

# Searching for New Relations among the Eilenberg-Zilber maps

Owen Abma | USRI Summer 2021

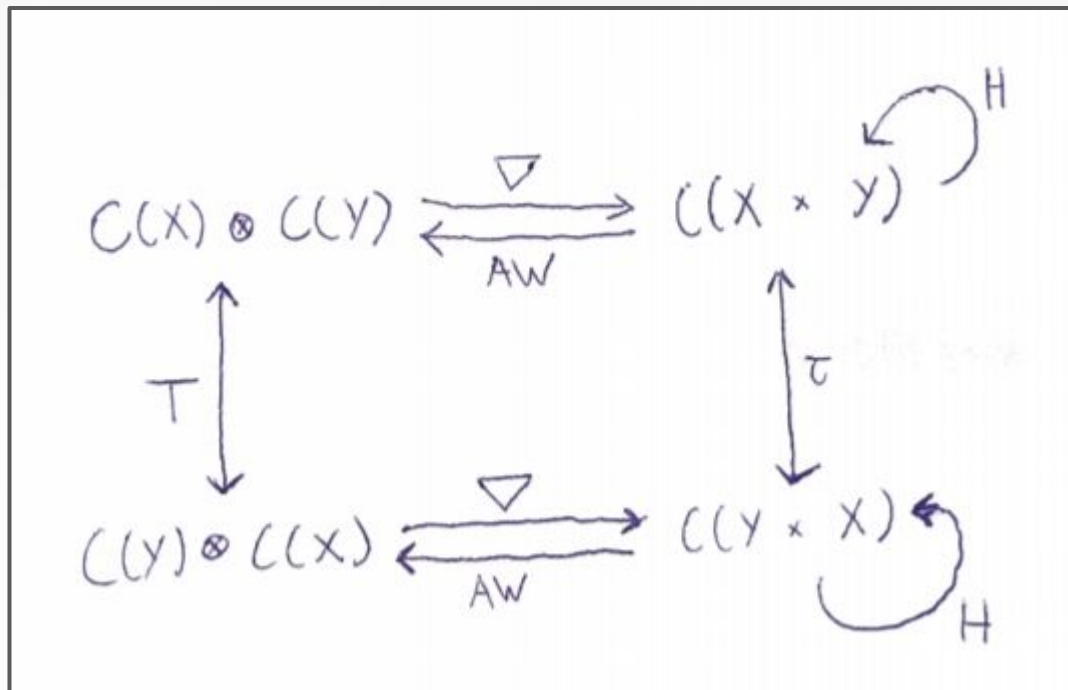
## Motivation

The primary goal of this research project was to write a **computer program** that would aid in the discovery of new mathematical relations among the **Eilenberg-Zilber maps**.

# The Eilenberg-Zilber Maps

The Eilenberg-Zilber Maps relate the chain complex of the product of two simplicial sets with the the tensor product of the chain complexes.

