

Catecon

The Categorical Console

Catecon.net

©2021 Harry Dole

1 Introduction

The Categorical Console, [Catecon](http://Catecon.net), is an artifice for drawing categorical diagrams. The user specifies a target category and places morphisms from that category into their current diagram. The user may then form composites, products, coproducts, lambdas, and as such depending on the target category. Users may assert in a diagram which of its cells are commutative, if not otherwise determined to be equal. Diagrams may refer to other diagrams and from other users' diagrams.

Morphisms from some categories, such as PFS (Partial Finite Sets) used for computation, JavaScript or C++ code can be emitted for computation.

In short, the primary notion is a morphism. All notions herein encode as morphisms.

This article shows how this works. As it happens, perhaps not everything quite actually works so this article provides the gist of it.

If configured, a local server communicates with its next level server ultimately leading to catecon.net.

The naming convention for elements is global and depends on your username, typically */user/diagram/element*. Choose your username wisely.

The initial starting view shows several diagrams. Click on one to view it or search by name. Return to the catalog view by clicking on **Catecon** on the title bar.

In your diagram, a referenced diagram is removable if nothing is used from the referenced diagram.

2 Features

2.1 User Views

Catecon provides two primary user views: One for querying and viewing available diagrams on the server, and a view for examining one or more diagrams. The former is the catalog view, and the latter is the diagram view. In the diagram view mode, the user may be single diagram view to edit their diagram (if owned) or in multi-diagram mode to see diagrams loaded into the current user session. The user session is saved to local storage so the user can restart in a new browser session with the prior setup.

2.1.1 Catalog View

The catalog view allows the user to browse the diagrams available from the server.

Search for diagrams by name using the search tool. Clicking the face icon shows only your own diagrams. The sort by icons sort by name, reference count, or last time edited.



Search for a Diagram: Figure 1

Create a new diagram by giving its base name, which is the name underneath your username in the global namespace. The proper name is that shown for display purposes. The description is a short one-liner describing the diagram. Lastly is a choice of category for the diagram and a button to create the diagram. Upon creation the user is taken to view that empty diagram as the current diagram.

New diagram			
<input type="text" value="Base name"/>	<input type="text" value="Proper name"/>	<input type="text" value="Description"/>	<input type="text" value="PFS"/>

Create a New Diagram: Figure 2

2.1.2 Diagram View

Use the Esc key to toggle back and forth between single diagram mode and multi-diagram mode.

2.1.2.1 Single Diagram Viewing Mode or Diagram Edit Mode

The full browser screen is used in single diagram mode to show the current diagram. In this mode the user may edit their diagram (if they own it). Pressing the home key shows the entirety of the current diagram. The mouse wheel zooms the user's view of the diagram in/out.

2.1.2.2 Multi-Diagram or User Session Mode

In multi-diagram mode the current diagram has a green border. Pressing the home key shows the entirety of all visible diagrams in the user's session. With the cursor not over a diagram, the shift-mouse wheel zooms the entire view. With the cursor over a diagram, the shift-mouse wheel zooms that diagram within the session view.

The current diagram has a green border. To select a different diagram to be the current diagram double-click on it.

2.2 Category



The nLab definition is [here](#). A category combines morphisms and hence a substrate to build upon for diagrams. A diagram targets a category to form a view in that category.

Some categories are built into Catecon:

PFS: Partial Finite Sets. This category is used for computations in JavaScript and C++ among others. A TTY interface is available in the web browser along with a 3D viewer.

Cat: The category of smaller categories. **PFS** is an object of **Cat**.

CAT: The category of larger categories. **Cat** is an object of **CAT**.

2.3 Diagrams



The basic unit of thought is the diagram. In practical terms, you download diagrams to refer to, you draw your intent in the form of morphisms in a new diagram, and upload diagrams to share.

A diagram is formed from elements which are objects, morphisms, and text. The objects and morphisms are from the diagram's specified category (generally). The text is commentary only.

2.3.1 Objects



Objects are those upon which the morphisms act. One such action does nothing, and each object has an identity morphism as such.

2.3.2 Morphisms

→ *Morphisms* are everything and are drawn as some form of arrow. *Objects* serve to connect morphisms together so connecting sequences of morphisms form composites. Objects have their own assigned morphism in the form of an *identity*. Collections of morphisms with certain properties are known as *categories*. A morphism between categories is a *functor*. A *diagram* is a form of a functor. A morphism between functors is a *natural transformation*, and so on.

The categorical console allows for the construction of new morphisms in a categorical manner. If such a formed morphism contains morphisms assigned vernacular codes as Javascript or C++ then said morphism may be executed.

2.4 Cells

A *leg* is a sequence of connected morphisms (codomain to domain) in the diagram. Composites are formed from legs.

Two legs with the same first domain and last codomain form a *cell* in the diagram if it is minimal in the sense of containing no other cell.

A cell commutes if the composites formed by both legs are equal. Each cell in a diagram is marked in a manner as to whether the two paths can be determined to be equal or not.

The equality engine attempts to determine each cell's commutativity. The following symbols are the labels used by the equality engine.

2.4.1 Cell Definition

def A *cell definition* is a leg of length one defined to be other leg, typically a composite, or the named element of an object or morphism.

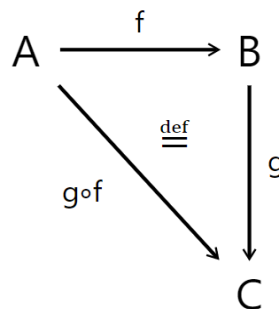


Figure 3

The morphism $g \circ f$ is the composite of the morphisms f and g .

