CAP – a categorical (re)organization of computer algebra

Mohamed Barakat

Topos Institute Colloquium
July 17, 2025

Universität Siegen

Joint work with Sebastian Posur, Kamal Saleh, Fabian Zickgraf, Tom Kuhmichel

Linear PDE system with polynomial coefficients

Motivating application: Compute the space

$$\mathrm{Sol}(\Delta) \coloneqq \left\{ \begin{pmatrix} f(x,y,z) \\ g(x,y,z) \end{pmatrix} \mid f, g \in C^\infty(\mathbb{R}^3, \mathbb{R}) \right\}$$

of smooth solutions of the linear PDE system

$$\underbrace{\begin{array}{rr} (\partial_y\partial_z - \frac{1}{3}\partial_z^2 + \frac{1}{3}\partial_x + \partial_y - \frac{1}{3}\partial_z) f \;+ & (\partial_y\partial_z - \frac{1}{3}\partial_z^2) g = 0 \\ (\partial_x\partial_z + \partial_z^2 + \partial_z) f \;+ & (\partial_x\partial_z + \partial_z^2) g = 0 \\ (\partial_z^2 - \partial_x + \partial_z) f \;+ & (3\partial_x\partial_y + \partial_z^2) g = 0 \\ \partial_x\partial_y f \quad & = 0 \\ (\partial_z^2 - \partial_x + \partial_z) f \;+ & (-3\partial_x^2 + \partial_z^2) g = 0 \\ \partial_x^2 f \quad & = 0 \\ \left(x\partial_z^2 - \left(x - \frac{3}{2}\right)\partial_x + \left(x + \frac{3}{2}\right)\partial_z + \frac{3}{2}\right) f \;+ & (x\partial_z^2 + \frac{3}{2}\partial_x + \frac{3}{2}\partial_z) g = 0 \\ (\partial_z^3 + 2\partial_z^2 + \partial_z) f \;+ & (\partial_z^3 + \partial_x\partial_z + \partial_z^2) g = 0 \end{array}}_{\Delta(f,g) = 0}$$

The algebraic approach – Step 1

Weyl **algebra** D	$D \coloneqq \mathbb{R}[x,y,z]\langle \partial_x, \partial_y, \partial_z \rangle$
matrix $\mathtt{m}_\triangle$ over D	$\mathtt{m}_\triangle \in D^{8\times 2}$
D-**module** $\mathcal{F}$ of smooth functions	$\mathcal{F} \coloneqq C^\infty(\mathbb{R}^3, \mathbb{R})$

$$\underbrace{\underset{\mathtt{m}_\triangle}{\overset{D^{8\times 2}}{\Psi}}}_{\begin{pmatrix} \partial_y\partial_z - \frac{1}{3}\partial_z^2 + \frac{1}{3}\partial_x + \partial_y - \frac{1}{3}\partial_z & \partial_y\partial_z - \frac{1}{3}\partial_z^2 \\ \partial_x\partial_z + \partial_z^2 + \partial_z & \partial_x\partial_z + \partial_z^2 \\ \partial_z^2 - \partial_x + \partial_z & 3\partial_x\partial_y + \partial_z^2 \\ \partial_x\partial_y & 0 \\ \partial_z^2 - \partial_x + \partial_z & -3\partial_x^2 + \partial_z^2 \\ \partial_x^2 & 0 \\ x\partial_z^2 - (x - \frac{3}{2})\partial_x + (x + \frac{3}{2})\partial_z + \frac{3}{2} & x\partial_z^2 + \frac{3}{2}\partial_x + \frac{3}{2}\partial_z \\ \partial_z^3 + 2\partial_z^2 + \partial_z & \partial_z^3 + \partial_x\partial_z + \partial_z^2 \end{pmatrix}} \cdot \underbrace{\underset{\psi}{\overset{\mathcal{F}^2}{\Psi}}}_{\begin{pmatrix} f \\ g \end{pmatrix}} = \underbrace{\underset{0}{\overset{\mathcal{F}^8}{\Psi}}}_{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}}$$

The algebraic approach – Step 2

- $D := \mathbb{R}[x_1, \ldots, x_n]\langle \partial_{x_1}, \ldots, \partial_{x_n} \rangle$, $\mathtt{m}_\Delta \in D^{p \times q}$, $\mathcal{F} = C^\infty(\mathbb{R}^n, \mathbb{R})$
- $\Delta : \mathtt{m}_\Delta \cdot \psi = 0, \quad \psi \in \mathcal{F}^q$.

