

# A diagram editor to mechanise categorical proofs

Ambroise Lafont (PARTOUT)

23 November 2023

# A bit of history

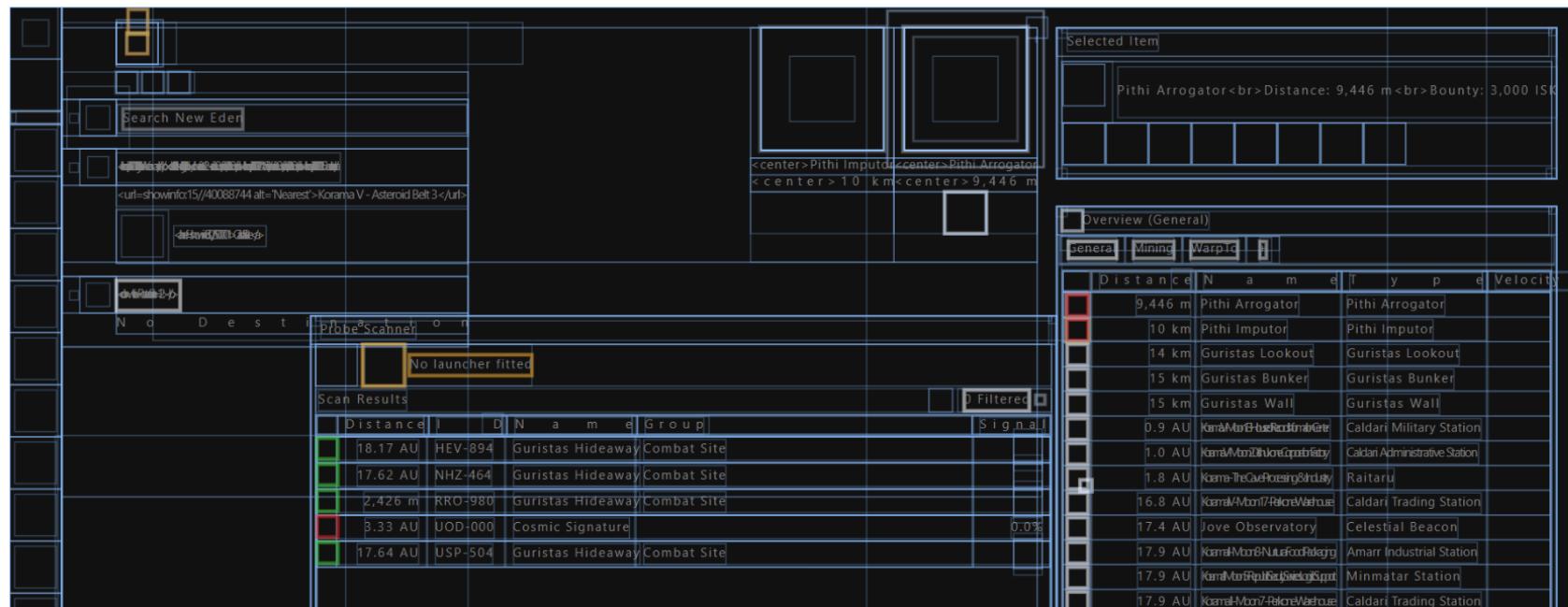
It all started with Covid19.



# Sanderling

Sanderling helps you read information from the [EVE Online](#) game client using memory reading.

Sanderling is the eyes of bots and monitoring tools. It helps programs see the game client in a structured way, detecting objects and reading information about the game world. It also reads the locations of elements in the game clients' graphical user interface (e.g., in-game windows, overview entries, buttons, etc.). You can use this information to interact with the game client using mouse input.



## Releases 39

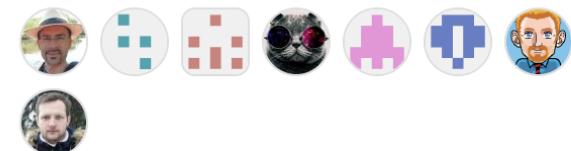
 v2023-01-03 Latest  
on Jan 3

+ 38 releases

## Packages

No packages published

## Contributors 8



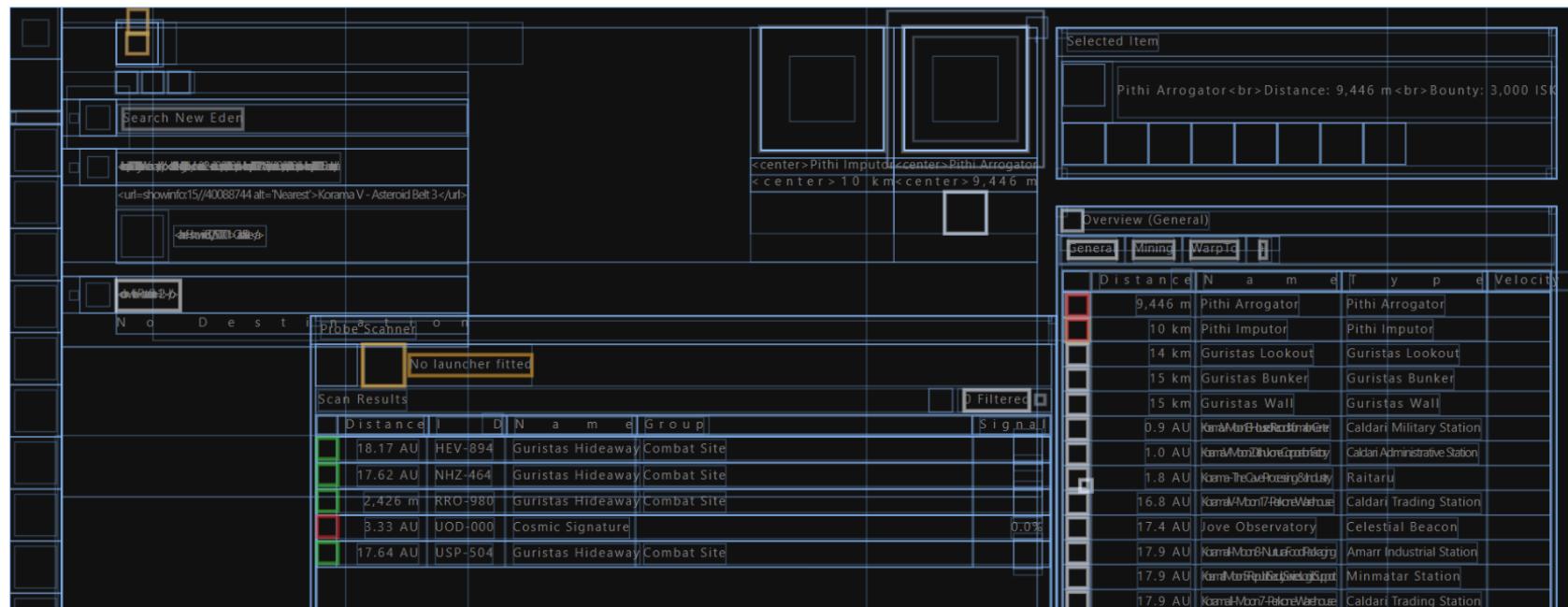
## Languages

 Elm 72.2%  C# 27.8%

# Sanderling

Sanderling helps you read information from the [EVE Online](#) game client using memory reading.

Sanderling is the eyes of bots and monitoring tools. It helps programs see the game client in a structured way, detecting objects and reading information about the game world. It also reads the locations of elements in the game clients' graphical user interface (e.g., in-game windows, overview entries, buttons, etc.). You can use this information to interact with the game client using mouse input.



## Releases 39

 v2023-01-03 Latest  
on Jan 3

+ 38 releases

## Packages

No packages published

## Contributors 8

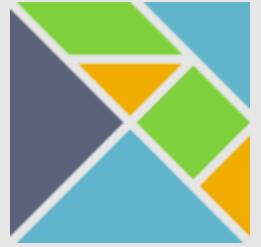


## Languages

 Elm 72.2%  C# 27.8%



Elm: A delightful language  
for reliable web applications.



# Elm: A delightful language for reliable web applications.

“[My favorite thing] is the feeling of joy and relaxation when writing Elm code.”

[Luca Mugnaini](#), Software Engineer, [Rakuten](#)



# Elm: A delightful language for reliable web applications.

“[My favorite thing] is the feeling of joy and relaxation when writing Elm code.”

[Luca Mugnaini](#), Software Engineer, [Rakuten](#)

Concretely:



# Elm: A delightful language for reliable web applications.

“[My favorite thing] is the feeling of joy and relaxation when writing Elm code.”

[Luca Mugnaini](#), Software Engineer, [Rakuten](#)

Concretely:

- A functional programming language that compiles to JavaScript



# Elm: A delightful language for reliable web applications.

“[My favorite thing] is the feeling of joy and relaxation when writing Elm code.”

[Luca Mugnaini](#), Software Engineer, [Rakuten](#)

## Concretely:

- A functional programming language that compiles to JavaScript
- A simple *reactive* architecture



# Elm: A delightful language for reliable web applications.

“[My favorite thing] is the feeling of joy and relaxation when writing Elm code.”

[Luca Mugnaini](#), Software Engineer, [Rakuten](#)

## Concretely:

- A functional programming language that compiles to JavaScript
- A simple *reactive* architecture

### ■ THE ELM ARCHITECTURE

`init` : ( Model, Cmd Msg )

`update` : Msg -> Model -> ( Model, Cmd Msg )

`subscriptions` : Model -> Sub Msg

`view` : Model -> Html Msg

# Afterthoughts: why you should not program in Elm

# Afterthoughts: why you should not program in Elm

- Is it still a thing?

# Afterthoughts: why you should not program in Elm

- Is it still a thing?

Alive slack (with occasional job opportunities), weekly newsletter, bimensual podcasts, ...

# Afterthoughts: why you should not program in Elm

- Is it still a thing?

Alive slack (with occasional job opportunities), weekly newsletter, bimensual podcasts, ...

But last compiler release: v0.19.1, Oct 2019

# Afterthoughts: why you should not program in Elm

- Is it still a thing?

Alive slack (with occasional job opportunities), weekly newsletter, bimensual podcasts, ...

But last compiler release: v0.19.1, Oct 2019

- Terrible JavaScript FFI

# Afterthoughts: why you should not program in Elm

- Is it still a thing?

Alive slack (with occasional job opportunities), weekly newsletter, bimensual podcasts, ...

But last compiler release: v0.19.1, Oct 2019

- Terrible JavaScript FFI

Asynchronous calls to JS only, as *commands*

# Afterthoughts: why you should not program in Elm

- Is it still a thing?

Alive slack (with occasional job opportunities), weekly newsletter, bimensual podcasts, ...

But last compiler release: v0.19.1, Oct 2019

- Terrible JavaScript FFI      (intentional!)

Asynchronous calls to JS only, as *commands*

# Afterthoughts: why you should not program in Elm

- Is it still a thing?

Alive slack (with occasional job opportunities), weekly newsletter, bimensual podcasts, ...

But last compiler release: v0.19.1, Oct 2019

- Terrible JavaScript FFI     (**intentional!**)

Asynchronous calls to JS only, as *commands*

## Why I'm leaving Elm

by [Luke Plant](#)

Posted in: [ELM](#), [WEB DEVELOPMENT](#) — April 4, 2020 14:28

Over the past year or so, I've reluctantly come to the conclusion I need to leave Elm and migrate to some other language (most likely Bucklescript via philip2), and I definitely cannot recommend it to anyone else. This post is about my reasons for that, which are mostly about the way in which the leadership behave.

2020: I need a pretext to program in Elm

# 2020: I need a pretext to program in Elm

- A game?

# 2020: I need a pretext to program in Elm

- A game?  
Too old for this...

# 2020: I need a pretext to program in Elm

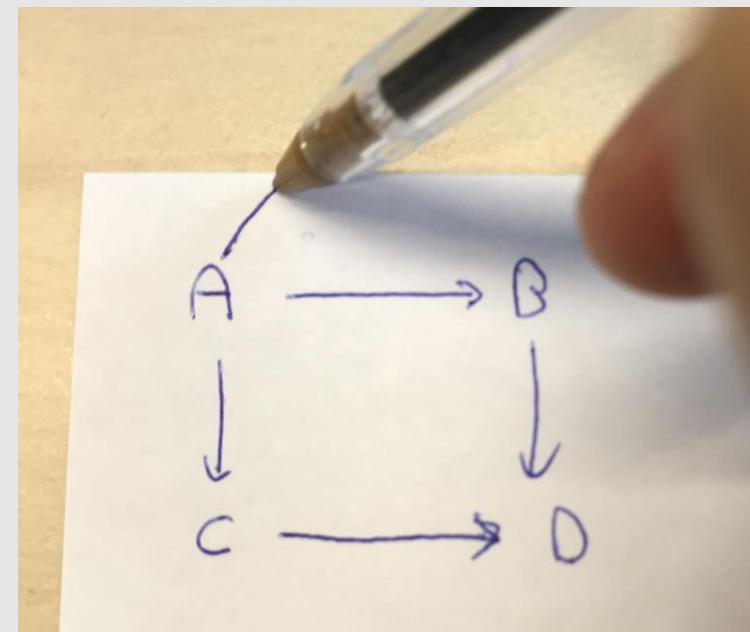
- A game?  
Too old for this...
- A categorical diagram editor

# 2020: I need a pretext to program in Elm

- A game?  
Too old for this...
- A categorical diagram editor

Drawing on paper is inconvenient:

no way of reorganising the diagram  
without redrawing everything.



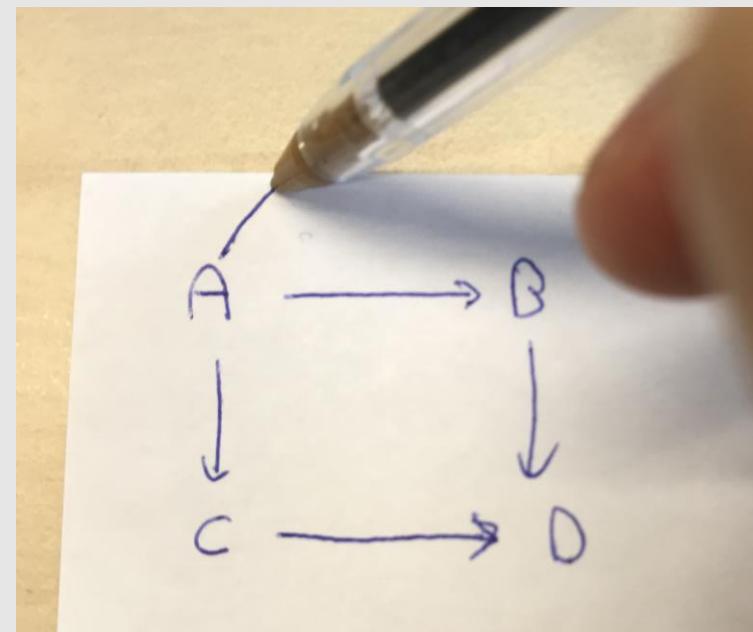
# 2020: I need a pretext to program in Elm

- A game?  
Too old for this...
- A categorical diagram editor

Drawing on paper is inconvenient:

no way of reorganising the diagram  
without redrawing everything.

