On the Syntax and Semantics of CAP

Sebastian Gutsche
Department of Mathematics
University of Siegen
sebastian.gutsche@uni-siegen.de

Sebastian Posur
Department of Mathematics
University of Siegen
sebastian.posur@uni-siegen.de

Øystein Skartsæterhagen
Department of Mathematical Sciences
Norwegian University of Science and Technology
oystein.skartsaterhagen@ntnu.no

Abstract

The CAP project (Categories, Algorithms, Programming) is a framework for implementing and computing with constructive categories. In this paper we explain the syntax and semantics of CAP by means of an example: the implementation of cokernels in the category of finitely presented modules. Although this example is quite simple, it reveals the necessary usage of dependent types for an appropriate specification of categorical constructions as well as the requirement to model homomorphisms as setoids rather than sets in our constructive framework for category theory.

1 Introduction

Category theory, a fundamental branch in mathematics, has two remarkable features that make it a valuable asset for computer algebra. First, as a meta theory of mathematical contexts, it provides a highly abstract and widely accepted language revealing links between seemingly different mathematical worlds. Second, despite of their abstractness, categorical theorems and constructions are often inherently algorithmic: many proofs in category theory claiming the existence of a mathematical object actually present a (sometimes hidden) way for its construction. Making the constructive aspects of category theory accessible on the computer is the central motivation of the on-going CAP project [GSP18] (the name CAP is an acronym for *Categories, Algorithms, Programming*).

The CAP project is a collection of software packages for category theory implemented in the computer algebra system `GAP` [GAP18]. Its purpose is to facilitate the implementation of concrete instances of categories, generic categorical algorithms, and category constructors, i.e., operations that create new categories out of given input categories. CAP's core system provides templates for categories possessing or equipped with extra structure, e.g., direct products, addition for morphisms, kernels and cokernels, tensor products, et cetera. On the one hand, these templates can be used to create instances of categories, for example the category of finite dimensional vector spaces over a computable field k or the category of finitely presented modules over a computable ring R (see [BLH11] for a definition of computable fields and rings). On the other hand, these templates provide the

In: O. Hasan, M. Pfeiffer, G. D. Reis (eds.): Proceedings of the Workshop Computer Algebra in the Age of Types, Hagenberg, Austria, 17-Aug-2018, published at http://ceur-ws.org

syntax for implementing generic categorical algorithms, such as the computation of specific differentials on a page of a spectral sequence in the context of an arbitrary abelian category.

The purpose of this paper is to explain the syntax and semantics of these templates by means of an example: we look at the categorical construction of a *cokernel* in a model for the category of finitely presented modules over a computable ring R. We will see that a convenient way to express the interdependencies in the specifications of a cokernel can be conveniently addressed by the usage of dependent types (Section 2). Moreover, modeling finitely presented modules in a constructive way will demonstrate the requirement to model homomorphisms as setoids rather than sets (Section 3).

The development of CAP started in December 2013. So far, four CAP related software packages[1] are distributed via the current `GAP` release[2], more packages still under development are available on the GitHub page[3] of the CAP project. Even more packages developed for CAP can be found on the GitHub page[4] of the `homalg`-project [hom17], these packages are usually marked with the suffix `ForCAP`. For a deeper discussion of CAP and its functionalities we refer the reader to [Gut17, Pos17b].

2 Syntax

CAP supports lots of important notions of category theory, which we also call *categorical constructions*. From a theoretical point of view, the specifications of a categorical construction may be expressed using dependent types. As a simple set-theoretic model for dependent functions and types in this paper, we will use the following definition.

Definition 2.1. Let A be a set and let $(B_a)_{a\in A}$ be an A-indexed family of sets. Then we denote the set of all sections of the natural projection $\uplus_{a\in A} B_a \to A$ by

$$\prod_{a\in A} B_a := \{\sigma : A \to \uplus_{a\in A} B_a \mid \sigma(a) \in B_a\}.$$

An element $\sigma \in \prod_{a\in A} B_a$ is called a **dependent function** of **dependent type** (or simply of **type**) $\prod_{a\in A} B_a$.