Interpret the matrix $\mathtt{m}_\Delta$ as a morphism of free D-modules:

Definition

Define the D-**module** M_Δ as the f.p. D-module

$$M_\Delta := D^{1\times q} / \operatorname{im}\left(D^{1\times p} \xrightarrow{\mathtt{m}_\Delta} D^{1\times q} \right) = D^{1\times q} / \left(D^{1\times p} \cdot \mathtt{m}_\Delta \right)$$
$$=: \operatorname{coker}\left(D^{1\times p} \xrightarrow{\mathtt{m}_\Delta} D^{1\times q} \right).$$

The residue classes $(\overline{e}_1, \ldots, \overline{e}_q)$ of the standard basis of the free D-module $D^{1\times q}$ is a generating system of M_Δ.

The rows of $\mathtt{m}_\Delta$ are the defining relations between $\overline{e}_1, \ldots, \overline{e}_q$:

$$\mathtt{m}_\Delta \cdot \begin{pmatrix} \overline{e}_1 \\ \vdots \\ \overline{e}_q \end{pmatrix} = 0.$$

The algebraic approach – Step 3

We therefore call $\begin{pmatrix} \overline{e}_1 \\ \vdots \\ \overline{e}_q \end{pmatrix}$ the abstract solution of $\mathtt{m}_\Delta \psi = 0$.

Lemma von Noether-Malgrange

The map

$$\begin{array}{ccc} \mathrm{Hom}(M_\Delta, \mathcal{F}) & \xrightarrow{\sim} & \mathrm{Sol}(\Delta, \mathcal{F}) \\ \varphi := (\overline{e}_i \mapsto f_i) & \mapsto & \psi := (f_i) \in \mathcal{F}^q \end{array}$$

is an isomorphism of $\mathbb{R}$-vector spaces.

The lemma implies that:

- $\mathrm{Sol}(\Delta, \mathcal{F})$ only depends on the isomorphism type of M_Δ.
- The D-module M_Δ can be studied *independent* of $\mathcal{F}$.
- A different generating set of M_Δ yields an equivalent system Δ' of linear PDEs with $M_\Delta \cong M_{\Delta'}$.

The algebraic approach – Step 4

Given a finitely presented D-module M:

The bidualizing spectral sequence

$$E^2_{pq} = \mathrm{Ext}^{-p}(\mathrm{Ext}^q(M,D),D)) \Longrightarrow M \quad \text{for } p+q=0$$

gives rise to the so-called **purity filtration** of M.

We can use this filtration to solve the above linear PDE system.

Software demo

The `homalg` project

Computing spectral sequences and their induced filtrations required computational models for:

- The abelian category D-$\mathbf{fpmod}$ of f.p. D-modules
- Diagram chasing in abelian categories

Both were realized in the `homalg` project:

- D-$\mathbf{fpmod}$ was implemented as an abelian category
- The only modular part of the implementation was D
- Depending on D, the implementation required various NF-algorithms up to noncommutative Gröbner bases
- Diagram chasing was realized by **generalized morphisms**

The motivation for the `CAP` project

- `homalg` was well-desigend for the intended application
- however, not modular enough to cover more applications
- implementing more complicated categories became increasingly difficult, e.g.,
- generalizing from f.p. modules to coh. sheaves was a pain

Rectify: Take category theory more seriously

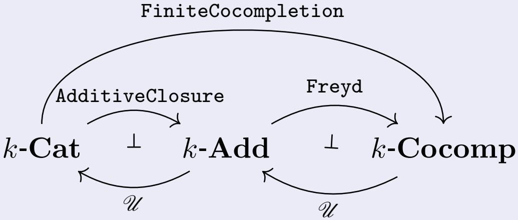
- category theory should guide all design decisions
- categories, functors, ... should become first class citizens
- turn category theory into a programming language:
- write all algorithms using categorical vocabulary

Revisiting D-**fpmod**

What is D-**fpmod** categorically?

- View D as a k-linear category on one object
- D-**fpmod** is the *finite colimit completion* of D

`FiniteCocompletion` as a **categorical tower** of biadjunctions



- `AdditiveClosure` formally adds direct sums
- `AdditiveClosure` invents matrices
- `Freyd` formally adds cokernels
- `Freyd` is a quotient of the arrow category

Free-forgetful 2-adjunctions

The above tower of categorical constructors is typically composed of several free-forgetful 2-adjunctions

$$\mathcal{D} \underset{\mathscr{U}}{\overset{\mathscr{L}}{\underset{\bot}{\rightleftarrows}}} \mathcal{E}$$

between a 2-category $\mathcal{D}$ of categories (called **doctrine**) and another doctrine $\mathcal{E}$ of categories with extra structure.