## DEMO



# How to test the editor

A web app that runs locally in your browser

<https://amblafont.github.io/graph-editor/index.html>

# How to test the editor

A web app that runs locally in your browser

<https://amblafont.github.io/graph-editor/index.html>

A standalone desktop program

- Embeds the web app using electron
  - Additional features

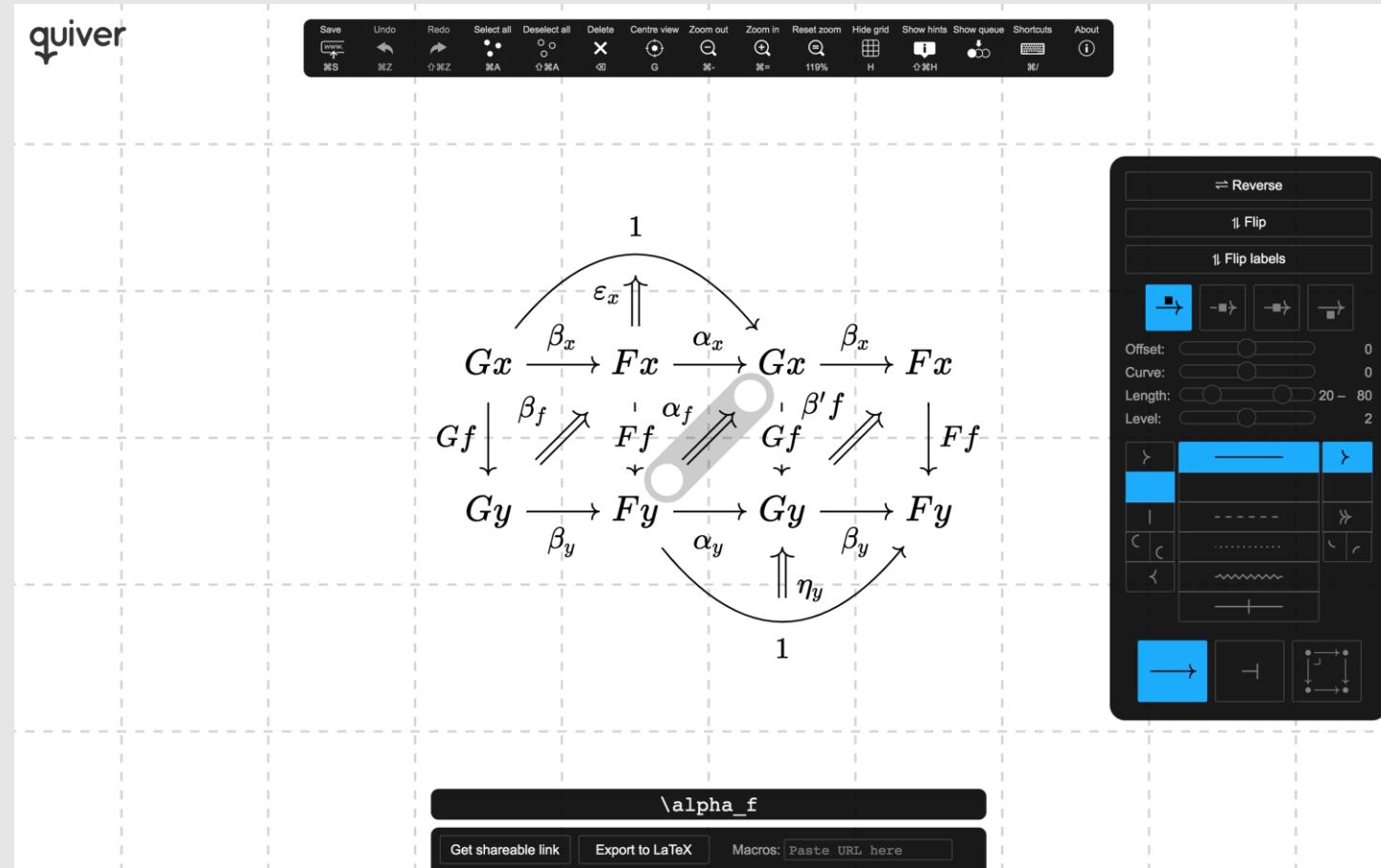
# Nov. 2020: An existential crisis

# Nov. 2020: An existential crisis

Nathanel Arkhor releases quiver, “*a modern, graphical editor for commutative and pasting diagrams, capable of rendering high-quality diagrams for screen viewing, and exporting to LaTeX via tikz-cd.*”

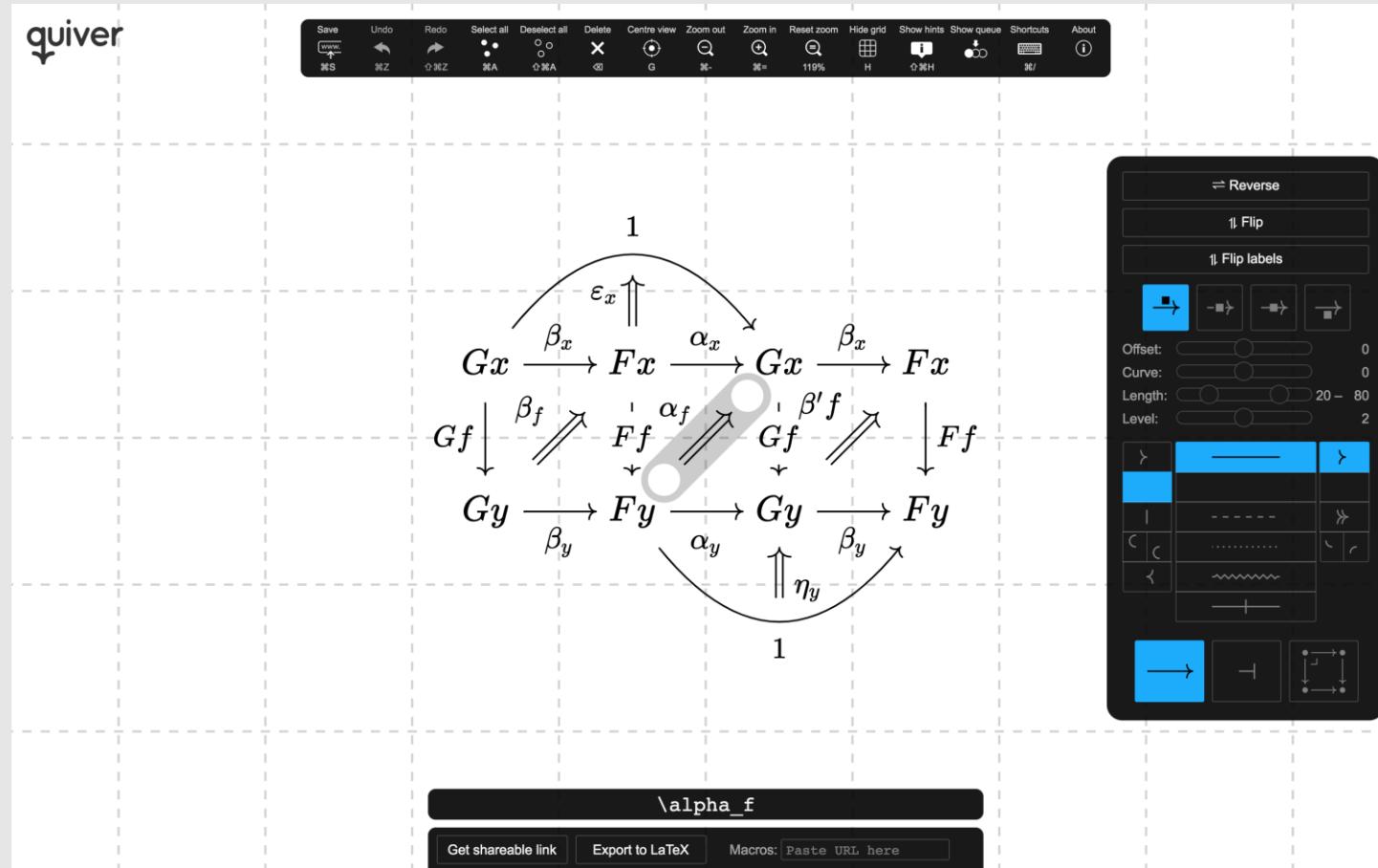
# Nov. 2020: An existential crisis

Nathanel Arkhor releases quiver, “*a modern, graphical editor for commutative and pasting diagrams, capable of rendering high-quality diagrams for screen viewing, and exporting to LaTeX via tikz-cd.*”



# Nov. 2020: An existential crisis

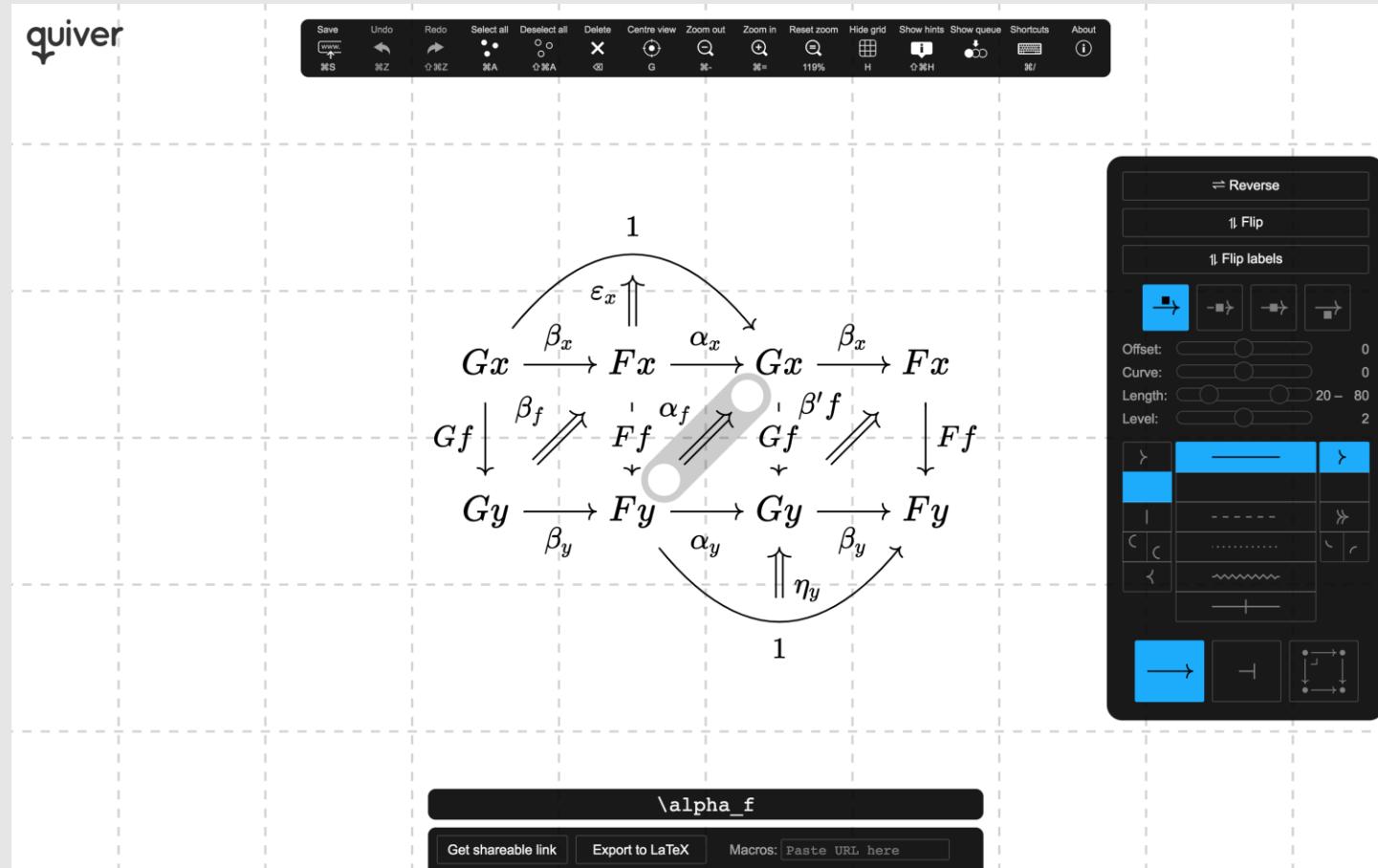
Nathaniel Arkhor releases quiver, “*a modern, graphical editor for commutative and pasting diagrams, capable of rendering high-quality diagrams for screen viewing, and exporting to LaTeX via tikz-cd.*”



That's what I have been looking for!

# Nov. 2020: An existential crisis

Nathaniel Arkhor releases quiver, “*a modern, graphical editor for commutative and pasting diagrams, capable of rendering high-quality diagrams for screen viewing, and exporting to LaTeX via tikz-cd.*”

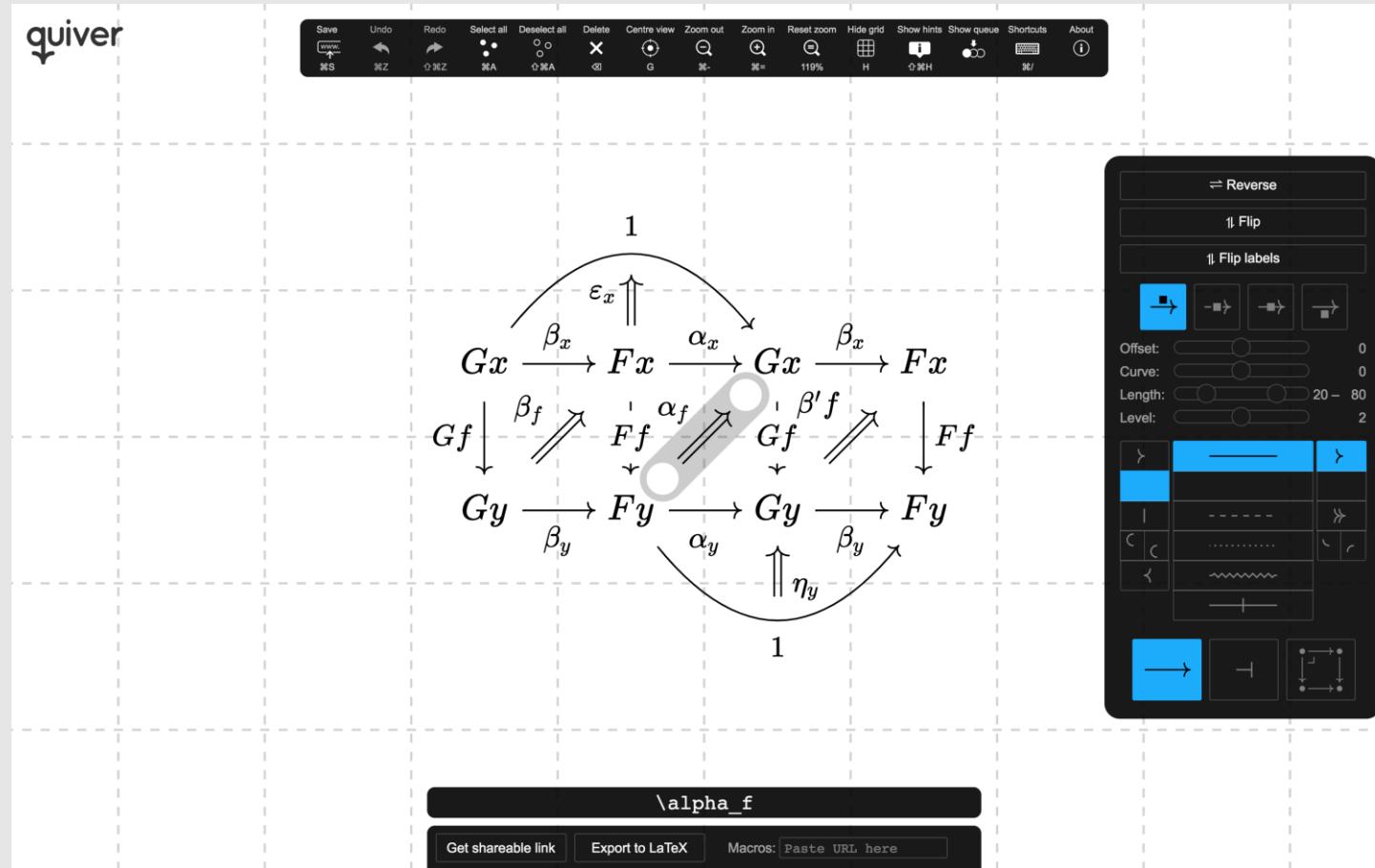


That's what I have been looking for!

Can I implement my own editing style and shortcuts on top of it?