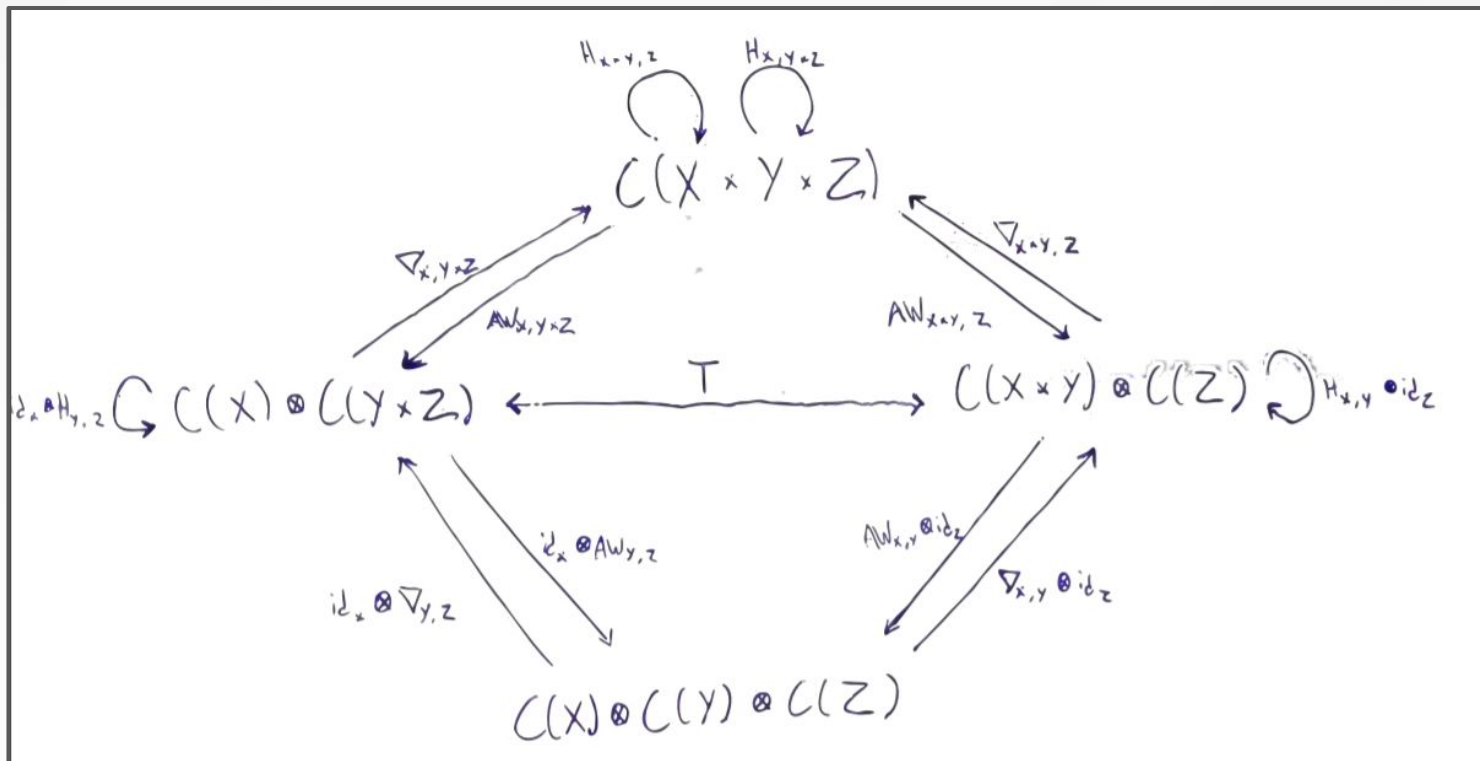
The maps interact with one another as shown on this diagram, with some known relations below.



$$(3.6) \quad AW \nabla = 1, \quad \nabla AW = 1 + d(H), \quad H \nabla = 0, \quad AW H = 0, \quad H H = 0.$$

# The Eilenberg-Zilber Maps

As more factors are added, these diagrams get more complex.



(this diagram doesn't even include the swap maps  $T$  and  $\tau$  - the full sized diagram would have 6 of these quadrilateral structures)

## The Program

The programming language being used for this project is Julia 1.6.1, a young yet powerful language that works very well in mathematical applications.

Julia has similar syntax to Python, which made it relatively easy to pick up and start using in a short amount of time.



# The Program

The first version of the program was used to solve for new relations in two factors.

