CAP: categories, algorithms, programming

S. Gutsche (University of Siegen)
S. Posur (University of Siegen)

sebastian.gutsche@uni-siegen.de
sebastian.posur@uni-siegen.de

What is CAP?

CAP stands for categories, algorithms, programming and is an open source software project for constructive category theory written in GAP (groups, algorithms, programming) [4]. CAP's core package includes tools to facilitate the implementation of categories in GAP. The other CAP packages mainly include implementations of various category constructors, i.e., operations that output concrete instances of categories. The idea of category constructors lies at the heart of CAP: construct fairly complex computational contexts from easily graspable building blocks.

The development of CAP's core package started in December 2013 by the authors of this article followed by major contributions of Øystein Skartsæterhagen in 2015. CAP's core package along with three other packages[1] are distributed via the current GAP release[2], more packages can be found on GitHub[3].

As a quick way to learn about CAP and test its features, you can try CAP in a Jupyter notebook interactively in Binder[4].

Applications of CAP

CAP is used for redesigning the categorical foundations of the homalg project [7] and in the development of a new version of QPA, a package for computations with quivers and path algebras [11]. CAP has been used in the search for equivariant vector bundles on projective space [10], in the computation of associators in skeletal representation categories [9], in modeling coherent sheaves on toric varieties [6] and for performing cohomology computations within this context [2]. Furthermore, Dupont and Juteau implemented an algorithm in CAP for the computation of periods associated to hyperplane arrangements decorated with two colors [3].

The purpose of CAP

Every computer algebra system (CAS) faces the question of how to organize the mathematical entities that it wishes to support. Desirable features of such an organization are:

1. Unambiguous specifications: providing input/output types for the operations.
2. High composability: offering sufficiently many operations that allow to quickly write programs for the computational exploration of new mathematical ideas.
3. Intuitive usage: being designed for mathematicians as human beings.

CAP's approach to achieve these features is the organization of all its data in the language of category theory. Every object or morphism created within a CAP session belongs to exactly one category, which can be thought of as being part of its type.

Choosing category theory as an organizational principle allows the definition of clearly specified operations for concepts of interest in computer algebra like subobjects or quotients, image factorizations, (co)kernels, tensor products, or Hom-functors. Moreover, a system that strictly adheres to purely categorical specifications automatically benefits from the impressive expressiveness of category theory, allowing to compose complicated systems from fairly easy building blocks. Last, category theory nowadays is a widely accepted link between different areas of mathematics, and thus a suitable choice of a common language.

[1] These three packages are: `LinearAlgebraForCAP` (computing with finite dimensional vector spaces), `ModulePresentationsForCAP` (computing with finitely presented modules), `GeneralizedMorphismsForCAP` (performing diagram chases in homological algebra).

[2] Version 4.10.0, as of January 2019

[3] The CAP project GitHub repository: `https://github.com/homalg-project/CAP_project`

[4] Trying CAP in a Jupyter notebook: `https://mybinder.org/v2/gh/sebastianpos/cap-aachen2018/master`

How CAP works

CAP is ultimately designed for computational purposes. It can be used as a calculator that operates on objects and morphisms of some specific instance of a category. We will demonstrate the usage of CAP by means of a simple guiding example: $\mathrm{vec}_{\mathbb{Q}}$, the category of finite dimensional $\mathbb{Q}$-vector spaces. Later, we discuss more advanced examples.

Computer-friendly models of categories

We consider a category $\mathbf{C}$ to be **computable** if we have data structures for both the objects in $\mathrm{Obj}_{\mathbf{C}}$ and the morphisms in $\mathrm{Hom}_{\mathbf{C}}(A, B)$, along with algorithms for composing morphisms, deciding their equality, and constructing identities $\mathrm{id}_A \in \mathrm{Hom}_{\mathbf{C}}(A, A)$, where $A, B \in \mathrm{Obj}_{\mathbf{C}}$.

Example We can construct an example of a computable category as follows: we let the objects simply be given by $\mathbb{N}_0$. We define the set of homomorphisms from $m \in \mathbb{N}_0$ to $n \in \mathbb{N}_0$ as $\mathbb{Q}^{m \times n}$. Composition is induced by matrix multiplication (with swapped arguments), identities are given by identity matrices. We call this category $\mathrm{mat}_{\mathbb{Q}}$, the category of matrices[5] over $\mathbb{Q}$.