# Nov. 2020: An existential crisis

Nathaniel Arkhor releases quiver, “*a modern, graphical editor for commutative and pasting diagrams, capable of rendering high-quality diagrams for screen viewing, and exporting to LaTeX via tikz-cd.*”



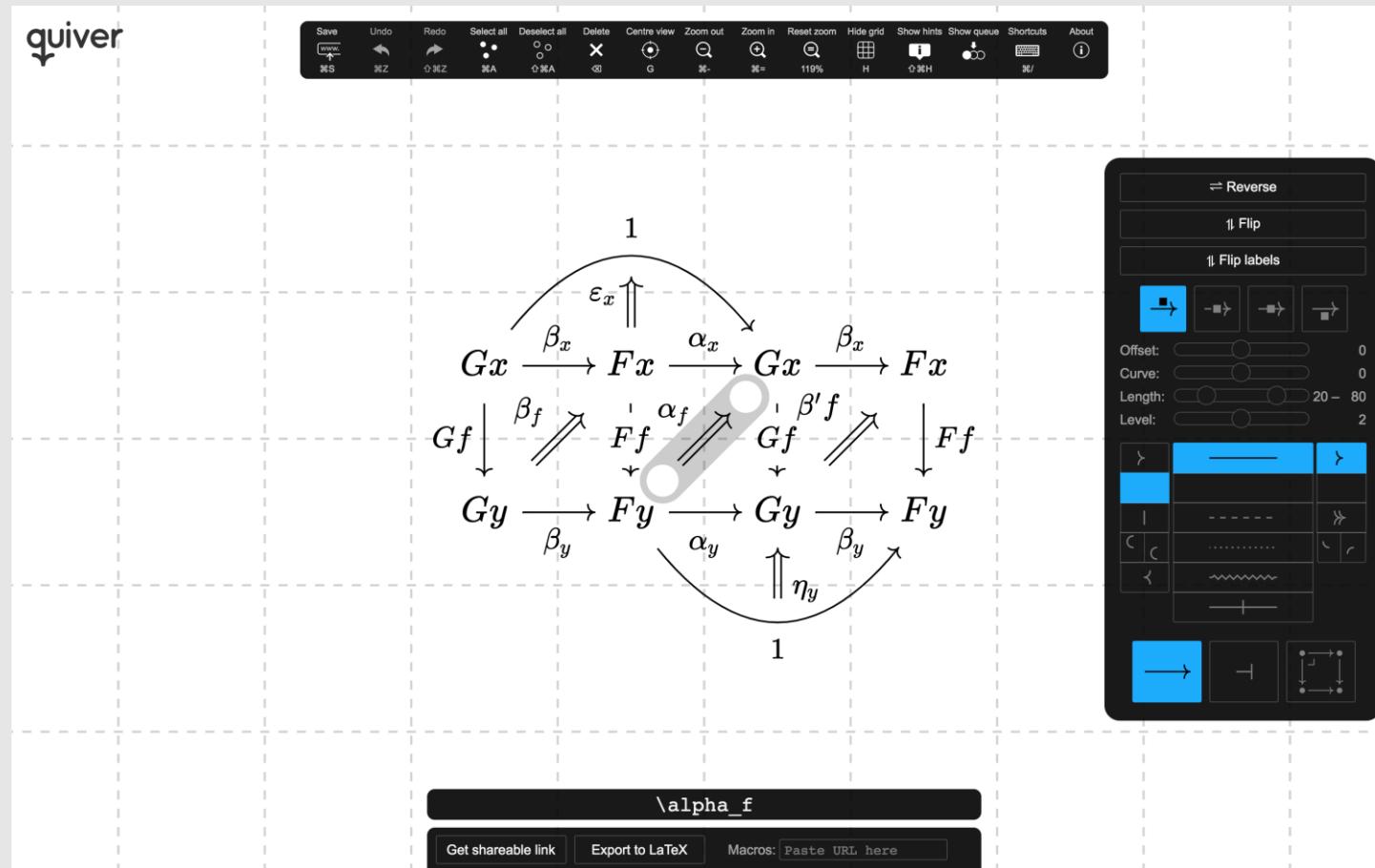
That's what I have been looking for!

Can I implement my own editing style and shortcuts on top of it?

In principle yes, but...

# Nov. 2020: An existential crisis

Nathaniel Arkhor releases quiver, “*a modern, graphical editor for commutative and pasting diagrams, capable of rendering high-quality diagrams for screen viewing, and exporting to LaTeX via tikz-cd.*”



That's what I have been looking for!

Can I implement my own editing style and shortcuts on top of it?

In principle yes, but...  
it is implemented in JavaScript 

# Comparison with quiver

About the same size (around 10k of LoC)

	Quiver	My editor
<b>Programming Languages</b>	<p>Languages</p>  <ul style="list-style-type: none"><li>JavaScript 90.7%</li><li>CSS 5.3%</li><li>TeX 2.4%</li><li>Other 1.6%</li></ul>	<p>Languages</p>  <ul style="list-style-type: none"><li>Elm 72.6%</li><li>HTML 12.7%</li><li>TeX 8.5%</li><li>TypeScript 4.7%</li><li>JavaScript 1.5%</li></ul>
<b>Styling options</b>	+	-
<b>User-friendly</b>	+	-
<b>Editing features</b>	-	<p>+</p> <p>Tabs, copy &amp; paste, find &amp; replace, expand selection to connected components, ...</p>

# Comparison with quiver

About the same size (around 10k of LoC)

	Quiver	My editor
Programming Languages	<p>Languages</p>  <ul style="list-style-type: none"><li>JavaScript 90.7%</li><li>CSS 5.3%</li><li>TeX 2.4%</li><li>Other 1.6%</li></ul>	<p>Languages</p>  <ul style="list-style-type: none"><li>Elm 72.6%</li><li>HTML 12.7%</li><li>TeX 8.5%</li><li>TypeScript 4.7%</li><li>JavaScript 1.5%</li></ul>
Styling options	+	-
User-friendly	+	-
Editing features	-	<p>+</p> <p>Tabs, copy &amp; paste, find &amp; replace, expand selection to connected components, ...</p>
LaTeX export	yes	yes <sup>1</sup>

<sup>1</sup> Implemented by Tom Hirschowitz

# Easy editing of LaTeX diagrams

# Easy editing of LaTeX diagrams

```
% YADE DIAGRAM {"graph":{"edges":[{"from":0,"id":4,"label":{"isPullshout":  
% GENERATED LATEX  
\begin{tikzpicture}[every node/.style={inner sep=5pt,outer sep=0pt,anchor  
\node (0) at (9.476190476190476em, -3.3154761904761907em) {$a$} ;  
\node (1) at (19em, -3.3154761904761907em) {$b$} ;
```

example.tex

# Easy editing of LaTeX diagrams

diagram data

```
% YADE DIAGRAM {"graph":{"edges":[{"from":0,"id":4,"label":{"isPullshout":  
% GENERATED LATEX  
\begin{tikzpicture}[every node/.style={inner sep=5pt,outer sep=0pt,anchor  
\node (0) at (9.476190476190476em, -3.3154761904761907em) {$a$} ;  
\node (1) at (19em, -3.3154761904761907em) {$b$} ;
```

example.tex

# Easy editing of LaTeX diagrams



standalone version, running in the background

```
% YADE DIAGRAM {"graph": {"edges": [{"from": 0, "id": 4, "label": {"isPullshout": true}}, {"to": 1}], "nodes": [{"id": 0, "label": "a"}, {"id": 1, "label": "b"}]}}, {"text": "% GENERATED LATEX", "x": 100, "y": 150}, {"text": "\\begin{tikzpicture}[every node/.style={inner sep=5pt,outer sep=0pt,anchor=center]", "x": 100, "y": 170}, {"text": "\\node (0) at (9.476190476190476em, -3.3154761904761907em) {$a$};", "x": 100, "y": 190}, {"text": "\\node (1) at (19em, -3.3154761904761907em) {$b$};", "x": 100, "y": 210}, {"text": "]", "x": 100, "y": 230}, {"text": "}", "x": 100, "y": 250}, {"text": "example.tex", "x": 100, "y": 300}], [{"x": 100, "y": 280, "x2": 100, "y2": 300}], [{"x": 100, "y": 300}]]
```

diagram data

# Easy editing of LaTeX diagrams



standalone version, running in the background

```
% YADE DIAGRAM {"graph": {"edges": [{"from": 0, "id": 4, "label": {"isPullshout": true}}, {"to": 1}], "nodes": [{"id": 0, "label": "a"}, {"id": 1, "label": "b"}]}, "id": 1}
```

example.tex

- (1) Remove the generated latex

diagram data

# Easy editing of LaTeX diagrams



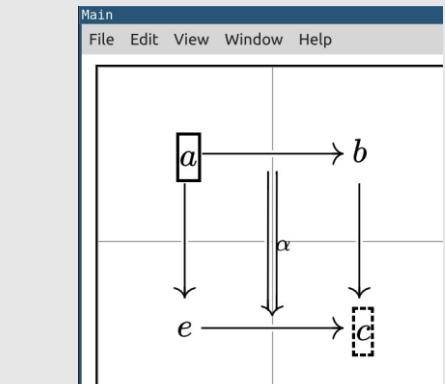
standalone version, running in the background

(1) Remove the generated latex

⇒ (2) the editor loads the diagram

```
% YADE DIAGRAM {"graph": {"edges": [{"from": 0, "id": 4, "label": {"isPullshout": true}}, {"from": 1, "id": 5, "label": {"isPushshout": true}}], "nodes": [{"id": 0, "label": "a"}, {"id": 1, "label": "b"}]}, "version": 1}
```

example.tex



# Easy editing of LaTeX diagrams

# Main

File Edit View Window Help

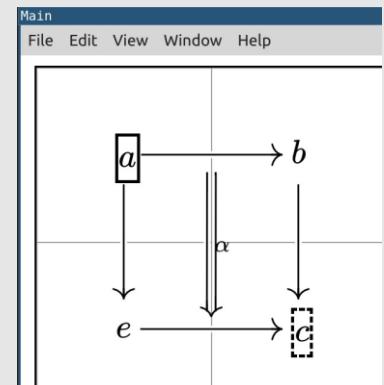
# Diagram editor

standalone version, running in the background

(1) Remove the generated latex      ⇒ (2) the editor loads the diagram

(3) Edit & Save the diagram in the editor

## diagram data



# Easy editing of LaTeX diagrams



standalone version, running in the background

```
% YADE DIAGRAM {"graph": {"edges": [{"from": 0, "id": 4, "label": {"isPullshout": true}, "to": 1}], "nodes": [{"id": 0, "label": "a"}, {"id": 1, "label": "b"}]}}, {"x": 9.476190476190476, "y": -3.3154761904761907}, {"x": 19, "y": -3.3154761904761907}
```

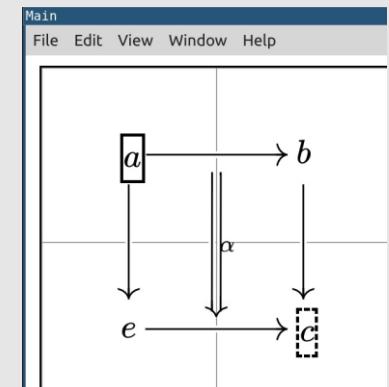
example.tex

(1) Remove the generated latex      ⇒ (2) the editor loads the diagram

(3) Edit & Save the diagram in the editor

⇒ (4) the editor updates example.tex  
(generated LaTeX & diagram data)

diagram data



# Easy editing of LaTeX diagrams



standalone version, running in the background

```
% YADE DIAGRAM {"graph": {"edges": [{"from": 0, "id": 4, "label": {"isPullshout": true}, "to": 1}], "nodes": [{"id": 0, "label": "a"}, {"id": 1, "label": "b"}]}, "id": 1, "label": "b"} ;
```

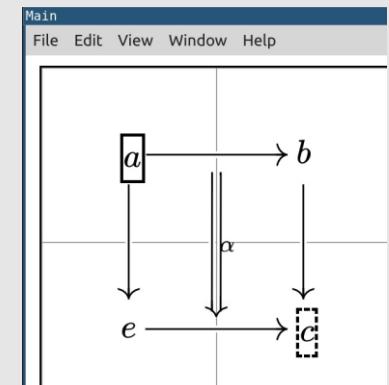
example.tex

(1) Remove the generated latex      ⇒ (2) the editor loads the diagram

(3) Edit & Save the diagram in the editor

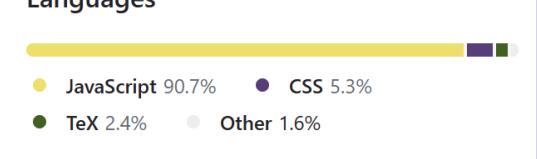
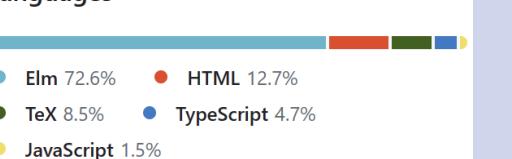
⇒ (4) the editor updates example.tex  
(generated LaTeX & diagram data)

diagram data

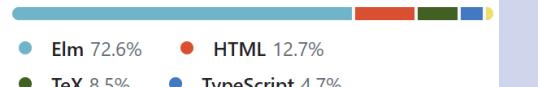


## DEMO

# Comparison with quiver

	Quiver	My editor
Programming Languages	<p>Languages</p>  <ul style="list-style-type: none"><li>JavaScript 90.7%</li><li>CSS 5.3%</li><li>TeX 2.4%</li><li>Other 1.6%</li></ul>	<p>Languages</p>  <ul style="list-style-type: none"><li>Elm 72.6%</li><li>HTML 12.7%</li><li>TeX 8.5%</li><li>TypeScript 4.7%</li><li>JavaScript 1.5%</li></ul>
Styling options	+	-
User-friendly	+	-
Editing features	-	<p>+</p> <p>Tabs, Copy &amp; paste, find &amp; replace, expand selection to connected components, ...</p>
LaTeX export	yes	yes

# Comparison with quiver

	Quiver	My editor
Programming Languages	<p>Languages</p>  <ul style="list-style-type: none"><li>JavaScript 90.7%</li><li>CSS 5.3%</li><li>TeX 2.4%</li><li>Other 1.6%</li></ul>	<p>Languages</p>  <ul style="list-style-type: none"><li>Elm 72.6%</li><li>HTML 12.7%</li><li>TeX 8.5%</li><li>TypeScript 4.7%</li><li>JavaScript 1.5%</li></ul>
Styling options	+	-
User-friendly	+	-
Editing features	-	<p>+</p> <p>Tabs, Copy &amp; paste, find &amp; replace, expand selection to connected components, ...</p>
LaTeX export	yes	yes
A nice name	+	-

I used to call my editor YADE.

I used to call my editor YADE.

# Yet Another Diagram Editor

I used to call my editor YADE.

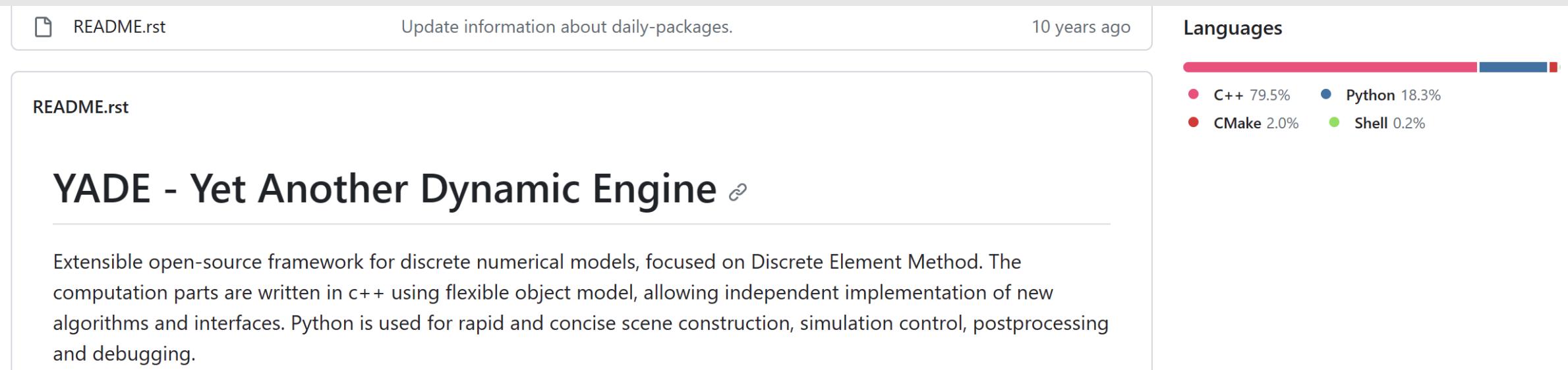
# Yet Another Diagram Editor

This is already taken 😞

I used to call my editor YADE.

# Yet Another Diagram Editor

This is already taken 😞



Now I call it Coreact YADE 🤡

Now I call it Coreact YADE 🤓



**ANR Project<sup>1</sup>** (2023 - 2027): Coq-based Rewriting: Towards Executable Applied Category Theory



Now I call it Coreact YADE 🤡



**ANR Project<sup>1</sup>** (2023 - 2027): Coq-based Rewriting: Towards Executable Applied Category Theory

*Develop and maintain an interactive Coq-based wiki system for categorical rewriting theory*



<sup>1</sup> <https://coreact.wiki/>

Now I call it Coreact YADE 🎨



**ANR Project<sup>1</sup>** (2023 - 2027): Coq-based Rewriting: Towards Executable Applied Category Theory

*Develop and maintain an interactive Coq-based wiki system for categorical rewriting theory*



- Led by N. Behr (IRIF)

Now I call it Coreact YADE 🤡



**ANR Project<sup>1</sup>** (2023 - 2027): Coq-based Rewriting: Towards Executable Applied Category Theory

*Develop and maintain an interactive Coq-based wiki system for categorical rewriting theory*



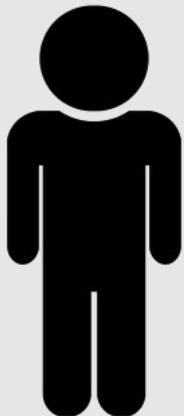
- Led by N. Behr (IRIF)
- Involving (at LIX): myself, S. Mimram, B. Werner, N. Zeilberger.