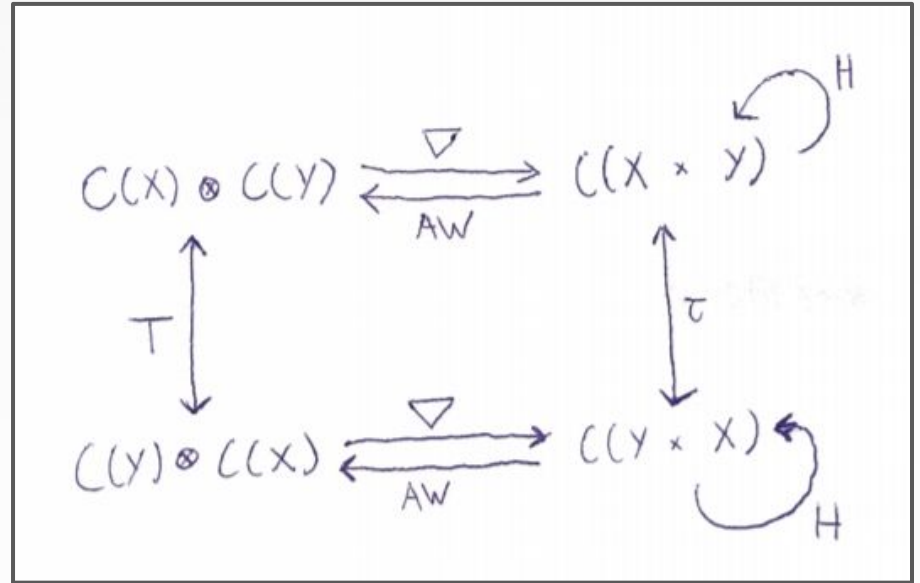
However, since solving for relations in two factors requires very little computation, it was unlikely that I would discover anything new.

The main point of this first version was to create a framework for how a larger program (for more than two factors) would work

# The Program - Main Idea

The process of this program is as follows:

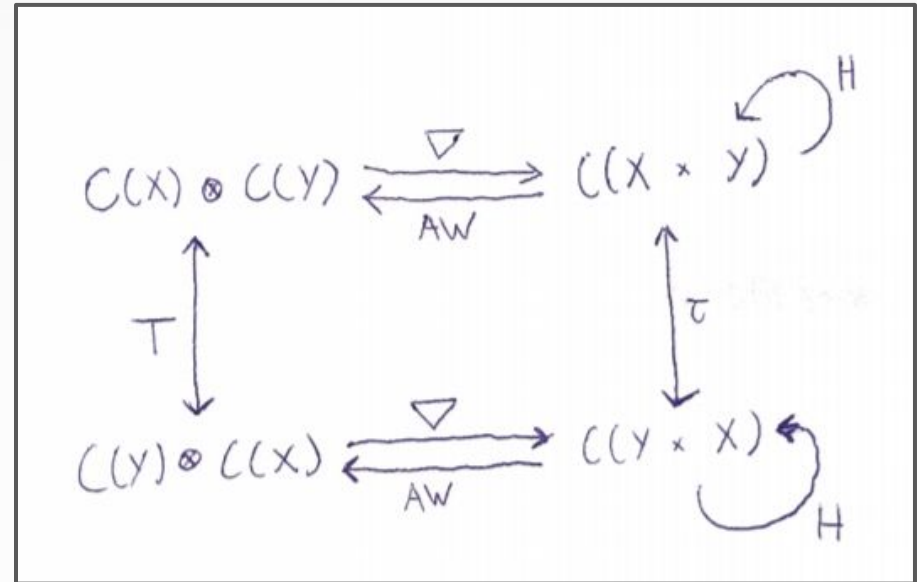
1. Generate all legal, nontrivial permutations of maps
2. Sort these permutations by domain, target, and dimension
3. For each category, put the resulting simplices into a matrix, and using basic linear algebra, solve for all linear dependencies



# The Program - Step by Step

## Step 1: Generate all legal, nontrivial permutations of maps

The “path” of a simplex (maps that have already been applied and their order) and the space it is in both affect which maps are allowed to be applied.