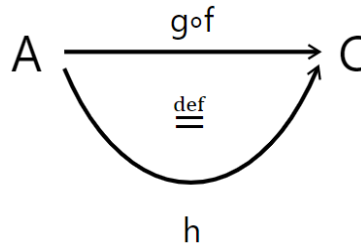


Figure 4

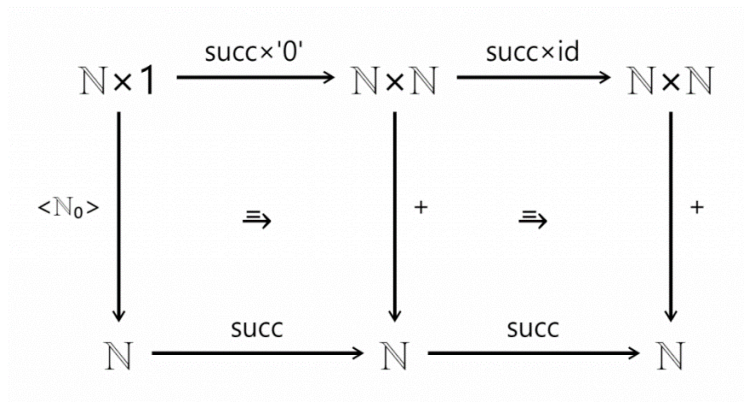
The morphism h is the named morphism for $g \circ f$, or $h = g \circ f$.

2.4.2 Assertion



When seen this means an *assertion* has been declared that the two legs of the cell are equal. The two legs are loaded into the equality engine as equal.


For example, here is the definition of addition on the natural numbers using two assertions:




Example of Two Assertions: Figure 5

2.4.3 Indeterminate



If the equality engine cannot determine if a cell commutes or not, then it is marked as *indeterminate*. Click on this and then click  to assert in the cell's diagram that the two legs are equal.

2.4.4 Commutative

 The two legs of the cell have been determined to be equal by the equality engine.


2.5 Navigation Bar (Navbar)




Navbar Buttons: Figure 6

The Navigation Bar is the topmost bar. It is gray if not logged in and black if logged in. It disappears after a while if the user is idle.


2.5.1 Access

 This panel allows login access to the server for uploading diagrams. When logged in it shows the username, email, and permissions on the server. The user may log out or reset the account's password.


2.5.2 Help

 Review terms and conditions, credits, contact information et al.

2.5.3 Settings

 Various settings, defaults and internal info are seen here.

2.5.4 TTY

 Shows the TTY output and error log.


Clear the section's TTY.

Download the file.

Download the error log.

The editing log is shown if the user owns the current diagram.

2.5.5 3D

 A 3D interface for the PFS category is built in and available from the 3D panel. The diagram hdoie/threeD provides morphisms that can update the 3D viewing area.


2.6 Toolbar

The toolbar is the primary means for interacting with elements in Catecon. Click on one or more elements to see which actions are available for your specified context.

For example, when the user clicks on nothing in the diagram view, the toolbar appears as the following:



Diagram Toolbar: Figure 7

 Allows the toolbar to be moved about the view.



 The toolbar is removed from the view. Clicking again on nothing also hides the toolbar.

2.7 Tools and Actions


Actions are the primary means of interacting with the categorical console. Depending on what has been selected, the toolbar shows what actions are available on that selected set. Pushing the button activates the action.

Many actions are editing commands, and the user must own the diagram to edit it.

2.7.1 Align Horizontally, Vertically

  Select two or more objects, and if out of alignment, align them horizontally or vertically.

2.7.2 Cell Tools

 Show the current cells in the diagram and their state (composite, computed, named, not equal, indeterminate). A given cell may be viewed by clicking on its view icon.

2.7.3 Composite



Select in sequence a leg of connected morphisms and hit the composite button:

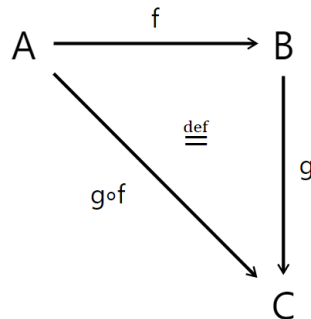


Figure 8

2.7.4 Copy Elements



Create a copy of the selected element and place it nearby.

2.7.5 Delete Elements



Remove an element from the diagram. If the element is used, that is its reference count is positive, then it cannot be deleted.

2.7.6 Diagram Tools

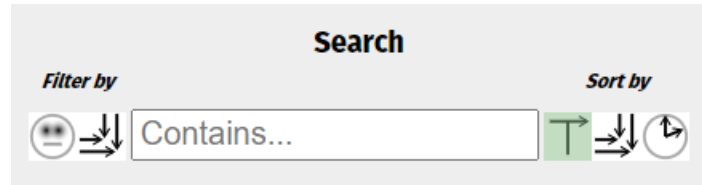


Clicking the diagram button on the toolbar shows the diagram search area by default and gives access to creating a new diagram or a copy of the current one, plus a means to upload a JSON file to load a diagram.



Secondary Tools for Diagrams: Figure 9


Shows the diagram search area:



Search for Diagrams: Figure 10

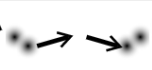
On the left side the user may filter the search results by what they own or by what are the reference diagrams for the current diagram. On the right side the found diagrams may be sorted by name, reference count, or timestamp.

- ✓ Show the section of the toolbar to create a new diagram that is either empty or a copy of the current one.