<sup>1</sup> <https://coreact.wiki/>

# The Coq proof assistant

A software to write formal proofs interactively

File editor, e.g., Visual Studio Code + coq-lsp extension



goals.v > ...

```
17
18 Lemma failingBullet
19 | : (1 = 1) /\ (21 = 21 /\ 22 = 22) /\ (3 = 3).
20 Proof.
21 split; [|split].
22 -
23 -
24 Qed.
25
26 Lemma failingsubProof
27 | : (1 = 1) /\ (21 = 21 /\ 22 = 22) /\ (3 = 3).
28 Proof.
29 | apply
30 |   You, 3 seconds ago • Uncommitted changes
31 Qed.
32
33 Lemma anotherFailing : nat * nat * nat.
```

Diagram illustrating the interaction between the user and the Coq proof assistant:

- The user (represented by a black person icon) interacts with the "Edited file" (the code in the file editor).
- The "Edited file" is processed by the Coq proof assistant.
- The Coq proof assistant provides "Feedback from Coq" (error messages and goal status).
- Blue arrows indicate the flow of information between the user, the edited file, and the Coq feedback.

Edited file

goals.v:30:9

▼ Goals (1)  
▼ Goal (1)

```
1 = 1 /\ (21 = 21 /\ 22 = 22) /\ (3 = 3)
```

► Messages (0)  
▼ Error Browser  
The variable Qed was not found in scope

Diagram illustrating the Coq proof assistant interface:

- The interface shows a goal state:  $1 = 1 \wedge (21 = 21 \wedge 22 = 22) \wedge (3 = 3)$ .
- An error message is displayed: "The variable Qed was not found in scope".
- A small icon of a rooster is present in the bottom right corner.

Feedback from Coq

Coq



# The Coq proof assistant

A software to write formal proofs interactively

File editor, e.g., Visual Studio Code + coq-lsp extension



goals.v > ...

```
17
18 Lemma failingBullet
19 | : (1 = 1) /\ (21 = 21 /\ 22 = 22) /\ (3 = 3).
20 Proof.
21 split; [|split].
22 -
23 -
24 Qed.
25
26 Lemma failingsubProof
27 | : (1 = 1) /\ (21 = 21 /\ 22 = 22) /\ (3 = 3).
28 Proof.
29 | apply
30 | Qed.
31
32 Lemma anotherFailing : nat * nat * nat.
33
```

You, 3 seconds ago • Uncommitted changes

This diagram shows a user interacting with a code editor (Visual Studio Code) containing a Coq script named 'goals.v'. The user has made changes to the script, specifically adding a new lemma 'failingsubProof' and a new lemma 'anotherFailing'. A large blue double-headed arrow indicates the bidirectional communication between the user and the Coq system.

▼ goals.v:30:9

▼ Goals (1)

▼ Goal (1)

1 = 1 /\ (21 = 21 /\ 22 = 22) /\

► Messages (0)

▼ Error Browser

The variable Qed was not found in

This diagram shows the Coq interface displaying the state of the proof. It shows a goal involving three equalities. Below the goal, the 'Error Browser' panel indicates that the variable 'Qed' was not found. A large blue double-headed arrow indicates the bidirectional communication between the user and the Coq system.

Coq

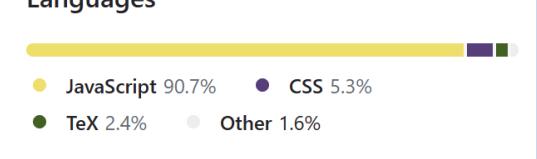
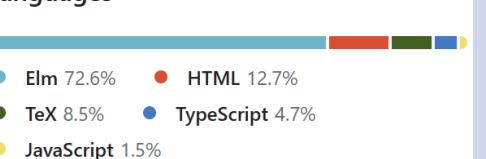


Edited file

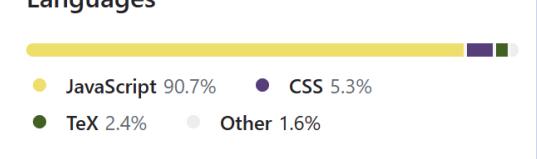
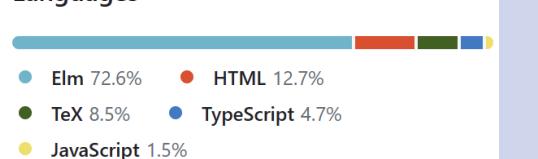
Feedback from Coq

There is also a web version of Coq (no installation required): <https://coq.vercel.app/>

# Comparison with quiver

	Quiver	My editor
<b>Programming Languages</b>	<p>Languages</p>  <ul style="list-style-type: none"><li>JavaScript 90.7%</li><li>CSS 5.3%</li><li>TeX 2.4%</li><li>Other 1.6%</li></ul>	<p>Languages</p>  <ul style="list-style-type: none"><li>Elm 72.6%</li><li>HTML 12.7%</li><li>TeX 8.5%</li><li>TypeScript 4.7%</li><li>JavaScript 1.5%</li></ul>
<b>Styling options</b>	+	-
<b>User-friendly</b>	+	-
<b>Editing features</b>	-	<p>+</p> <p>Tabs, Copy &amp; paste, find &amp; replace, expand selection to connected components, ...</p>
<b>LaTeX export</b>	yes	yes
<b>A nice name</b>	+	-

# Comparison with quiver

	Quiver	My editor
Programming Languages	<p>Languages</p>  <ul style="list-style-type: none"> <li>JavaScript 90.7%</li> <li>CSS 5.3%</li> <li>TeX 2.4%</li> <li>Other 1.6%</li> </ul>	<p>Languages</p>  <ul style="list-style-type: none"> <li>Elm 72.6%</li> <li>HTML 12.7%</li> <li>TeX 8.5%</li> <li>TypeScript 4.7%</li> <li>JavaScript 1.5%</li> </ul>
Styling options	+	-
User-friendly	+	-
Editing features	-	<p>+</p> <p>Tabs, Copy &amp; paste, find &amp; replace, expand selection to connected components, ...</p>
LaTeX export	yes	yes
A nice name	+	-
Mechanisation features	-	+

# Diagrammatic reasoning in category theory

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c: Fc \rightarrow Gc)_c$  such that:

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c: Fc \rightarrow Gc)_c$  such that:

$$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c: Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)



$$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c: Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)



$$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

Or is it  $Ff \circ \alpha_{c_2} = \alpha_{c_1} \circ Gf$ ?

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c: Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)



$$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

Or is it  $Ff \circ \alpha_{c_2} = \alpha_{c_1} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c: Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)

$$\underbrace{\quad}_{\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf}$$

Or is it  $Ff \circ \alpha_{c_2} = \alpha_{c_1} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?

Diagrammatically,

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c:Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)

$$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

Or is it  $Ff \circ \alpha_{c_2} = \alpha_{c_1} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?

Diagrammatically,

$$\begin{array}{ccc} Fc_1 & & \\ Ff \downarrow & & \\ Fc_2 & & \end{array}$$

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c:Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)

$$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

Or is it  $Ff \circ \alpha_{c_2} = \alpha_{c_1} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?

Diagrammatically,

$$\begin{array}{ccc} Fc_1 & & Gc_1 \\ \downarrow Ff & & \downarrow Gf \\ Fc_2 & & Gc_2 \end{array}$$

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c:Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)

$$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

Or is it  $Ff \circ \alpha_{c_2} = \alpha_{c_1} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?

Diagrammatically,

$$\begin{array}{ccc} Fc_1 & \xrightarrow{\alpha_{c_1}} & Gc_1 \\ Ff \downarrow & & \downarrow Gf \\ Fc_2 & & Gc_2 \end{array}$$

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c:Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)

$$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

Or is it  $Ff \circ \alpha_{c_2} = \alpha_{c_1} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?

Diagrammatically,

$$\begin{array}{ccc} Fc_1 & \xrightarrow{\alpha_{c_1}} & Gc_1 \\ Ff \downarrow & & \downarrow Gf \\ Fc_2 & \xrightarrow{\alpha_{c_2}} & Gc_2 \end{array}$$

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c:Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)

$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$

Or is it  $Ff \circ \alpha_{c_2} = \alpha_{c_1} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?

Diagrammatically,

$$\begin{array}{ccc} Fc_1 & \xrightarrow{\alpha_{c_1}} & Gc_1 \\ Ff \downarrow & = & \downarrow Gf \\ Fc_2 & \xrightarrow{\alpha_{c_2}} & Gc_2 \end{array}$$

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c:Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)

$$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

Or is it  $Ff \circ \alpha_{c_2} = \alpha_{c_1} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?

Diagrammatically,

$$\begin{array}{ccc} Fc_1 & \xrightarrow{\alpha_{c_1}} & Gc_1 \\ Ff \downarrow & = & \downarrow Gf \\ Fc_2 & \xrightarrow{\alpha_{c_2}} & Gc_2 \end{array}$$

$$Gf \circ \alpha_{c_1} = \alpha_{c_2} \circ Ff$$

# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

A family of morphisms  $(\alpha_c:Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)

$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ f$

Or is it  $Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?

Diagrammatically,

$$\begin{array}{ccc} Fc_1 & \xrightarrow{\alpha_{c_1}} & Gc_1 \\ Ff \downarrow & = & \downarrow Gf \\ Fc_2 & \xrightarrow{\alpha_{c_2}} & Gc_2 \end{array}$$

$$Gf \circ \alpha_{c_1} = \alpha_{c_2} \circ Ff$$



# Diagrammatic reasoning in category theory

Definition of a natural transformation  $\alpha: F \Rightarrow G$  ?

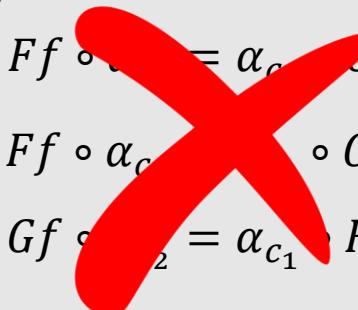
A family of morphisms  $(\alpha_c:Fc \rightarrow Gc)_c$  such that:

Computer-friendly (text)

$\forall f: c_1 \rightarrow c_2, \quad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ f$

Or is it  $Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$ ?

Or is it  $Gf \circ \alpha_{c_2} = \alpha_{c_1} \circ Ff$ ?



Diagrammatically,

$$\begin{array}{ccc} Fc_1 & \xrightarrow{\alpha_{c_1}} & Gc_1 \\ Ff \downarrow & = & \downarrow Gf \\ Fc_2 & \xrightarrow{\alpha_{c_2}} & Gc_2 \end{array}$$

$$Gf \circ \alpha_{c_1} = \alpha_{c_2} \circ Ff$$



...unless  $\circ$  is defined the other way around

## Natural transformations compose:

If  $\alpha: F \Rightarrow G$  and  $\beta: G \Rightarrow H$  are natural, then so is  $(\beta_c \circ \alpha_c:Fc \rightarrow Hc)_c$

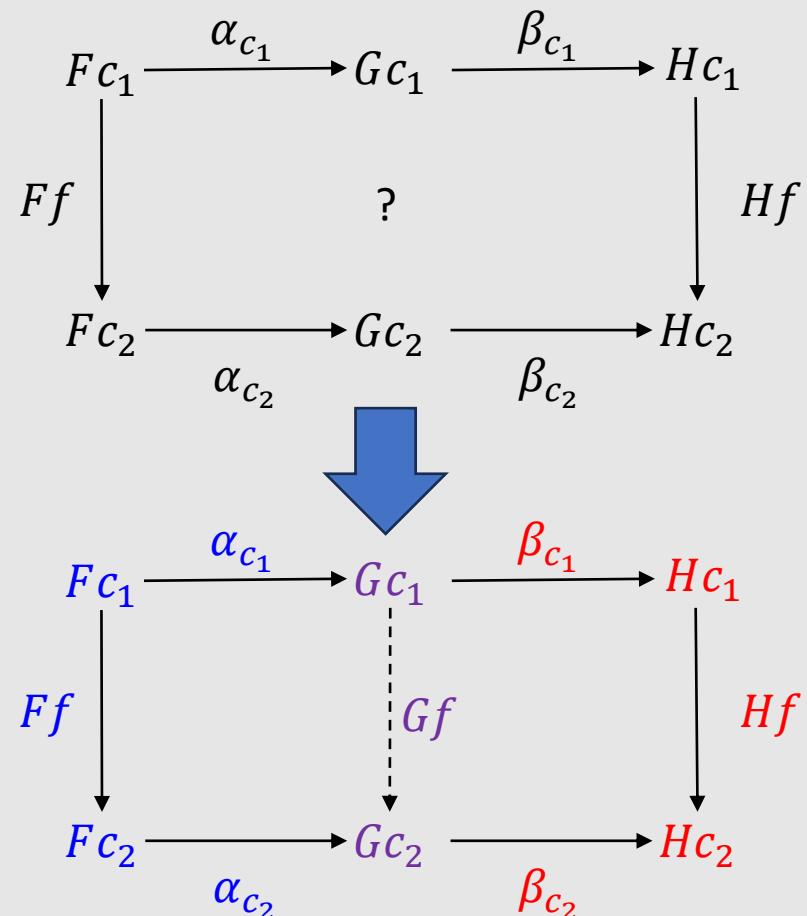
Diagrammatic proof

$$\begin{array}{ccccc} Fc_1 & \xrightarrow{\alpha_{c_1}} & Gc_1 & \xrightarrow{\beta_{c_1}} & Hc_1 \\ \downarrow Ff & & \downarrow ? & & \downarrow Hf \\ Fc_2 & \xrightarrow{\alpha_{c_2}} & Gc_2 & \xrightarrow{\beta_{c_2}} & Hc_2 \end{array}$$

# Natural transformations compose:

If  $\alpha: F \Rightarrow G$  and  $\beta: G \Rightarrow H$  are natural, then so is  $(\beta_c \circ \alpha_c:Fc \rightarrow Hc)_c$

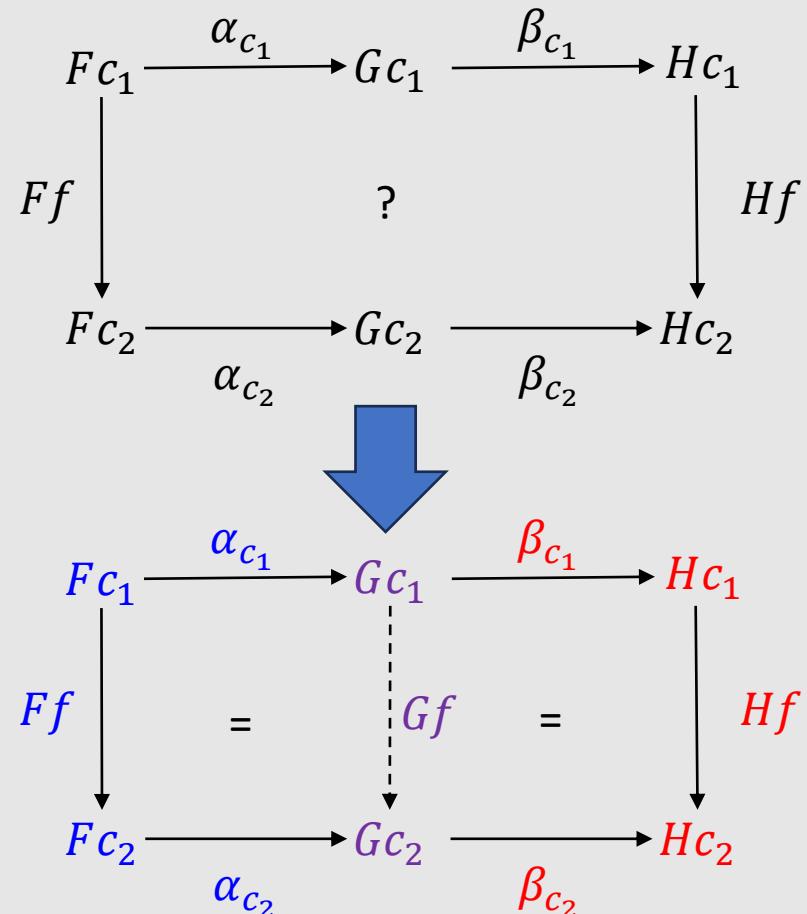
Diagrammatic proof



# Natural transformations compose:

If  $\alpha: F \Rightarrow G$  and  $\beta: G \Rightarrow H$  are natural, then so is  $(\beta_c \circ \alpha_c: Fc \rightarrow Hc)_c$

