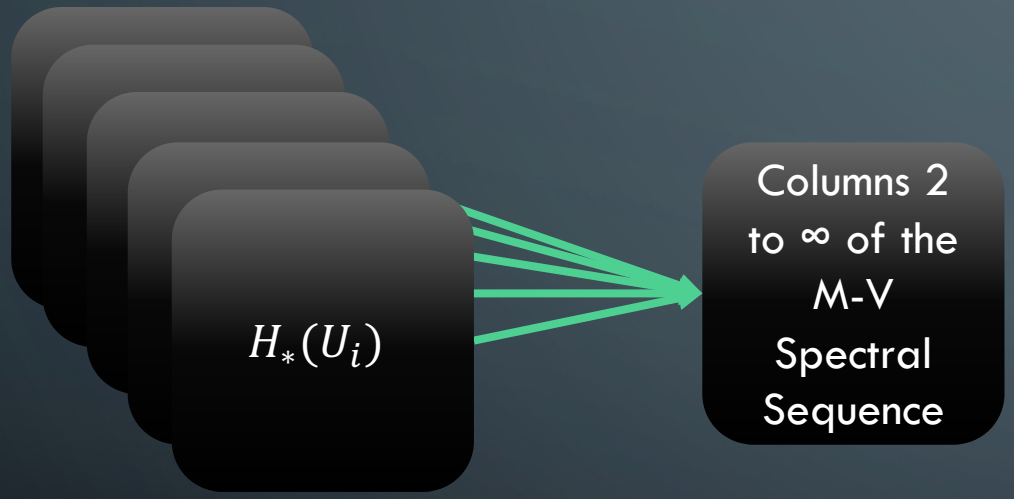
Co-Presentations of Persistence Modules

Mikael Vejdemo-Johansson (CUNY, visiting TU München)

Primoz Skraba (Queen Mary University of London)

We are trying to make the Mayer-Vietoris Spectral Sequence performant

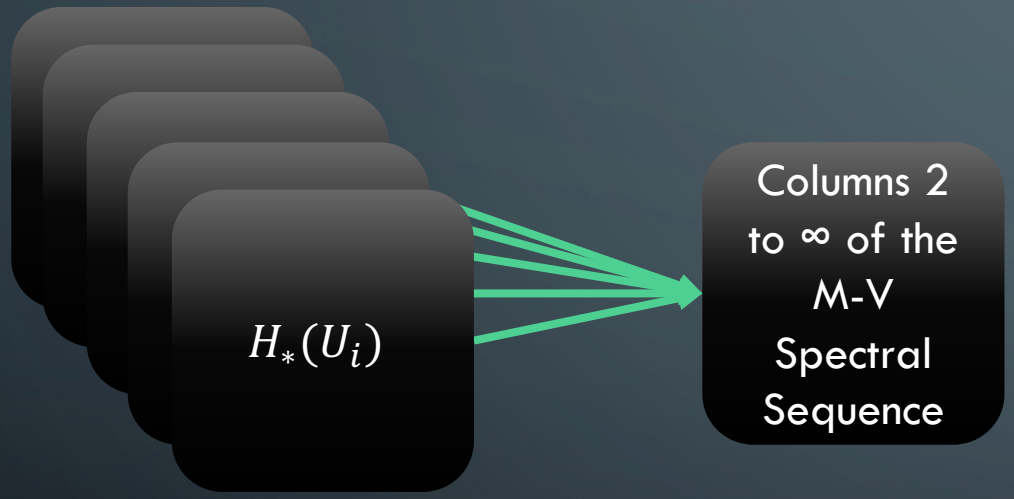


Key requirements:

Avoid computing in $\bigoplus H_*(U_i)$ if at all possible.

If we have to work in $\bigoplus H_*(U_i)$, make it very simple.