As an example of a categorical construction and its specifications, we will discuss the notion of a *cokernel*. Note that a cokernel can be defined in the context of a category enriched over abelian groups, i.e., its homomorphism sets are abelian groups, and composition of morphisms distributes over addition.

Definition 2.2. Let $\mathbf{A}$ be a category enriched over abelian groups. Given objects $A, B \in \mathbf{A}$ and a morphism $\phi \in \mathrm{Hom}_{\mathbf{A}}(A, B)$, a **cokernel of** ϕ consists of the following data:

1. An object $C \in \mathbf{A}$.
2. A morphism $\pi : B \to C$ such that $\pi \circ \phi = 0$.
3. A dependent function u mapping any pair (T, τ) consisting of an object $T \in \mathbf{A}$ and a morphism $\tau : B \to T$ such that $\tau \circ \phi = 0$ to a morphism $u(T, \tau) : C \to T$ which has to be uniquely determined by the property $\tau = u(T, \tau) \circ \pi$.

The category $\mathbf{A}$ **has cokernels** if it comes equipped with a dependent function mapping any morphism $\phi \in \mathrm{Hom}_{\mathbf{A}}(A, B)$ for $A, B \in \mathbf{A}$ to a cokernel (C, π, u) of ϕ.

CAP provides the following three primitives accessing the three components of the triple (C, π, u) for an additive category $\mathbf{A}$ having cokernels:

1. CokernelObject :
$$\mathrm{Hom}_{\mathbf{A}}(A, B) \to \mathrm{Obj}_{\mathbf{A}} : \phi \mapsto C.$$

[1] These packages are:

- `CAP` (the core system)
- `LinearAlgebraForCAP` (an implementation of the category of finite dimensional vector spaces)
- `ModulePresentationsForCAP` (an implementation of the category of finitely presented modules)
- `GeneralizedMorphismsForCAP` (an implementation of additive relations in abelian categories)

[2] Version 4.9.1, as of May 2018

[3] `https://github.com/homalg-project/CAP_project`

[4] `https://github.com/homalg-project`

2. CokernelProjection :
$$\prod_{\phi\in\mathrm{Hom}_{\mathbf{A}}(A,B)} \mathrm{Hom}_{\mathbf{A}}\left(B, \mathrm{CokernelObject}(\phi)\right) : \phi \mapsto \pi.$$

3. CokernelColift :
$$\prod_{\substack{\phi\in\mathrm{Hom}_{\mathbf{A}}(A,B)\\ \tau\in\{\sigma\in\mathrm{Hom}_{\mathbf{A}}(B,T)\mid\sigma\circ\phi=0\}}} \mathrm{Hom}_{\mathbf{A}}\left(\mathrm{CokernelObject}(\phi), T\right) : (\phi,\tau) \mapsto u(T,\tau).$$

We also wrote down the dependent types of these primitives for given objects A, B, T, in order to highlight their interdependencies. For example, the dependent type of the primitive CokernelProjection tells us that given a morphism $\phi : A \to B$, the output CokernelProjection(ϕ) will be a morphism $B \to \mathrm{CokernelObject}(\phi)$, i.e., a morphism with range depending on the primitive CokernelObject.

These three primitives suffice for building up other functionalities of the cokernel, e.g., its functoriality.