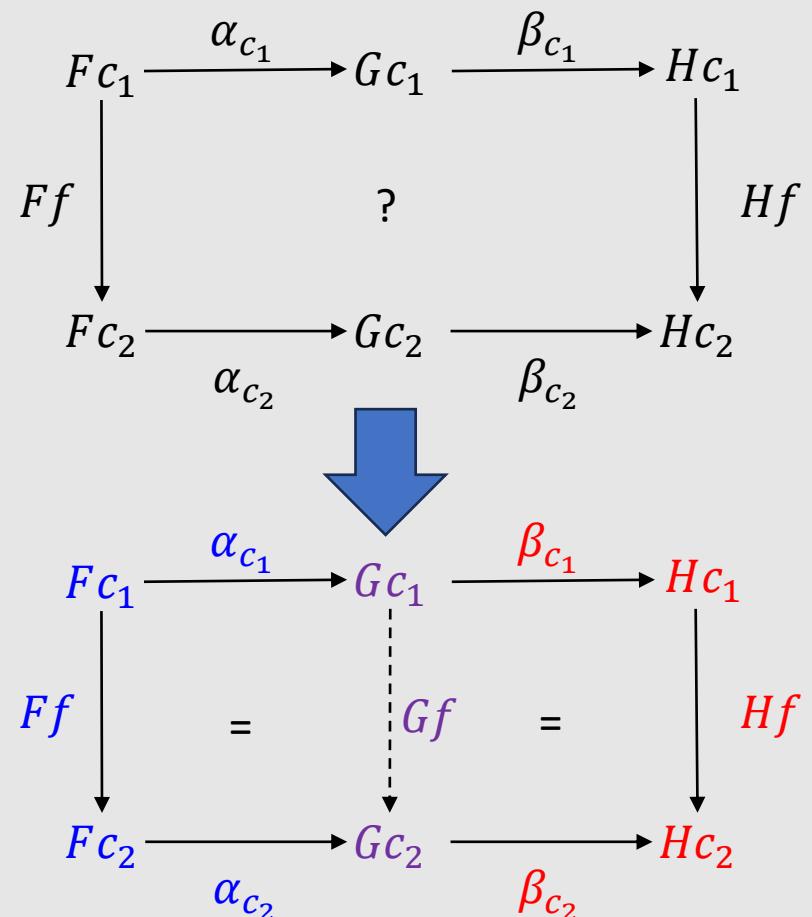
Diagrammatic proof



# Natural transformations compose:

If  $\alpha: F \Rightarrow G$  and  $\beta: G \Rightarrow H$  are natural, then so is  $(\beta_c \circ \alpha_c:Fc \rightarrow Hc)_c$

Diagrammatic proof



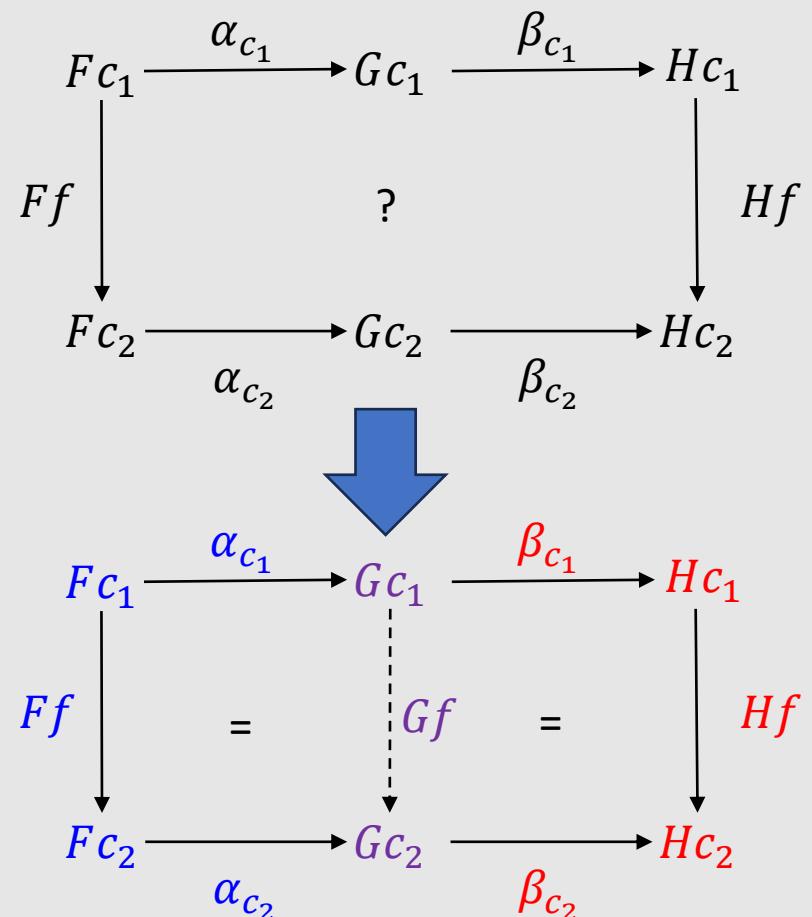
Computer-friendly proof

$$\begin{aligned}
 & Hf \circ \beta_{c_1} \circ \alpha_{c_1} \\
 = & \beta_{c_2} \circ Gf \circ \alpha_{c_1} \\
 = & \beta_{c_2} \circ \alpha_{c_2} \circ Ff
 \end{aligned}$$

# Natural transformations compose:

If  $\alpha: F \Rightarrow G$  and  $\beta: G \Rightarrow H$  are natural, then so is  $(\beta_c \circ \alpha_c:Fc \rightarrow Hc)_c$

Diagrammatic proof



Computer-friendly proof

$$\begin{aligned} & Hf \circ \beta_{c_1} \circ \alpha_{c_1} \\ &= \beta_{c_2} \circ Gf \circ \alpha_{c_1} \\ &= \beta_{c_2} \circ \alpha_{c_2} \circ Ff \end{aligned}$$

# Natural transformations compose:

If  $\alpha: F \Rightarrow G$  and  $\beta: G \Rightarrow H$  are natural, then so is  $(\beta_c \circ \alpha_c: Fc \rightarrow Hc)_c$

Diagrammatic proof

$$\begin{array}{ccccc}
 Fc_1 & \xrightarrow{\alpha_{c_1}} & Gc_1 & \xrightarrow{\beta_{c_1}} & Hc_1 \\
 \downarrow Ff & & \downarrow ? & & \downarrow Hf \\
 Fc_2 & \xrightarrow{\alpha_{c_2}} & Gc_2 & \xrightarrow{\beta_{c_2}} & Hc_2 \\
 & & \downarrow \text{blue arrow} & & \\
 & & Fc_1 & \xrightarrow{\alpha_{c_1}} & Gc_1 \xrightarrow{\beta_{c_1}} Hc_1 \\
 & & \downarrow Ff & = & \downarrow Hf \\
 & & Fc_2 & \xrightarrow{\alpha_{c_2}} & Gc_2 \xrightarrow{\beta_{c_2}} Hc_2
 \end{array}$$

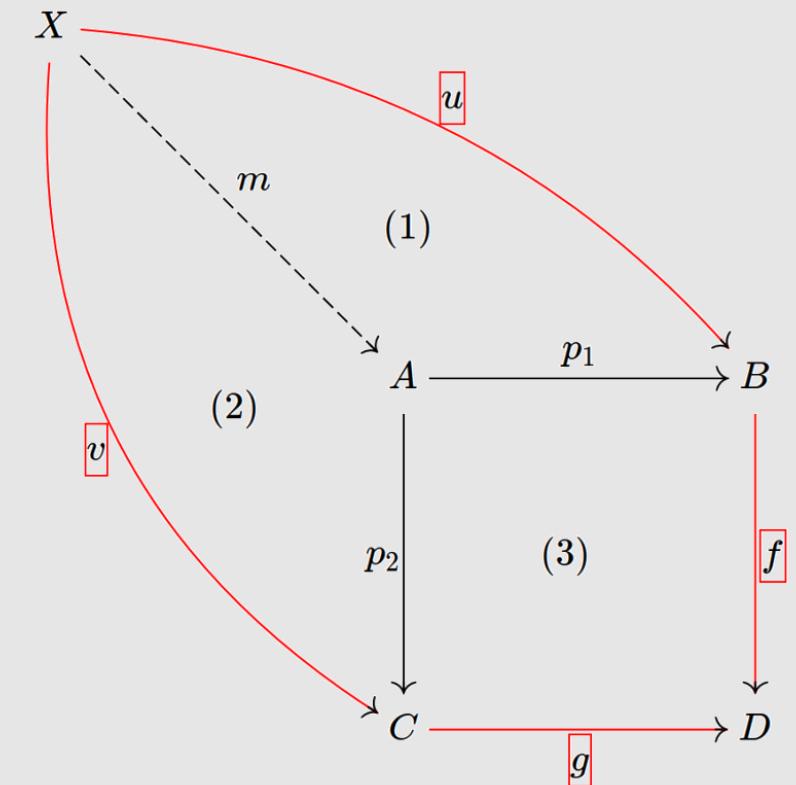
Automatic generation?

Computer-friendly proof

$$\begin{aligned}
 & Hf \circ \beta_{c_1} \circ \alpha_{c_1} \\
 & = \beta_{c_2} \circ Gf \circ \alpha_{c_1} \\
 & = \beta_{c_2} \circ \alpha_{c_2} \circ Ff
 \end{aligned}$$

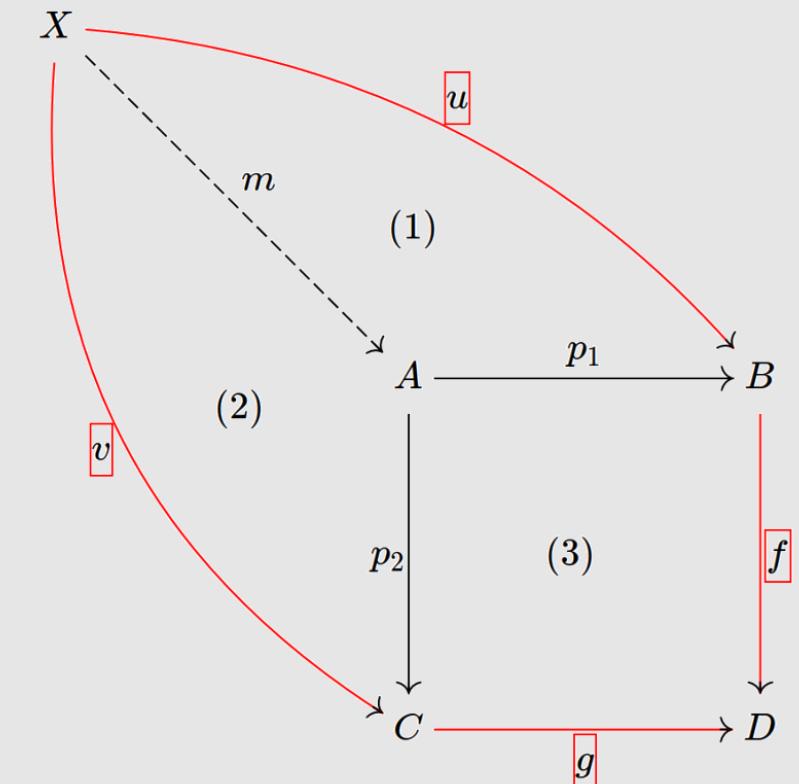


# Proof generation: sketch of the algorithm



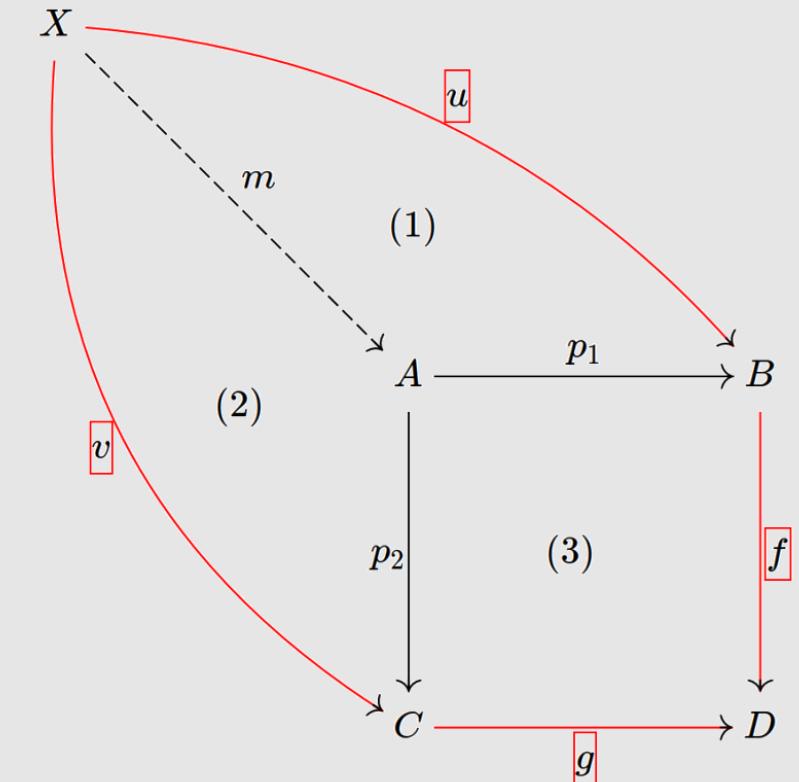
# Proof generation: sketch of the algorithm

- 1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”



# Proof generation: sketch of the algorithm

- 1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”  
(1)  $u \rightarrow p_1 \circ m$

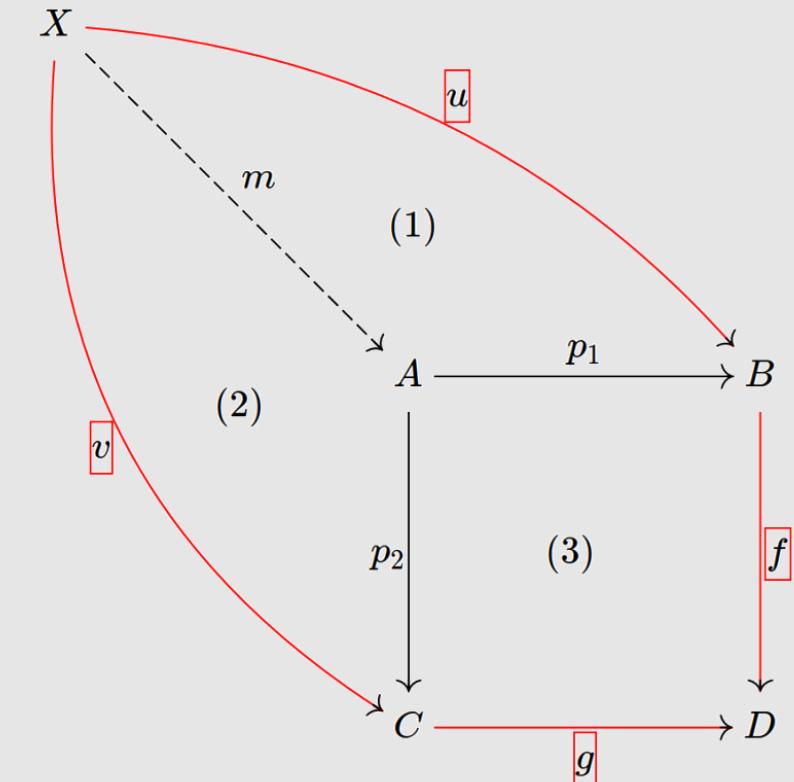


# Proof generation: sketch of the algorithm

- 1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”

$$(1) u \rightarrow p_1 \circ m$$

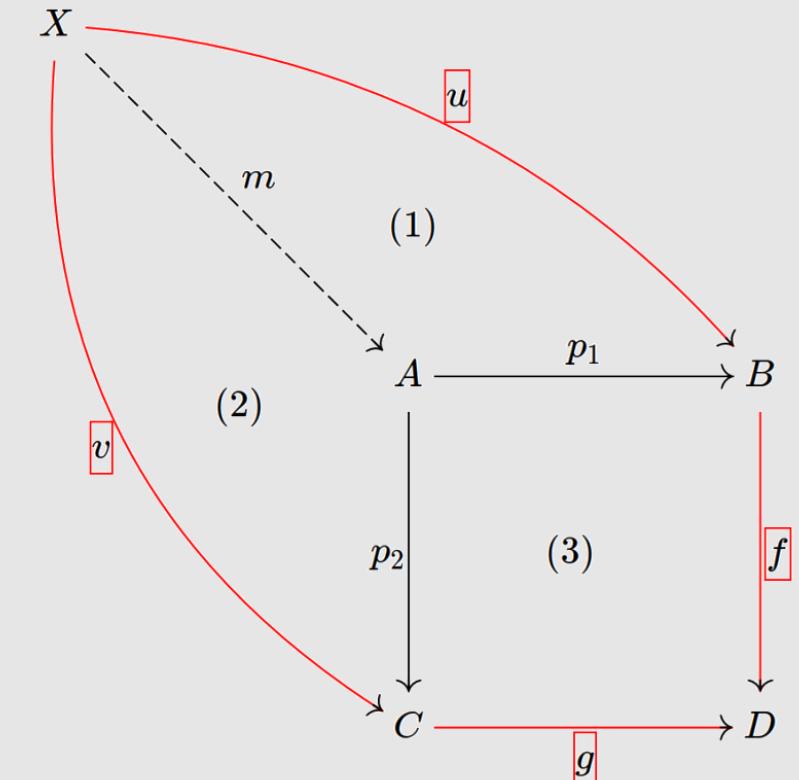
$$(2) p_2 \circ m \rightarrow v$$



# Proof generation: sketch of the algorithm

- 1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”

$$\begin{aligned}(1) \quad & u \rightarrow p_1 \circ m \\(2) \quad & p_2 \circ m \rightarrow v \\(3) \quad & f \circ p_1 \rightarrow g \circ p_2\end{aligned}$$

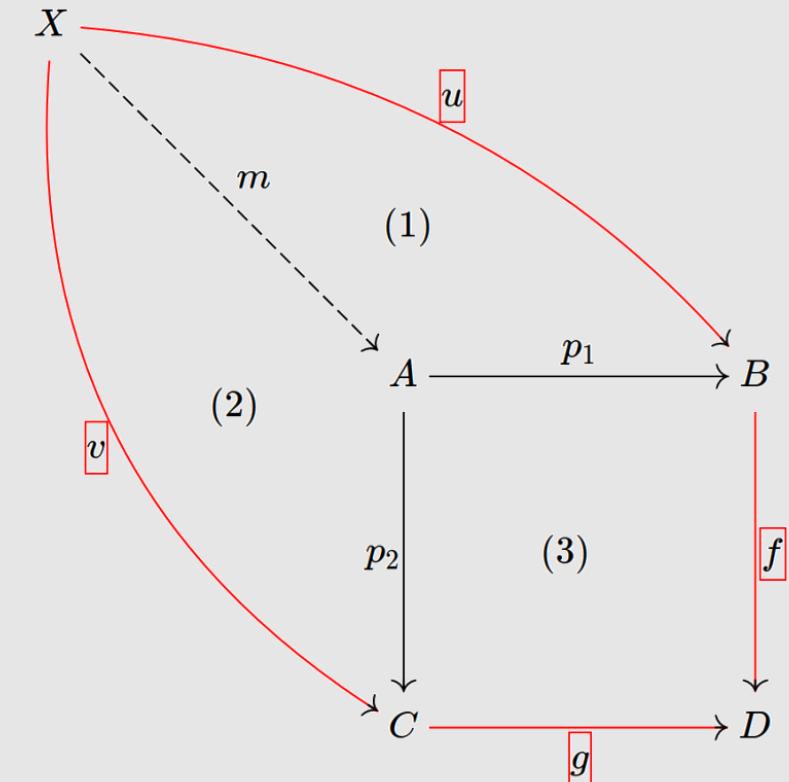


# Proof generation: sketch of the algorithm

1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”

- (1)  $u \rightarrow p_1 \circ m$
- (2)  $p_2 \circ m \rightarrow v$
- (3)  $f \circ p_1 \rightarrow g \circ p_2$

2) Identify the top right branch of the outer diagram.