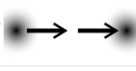
 Show the section of the toolbar to allow selecting a diagram JSON file to upload as a new **JSON** diagram. If the diagram is already loaded in the browser, it is then removed and replaced by the uploaded diagram. The user must own the diagram.

2.7.7 Distribute, De-Distribute

2.7.8 Detach Domain, Codomain

 Select a morphism and detach its domain or codomain if there are other morphisms connected to that domain or codomain.


2.7.9 Domain, Codomain Morphisms

 List morphisms that have the selected object as its domain or codomain. Clicking on a listed morphism places the morphism in the current diagram with the selected object as domain or codomain.

See *Fusing* under mouse actions for moving an object to fuse with another.

2.7.10 Evaluate Morphism

2.7.11 Factor, Cofactor Morphisms

 Projections, injections, deltas, folds, twists, permutations, and ilk are lumped into a construct known as [co]factor morphisms.

Create a factor morphism which basically means you can project out pieces of a product object, or you can make copies (delta), or add the terminal object $*$ to the codomain product.

Create a cofactor morphism which basically means you can inject pieces of a coproduct object, or you can make folds of the same object, or add the initial object $*$ to the domain coproduct.

For example, to project off the first term of a product select the 0th index.

2.7.12 Finite Object



Create a finite object of a determinate or indeterminate size.

2.7.13 Flip Morphism Name

A simple way to move a morphism's name from one side of the arrow to the other. Double-click on the morphism's name to activate.

2.7.14 Help



Select an element and produce a textual description. If allowed, other activities can be done such as changing the element's base name, proper name, or description. For morphisms the domain or codomain may be changed.

For a selected morphism and if allowed, the domain or codomain may be changed if the reference count is low.

The homset index refers to the curvature of the arrow between the two objects. The user may adjust this for a more pleasing layout.

2.7.14.1 Assertion Tools

If a diagram cell is selected, then its help section shows the assertion tools.



Set the cell to be commutative. The appropriate equivalences are loaded into the equality engine.



Set the cell to be non-commutative. The appropriate non-equivalences are loaded into the equality engine.



Hide the cell. There is still a small object visible that can be selected to change this status. The purpose is simply to ignore cells that the user does not want to see flagged as indeterminate.

2.7.14.2 Language Tools


If supported access to vernacular programming language tools may be shown.

2.7.14.2.1 Javascript

JS Show the Javascript tools for this morphism. Typically, this allows for a text editor to define the Javascript code for the morphism. The morphism may be marked as Javascript async, but that is not usual.

2.7.14.2.2 C++


C++ If possible, C++ code is shown for an object or morphism. If the selected element is complex, such as a product, composite, or what not, then the code is generated appropriately for those constructs. If an input factor to a generated morphism has a morphism from TTY, then that is automatically inserted into the C++ code. Thus, when compiled and run, the inputs are read from stdin. Similarly, those outputs with morphisms to TTY are automatically inserted and written to stdout as needed.

 If the selected element is a bare object or morphism, and the represented element belongs to the current diagram, then if permitted the user may edit C++ code representing the object or morphism. For example, the add function $\mathbb{F}_{64} \times \mathbb{F}_{64} \rightarrow \mathbb{F}_{64}$ may have C++ code assigned as:


```
%2 = %0 + %1;
```

The %0, %1, and %2 are appropriately substituted when the morphism is generated as C++ code.

A diagram itself may have code assigned to it that is inserted at the beginning of the generation process. This is where #include files may be stated so they are included ahead of the user's code. All reference diagrams are scanned to include why they require as well.

 The user may download the displayed C++ code as needed.

2.7.15 Home View

 View all the elements in the diagram. If in multi-diagram viewing mode, then view all diagrams in the user session.

2.7.16 Homset



Select two objects and see the homset, or all the available morphisms from the first to the second. Selecting one such places the morphism between those selected two objects.

Or create a new bare morphism between the two objects.

2.7.17 Hom



Form a hom object or hom morphism depending on what was selected, two objects or two morphisms.

2.7.18 Identity

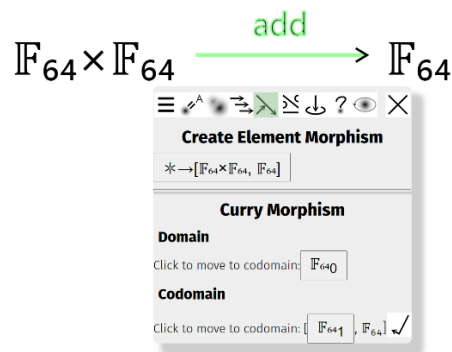
1

For a selected object in the diagram, click on this button to place in the diagram its assigned identity morphism.

2.7.19 Lambda



Form a lambda morphism from a single selected morphism by using a popup dialog. For example, for the following add function the second input factor is transferred to the codomain for the resulting morphism $\mathbb{F}_{64} \rightarrow [\mathbb{F}_{64}, \mathbb{F}_{64}]$. Clicking on one of the factor buttons in the domain or codomain portion of the dialog flips it back and forth between the codomain and domain.



Create Factor Morphism: Figure 11

At the top of the above dialog there is also the opportunity to create an element morphism $* \rightarrow [\mathbb{F}_{64} \times \mathbb{F}_{64}, \mathbb{F}_{64}]$ representing the selected morphism.

2.7.20 Morphism Assembly



A diagram may also be used to assemble a morphism. For example, a string of arrows connecting codomain to domain can be formed into a composite. Click on an object connected to morphisms and the morphism assembly button is available. See more details in the Morphism Assembly section.