Example 2.3. Given a commutative diagram in $\mathbf{A}$ of the form
$$D := \quad \begin{array}{ccc} A & \xrightarrow{\alpha} & B \\ {\scriptstyle\nu}\downarrow & & \downarrow{\scriptstyle\mu} \\ A' & \xrightarrow{\alpha'} & B' \end{array}$$
the functoriality of the cokernel is given by the term
$$\mathrm{CokernelFunctorial}(D) := \mathrm{CokernelColift}\left(\alpha, \mathrm{CokernelProjection}(\alpha')\circ\mu\right). \qquad (\dagger)$$
$$\begin{array}{ccccc} A & \xrightarrow{\alpha} & B & \xrightarrow{\mathrm{CokernelProjection}(\alpha)} & \mathrm{CokernelObject}(\alpha) \\ {\scriptstyle\nu}\downarrow & & \downarrow{\scriptstyle\mu} & & \downarrow{\scriptstyle\mathrm{CokernelFunctorial}(D)} \\ A' & \xrightarrow{\alpha'} & B' & \xrightarrow[\mathrm{CokernelProjection}(\alpha')]{} & \mathrm{CokernelObject}(\alpha') \end{array}$$

The primitives for categorical constructions provided by CAP are powerful enough for a functorial implementation of a spectral sequence algorithm working in the context of an arbitrary abelian category. Such an algorithm takes as arguments a morphism of (linearly) descending filtered cochain complexes $F^\bullet A^\bullet \to F^\bullet B^\bullet$ and a triple of integers (r, p, q) where $r \geqslant 0$. The output is the (p, q)-th differential on the r-th page of the associated spectral sequence connected in a commutative diagram of the form
$$\begin{array}{ccc} {}_{F^\bullet A^\bullet}E_r^{pq} & \xrightarrow{\partial} & {}_{F^\bullet A^\bullet}E_r^{p+r,q-(r-1)} \\ \downarrow & & \downarrow \\ {}_{F^\bullet B^\bullet}E_r^{pq} & \xrightarrow{\partial} & {}_{F^\bullet B^\bullet}E_r^{p+r,q-(r-1)} \end{array}$$
induced by the functoriality of spectral sequences. To see how an implementation of such a high-level categorical construction can be realized with CAP's primitives see [Pos17b, Chapter 2].

3 Semantics

The purpose of CAP is to model categories. Classically, a set of objects $\mathrm{Obj}_{\mathbf{A}}$ and a *set* of morphisms $\mathrm{Hom}_{\mathbf{A}}(A, B)$ for all pairs $A, B \in \mathrm{Obj}_{\mathbf{A}}$ are part of the data defining a (small) category $\mathbf{A}$.

CAP models a slightly more general and computer-friendlier notion of a category: homomorphisms $\mathrm{Hom}_{\mathbf{A}}(A, B)$ are not only sets but *setoids*, i.e., a set equipped with an equivalence relation on it as an *extra datum*. The formal definition of this kind of category looks as follows:

Definition 3.1. A CAP **category A** consists of the following data:

1. A set $\mathrm{Obj}_{\mathbf{A}}$ of **objects**.
2. For every pair $A, B \in \mathrm{Obj}_{\mathbf{A}}$, a set $\mathrm{Hom}_{\mathbf{A}}(A, B)$ of **morphisms**. If two morphisms $\alpha, \beta \in \mathrm{Hom}_{\mathbf{A}}(A, B)$ are equal as elements of this set, we say they are ***equal***.
3. For every pair $A, B \in \mathrm{Obj}_{\mathbf{A}}$, an equivalence relation $\sim_{A,B}$ on $\mathrm{Hom}_{\mathbf{A}}(A, B)$. If $\alpha \sim_{A,B} \beta$ for two morphisms $\alpha, \beta \in \mathrm{Hom}_{\mathbf{A}}(A, B)$, we say they are ***congruent***.
4. For every $A \in \mathrm{Obj}_{\mathbf{A}}$, an **identity morphism** $\mathrm{id}_A \in \mathrm{Hom}_{\mathbf{A}}(A, A)$.
5. For every triple $A, B, C \in \mathrm{Obj}_{\mathbf{A}}$, a **composition function**

$$\circ : \mathrm{Hom}_{\mathbf{A}}(B, C) \times \mathrm{Hom}_{\mathbf{A}}(A, B) \to \mathrm{Hom}_{\mathbf{A}}(A, C)$$