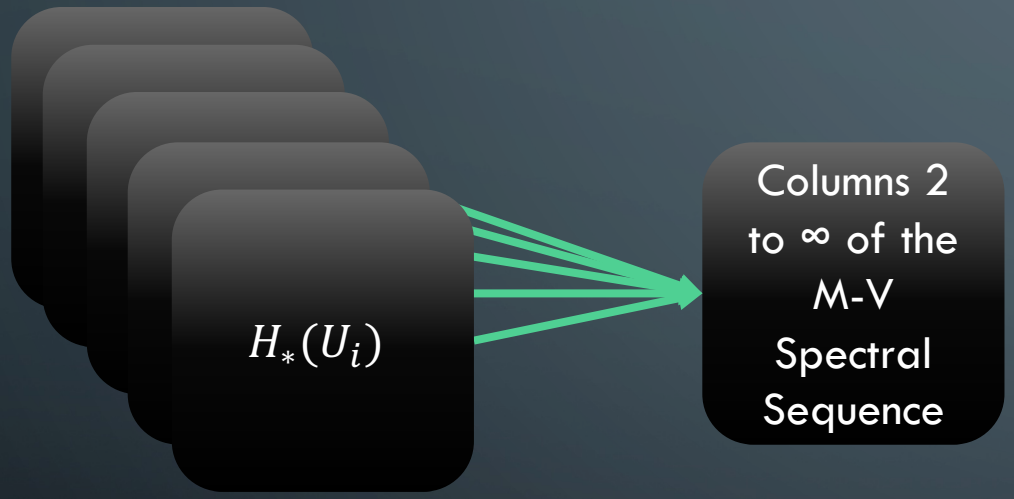
We are trying to make the Mayer-Vietoris Spectral Sequence performant



Key requirements:
Avoid computing in $\bigoplus H_*(U_i)$ if at all possible.
If we have to work in $\bigoplus H_*(U_i)$, make it very simple.

M-V S-S in homology: everything flows to the column with $\bigoplus H_*(U_i)$. Homology is a cokernel of a kernel, kernel is expensive to compute with presentations, and cokernel is a quotient of the target space of the map.

We are trying to make the Mayer-Vietoris Spectral Sequence performant



Key requirements:
Avoid computing in $\bigoplus H_*(U_i)$ if at all possible.
If we have to work in $\bigoplus H_*(U_i)$, make it very simple.

M-V S-S in homology: everything flows to the column with $\bigoplus H_*(U_i)$. Homology is a cokernel of a kernel, kernel is expensive to compute with presentations, and cokernel is a quotient of the target space of the map.

Our idea (coming soon to a preprint server near you) is to work with the **cohomology** spectral sequence, where computations flow away from $\bigoplus H^*(U_i)$, and the only operation that deals with all of $\bigoplus H^*(U_i)$ is one kernel computation.

Making Kernel Computations Simpler…

Moving to cohomology dualizes everything.

…why not dualize our representation of the persistence module itself?

We know (e.g. Škraba – Vejdemo-Johansson, arXiv:1302.2015) that with finitely presented persistence, cokernels are easy (just extend the presentation), kernels are difficult (repeated computations of free kernels).

Co-presentations turn out to provide the dual situation: **kernels** are **easy**, and **cokernels** are **difficult**.

What do we mean by Persistence Module?

The field seems to be converging towards this:

> A **Persistence Module** is a **representation** of the total order of $\mathbb{R}$;
>
> where a representation over a ring R of a category C is a functor $C \rightarrow R\text{-Mod}$.
>
> In other words, for every real value x, we have an R-module M_x, and whenever $x < y$, we have an R-linear map $M_x \rightarrow M_y$.

These representations behave a lot like modules, and can be viewed as modules over the **path algebra** $R[\mathbb{R},<]$.

What do we mean by Persistence Module?

From here, we develop familiar constructions from module theory:

- A sufficiently non-degenerate persistence module **decomposes** into a direct sum of **interval modules**.
- An interval module is determined by its start- and end-points.
- A persistence module is
  - Projective (free) if all interval modules are on the form $[a,\infty)$.
  - Injective if all interval modules are on the form $(-\infty,a]$ or $(-\infty,a)$.
- All sufficiently non-degenerate finitely generated persistence modules have a **projective presentation**.

Projectively Presented Persistence Modules

A Projective Presentation of a persistence module M is a pair of projective modules G (generators) and R (relations) such that: $0 \to R \to G \to M \to 0$ is a short exact sequence.

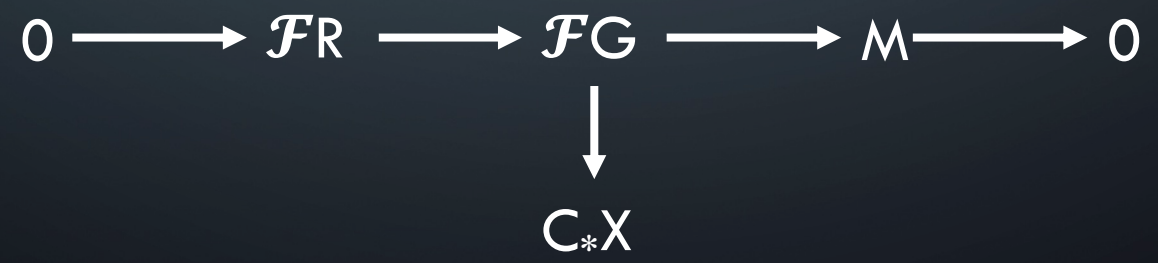
In other words, a presentation is a map between projective modules whose **cokernel** is the presented module.