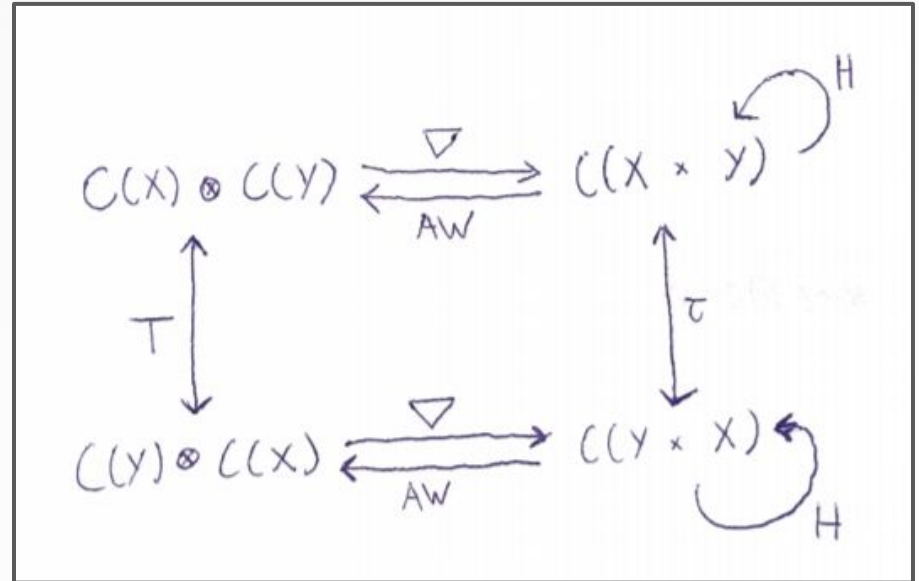


# The Program - Step by Step

## Step 1: Generate all legal, nontrivial permutations of maps

The program was written to avoid discovering

- Trivial relations (e.g. swapping twice is the identity)
- Illegal relations (e.g. AW cannot be applied twice in a row)
- Known properties (e.g. applying H twice yields zero)



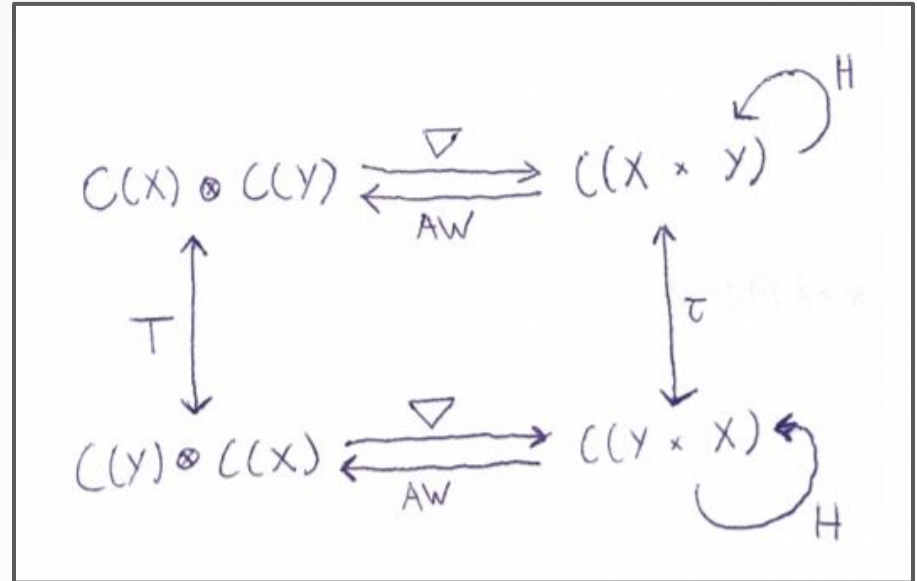
# The Program - Step by Step

## Step 2: Sort these permutations by domain, target, and dimension

By sorting these permutations of maps by:

- Domain (initial space)
- Target (final space)
- Dimension (number of times  $H$  is applied)

The calculations in step 3 were much easier for Julia to handle.