There exists an equivalence of categories

$$\mathrm{mat}_{\mathbb{Q}} \simeq \mathrm{vec}_{\mathbb{Q}}$$

that identifies $n \in \mathbb{N}_0$ with the row space $\mathbb{Q}^{1 \times n}$. Due to the simplicity of its data structures, the category $\mathrm{mat}_{\mathbb{Q}}$ is undoubtedly very computer-friendly, and as long as we only care about categorical notions, working within $\mathrm{mat}_{\mathbb{Q}}$ is as good as working within $\mathrm{vec}_{\mathbb{Q}}$, just like replacing a group G with an isomorphic group H is harmless as long as we only care about isomorphism-invariant properties of G.

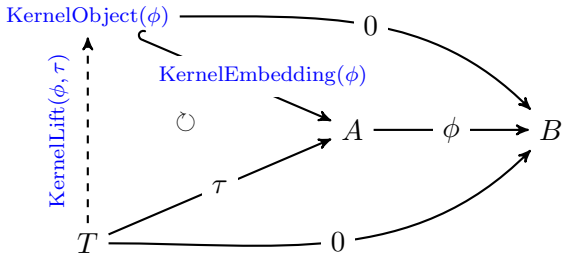
Here, we see a first benefit of categorical organization: the notion of an equivalence of categories makes us very flexible in our choice of data structures, and lets us, in the context of our guiding example, think about elements in $\mathbb{N}_0$ as if they were finite dimensional vector spaces.

Categorical language

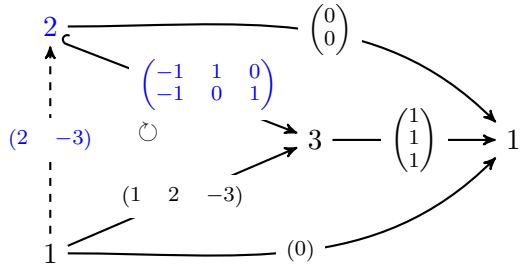
Most of CAP's primitives are constructive interpretations of definitions found in standard textbooks on category theory. For example, the concept of kernels within an Ab-category[6] is realized by three operations:

1. Given $\phi : A \to B$, compute an object $\mathrm{KernelObject}(\phi)$.
2. Given $\phi : A \to B$, compute a morphism $\mathrm{KernelEmbedding}(\phi) : \mathrm{KernelObject}(\phi) \to A$ s.t. $\phi \circ \mathrm{KernelEmbedding}(\phi) = 0$.
3. Given $\phi : A \to B$ and $\tau : T \to A$ such that
$$\phi \circ \tau = 0,$$
compute a uniquely determined morphism $\mathrm{KernelLift}(\phi, \tau) : T \to \mathrm{KernelObject}(\phi)$ s.t. $\mathrm{KernelEmbedding}(\phi) \circ \mathrm{KernelLift}(\phi, \tau) = \tau$.

We depict the three algorithms in a diagram:



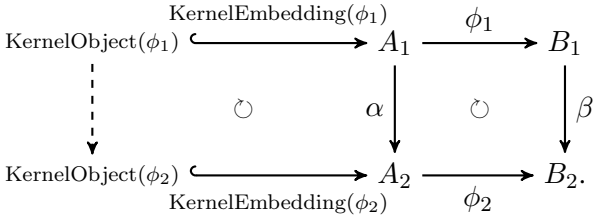
Here is an example of how the three algorithms could act on a specific input in $\mathrm{mat}_{\mathbb{Q}}$ (remember, the objects are simply elements in $\mathbb{N}_0$):



We see that within $\mathrm{mat}_{\mathbb{Q}}$ the operation $\mathrm{KernelObject}$ merely yields the dimension of the row syzygies, whereas $\mathrm{KernelEmbedding}$ outputs an actual basis. Finally, $\mathrm{KernelLift}$ expresses any matrix containing row syzygies as a linear combination w.r.t. the basis given by $\mathrm{KernelEmbedding}$.

Other examples of categorical notions supported by CAP are limits like direct products, terminal objects, equalizers, and pullbacks, colimits like cokernels, coproducts, initial objects, coequalizers, and pushouts, universal constructions like images and coimages, and additional categorical structures like tensor products and internal homs.

One needs a little practice in order to get used to expressing familiar set-theoretic constructions in the language of category theory. We give an example. Consider the following diagram



[5] Note that matrices form the morphisms in this category, not the objects.

[6] An Ab-category is a category whose homomorphism sets are abelian groups s.t. the composition is bilinear. In particular, there exists a zero morphism (simply denoted by 0) between any two objects.