Persistence Modules are nice enough that all (finitely generated) projective modules are free: we can take an $\mathbb{R}$-graded set S and a free persistence module construction $\mathcal{F}S$ to make a free module, where the degrees of elements in S give us the starting points of the interval modules.

Freely ~~Projectively~~ Presented Persistence Modules

Free modules are nice: R-linear maps between them can be given by matrices, once we have established a basis choice. We will often write finitely presented modules as $\langle G|R\rangle$,and discuss them in terms of their presentation matrices.

Very often we work with ambient (co)chain complexes and track representatives there. This fits in the presentation framework by adding a map from the generators. Thus a typical freely presented persistence module is given by a diagram:



Algebraic Operations with Freely Presented Modules

Through the following slides, we will present some commonly needed algebraic operations. All of these first establish a new presentation map as a block matrix, then row-reduce that presentation matrix, and finally prune any basis elements that end up unused by the new presentation matrix.

Algebraic Operations with Freely Presented Modules Direct Sum

Given two modules $\mathrm{M} = \langle G_M | R_M \rangle$ and $\mathrm{N} = \langle G_N | R_N \rangle$ with presentation maps ι_M, ι_N, the direct sum is given by:

$$0 \to \mathcal{F}(R_M \sqcup R_N) \xrightarrow{\begin{pmatrix} \iota_M & 0 \\ 0 & \iota_N \end{pmatrix}} \mathcal{F}(G_M \sqcup G_N) \to M \oplus N \to 0$$

Algebraic Operations with Freely Presented Modules Image

Given a map $M \to N$ represented by a matrix $f: \mathcal{F}G_M \to \mathcal{F}G_N$, the image is given by reducing the matrix f to a reduced matrix F. To get the relations in $\mathcal{F}R_N$ represented in a new presentation, we reduce the block matrix

$$\begin{pmatrix} F & \iota_N \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Algebraic Operations with Freely Presented Modules Cokernel

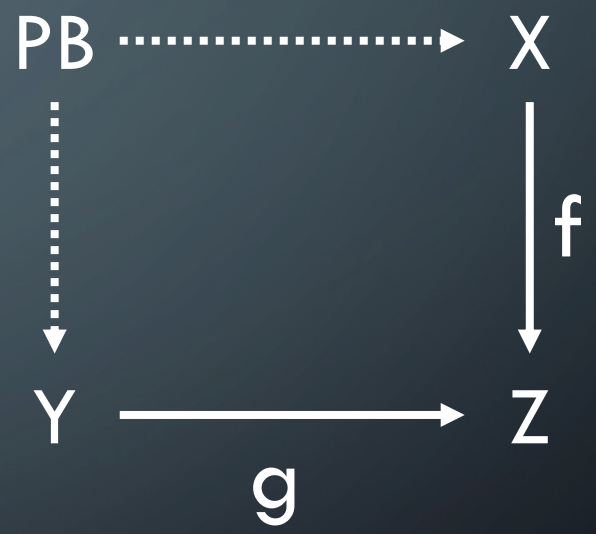
Given a map $M \to N$ represented by a matrix $f : \mathcal{F}G_M \to \mathcal{F}G_N$, we simply need to add the images of basis elements under f to the relations for N:

$$0 \to \mathcal{F}(\iota_N R_N \sqcup f G_M) \xrightarrow{(f \quad \iota_N)} \mathcal{F}G_N \to coker\ f \to 0$$

Reducing this presentation matrix prunes out redundancies in the presentation.

Algebraic Operations with Freely Presented Modules Pullbacks

Recall that a pullback completes the diagram to the right with universal choices for the maps $PB \to X$ and $PB \to Y$.