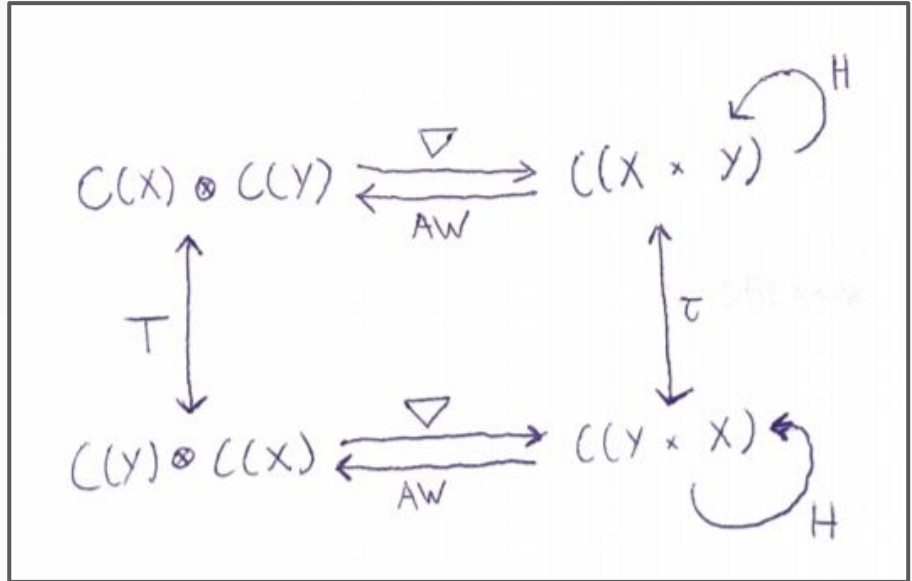


## The Program - Step by Step

**Step 3: For each category, put the resulting simplices into a matrix, and using basic linear algebra, solve for all linear dependencies**

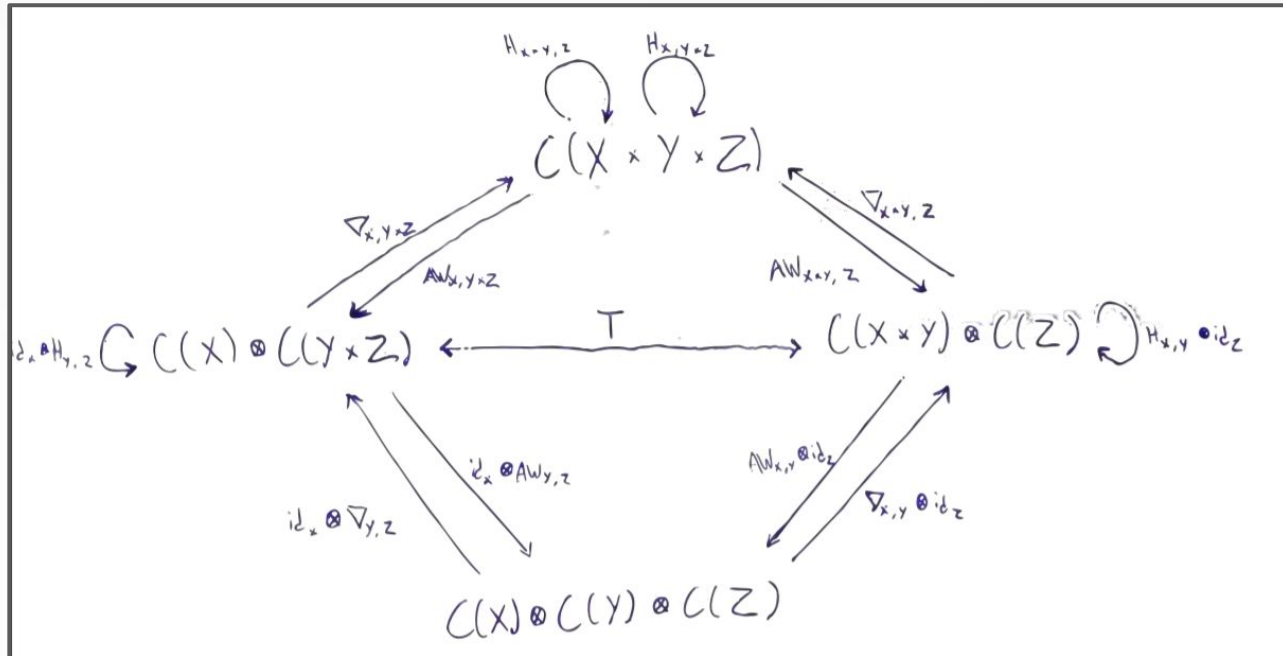
This was initially performed with a linear algebra package that I had coded myself.

If the program found a linear dependency, it was able to return the relation that was implied by this dependency.



## The Program - Moving Up



















Though no new relations were discovered in 2 factors, the program worked as it was supposed to, so it was scaled up to do the same process with three factors.



## The Program - Moving Up

Working in 3 factors is much more complex and took a significantly larger amount of work

As you can see, this process required many drafts until it was finally right.

