tikzcd} A \arrow[d, "\alpha"'] \arrow[dr, dashed, blue] & R_A \arrow[l, "\rho_A"'] \arrow[d, dashed, "\exists \rho_\alpha"] \\ B & R_B \arrow[l, "\rho_B"] \arrow[ul, phantom, "\circlearrowleft"] \end{tikzcd} \quad = \quad 0$$

- $\mathbf{A} = \mathrm{Rows}_R$: linear system $X \cdot D = E$ for matrices D, E in R.
- $\mathbf{A} = \mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}$: 2-sided linear system

Equality of morphisms in Freyd categories

Example

In $k[x_1, \ldots, x_n]$ we can solve all linear equations

Equality of morphisms in Freyd categories

Example

In $k[x_1, \dots, x_n]$ we can solve all linear equations using Gröbner bases

Equality of morphisms in Freyd categories

Example

In $k[x_1, \ldots, x_n]/I$ we can solve all linear equations using Gröbner bases

Example

In $S^{-1}k[x_1, \ldots, x_n]/I$ we can solve all linear equations using Gröbner bases

Equality of morphisms in Freyd categories

Example

In $S^{-1}k[x_1, \ldots, x_n]/I$ we can solve all linear equations using Gröbner bases if we can also algorithmically create elements $s \in S \cap J$ (if they exist) for any given ideal $J \subseteq k[x_1, \ldots, x_n]/I$.

Example

In $S^{-1}k[x_1, \dots, x_n]/I$ we can solve all linear equations using Gröbner bases if we can also algorithmically create elements $s \in S \cap J$ (if they exist) for any given ideal $J \subseteq k[x_1, \dots, x_n]/I$.

Nonexample

Let F be the free group in 10 generators.

Equality of morphisms in Freyd categories

Example

In $S^{-1}k[x_1, \dots, x_n]/I$ we can solve all linear equations using Gröbner bases if we can also algorithmically create elements $s \in S \cap J$ (if they exist) for any given ideal $J \subseteq k[x_1, \dots, x_n]/I$.

Nonexample

Let F be the free group in 10 generators. In $\mathbb{Q}[F \times F]$

Equality of morphisms in Freyd categories

Example

In $S^{-1}k[x_1, \ldots, x_n]/I$ we can solve all linear equations using Gröbner bases if we can also algorithmically create elements $s \in S \cap J$ (if they exist) for any given ideal $J \subseteq k[x_1, \ldots, x_n]/I$.

Nonexample

Let F be the free group in 10 generators. In $\mathbb{Q}[F \times F]$ the existence of a solution of a given linear system $X \cdot D = E$ is **computationally undecidable**.

Equality of morphisms in Freyd categories

Example

In $S^{-1}k[x_1, \ldots, x_n]/I$ we can solve all linear equations using Gröbner bases if we can also algorithmically create elements $s \in S \cap J$ (if they exist) for any given ideal $J \subseteq k[x_1, \ldots, x_n]/I$.

Nonexample

Let F be the free group in 10 generators. In $\mathbb{Q}[F \times F]$ the existence of a solution of a given linear system $X \cdot D = E$ is **computationally undecidable**. This is based on an example by Collins of a f.p. group with 10 generators with unsolvable word problem.

Equality of morphisms in Freyd categories

Nonexample (P., arXiv:1712.03492)

We can construct a ring R with an algorithm for solving linear systems $X \cdot D = E$

Equality of morphisms in Freyd categories

Nonexample (P., arXiv:1712.03492)

We can construct a ring R with an algorithm for solving linear systems $X \cdot D = E$, but the existence of a solution of a given 2-sided linear system is **computationally undecidable**.