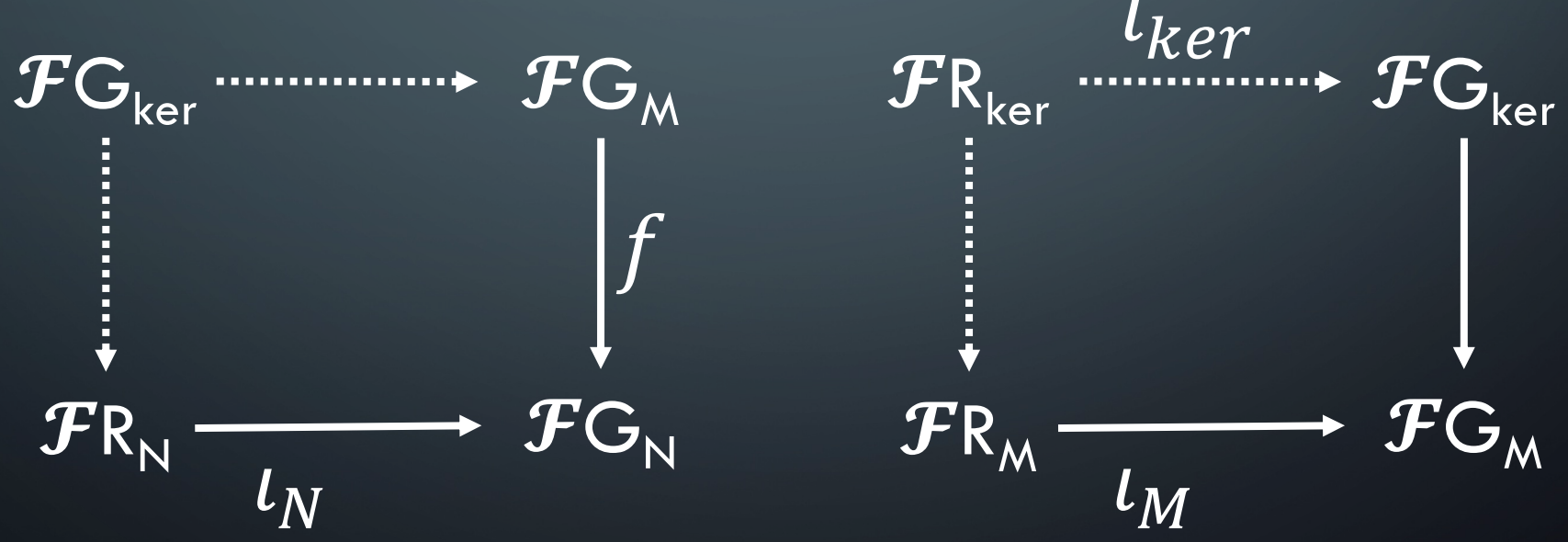


This is a kernel computation, but if X, Y, Z are free, so is PB, and the kernel computation can be done by reducing the matrix

$$\begin{pmatrix} f & g \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Algebraic Operations with Freely Presented Modules Kernel

Given a map $M \to N$ represented by a matrix $f : \mathcal{F}G_M \to \mathcal{F}G_N$, the kernel can be computed by a sequence of two pullbacks:



If we need to dualize, then let's dualize! Cohomology, large computations, co-presentations

In work for an upcoming preprint (and paper), we found ourselves switching to cohomology to reduce computational load – and realized some things were easier to express after dualizing more consistently than before.

In de Silva - Vejdemo-Johansson - Morozov (2009, 2010) on persistent cohomology, we argued for **reversing time** to get projectively presented cohomology modules.

Today, instead, we will describe **injective co-presentations**.

Injective building blocks: cofree modules

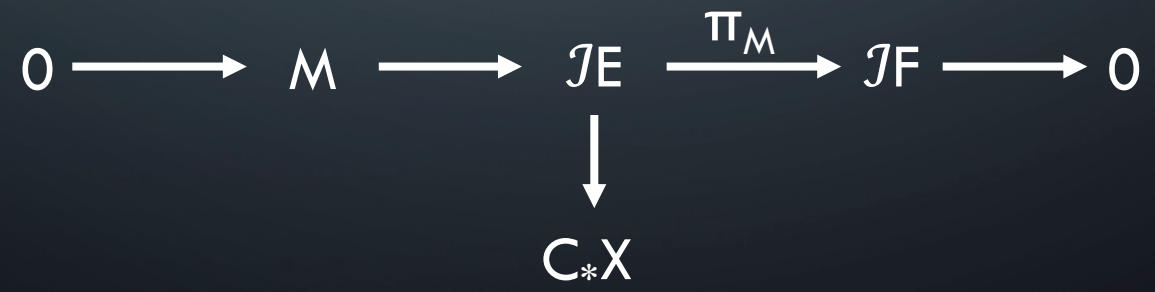
Following the recipe for a free module, we define a **cofree module** given by a graded set of generators to be the direct sum of interval modules $(-\infty, |a|]$ for generator set elements a in degree $|a|$. We will write $\mathcal{I}S$ for the cofree module on the generator set S.