   compatible with the equivalence relation, i.e., if $\alpha, \alpha' \in \mathrm{Hom}_{\mathbf{A}}(A, B)$, $\beta, \beta' \in \mathrm{Hom}_{\mathbf{A}}(B, C)$, $\alpha \sim_{A,B} \alpha'$ and $\beta \sim_{B,C} \beta'$, then $\beta \circ \alpha \sim_{A,C} \beta' \circ \alpha'$.
6. For all $A, B \in \mathrm{Obj}_{\mathbf{A}}$, $\alpha \in \mathrm{Hom}_{\mathbf{A}}(A, B)$, we have

$$(\mathrm{id}_B \circ \alpha) \sim_{A,B} \alpha$$

   and

$$\alpha \sim_{A,B} (\alpha \circ \mathrm{id}_A).$$

7. For all $A, B, C, D \in \mathrm{Obj}_{\mathbf{A}}$, $\alpha \in \mathrm{Hom}_{\mathbf{A}}(A, B)$, $\beta \in \mathrm{Hom}_{\mathbf{A}}(B, C)$, $\gamma \in \mathrm{Hom}_{\mathbf{A}}(C, D)$, we have

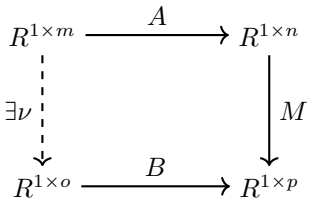
$$((\gamma \circ \beta) \circ \alpha) \sim_{A,D} (\gamma \circ (\beta \circ \alpha))$$

Remark 3.2. As it will be illustrated in Example 3.4, an implementation of a CAP category **A** does not need data structures for the *sets* $\mathrm{Obj}_{\mathbf{A}}$ or $\mathrm{Hom}_{\mathbf{A}}(A, B)$ for two $A, B \in \mathrm{Obj}_{\mathbf{A}}$, as they are not necessary to carry out the computations CAP is designed for. A proper implementation of a category **A** needs a data structure for the *elements* of these sets, i.e., for the objects $A \in \mathrm{Obj}_{\mathbf{A}}$ and for the morphisms $\phi \in \mathrm{Hom}_{\mathbf{A}}(A, B)$ for all objects $A, B \in \mathrm{Obj}_{\mathbf{A}}$. Note that the data structure for $\phi \in \mathrm{Hom}_{\mathbf{A}}(A, B)$ must be the same for all pairs of objects $A, B \in \mathrm{Obj}_{\mathbf{A}}$.

Remark 3.3. In terms of higher category theory, a CAP category is a 2-category such that the 2-morphism sets are either empty or a singleton, and such that its underlying object class is a set. Using this point of view, we can derive the notion of a functor between CAP categories: a CAP functor consists of an object and a morphism function such that the usual axioms of a functor hold up to *congruence*.

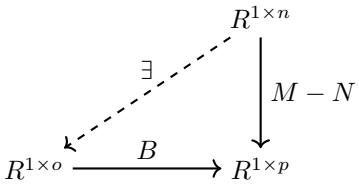
Given a CAP category **A**, passing to the quotient sets $\mathrm{Hom}_{\mathbf{A}}(A, B)/\sim_{A,B}$ gives rise to a classical category $\overline{\mathbf{A}}$, because all constructions and axioms respect the congruence for morphisms. It is usually the case that we actually want to compute with $\overline{\mathbf{A}}$, but that it is easier to implement a CAP category **A** giving rise to $\overline{\mathbf{A}}$. We demonstrate this principle by means of an example.

Example 3.4. Let R-fpmod be the category of finitely presented left R-modules for a computable ring R. We are going to model R-fpmod by a CAP category R-fpres. We define $\mathrm{Obj}_{R\text{-fpres}}$ as the set of all matrices with entries in R. Note that each such matrix $A \in R^{m \times n}$ can be interpreted as a homomorphism between free modules $R^{1\times m} \xrightarrow{A} R^{1\times n} \in R$-fpmod presenting its cokernel. For $A \in R^{m\times n}, B \in R^{o\times p}$, we define $\mathrm{Hom}_{R\text{-fpmod}}(A, B)$ as the set of matrices $M \in R^{n\times p}$ such that the following diagram can be completed to a commutative diagram by inserting a matrix ν on the left:



Note that by the functoriality of the cokernel, such a diagram induces a morphism between the modules presented by A and B independent of the choice of ν (since ν does not appear in the CAP term (Example 2.3, (†)) defining CokernelFunctorial). Conversely, every morphism in R-fpmod between the cokernels can be lifted to such a diagram since row modules are projective.