2.7.21 Morphism Tools



A list of morphisms available in the diagram is displayed with a means of searching for them by name. Clicking on one of the listed morphisms places it on the diagram.

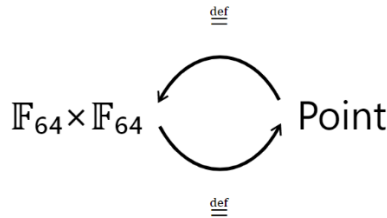
A new morphism may also be created by the user providing a unique basename, and optional proper name and description, and mandatory domain and codomain.

2.7.22 Named Element



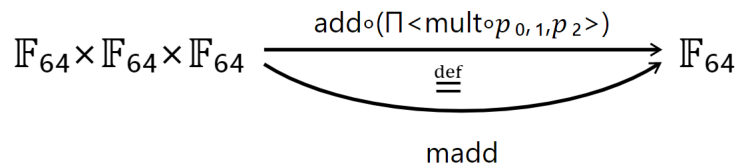
Create a named object or morphism from an element representing it.

For a named object, two identities are created between the two identical objects.



Named Object: Figure 12

For a named morphism, the named morphism is in the same hom-set and appropriate equivalences are loaded into the equality engine.



Named Morphism: Figure 13

The named elements of a diagram may be seen as its externally available elements.

2.7.23 Object Tools



A list of objects available in the diagram is displayed with a means of searching for them by name. Clicking on one of the listed objects places it on the diagram.

A new object may also be created by the user providing a unique basename, and optional proper name and description.

2.7.24 Product, Coproduct Assembly



To form a product assembly, select several morphisms with a common domain.
To form a coproduct assembly, select several morphisms with a common codomain.

In the following figure two selected morphisms have their product assembly shown as the vertical morphism:

$$\begin{array}{ccccc}
 \mathbb{F}_{64} & \xleftarrow{p_2} & \mathbb{F}_{64} \times \mathbb{F}_{64} \times \mathbb{F}_{64} & \xrightarrow{\text{mult} \circ p_{0,1}} & \mathbb{F}_{64} \\
 & & \downarrow \Pi \langle \text{mult} \circ p_{0,1}, p_2 \rangle & & \\
 & & \mathbb{F}_{64} \times \mathbb{F}_{64} & &
 \end{array}$$

Product Assembly: Figure 14

2.7.25 Product, Coproduct

\times $+$ Select two or more objects (only) or morphisms (only) and click to form the [co]product of those morphisms. The order is that given by the sequence of selecting the elements.

2.7.26 Recursion

Select a bare morphism and another morphism which uses it, then the second morphism can be set as the *recursor*, or recursive function, for the bare morphism.

2.7.27 [Co]Reference Morphism



Select one more projections or insertions and tag as references or coreferences with this button. Alternately, remove the reference tag by clicking again.

2.7.28 Run Morphism

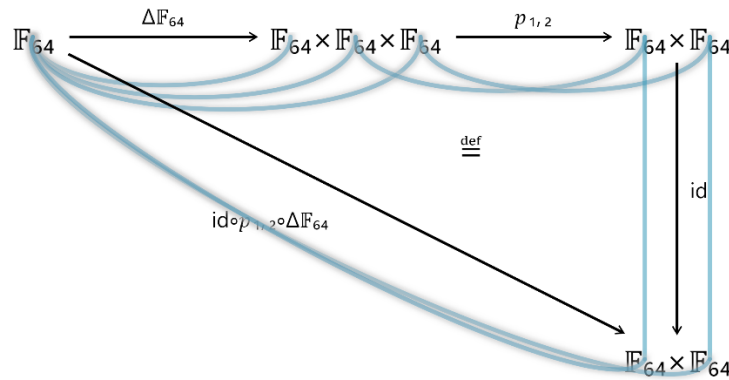


Create data on the object or morphism. There must be a handler for the selected object or codomain to form a datum. If an object is selected, then creating data means creating a new finite object as domain to a morphism with the selected object as codomain. If a morphism is selected, then the domain must be finite, and the codomain have an input handler. Products and coproducts are handled intrinsically. A datum for a hom object would be to pick a morphism from that homset.

2.7.29 String Graph



Show the selected morphism's string graph (if it has one). The graph action on a morphism produces the links (or strings) between the terms in the object expressions that have the same value (better: confluence). For example, an identity could have a solid blue bar connecting the domain and codomain. Connecting links have the same random color. A collection of connected links of the same color is like a variable in a vernacular language.



String Graph of Four Morphisms: Figure 15

2.7.30 Swap Morphism Domain with Codomain




If the underlying morphism's reference count is low, then its domain may be swapped with the codomain.



View the selected objects.

2.7.31 Text

 Show the text instantiated in the current diagram. The user may search the text for key words, delete the text, or zoom the view to the text. Additionally, the user may create new text at the location clicked on the diagram.

2.8 Morphism Assembly

A morphism may be assembled from a connected blob of arrows in a diagram by a process we called *form morphism*. First, we examine examples to see why and how to do this, then step into the basics and general form.

2.8.1 Examples

Products, coproducts, references, and homsets may be formed as well. In the following the terms describing the formation of an assembly are discussed. Note that in this process we talk of the arrows in diagram's index or domain category, as drawn on the screen. The morphisms in the diagram's codomain or target category are referred to as the *target* morphisms in this discussion.