A module M is **injectively presented** or **co-presented** if it fits in a short exact sequence, as the kernel of a map from co-generators E to co-relations F:

$$0 \to M \to \mathcal{I}E \to \mathcal{I}F \to 0$$

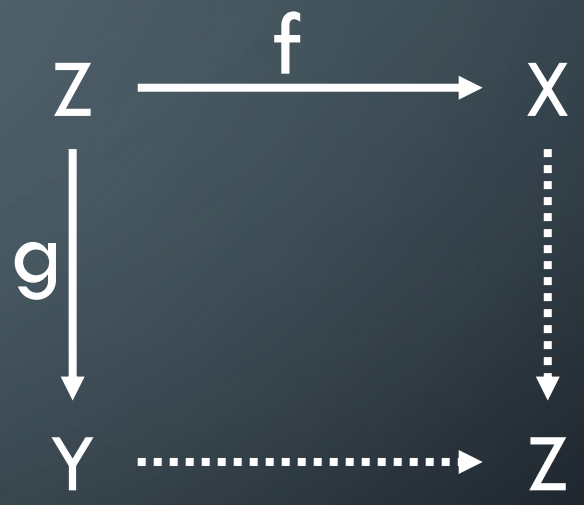
Co-Presented Persistence Modules

To account for an ambient (co)chain complex, the same construction with a representative map can be used for co-presentations:



Algebraic Operations with (Co)Freely Presented Modules
Pushouts

Recall that a pushout completes the diagram to the right with universal choices for the maps $X \to PO$ and $Y \to PO$.

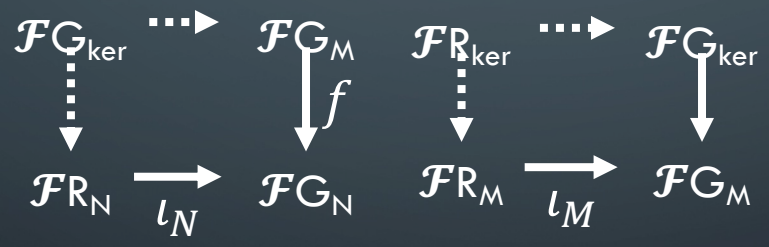
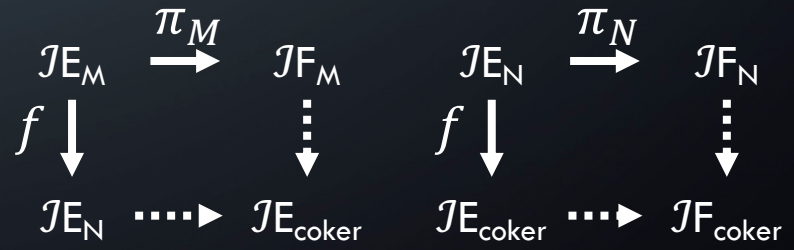


This is a co-kernel computation, but if X, Y, Z are co-free, so is PO, and the co-kernel computation can be done by reducing the matrix

$$\begin{pmatrix} f & 1 & 0 \\ g & 0 & 1 \end{pmatrix}$$

and thus compute the quotient module $X \oplus Y / f(z) - g(z)$

Algebraic Operations Everything All At Once

	Projective	Injective
Direct Sum	$\begin{pmatrix} \iota_M & 0 \\ 0 & \iota_N \end{pmatrix}$	$\begin{pmatrix} \pi_M & 0 \\ 0 & \pi_N \end{pmatrix}$
Image	$\begin{pmatrix} F & \iota_N \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\pi_N \circ F$
Kernel		$\begin{pmatrix} F \\ \pi_M \end{pmatrix}$
Cokernel	$(F \quad \iota_N)$	

Transferring between Presentations and Co-Presentations

To go between representations:

1. Split (co)generators into finite and infinite supported interval modules
2. Pair finite interval module generators with their corresponding (co)relation
3. Introduce new (co)generators at $\pm\infty$, paired with the (co)generators that were not paired

Passing through this construction twice gives isomorphic **modules** but not isomorphic **presentations**, since anything with an infinite lifetime gets a *virtual* endpoint at infinity before dualizing.

Thank you for your attention!!!

Coming soon to a preprint server near you:

- All these constructions, with examples.
- Asymptotic complexity for all the algebraic operations.
- The Mayer-Vietoris Spectral Sequence in cohomology, performant for parallelizing computations, and conditions for how to ensure performance.