<input type="checkbox"/>	 _2Variable Relation Solver.ipynb
<input type="checkbox"/>	 _3Variable Relation Solver.ipynb
<input type="checkbox"/>	 _3Variable Relation Solver_V2.ipynb
<input type="checkbox"/>	 _3Variable Relation Solver_V3.ipynb
<input type="checkbox"/>	 _3Variable_Relation_Solver_V4_with_Nemo.ipynb
<input type="checkbox"/>	 _3Variable_Relation_Solver_V5_with_Nemo.ipynb
<input type="checkbox"/>	 _Known Relations.ipynb
<input type="checkbox"/>	 _Learning Julia.ipynb
<input type="checkbox"/>	 _myfunctions.ipynb
<input type="checkbox"/>	 _myfunctions_nmo.ipynb
<input type="checkbox"/>	 _new_idea.ipynb
<input type="checkbox"/>	 _new_idea_v2.ipynb
<input type="checkbox"/>	 _Relations Practice.ipynb
<input type="checkbox"/>	 _Rough Draft - RelationGeneration with 2 variables.ipynb
<input type="checkbox"/>	 _Searching for New Relations-Copy1.ipynb
<input type="checkbox"/>	 _Searching for New Relations.ipynb
<input type="checkbox"/>	 _Simplicial Sets Testing.ipynb
<input type="checkbox"/>	 _solverfunctions_nmo.ipynb

# Challenges

The largest challenges that came with this project were:

1. Avoiding trivial relations
2. Locating and fixing inefficient code
3. Lack of computing power

# Challenges

## **Avoiding trivial relations**

Due to the complex definition of the Eilenberg-Zilber maps, it was very difficult to completely avoid trivial consequences, and often required a lot of tweaking to fix.

# Challenges

## **Locating and fixing inefficient code**

Although I was able to quickly pick up Julia's basic syntax, there were still many instances where I was writing inefficient code, resulting in a very slow program.

Over the course of the summer, I was able to improve greatly and quickly recognize any poor programming errors made.



# Challenges

## **Lack of computing power**

Unfortunately since I am using a laptop for this project, I am limited by the amount of power and memory it contains.

This became an issue towards the end of the summer, when the length of the permutations that I could generate developed a cap around 7.

# Results

Over the course of the summer, I have discovered two new relations, shown below.

$$\begin{aligned}
 & (HY \times Z, X \circ \tau_{X,Z} \circ \tau_{X,Y} \circ H_{X \times Y, Z} \circ H_{X, Y \times Z}) + (HY \times Z, X \circ H_{Y, Z \times X} \circ \tau_{X,Z} \circ \tau_{X,Y} \circ H_{X \times Y, Z}) \\
 & \quad = \\
 & (HY, Z \times X \circ \tau_{X,Z} \circ \tau_{X,Y} \circ H_{X \times Y, Z} \circ H_{X, Y \times Z}) + (HY \times Z, X \circ H_{Y, Z \times X} \circ \tau_{X,Z} \circ \tau_{X,Y} \circ H_{X, Y \times Z}) \\
 & \quad = \\
 & \quad \quad \quad AW_{Y \times Z, X} \circ \tau_{X,Z} \circ \tau_{X,Y} \circ H_{X \times Y, Z} \circ H_{X, Y \times Z} \\
 & \quad \quad \quad = \\
 & (AW_{Y \times Z, X} \circ \tau_{X,Z} \circ H_{Y \times X, Z} \circ H_{Y, X \times Z} \circ \tau_{X,Y}) + (AW_{Y \times Z, X} \circ H_{Y, Z \times X} \circ \tau_{X,Z} \circ \tau_{X,Y} \circ H_{X, Y \times Z})
 \end{aligned}$$

As you can see, these are much more complex than some of the known relations.

$$(3.6) \quad AW \nabla = 1, \quad \nabla AW = 1 + d(H), \quad H \nabla = 0, \quad AW H = 0, \quad H H = 0.$$

## Conclusion

This project was a fantastic opportunity for me to learn about programming and high-level mathematics and I am satisfied with my results, though I do believe that there are more relations that can be found with enough time and computing power.

Thank you for reading/listening!