2.8.1.1 Multiply-Add

Take the notion of a multiply-add operation of three input numbers. To form the final morphism takes basic actions that look like this resulting diagram:

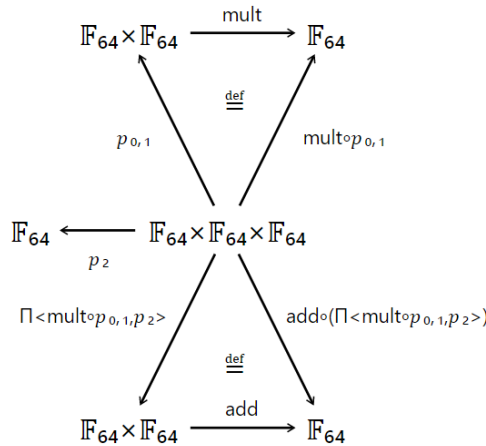
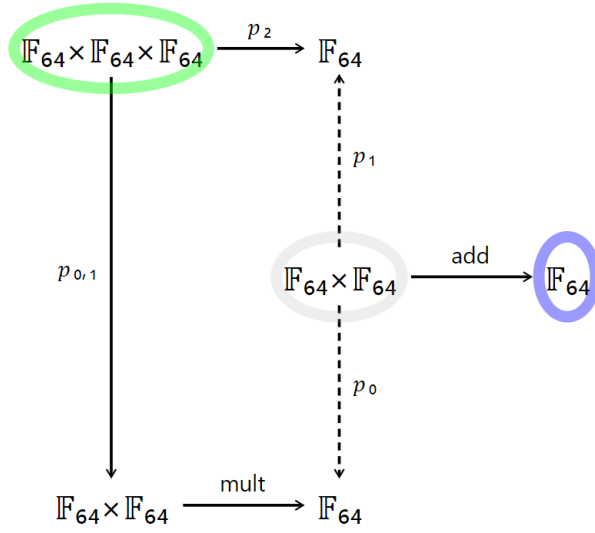


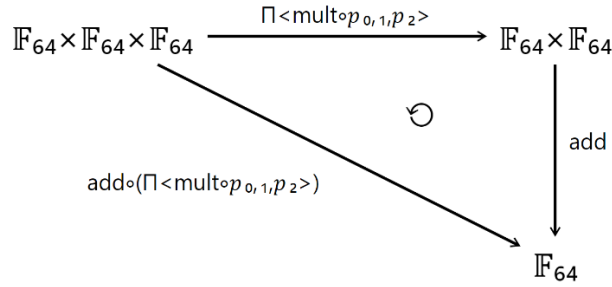
Figure 16

Alternately, the following figure shows a morphism assembly that forms the same final morphism:



Blob to Assemble Multiply/Add: Figure 17

The object circled in green is the input and the one in blue is the output. The two dashed projections p_0 and p_1 are references on how to build the arguments for the add. The two branches from the input form a product assembly. The final morphism is seen here:

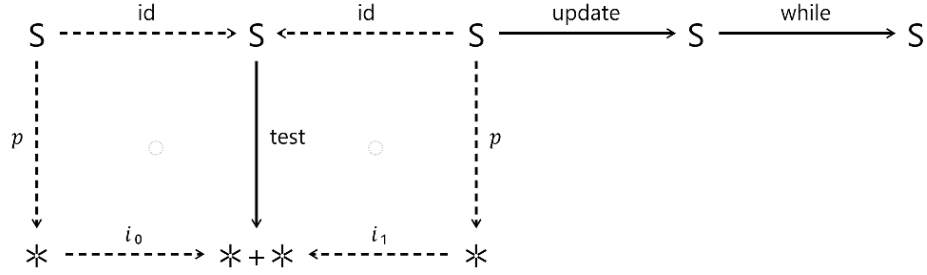


The Assembled Multiply/Add: Figure 18

Evidently the second construction is simpler and easier to understand.

2.8.1.2 While

The notion of ‘while’ in a programming language typically has a function test: $S \rightarrow *+*$ and a function update: $S \rightarrow S$ to run if the test is true. Assemble the ‘while’ of these two morphisms by using the test to decide which branch of the diagram to take. Once the ‘while(test, update)’ morphism is formed, set ‘while’ to have the formed morphism as its recursor.



Assemble a While Statement: Figure 19

In the above figure the dashed i_0 is a coreference represents taking the false branch. Upon that happening the output S is then from the upper left dashed id reference morphism, so the state is unchanged. The i_1 coreference represents taking the true branch. Upon that happening, the state S is taken through the other id reference, and sent through the composite of update and while , so the output state is updated and checked again with ‘while’ to see if it keeps going recursively.

2.8.2 Basics

We state a few basic terms to discuss how these morphism assemblies work.

A *domain morphism* of a diagram object is a morphism whose domain is that object. The *domains* of an object are its domain morphisms. In Figure 17 the object $\mathbb{F}_{64} \times \mathbb{F}_{64} \times \mathbb{F}_{64}$ has two domain morphisms, $p_{0,1}$ and p_2 .

A *codomain morphism* of a diagram object is a morphism whose codomain is that object. The *codomains* of an object are its codomain morphisms. In Figure 19 the object $*+*$ has three codomains: test , i_0 , i_1 .

A *blob* is a maximally connected collection of diagram objects and morphisms by domains and codomains in a diagram’s index category.

A *reference* in a blob is a diagram morphism tagged as a reference and whose target morphism is a factor morphism or an identity. The target codomain may be the terminal object (factor -1), or the factors must be unique and not -1. A factor may also be $[]$ meaning a targeted identity morphism is tagged as a reference. In Figure 17 the morphisms p_0 and p_1 are references. In Figure 19 the two p ’s and the two id ’s are references.