Equality of morphisms in Freyd categories

Nonexample (P., arXiv:1712.03492)

We can construct a ring R with an algorithm for solving linear systems $X \cdot D = E$, but the existence of a solution of a given 2-sided linear system is **computationally undecidable**.

$$\rightsquigarrow$$

We can decide equality of morphisms in $\mathcal{A}(\mathrm{Rows}_R)$

Equality of morphisms in Freyd categories

Nonexample (P., arXiv:1712.03492)

We can construct a ring R with an algorithm for solving linear systems $X \cdot D = E$, but the existence of a solution of a given 2-sided linear system is **computationally undecidable**.

$$\rightsquigarrow$$

We can decide equality of morphisms in $\mathcal{A}(\mathrm{Rows}_R)$, but not in $\mathcal{A}\big(\mathcal{A}(\mathrm{Rows}_R)^{\mathrm{op}}\big)$.

Algorithms in Freyd categories

More delicate algorithmic issues:

- **Kernels** in Freyd categories. ✓
- **Equality** of morphisms in Freyd categories.

Algorithms in Freyd categories

More delicate algorithmic issues:

- **Kernels** in Freyd categories. ✓
- **Equality** of morphisms in Freyd categories.

Algorithms in Freyd categories

More delicate algorithmic issues:

- **Kernels** in Freyd categories. ✓
- **Equality** of morphisms in Freyd categories. ✓

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

Categorical algorithms

Categorical algorithms

Categorical algorithms for abelian categories

Categorical algorithms

Categorical algorithms for abelian categories

- Abelian constructions like pullback or pushout

Categorical algorithms

Categorical algorithms for abelian categories

- Abelian constructions like pullback or pushout
- Constructive diagram chases

Categorical algorithms

Categorical algorithms for abelian categories

- Abelian constructions like pullback or pushout
- Constructive diagram chases
- Spectral sequence algorithm

Categorical algorithms

Categorical algorithms for abelian categories

- Abelian constructions like pullback or pushout
- Constructive diagram chases
- Spectral sequence algorithm

Computational applications of Freyd categories

Categorical algorithms

Categorical algorithms for abelian categories

- Abelian constructions like pullback or pushout
- Constructive diagram chases
- Spectral sequence algorithm

Computational applications of Freyd categories

- Computing sets of natural transformations, e.g., $\mathrm{Hom}\left(\mathrm{Tor}_j(M,-), \mathrm{Ext}^i(A,-)\right)$

Categorical algorithms

Categorical algorithms for abelian categories

- Abelian constructions like pullback or pushout
- Constructive diagram chases
- Spectral sequence algorithm

Computational applications of Freyd categories

- Computing sets of natural transformations, e.g., Hom $\left(\mathrm{Tor}_j(M,-), \mathrm{Ext}^i(A,-) \right)$
- Construct free abelian categories

Categorical algorithms

Categorical algorithms for abelian categories

- Abelian constructions like pullback or pushout
- Constructive diagram chases
- Spectral sequence algorithm

Computational applications of Freyd categories

- Computing sets of natural transformations, e.g., Hom $\left(\mathrm{Tor}_j(M,-), \mathrm{Ext}^i(A,-) \right)$
- Construct free abelian categories $\rightsquigarrow$ prove homological theorems

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

CAP

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

CAP

CAP - Categories, Algorithms, Programming (Gutsche, P., Skartsæterhagen)

CAP is a software project in GAP facilitating the implementation of

1. specific instances of categories,
2. category constructors,
3. categorical algorithms.

CAP Days 2018 in Siegen: 8/28/2018 - 8/31/2018

References

Maurice Auslander, *Coherent functors*, Proc. Conf. Categorical Algebra (La Jolla, Calif., 1965), Springer, New York, 1966, pp. 189–231. MR MR0212070 (35 #2945)

Mohamed Barakat and Markus Lange-Hegermann, *An axiomatic setup for algorithmic homological algebra and an alternative approach to localization*, J. Algebra Appl. **10** (2011), no. 2, 269–293, (`arXiv:1003.1943`). MR 2795737 (2012f:18022)

Peter Freyd, *Representations in abelian categories*, Proc. Conf. Categorical Algebra (La Jolla, Calif., 1965), Springer, New York, 1966, pp. 95–120. MR 0209333

Sebastian Posur, *A constructive approach to Freyd categories*, ArXiv e-prints (2017), (`arXiv:1712.03492`).