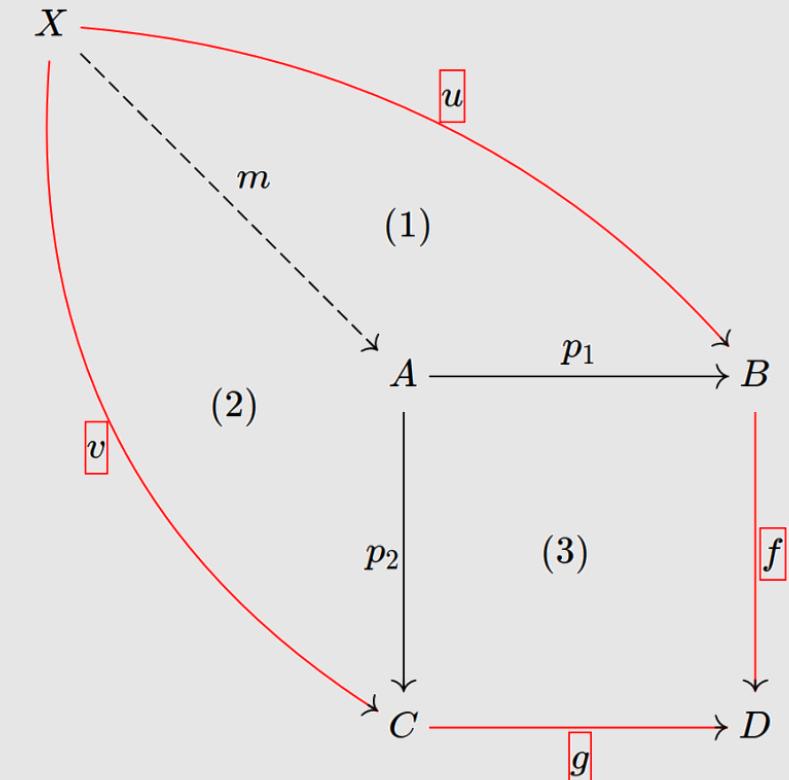
# Proof generation: sketch of the algorithm

1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”

- (1)  $u \rightarrow p_1 \circ m$
- (2)  $p_2 \circ m \rightarrow v$
- (3)  $f \circ p_1 \rightarrow g \circ p_2$

2) Identify the top right branch of the outer diagram.

$$f \circ u$$



# Proof generation: sketch of the algorithm

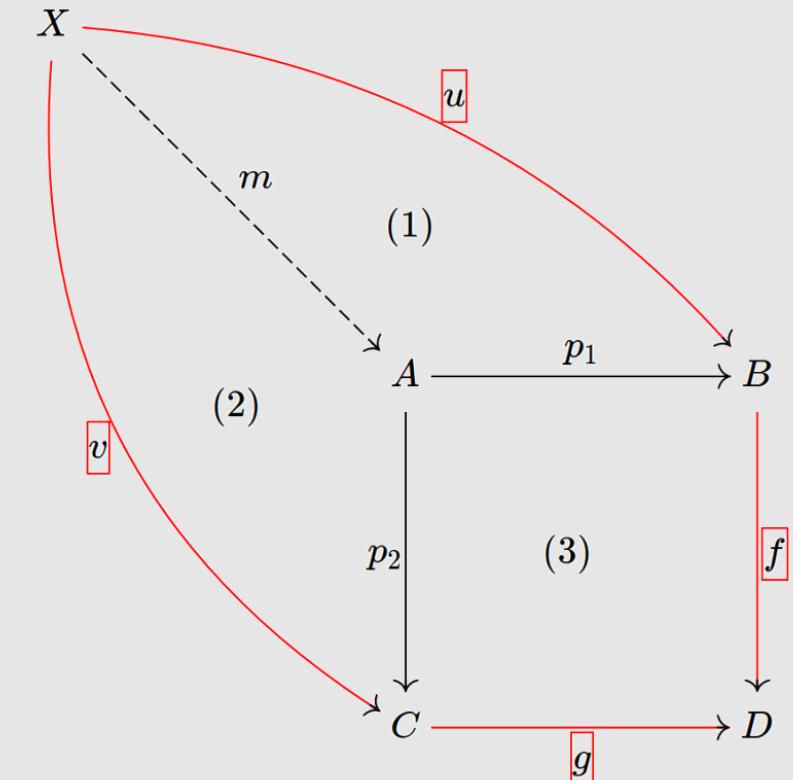
1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”

- (1)  $u \rightarrow p_1 \circ m$
- (2)  $p_2 \circ m \rightarrow v$
- (3)  $f \circ p_1 \rightarrow g \circ p_2$

2) Identify the top right branch of the outer diagram.

$$f \circ u$$

3) Use greedily the rewrite rules until reaching the bottom left branch



# Proof generation: sketch of the algorithm

1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”

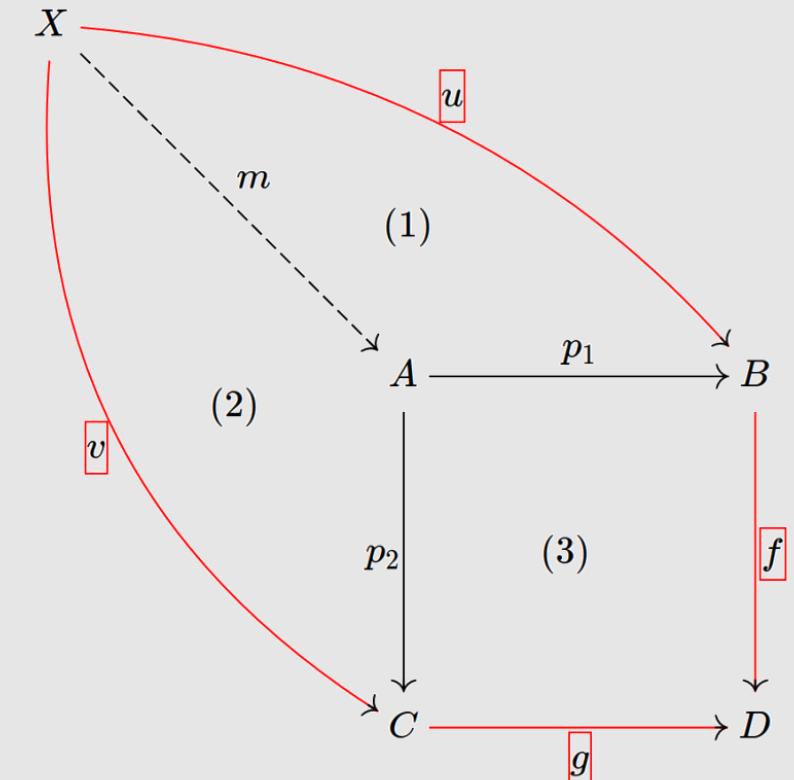
- (1)  $u \rightarrow p_1 \circ m$
- (2)  $p_2 \circ m \rightarrow v$
- (3)  $f \circ p_1 \rightarrow g \circ p_2$

2) Identify the top right branch of the outer diagram.

$$f \circ u$$

3) Use greedily the rewrite rules until reaching the bottom left branch

$$f \circ u \xrightarrow{(1)} f \circ p_1 \circ m \xrightarrow{(3)} g \circ p_2 \circ m \xrightarrow{(2)} g \circ v$$



# Proof generation: sketch of the algorithm

1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”

- (1)  $u \rightarrow p_1 \circ m$
- (2)  $p_2 \circ m \rightarrow v$
- (3)  $f \circ p_1 \rightarrow g \circ p_2$

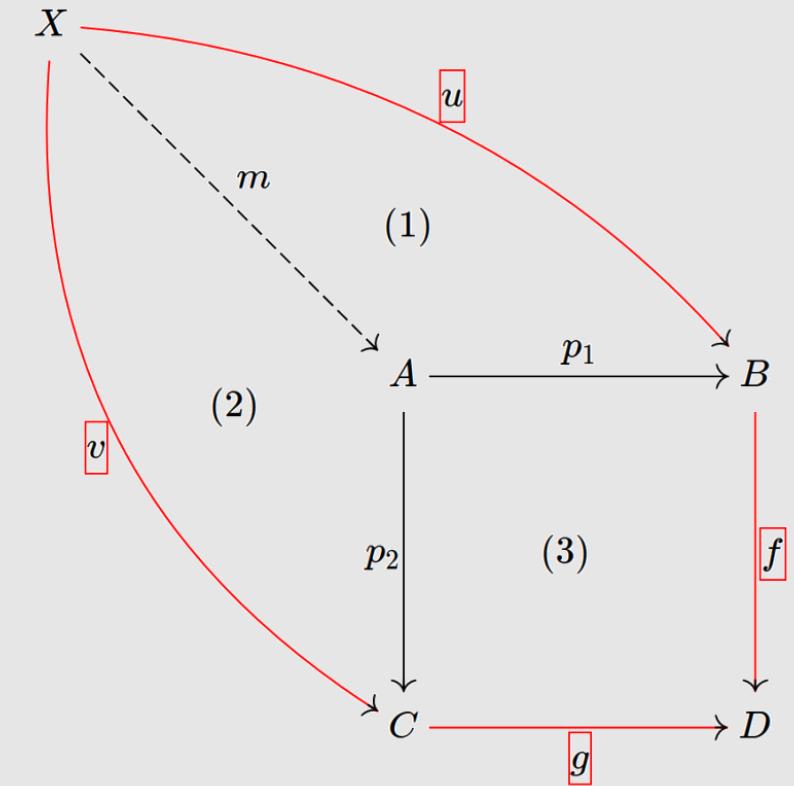
2) Identify the top right branch of the outer diagram.

$$f \circ u$$

3) Use greedily the rewrite rules until reaching the bottom left branch

$$f \circ u \xrightarrow{(1)} f \circ p_1 \circ m \xrightarrow{(3)} g \circ p_2 \circ m \xrightarrow{(2)} g \circ v$$

4) Remember the rewrite steps and generate the coq proof script accordingly.



# Proof generation: sketch of the algorithm

1) Save all subdiagrams as rewrite rules: “top right branch”  $\rightarrow$  “bottom left branch”

- (1)  $u \rightarrow p_1 \circ m$
- (2)  $p_2 \circ m \rightarrow v$
- (3)  $f \circ p_1 \rightarrow g \circ p_2$

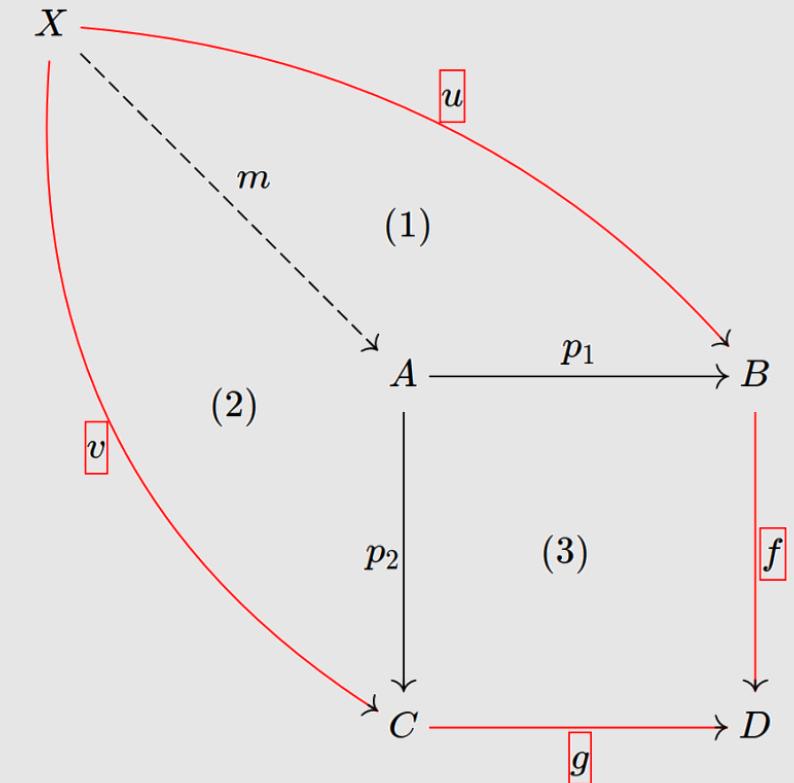
2) Identify the top right branch of the outer diagram.

$$f \circ u$$

3) Use greedily the rewrite rules until reaching the bottom left branch

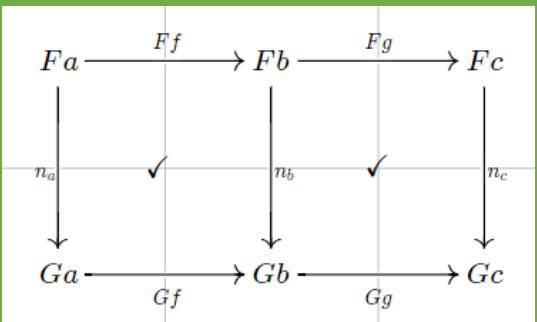
$$f \circ u \xrightarrow{(1)} f \circ p_1 \circ m \xrightarrow{(3)} g \circ p_2 \circ m \xrightarrow{(2)} g \circ v$$

4) Remember the rewrite steps and generate the coq proof script accordingly.  
If a Coq proof is provided inside a subdiagram, use it to justify the rewrite step.



# Architecture

Diagram editor  
(standalone version)



Generate proofs

Custom  
vscode  
extension  
(building  
upon  
coq-lsp)

Visual Studio Code



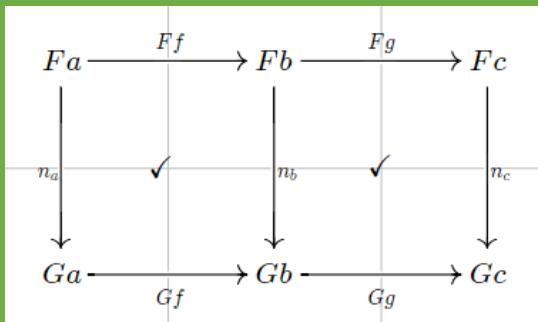
```
goals.v:30:9
  ▼Goals (1)
    ▼Goal (1)
      1 = 1 /\ (21 = 21 /\ 22 = 22) /\ (3 = 3).
      1 = 1 /\ (21 = 21 /\ 22 = 22) /-
      ▷ Messages (0)
      ▷ Error Browser
        The variable Qed was not found in this scope.
```

The screenshot shows a Coq proof script in the VS Code editor. The script includes a lemma named 'failingBullet' with a failing proof attempt involving a 'split' tactic. It also includes a lemma 'failingsubProof' with a failing proof attempt involving an 'apply' tactic. A message in the 'Error Browser' panel indicates that the variable 'Qed' was not found in the current scope. The Coq logo is visible in the bottom right corner of the code editor.