A *coreference* in a blob is a morphism tagged as a coreference and whose target morphism is a cofactor morphism or an identity. The factors must be unique and have no initial objects. The factor may also be $[]$ meaning a targeted identity morphism is tagged as a coreference. In Figure 19 the morphisms i_0 and i_1 are coreferences.

A diagram object is said to be a *reference object* and *have references* if it is the domain of a reference. An object is said to be *referenced* if it is the codomain of a reference. Each reference object must have its factors from all its references to be unique and cover the object. In other words, having values on the codomains of that object's references we know how to uniquely assemble a value on the object through its references' factors. If at most one of the references is an identity, then that value is taken first, and the other factors override. In general, a reference object is a product object (unless the reference is only an identity).

A diagram object is said to *have coreferences* if it is the codomain of a coreference. An object is a *coreference* if it has coreferences. An object is said to be *coreferenced* if it is the domain of a coreference. If an object has coreferences, then the factors from all those coreferences must cover the object. Uniqueness of those factors is not required as activation of more than one is allowed. If at most one of the coreferences is an identity, that option is taken last.

The *domain count* of a diagram object is the number of morphisms in its domains that are not references nor coreferences. The *codomain count* is the number of morphisms in its codomains that are not references or coreferences. The *reference count* is the number of references in its domains. The *coreference count* is the number of coreferences in its domains. The *use count* is the number of references or coreferences in its codomains.

An *input* in a blob is an object with a zero codomain count and positive domain count. In other words, the input can be (co)referenced but not set and must be consumed. In Figure 1 the object A is an input.

A *source* in a blob is a diagram object that looks like an input but satisfies one of the following.

- The domain and reference counts are positive, and the codomain count is zero. In other words, the object is constructed from its references and consumed.

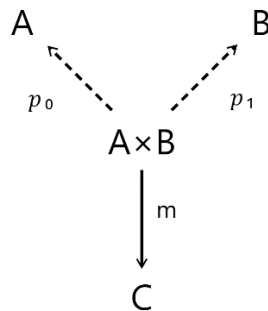
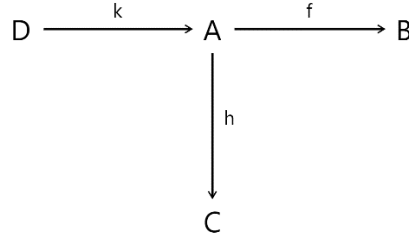


Figure 20

- The codomain count is one and the domain count is greater than one. In other words, the input is forked down two separate paths to be later put back together in a product assembly. In the following figure the object A is this type of source:



A is a Source: Figure 21

- The coreference count is one and either the domain or use count is positive. In other words, the object is determined by its coreference and either consumed or is referred to.

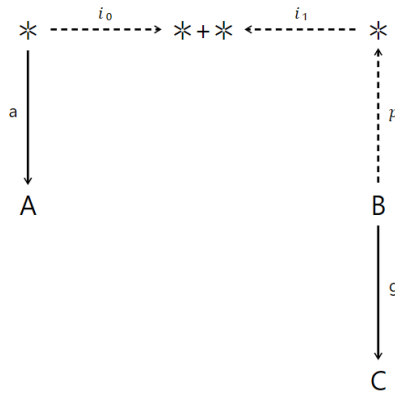


Figure 22

In other words, we form morphisms starting from inputs and sources.

An *output* in a blob is a diagram object with zero domain and use counts, and either the codomain count is one, or the reference count is positive, or the coreference count is positive. In Figure 16 the object D is an output.

Use the following steps to form a morphism from a blob.

2.8.3 Steps

The following steps describe the process of forming a morphism from a blob. Not all blobs can form morphisms. For example, references to out-of-scope objects are not good. When it is time to build an object from its references, if the object is not covered then the formation fails.

2.8.3.1 Form Composites

Form the composite morphism from a codomain-to-domain connected list of arrows starting from an input or source and ending in an output or source. This basically reduces the graph.

$$A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D$$

Assemble a Composite: Figure 23

2.8.3.2 Form Product Assemblies

When more than one composite morphism starts from an input or source, then form the product assembly morphism with those composites.

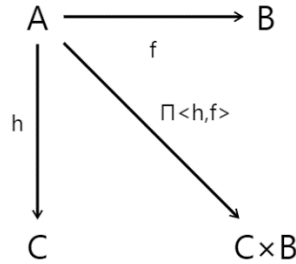


Figure 24

2.8.3.3 Form Reference Section

If a morphism's domain is referenced, then form the product assembly of the domain's identity with that morphism. The additional A as seen in the resulting codomain is known as the *reference section* of the forming morphism.

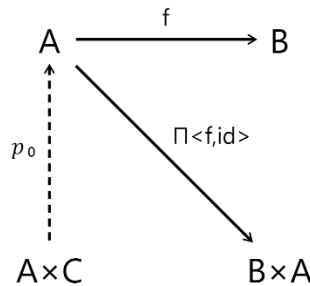











Figure 25








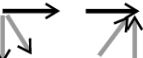

2.8.3.4 Form Hom Assemblies

To form morphisms from a coreferenced object A where the current state of the forming morphism is S , form the morphism starting from the coreference's domain. In other words, we have $S \rightarrow A$ given. For the i 'th coreference, that forms a morphism in the hom set $[S, S]$. Then a map from the coreferenced object A to the hom set $[S, S]$ is formed $A \rightarrow [S, S]$. This then represents the morphism to be executed depending on the coreferenced object A . The morphism $S \rightarrow A \rightarrow [S, S]$ is then product assembled with the identity $S \rightarrow S$ to get $S \rightarrow [S, S] \times S$. Following with the evaluation $e: [S, S] \times S \rightarrow S$ yields the final formed morphism $S \rightarrow [S, S] \times S \rightarrow S$.