In our definition of the homomorphism sets, two morphisms $M, N \in \mathrm{Hom}_{R\text{-fpres}}(A, B)$ are *equal* if they are equal as matrices. We say M and N are *congruent* if and only if they induce equal morphisms between the modules presented by A and B, which is the case if and only if there exists a matrix rendering the diagram



commutative (this is a direct consequence of the comparison theorem [Wei94]).

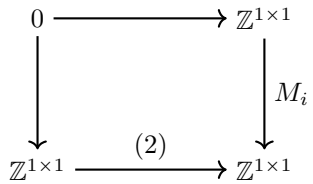
Thus, we equipped the homomorphism sets $\mathrm{Hom}_{R\text{-fpres}}(A, B)$ with an equivalence relation such that passing to the quotient yields a category $\overline{R\text{-fpres}}$ equivalent to R-fpmod. We can see the advantage of the model R-fpres over $\overline{R\text{-fpres}}$ when we start defining the *function*

$$\mathrm{CokernelObject} : \mathrm{Hom}_{R\text{-fpres}}(A, B) \to \mathrm{Obj}_{R\text{-fpres}}$$

for given $A, B \in R$-fpres. In the case of R-fpres, for $M \in R^{n\times p}$, we can simply set

$$\mathrm{CokernelObject}(M) := \begin{pmatrix} M \\ B \end{pmatrix}$$

which yields a function since *equal* input yields *equal* output. The same mapping rule in the context of $\overline{R\text{-fpres}}$ does not yield a function: For example, $M_1 = (0)$ and $M_2 = (2)$ both represent the same module homomorphism in



for $i = 1, 2$, but

$$\mathrm{CokernelObject}(M_1) = \begin{pmatrix} 0 \\ 2 \end{pmatrix} \neq \begin{pmatrix} 2 \\ 2 \end{pmatrix} = \mathrm{CokernelObject}(M_2)$$

on the level of matrices and thus on the level of objects in $\overline{R\text{-fpres}}$. This issue can be fixed by making (possibly unnatural) choices of representatives, but this can be very expensive in an actual implementation.

We further define

$$\mathrm{CokernelProjection}(M) := I_p$$

where I_p denotes the $p \times p$ identity matrix, and

$$\mathrm{CokernelColift}(M, T) := T$$

which are dependent functions of the correct types for our model R-fpres.

The following interpretation underlines the naturality of our model R-fpres: not only is CokernelObject a function in the context of R-fpres, but actually a functor between CAP categories. This can be made precise as follows: $\mathrm{Hom}_{R\text{-fpres}}(A, B)$ equipped with its equivalence relation can be seen as a category, where there is a morphism from M to N if and only if $M \sim N$. Furthermore, every category trivially can be turned into a CAP category, so $\mathrm{Hom}_{R\text{-fpres}}(A, B)$ is also a CAP category. The primitive CokernelObject can now be regarded as a CAP functor

$$\mathrm{CokernelObject} : \mathrm{Hom}_{R\text{-fpres}}(A, B) \to R\text{-fpres}$$

whose action on morphisms $\mathrm{CokernelObject}(M \sim M')$ is given by

$$\text{CokernelObject}(M) \xrightarrow{\text{CokernelColift}(M,\,\text{CokernelProjection}(M'))} \text{CokernelObject}(M').$$

It is well-defined since it respects composition and identities up to congruence and thus defines a CAP functor.