Software demo

FiniteCompletions

The dual category construction is also a 2-adjunction on each doctrine

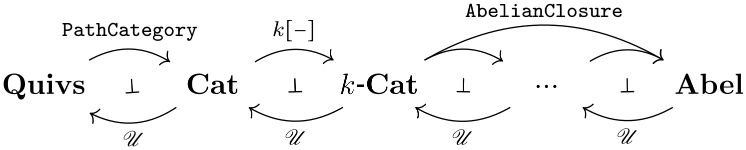
$$\mathcal{D} \underset{\mathscr{R} = \texttt{Opposite}}{\overset{\mathscr{L} = \texttt{Opposite}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\perp}}}} \mathcal{D}^{\mathrm{co-dual}}$$

Implementing `Opposite` requires a lot of meta programming.

More categorical towers of biadjunctions

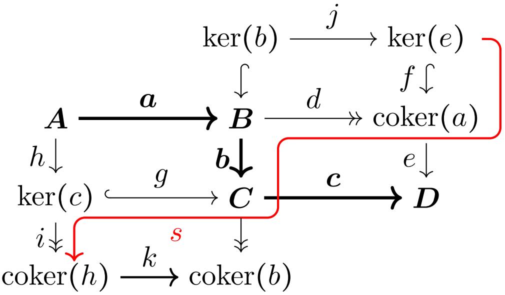
- $\texttt{CoFreyd} \coloneqq \texttt{Opposite} \circ \texttt{Freyd} \circ \texttt{Opposite}$
- $\texttt{FiniteCompletion} \coloneqq \texttt{Opposite} \circ \texttt{FiniteCocompletion} \circ \texttt{Opposite}$
- $\texttt{FpCoPreSheaves} \coloneqq \texttt{Opposite} \circ \texttt{FpPreSheaves} \circ \texttt{Opposite}$

A longer categorical tower of biadjunction yields `AbelianClosure` as a **categorical tower** of 2-*adjunctions*:



Simplest diagram chasing: The connecting morphism

Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.



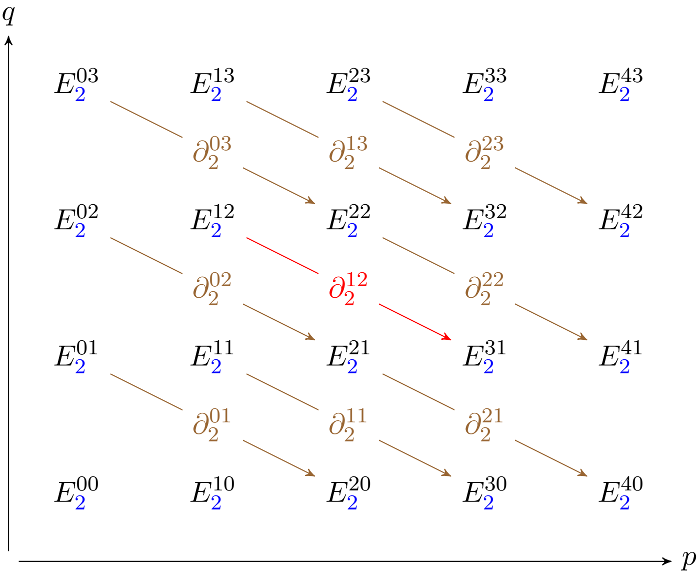
$\rightsquigarrow$ $\exists$ an *ess. unique natural* morphism $\ker(e) \xrightarrow{s} \operatorname{coker}(h)$ with $\ker(b) \xrightarrow{j} \ker(e) \xrightarrow{s} \operatorname{coker}(h) \xrightarrow{k} \operatorname{coker}(b)$ an exact sequence.

Software demo

`https://homalg-project.github.io/nb/SnakeInFreeAbelian`

Exercise: Along the same lines treat spectral sequences of bicomplexes.

Spectral sequences of bicomplexes



Examples of categorical towers

We can model

- free left R-modules of finite rank via $\mathcal{C}(R)^{\oplus}$
- free right R-modules of finite rank via $(\mathcal{C}(R)^{\oplus})^{\mathrm{op}}$
- finitely presented left R-modules via $\mathbf{Freyd}(\mathcal{C}(R)^{\oplus})$
- finitely presented right R-modules via $\mathbf{Freyd}((\mathcal{C}(R)^{\oplus})^{\mathrm{op}})$
- quivers via $\mathbf{Func}(\mathcal{C}(\mathfrak{A} \rightrightarrows \mathfrak{V}), \mathbf{Sets})$
- ZX-diagrams via $\mathbf{Sub}(\mathbf{Csp}(\mathbf{Slice}(\mathbf{Func}(\mathcal{C}(\mathfrak{A} \rightrightarrows \mathfrak{V}), \mathbf{Sets}))))$
- free Abelian categories for theorem proving via $\mathbf{Freyd}(\mathbf{Freyd}(-^{\mathrm{op}})^{\mathrm{op}})$
- linear representations of a group G over a field k via $\mathbf{Func}(\mathcal{C}(G), k^{\oplus})$
- radical ideals of a ring R via $\mathbf{StablePoset}(\mathbf{Poset}(\mathbf{Slice}(\mathcal{C}(R)^{\oplus})))$