The task is to find the dashed arrow which renders the left square commutative assuming we are given the solid arrows. If this is a diagram of abelian groups, one can argue set-theoretically: for an element $a \in A_1$ in the kernel of ϕ_1, we have $\phi_2(\alpha(a)) = \beta(\phi_1(a)) = 0$. Thus, $\alpha(a)$ lies in the kernel of ϕ_2 and we can construct the dashed arrow by restricting the source and range of α.

In the language of category theory, we can simply state that, due to the commutativity of the right square, the following closed formula is well-defined and yields the desired dashed arrow:

$$\mathrm{KernelLift}(\phi_2, \alpha \circ \mathrm{KernelEmbedding}(\phi_1)). \quad (1)$$

Note that this formula is easy to implement, we don't need any accessing of "underlying elements". Moreover, this formula is valid in any Ab-category with kernels.

Finitely presented modules

To give another example of such a category, we take a look at finitely presented modules[7] over a ring R.

Definition An R-module M is **finitely presented** if there exist $m, n \in \mathbb{N}_0$, $r_1, \ldots, r_m \in R^{1\times n}$ such that $M \cong \frac{R^{1\times n}}{\langle r_1, \ldots, r_m \rangle}$.

Finitely presented modules form a subcategory $R\text{-fpmod}$ of the category of all R-modules. For designing a computable model of $R\text{-fpmod}$, we first need to find a data structure for the objects. Depending on the information we are given about R, we have several competing choices.

Example Any matrix $M \in R^{m\times n}$ can be interpreted as the finitely presented module that is defined by

$$\frac{R^{1\times n}}{\langle \text{Rows of } M \rangle}.$$

This is the data structure used by `homalg` [7].

Example Assume that R is a coherent ring, i.e., the row syzygies of any matrix with entries in R are finitely generated. Then any pair of matrices $M \in R^{m\times n}$, $N \in R^{o\times n}$ gives rise to the module

$$\frac{\langle \text{Rows of } M \rangle}{\langle \text{Rows of } M \rangle \cap \langle \text{Rows of } N \rangle}$$

which is finitely presented due to the coherence of R. This is the data structure used for example by MACAULAY2 [5].

Example Assume that R is a principal ideal domain (PID). Then any proper chain of elements $d_1 | \ldots | d_m$ and any $n \in \mathbb{N}_0$ define the finitely presented module

$$R^{1\times n} \oplus \left(\bigoplus_{i=1}^{m} \frac{R}{\langle d_i \rangle} \right)$$

and all finitely presented modules arise in this way due to the structure theorem for PIDs.

If we impose in all three examples the extra condition of R being a computable ring (see [1]), then we can build up three different computable models of $R\text{-fpmod}$ from these three different data structures of the objects. In this case, CAP actually encourages the implementation of three different categories, one for each data structure. CAP provides interfaces for the creation of functors which can be thought of as tools for the conversion of data structures. Moreover, the categorical language that operates within each different model is always the same: a categorical term as presented in equation (1) can always be interpreted in any of the three models above, regardless of the underlying data structures.

Category constructors

Categories in CAP are first class citizens, i.e., they can be part of the input/output of operations. We call any operation that outputs a category a **category constructor**. Category constructors are helpful for quickly building up computable models[8] of categories.

Example CAP offers a category constructor for **Serre quotients**: given an abelian category $\mathbf{A}$ and a Serre subcategory $\mathbf{C}$ (given by a membership function), there is an abelian category $\frac{\mathbf{A}}{\mathbf{C}}$ universal with the property that objects of $\mathbf{C}$ are treated as zero objects within $\mathbf{A}$. This category can be employed to perform computations with coherent sheaves on toric varieties [6].

Example Given an abelian category $\mathbf{A}$, its **generalized morphism category** $\mathrm{G}(\mathbf{A})$ can be employed to perform diagram chases constructively [9]. All operations in $\mathrm{G}(\mathbf{A})$ are derived from the operations in $\mathbf{A}$.

If one wishes to perform diagram chases with coherent sheaves on toric varieties, one could simply concatenate the Serre quotient constructor with the generalized morphism constructor.

During our development and usage of CAP, we learned to appreciate more and more the impressive power of category theory as an organizational principle. It lets one build up complex mathematical entities step by step from easily graspable concepts. Each step, if implemented with the necessary categorical rigor, interacts smoothly and coherently with the previous and the next step.

Example A good example for the benefits of category constructors are **Freyd categories**: given an additive category[9] $\mathbf{A}$, we can turn its morphisms to the objects of a new category $\mathcal{A}(\mathbf{A})$. Morphisms in $\mathcal{A}(\mathbf{A})$ are then defined as commutative squares considered up to homotopy [8]. Depending on the input category $\mathbf{A}$, we get computable models of different categories:

[7] We will only consider left modules in this article.