2.9 Keyboard Actions

Editing keys only work in single diagram or full screen viewing. A key may only function when the correct elements are selected.

Key	Mod	Diagram View
Arrow		If nothing selected, pan in the indicated direction for either single or multi-diagram viewing. Otherwise move the selected elements.
Home		 View everything
	Ctrl	Go to single diagram viewing and encompass all contents of the current diagram.
A		 Form a morphism assembly from a selected object.
	Ctrl	Select all elements.
C		 Copy the selected elements from the current diagram.
	Ctrl	Copy selected elements from the current diagram into the paste buffer.
D		 Show diagram toolbar.
	Shift	Show new diagram toolbar.
E		 Form the composite of the selected morphisms.
G		 Show the graphs of the morphisms.
H		 Show the new morphism toolbar with the chosen domain and codomain corresponding to the two selected objects.
	Shift	[,] Form the hom action on two objects or two morphisms.
I		1 Create an identity from the selected object.
L		 Show the lambda morphism toolbar.
	Ctrl	Open the TTY panel.
M		 Show the morphism toolbar.

	Shift	Show the new morphism toolbar.
O		 Show the object toolbar.
	Shift	Show the new object toolbar.
P		 Form the product of the selected objects or morphisms.
	Shift	 Form the coproduct of the selected objects or morphisms.
Q		 Show the help info for the selected element.
R		 Make the selected factor or cofactor morphism into a reference.
T		 Show the text toolbar.
	Shift	Show the new text toolbar.
V		 View the selected elements.
	Ctrl	Paste the contents of the paste buffer at the cursor.
Y		 Morphism assembly
Z	Ctrl	Undo the last edit action (maybe).
Delete		 Delete the selected elements.
	Shift	If morphisms are selected, delete their objects as well if possible.
Digit '1'		Place terminal object at mouse location. If an object is selected, then a morphism is placed from that object to the terminal object.
Escape		Close toolbar, close panels, change view mode between single diagram view (for editing) and multi-diagram view.
Equal '='		Zoom out
Minus '-'		Zoom in
Numpad		Pan in indicated direction.
Page Up/ Down		In multi-diagram viewing, the current diagram's Z-order in the view changes up or down accordingly.
Spacebar		Press the spacebar and move the mouse to pan the current diagram's view.

2.10 Mouse Actions

The following actions describe how the mouse interacts with the diagram view.

2.10.1 Select with Left Mouse Button

If the diagram is editable, select an element in the diagram and the toolbar appears. If nothing under the mouse click location, then the toolbar for the diagram appears.

2.10.2 Area Select with Click-Drag-Release

If the diagram is editable, select an area containing elements with a click-drag-release. Using the shift key adds those elements to the selected elements for that diagram.

2.10.3 Vertical Scroll with Mouse Wheel

Use the mouse wheel to scroll vertically in the diagram. If your mouse wheel comes equipped with left/right buttons, then you may use those to horizontally scroll the diagram view.

2.10.4 Side Scroll with Mouse Wheel Tilt

Some mice can tilt the mouse wheel to one side or the other. This then causes a left/right scroll in the diagram view.

2.10.5 Zoom with Shift Key/Mouse Wheel

Press the shift key and then use to mouse wheel to zoom in or out in the diagram view. This also works in the catalog view to adjust the size of the diagram images.

2.10.6 Fusing Objects by Dragging Objects

Drag an object to the same such object and a good glow show be seen. Release the mouse and the object fuses to the targeted object. Any morphisms attached to the dragged object are then set in place with the target object as domain or codomain.

2.10.7 Lock to Grid by Select, Drag then Shift Key

Select one or more elements, start to drag them, then hold the shift key. The elements then lock to the major edit grid as you move the mouse with the shift key depressed.

3 Implementation

The Catecon Node.js servers form a hierarchy. A server marked with a parent downloads reference diagrams from the parent server. Often this is a local host Node.js server that a user deals with daily. When ready diagrams from the local server are uploaded to its parent.

3.1 Node.js Web Server

The web server is made with Node.js and Express.js with a single Pug view.

3.1.1 Setup

Catecon runs on a Linux installation. For Windows, this can be the wsl or wsl2 subsystems.

Steps:

1. Change your current working directory to a clean directory.
2. Have node.js installed on your system.
3. Have your mysql daemon already running.
4. Run the install.sh file, but first rename it to installer.sh to avoid name collisions.
5. Edit your generated .env file accordingly.

The .env file for running the server is also created. It must be edited to add the MYSQL_PASSWORD for the local mysql server and for the correct location of the CAT_DIR which is the root location of the Catecon node application. See the section about the .env file below.

3.1.2 Web Server Directory Structure

.git	Typical git thing.
bin	Catecon support scripts, most notably the web server www
etc	Unused
lambda	Files for AWS Lambda support. Unused currently.
logs	Server log files are stored here during runs
mixins	This is Express related. Unused currently.
node_modules	Installation area for the node server.
public	Public area for the node Express web server. This includes user diagrams, javascript support files et al.
public/diagram	Location of user diagrams in the form public/diagram/username/...
public/js	Client Javascript code
public/ss	Stylesheet
public/svg	Icons
routes	Routes used by the Express server
sql	sql files needed to generate the Catecon database and tables
views	This is Express related, namely the views that are generated. Currently diagram.pug is the only one in use.