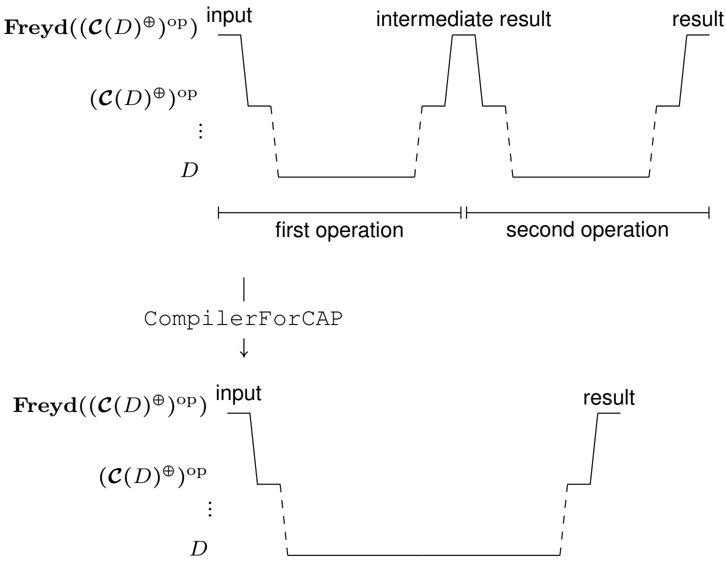
Advantages:

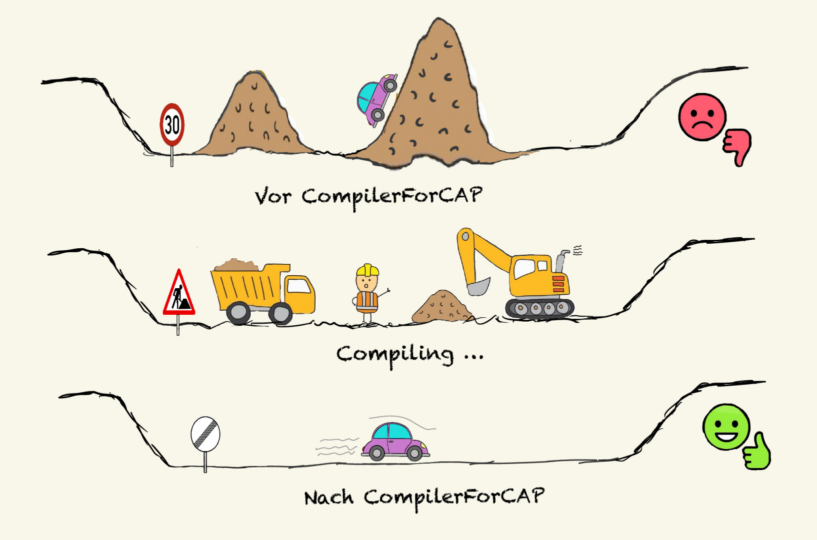
- **Reusability**: Building blocks can appear in multiple different contexts.
- **Separation of concerns**: Focus on a single concept at a time.
- **Verifiability**: Every constructor has a limited scope.
- **Emergence**: The whole is greater than the sum of its parts.

Effects on computer implementations

- **Efficient development** thanks to
  - reusability
  - separation of concerns
  - verifiability
  - emergence
- **Inefficient execution** due to computational overhead :-(
- Solution: compilation

Overhead of boxing and unboxing




30
Vor CompilerForCAP
Compiling ...
Nach CompilerForCAP

Benchmarks

Consider a computation in the categorical tower

$$\mathbf{Freyd}((\mathcal{C}(D)^{\oplus})^{\mathrm{op}}) \simeq \textbf{fpmod}\text{-}D$$

problem size	**original code** (s)	**compiled code** (s)	**factor**
1	0.2	0.05	≈ 5
2	2.4	0.06	≈ 50
3	19.1	0.07	≈ 250
4	118.9	0.09	≈ 1250
5	584.5	0.12	≈ 5000
10	N/A	0.35	N/A
20	N/A	1.34	N/A
30	N/A	3.53	N/A

We see a difference between “finishes in seconds” and “will never finish”.

Further applications

`CompilerForCAP` can also be used

- for removing additional sources of overhead,
- for **generating categorical code** from categorical towers,
- as a **proof assistant** for verifying categorical implementations.

Conclusion

- **Algorithmic category theory** is a high-level programming language.
- Using this language for building **categorical towers** allows
  - to reach a wide range of advanced and complex applications
  - allowing reusability, separation of concerns, verifiability, and emergence.
- This approach naturally comes with a computational overhead.
- `CompilerForCAP` can avoid this overhead, allowing us to make full use of the advantages of building categorical towers.

Thank you