[8] Reminder: this includes data structures for objects and morphisms, as well as algorithms for categorical operations.

[9] An additive category is an Ab-category with finite direct sums.

1. If R is a computable ring and mat_R the category of matrices over R, then $\mathcal{A}(\mathrm{mat}_R) \simeq R\text{-fpmod}$, and we have recovered our first model for finitely presented modules.

2. If S is a G-graded computable ring for an additively written abelian group G, then we let grmat_S denote the category of graded matrices[10] over S. Now, $\mathcal{A}(\mathrm{grmat}_S)$ is equivalent to the category of finitely presented graded S-modules.

3. The iterated construction $\mathcal{A}(\mathcal{A}(\mathrm{mat}_R)^{\mathrm{op}})$ is equivalent to the functor category generated by all finitely presented functors from R-fpmod to the category of abelian groups. In the case where R is computable and commutative, Tor- and Ext-functors are finitely presented and we can calculate the sets of natural transformations between any pair of such functors [8].

We see that category constructors can take massive advantage of reusing code for building up quite different mathematical entities.

Enhancing CAP

GAP programmers who want to implement their own computable category can make use of CAP's helpful derivation mechanism: if you install only a subset of the available categorical primitives, CAP will try to install as much of the remaining primitives as it can. For example, CAP knows the usual textbook derivation of pullbacks from kernels and direct sums: the pullback object of $A \xrightarrow{\alpha} B \xleftarrow{\gamma} C$ is given (up to isomorphism) by

$$\mathrm{KernelObject}\Big(\begin{pmatrix}\alpha\\-\gamma\end{pmatrix} : A \oplus C \longrightarrow B\Big).$$

CAP's current implementation of the computable model of R-fpmod (using single matrices as its data structure for the objects) was implemented by directly providing algorithms for 35 primitives, the total number of 181 available primitives are due to the derivation mechanism.

The future of CAP

We plan to extend our arsenal of category constructors within the realm of additive categories, dg categories, A_∞-categories, monoidal categories, and toposes. Moreover, we are interested in the study and effective application of type theory for our categorical approach to computer algebra.

We believe that computer algebra systems can greatly profit from a categorical organization both in their expressiveness and computational power. It is our goal to explore and reveal with CAP as many of these profits as possible.

References

[1] Mohamed Barakat and Markus Lange-Hegermann. An axiomatic setup for algorithmic homological algebra and an alternative approach to localization. *J. Algebra Appl.*, 10(2):269–293, 2011. (arXiv:1003.1943).

[2] Martin Bies. *Cohomologies of coherent sheaves and massless spectra in F-theory*. PhD thesis, Heidelberg U., 2018-02. (arXiv:1802.08860).

[3] Clement Dupont. *Periods of hyperplane arrangements and motivic coproduct*. PhD thesis, Université Pierre et Marie Curie - Paris VI, 2014. (https://tel.archives-ouvertes.fr/tel-01083524/document).

[4] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.9.1*, 2018. (http://www.gap-system.org).

[5] Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry. http://www.math.uiuc.edu/Macaulay2/.

[6] Sebastian Gutsche. *Constructive category theory and applications to algebraic geometry*. PhD thesis, University of Siegen, 2017. (http://dokumentix.ub.uni-siegen.de/opus/volltexte/2017/1241/).

[7] homalg project authors. The homalg project – Algorithmic Homological Algebra. (http://homalg-project.github.io), 2003–2017.

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

[9] Sebastian Posur. *Constructive Category Theory and Applications to Equivariant Sheaves*. PhD thesis, University of Siegen, 2017. (http://dokumentix.ub.uni-siegen.de/opus/volltexte/2017/1179/).

[10] Sebastian Posur. Constructing equivariant vector bundles via the bgg correspondence. *Journal of Symbolic Computation*, 91:57 – 73, 2019. MEGA 2017, Effective Methods in Algebraic Geometry, Nice (France), June 12-16, 2017.

[11] The QPA-team. *QPA – Quivers and path algebras*, 2012. (http://www.math.ntnu.no/~oyvinso/QPA/).

[10] An object in grmat_S is a finite list of elements in G, and a morphism between two such lists $(a_i)_i \in G^m$ and $(b_j)_j \in G^n$ for $m, n \in \mathbb{N}_0$ is a matrix $(s_{ij})_{ij} \in S^{m\times n}$ whose entries s_{ij} are either zero or homogeneous of degree $b_j - a_i$. Composition is given by matrix multiplication (with swapped arguments).