(+ Coq library for custom notations)

# Architecture

Diagram editor  
(standalone version)



Generate proofs

Show diagram  
under cursor

Custom  
vscode  
extension  
(building  
upon  
coq-lsp)

Visual Studio Code

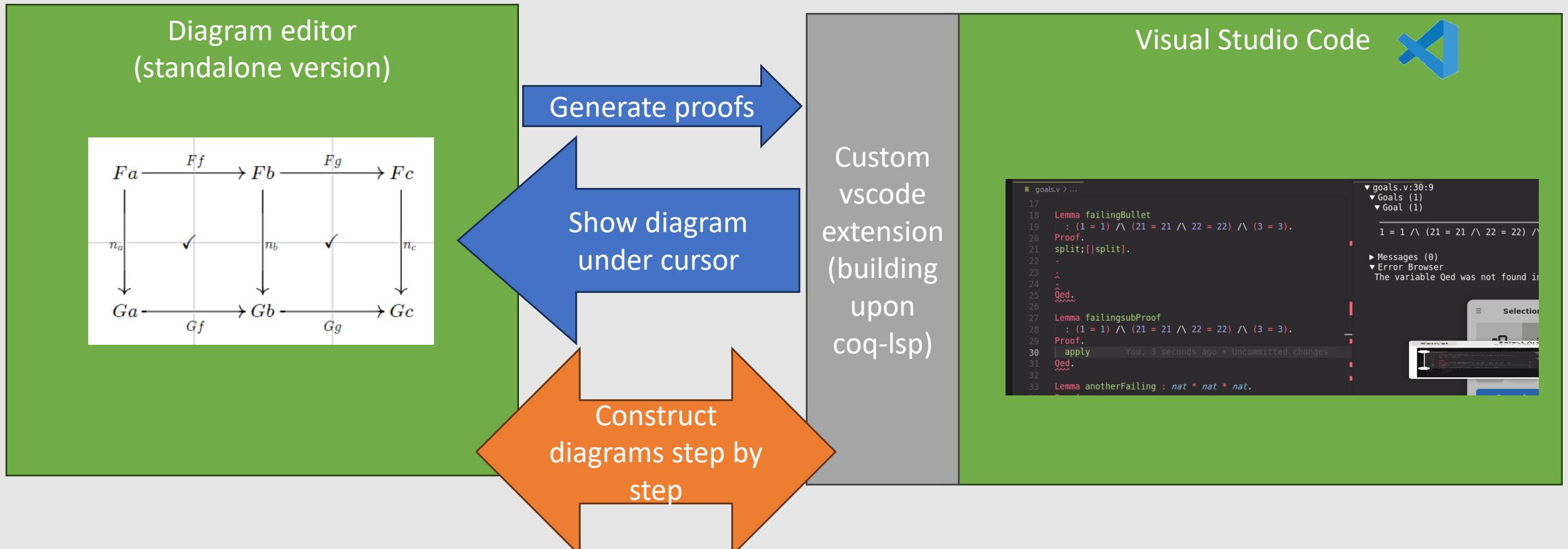


```
goals.v:30:9
  ▼Goals (1)
    ▼Goal (1)
      1 = 1 /\ (21 = 21 /\ 22 = 22) /\ (3 = 3).
      |
      QED was not found in the current context.
      |
      You, 3 seconds ago * Uncommitted changes
```

The screenshot shows a Coq proof script in the main editor area. The script includes a lemma named 'failingBullet' with a failing proof attempt involving a 'split' tactic. The 'Error Browser' panel on the right displays an error message: 'The variable Qed was not found in the current context.' Below the editor, a status bar indicates 'You, 3 seconds ago \* Uncommitted changes'.

(+ Coq library for custom notations)

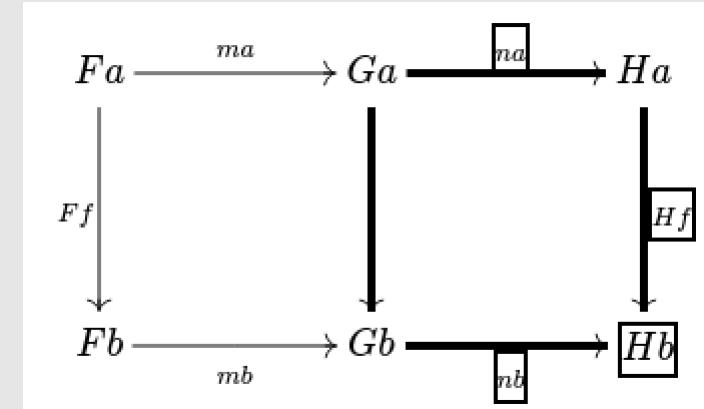
# Architecture



(+ Coq library for custom notations)

# Construction of a diagrammatic proof with Coq + YADE

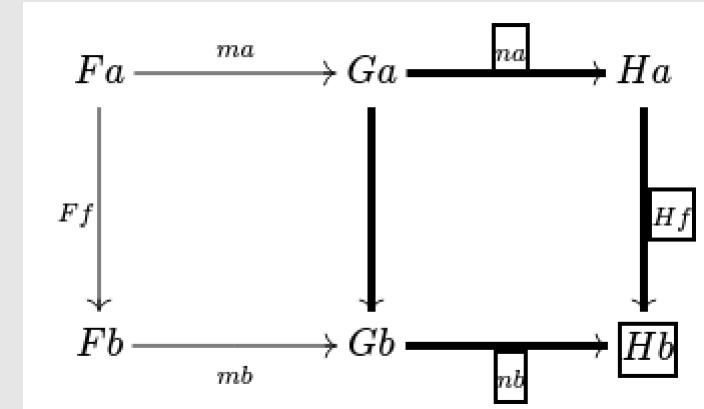
1) Select a subdiagram



# Construction of a diagrammatic proof with Coq + YADE

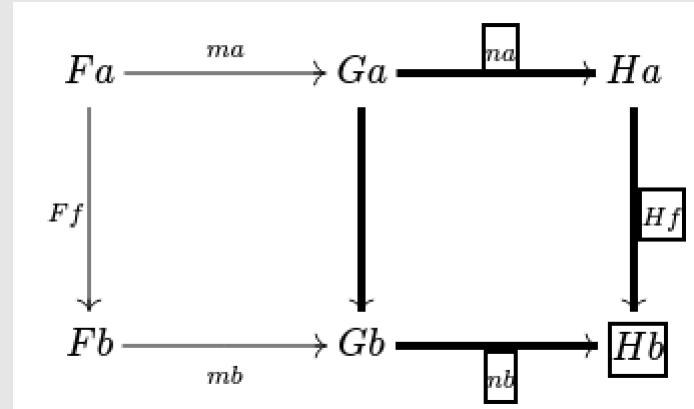
1) Select a subdiagram

2) Execute the export command in the diagram editor



# Construction of a diagrammatic proof with Coq + YADE

- 1) Select a subdiagram
- 2) Execute the export command in the diagram editor  
 $\Rightarrow$  vscode asks for a Coq proof of equality



# Construction of a diagrammatic proof with Coq + YADE

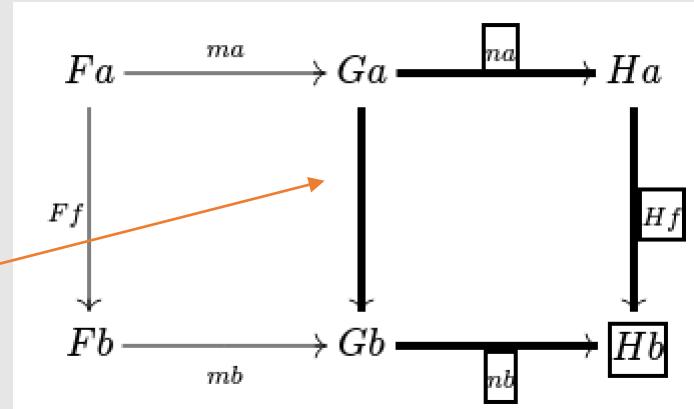
1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality

```
eassert (yade : { {n a} · H f = { _ } · {n b} }).  
{  
(* Coq proof to be written here by the user *)  
}
```

*unnamed arrow*



# Construction of a diagrammatic proof with Coq + YADE

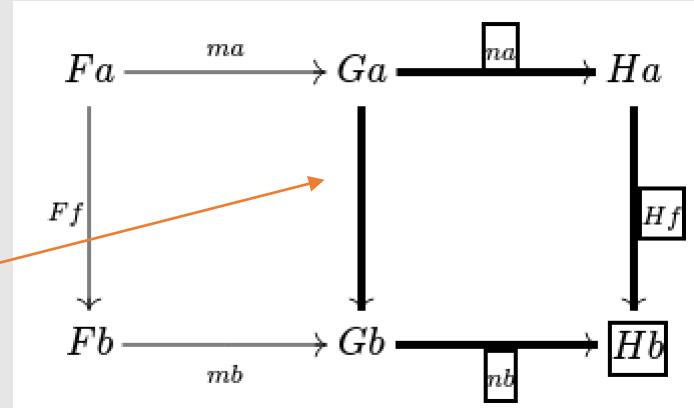
1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality

```
eassert (yade : { {n a} . H f = { _ } . { {n b} } }).  
{  
(* Coq proof to be written here by the user *)  
}
```

*unnamed arrow*



4) Complete the proof (here, by invoking naturality of  $n$ )

# Construction of a diagrammatic proof with Coq + YADE

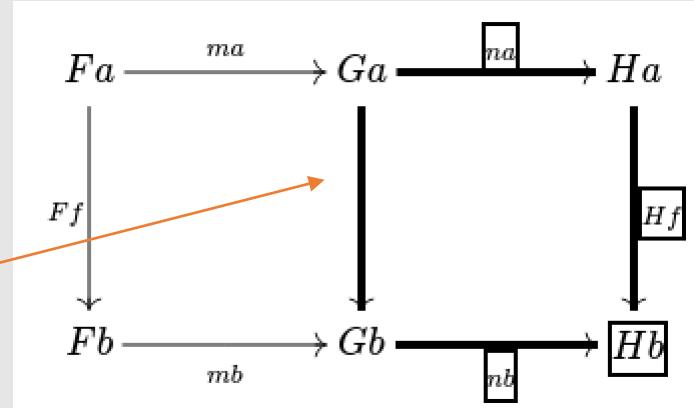
1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality

```
eassert (yade : { {n a} . H f = { _ } . { {n b} } }).  
{  
(* Coq proof to be written here by the user *)  
}
```

*unnamed arrow*



4) Complete the proof (here, by invoking naturality of  $n$ )

5) Execute the export command in vscode

# Construction of a diagrammatic proof with Coq + YADE

1) Select a subdiagram

2) Execute the export command in the diagram editor

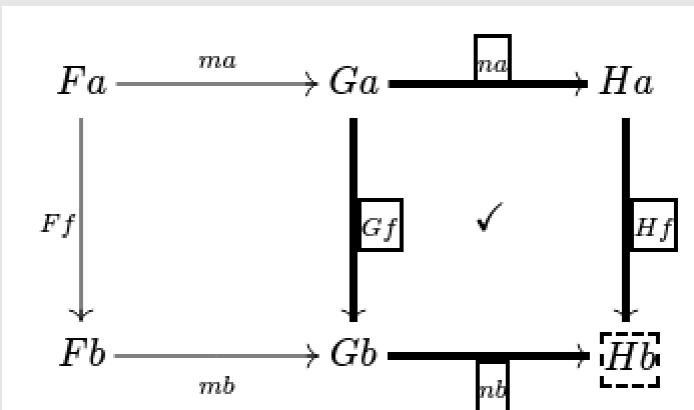
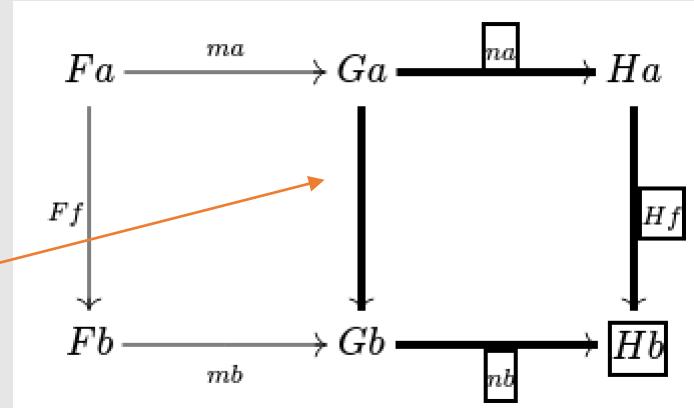
⇒ vscode asks for a Coq proof of equality

```
eassert (yade : { {n a} . H f = { _ } . {n b} }).  
{  
(* Coq proof to be written here by the user *)  
}
```

4) Complete the proof (here, by invoking naturality of  $n$ )

5) Execute the export command in vscode

⇒ The diagram gets completed in the editor window:



# Construction of a diagrammatic proof with Coq + YADE

1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality

```
eassert (yade : { {n a} . H f = { _ } . {n b} }).  
{  
(* Coq proof to be written here by the user *)  
}
```

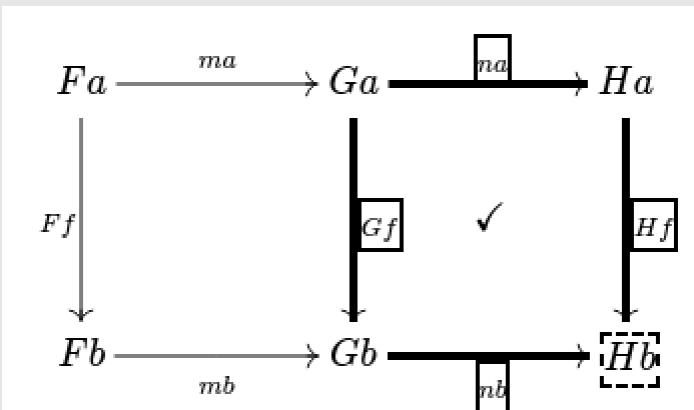
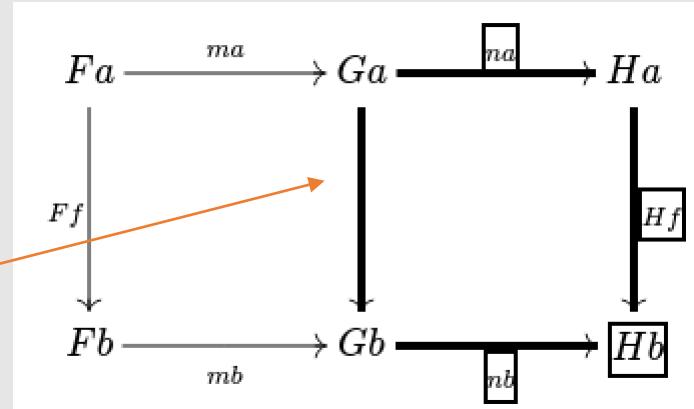
4) Complete the proof (here, by invoking naturality of  $n$ )

5) Execute the export command in vscode

⇒ The diagram gets completed in the editor window:

- The label of the unnamed arrow is inferred

*unnamed arrow*



# Construction of a diagrammatic proof with Coq + YADE

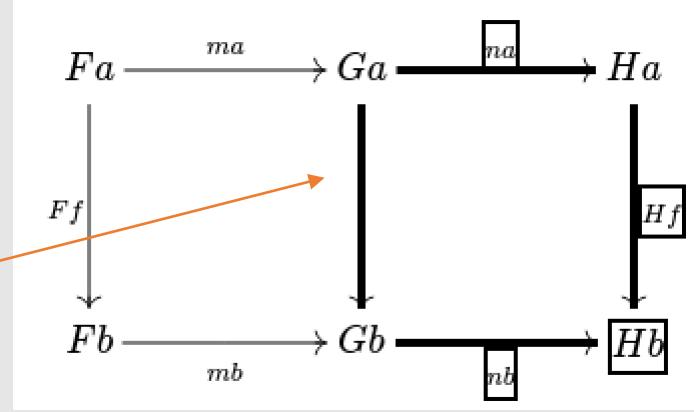
1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality

```
eassert (yade : { {n a} . H f = { _ } . {n b} }).  
{  
(* Coq proof to be written here by the user *)  
}
```