Problems similar to the issues with the cokernel arise when we want to deal with other categorical constructions, like kernels, pullbacks, or pushout, and the CAP category R-fpres provides a natural solution for all them.

Remark 3.5. In Example 3.4 we described concrete constructions for the three primitives

$$\text{CokernelObject}, \text{CokernelProjection}, \text{CokernelColift}$$

in a particular model of the category of finitely presented modules over a computable ring R. As we have seen in Example 2.3, from these primitives we can derive a generic algorithm for the primitive CokernelFunctorial. The core system of CAP offers various such automatic derivations, which come in handy in the implementation of concrete instances of categories. However, note that whenever performance is crucial, it is wise to substitute a primitive with a faster non-generic algorithm that might take advantage of attributes specific to the computational model in question.

Remark 3.6. The category of finitely presented modules can be seen as a special instance of the so-called Freyd category, which first appeared in [Fre66]. For a given additive category $\mathbf{P}$, its Freyd category $\mathcal{A}(\mathbf{P})$ can be constructed as a certain quotient category of the category of arrows in $\mathbf{P}$. The process of forming the Freyd category can be seen as a category constructor, it takes an additive category as input and constructs an additive category with cokernels as output:

$$\text{additive categories} \xrightarrow{\mathcal{A}(-)} \text{additive categories with cokernels}$$

If we apply this category constructor to Rows_R, i.e., the full subcategory of left R-modules generated by row modules $R^{1\times n}$ for $n \in \mathbb{N}_0$, we get the CAP category of finitely presented modules as output as it is modeled in Example 3.4.

The power of this abstraction lies in the fact that we can apply the category constructor $\mathcal{A}(-)$ not only to Rows_R, but to any additive category, in particular to $\mathcal{A}(\text{Rows}_R)$ itself. One can show that $\mathcal{A}(\mathcal{A}(\text{Rows}_R))$ is equivalent to the category of contravariant finitely presented functors on R-fpmod, i.e., contravariant functors mapping from R-fpmod to the category of abelian groups that arise as cokernels of natural transformations between representable functors. Thus, a proper implementation of the category constructor $\mathcal{A}(-)$ in CAP enables us to work computationally with such functors. For a detailed discussion of the constructive aspects of Freyd categories we refer the reader to [Pos17a].

Acknowledgements

Sebastian Gutsche and Sebastian Posur are supported by Deutsche Forschungsgemeinschaft (DFG) grant SFB-TRR 195: *Symbolic Tools in Mathematics and their Application.*

References

[BLH11] 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)

[Fre66] Peter Freyd, *Representations in abelian categories*, Proc. Conf. Categorical Algebra (La Jolla, Calif., 1965), Springer, New York, 1966, pp. 95–120. MR 0209333

[GAP18] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.9.1*, 2018, (`http://www.gap-system.org`).

[GSP18] Sebastian Gutsche, Øystein Skartsæterhagen, and Sebastian Posur, *The* `CAP` *project – Categories, Algorithms, Programming*, (`http://homalg-project.github.io/CAP_project`), 2013–2018.

[Gut17] Sebastian Gutsche, *Constructive category theory and applications to algebraic geometry*, Ph.D. thesis, University of Siegen, 2017, (`http://dokumentix.ub.uni-siegen.de/opus/volltexte/2017/1241/`).

[hom17] homalg project authors, *The* `homalg` *project – Algorithmic Homological Algebra*, (`http://homalg-project.github.io`), 2003–2017.

[Pos17a] Sebastian Posur, *A constructive approach to Freyd categories*, ArXiv e-prints (2017), (`arXiv:1712.03492`).

[Pos17b] Sebastian Posur, *Constructive category theory and applications to equivariant sheaves*, Ph.D. thesis, University of Siegen, 2017, (`http://dokumentix.ub.uni-siegen.de/opus/volltexte/2017/1179/`).

[Wei94] Charles A. Weibel, *An introduction to homological algebra*, Cambridge Studies in Advanced Mathematics, Cambridge University Press, 1994. MR MR1269324 (95f:18001)