3.1.3 Web Server Files

The following gives info regarding select files used by the web server.

3.1.3.1 .env

The configuration file for the server. Typical terms to specify are CAT_DIR and MYSQL_PASSWORD. If you have a different parent than catecon.net, then specify CAT_PARENT.

The AWS_USER_COG_REGION, AWS_USER_IDENTITY_POOL, and AWS_APP_ID are for user logins.

Set the CAT_DIR, replace USER with the user. Set NODE_ENV as desired.

```
CAT_PARENT='https://catecon.net'
CAT_URL='http://localhost:3000'
CAT_DIAGRAM_USER_LIMIT=1024
CAT_DIR='/home/USER/catecon'
CAT_SEARCH_LIMIT=128
CAT_SRVR_LOG='./logs'
CAT_SRVR_LOG_SIZE='100M'
HTTP_DIR='public'
HTTP_PORT=3000
HTTP_ADMINS='USER'
HTTP_UPLOAD_LIMIT='1mb'
MYSQL_HOST=localhost
MYSQL_PORT=3306
MYSQL_USER=root
MYSQL_PASSWORD=XXXXXX
MYSQL_DB=Catecon
AWS_DGRM_RGN='us-west-1'
AWS_DIAGRAM_URL='https://catecon-diagrams.s3-us-west-1.amazonaws.com'
AWS_USER_COG_REGION='us-west-2'
AWS_USER_IDENTITY_POOL='us-west-2_HKN5CKGDz'
AWS_APP_ID='fjc1c9b91pc83tmkm8b152pin'
NODE_ENV='development'
```

3.1.3.2 install.sh

Start the install process with this script, rename it to installer.sh. Does the initial git configuration and downloads the current source files. Installs required npm packages. Creates a skeleton .env configuration file.

3.1.3.3 package.json

The typical Node.js package file used to maintain the project.

3.1.3.4 app.js

The main web server application file. It is run by bin/www.

3.1.4 Debugging

Start the web server using the following command:

```
node -inspect-brk bin/www
```

Next enter the URL *chrome://inspect* into Chrome's address bar.

The click on the displayed link "Open dedicated DevTools for Node".

The debugger should then attach to the node.js web server.

Next press F8 to start the web server. You may need to press it more than once.

3.1.5 Development Web Server Restart

Download the latest updates from the source:

```
git pull
```

Kill the currently running web server:

```
killall node
```

Restart the local host web server:

```
nodemon bin/www
```

3.1.6 Production Web Server Restart

Download the latest updates from the source:

```
git pull
```

Next kill off all the currently running nodes on the system:

```
killall node
```

Then restart the web server:

```
forever bin/www restart &
```

3.2 MySQL Server

Running Catecon requires some form of backing store. In this case the MySQL relational database has been chosen but a non-relational database has been tried as well.

Start the mysql daemon.

```
sudo /etc/init.d/mysql start
```

3.2.1 Catecon Database

The Catecon database tracks users and diagrams. It is currently implemented via MySQL.

3.2.1.1 Diagram Table

```
CREATE TABLE `diagrams` (  
  `name` varchar(128) DEFAULT NULL,  
  `basename` mediumtext CHARACTER SET utf8 COLLATE utf8_bin NOT  
  NULL,  
  `user` mediumtext CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,  
  `codomain` mediumtext CHARACTER SET utf8 COLLATE utf8_bin NOT  
  NULL,  
  `description` mediumtext CHARACTER SET utf8 COLLATE utf8_bin  
  NOT NULL,  
  `properName` mediumtext CHARACTER SET utf8 COLLATE utf8_bin NOT  
  NULL,  
  `refs` longtext CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,  
  `timestamp` bigint(20) NOT NULL,  
  `refcnt` int(11) DEFAULT '0',  
  `cloudTimestamp` bigint(20) DEFAULT NULL,  
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `name` (`name`),  
  FULLTEXT KEY `DescriptionIndex` (`properName`)  
) ENGINE=InnoDB AUTO_INCREMENT=32 DEFAULT CHARSET=utf8mb4
```

3.2.1.2 User Table

Permissions is a token list string, e.g.: “admin mysql gcc node”.

maxDiagrams: A user has a maximum number of diagrams allowed on the server.

```
CREATE TABLE `users` (  
  `name` varchar (256) CHARACTER SET utf8 COLLATE utf8_bin NOT  
  NULL,  
  `email` varchar(256) CHARACTER SET utf8 COLLATE utf8_bin NOT  
  NULL,  
  `permissions` varchar(256) CHARACTER SET utf8 COLLATE utf8_bin  
  NOT NULL,  
  `maxDiagrams` bigint(20) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

4 Terms and Conditions

No hate. The purpose of this site is to promote math and computation.

Tracking: There is no tracking. If you get a user login, then you get user credentials in your browser from AWS.

Diagrams are stored in the user’s web browser local storage. Diagrams are not uploaded to catecon.net unless willed so by the user. If the user runs a local server, then the user’s edits may be immediately saved on their personal private server.

Viewing diagrams requires no signup. Anonymous users may edit their own diagrams but no uploading. These diagrams remain in their web browser local storage.

If a user uploads a diagram to a server, all users on that server may see it and reference it.

If your diagram is referenced either by you or others on a server, you may not be able to edit it or delete it from that server.