*unnamed arrow*

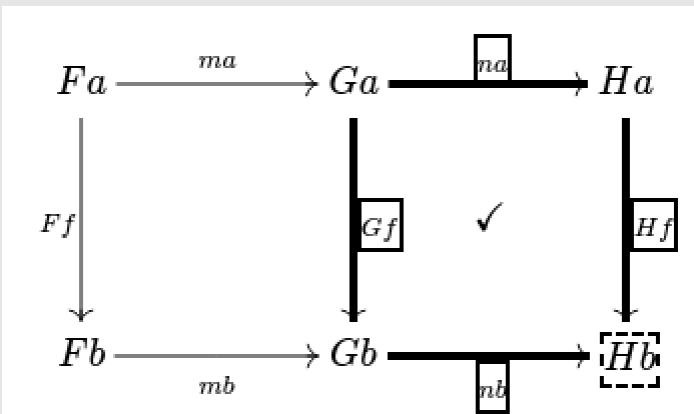


4) Complete the proof (here, by invoking naturality of  $n$ )

5) Execute the export command in vscode

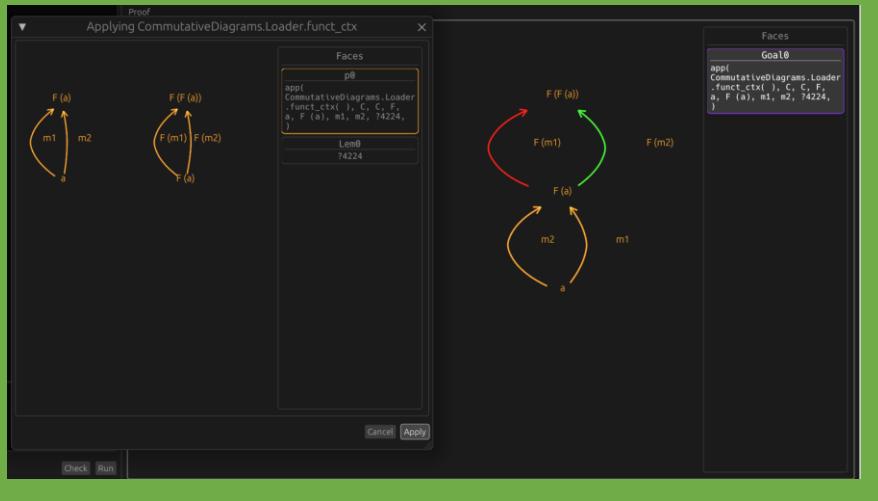
⇒ The diagram gets completed in the editor window:

- The label of the unnamed arrow is inferred
- The Coq proof is saved in the diagram (indicated by ✓)



# Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs

Graphical interface  
(standalone program,  
implemented in Rust)

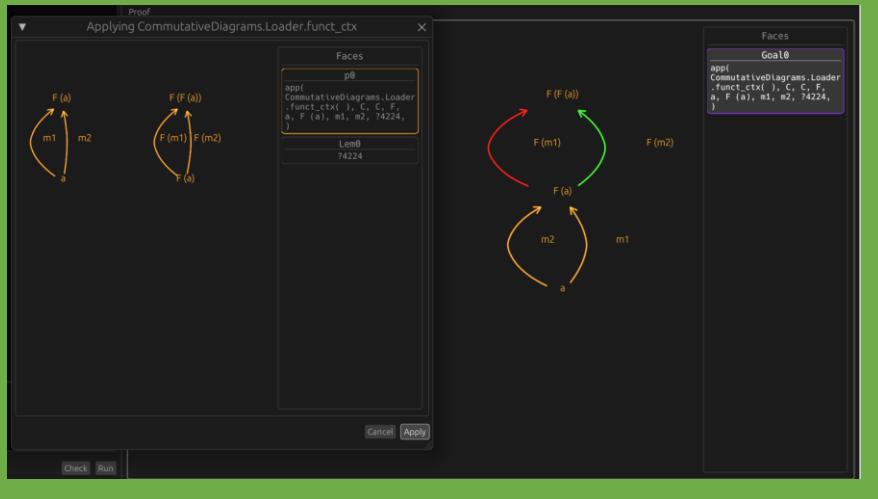


Coq  
plugin  
(ocaml)



# Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs

Graphical interface  
(standalone program,  
implemented in Rust)

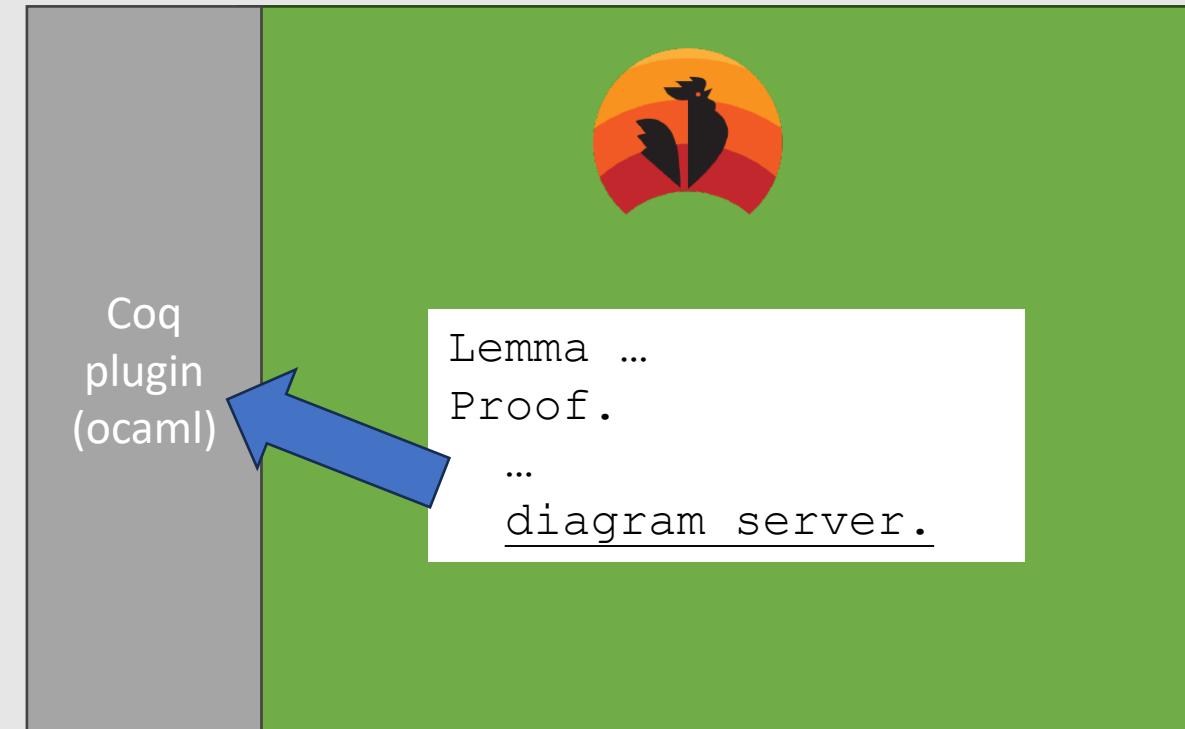
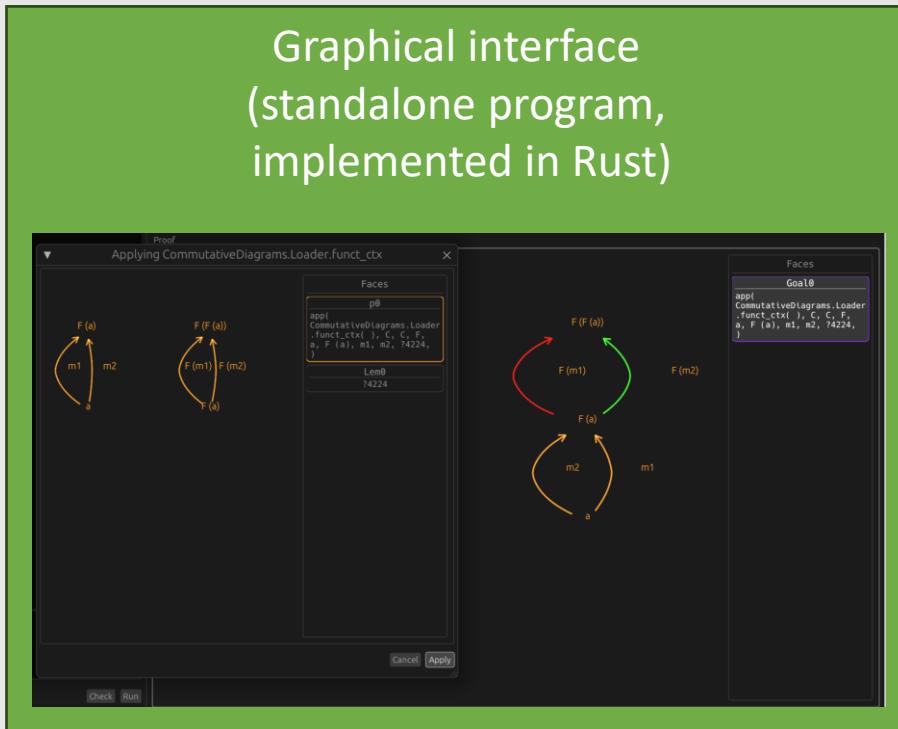


Coq  
plugin  
(ocaml)

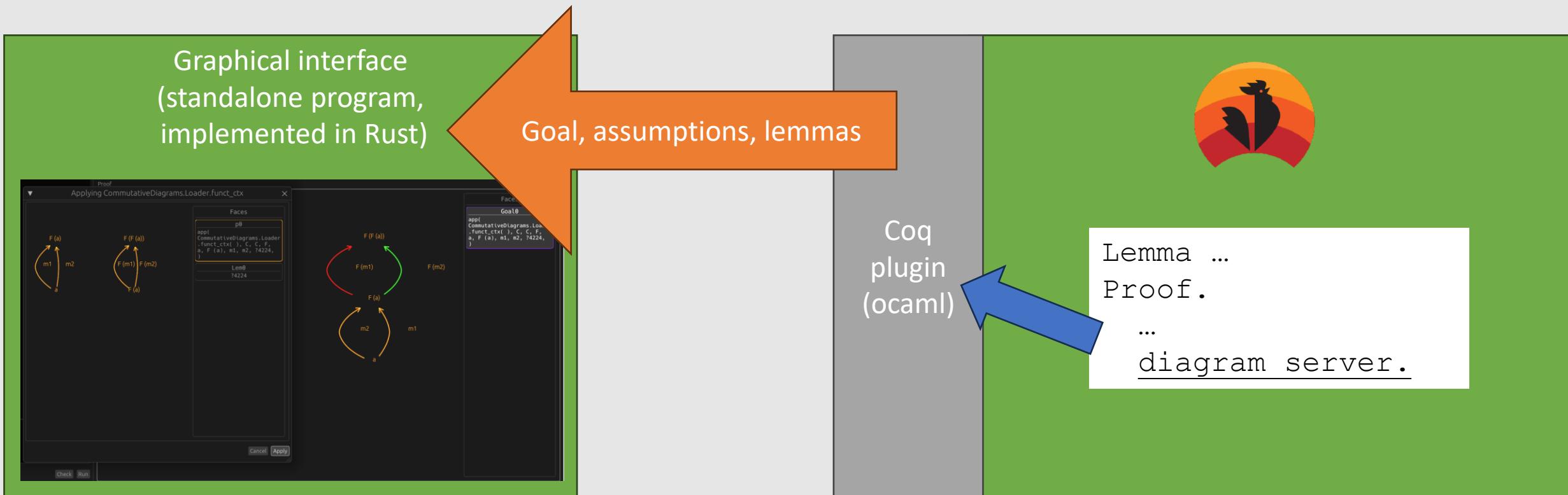


Lemma ...  
Proof.  
...  
diagram server.

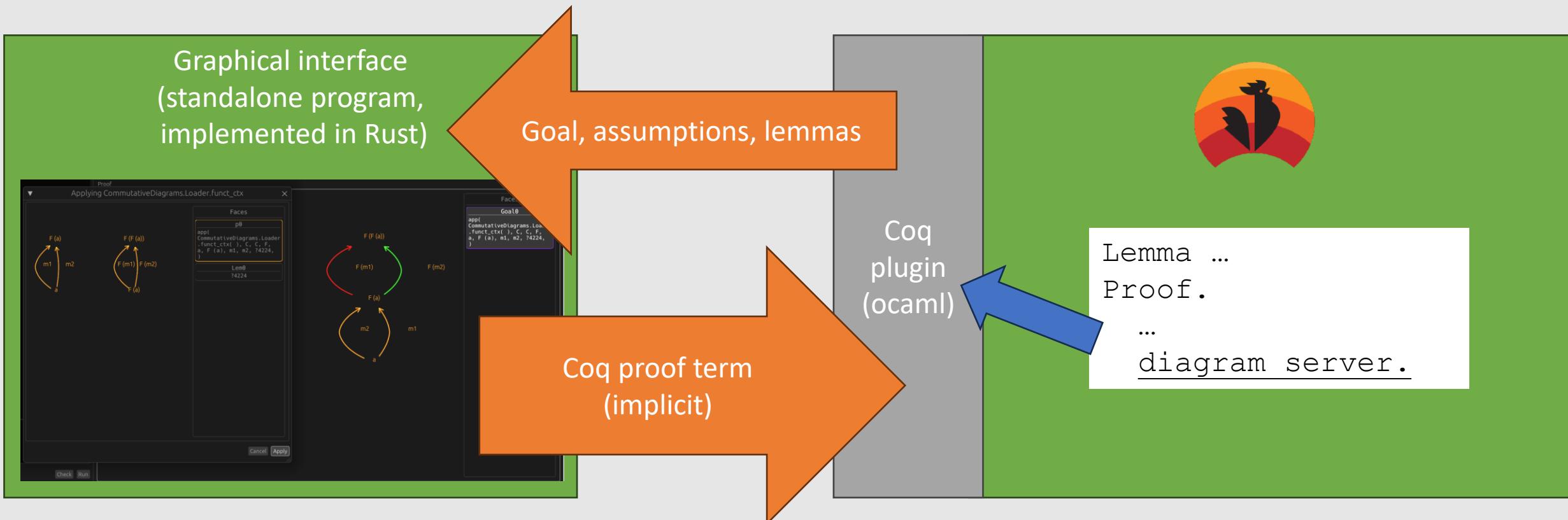
# Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs



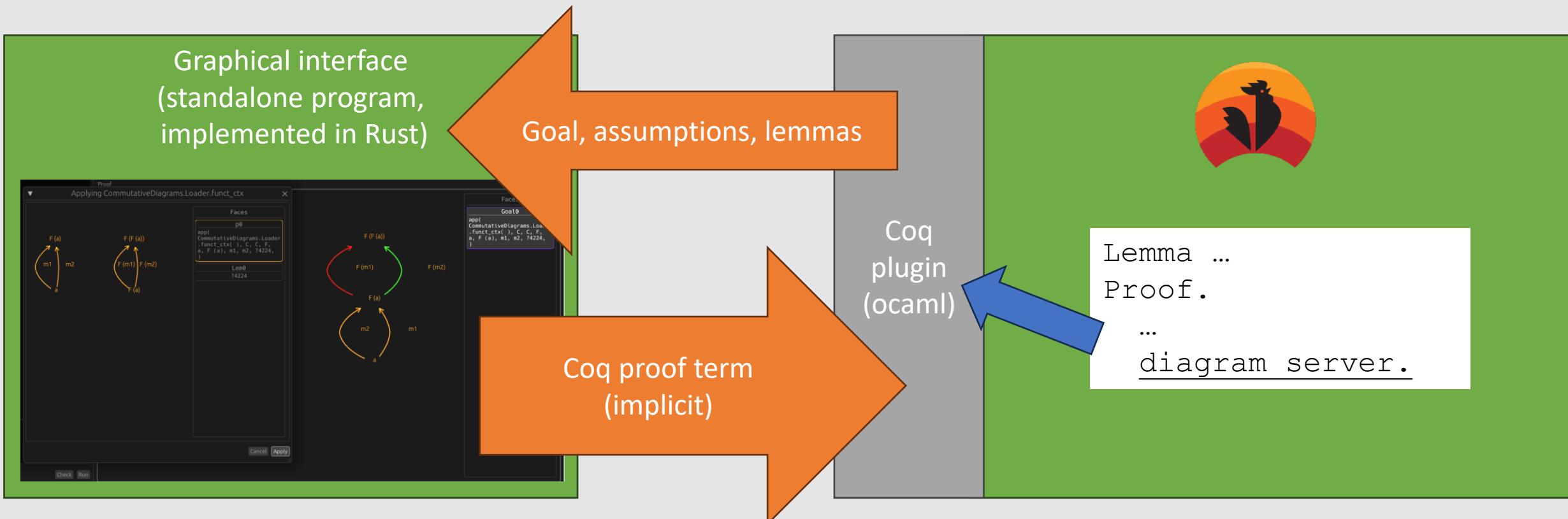
# Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs



# Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs

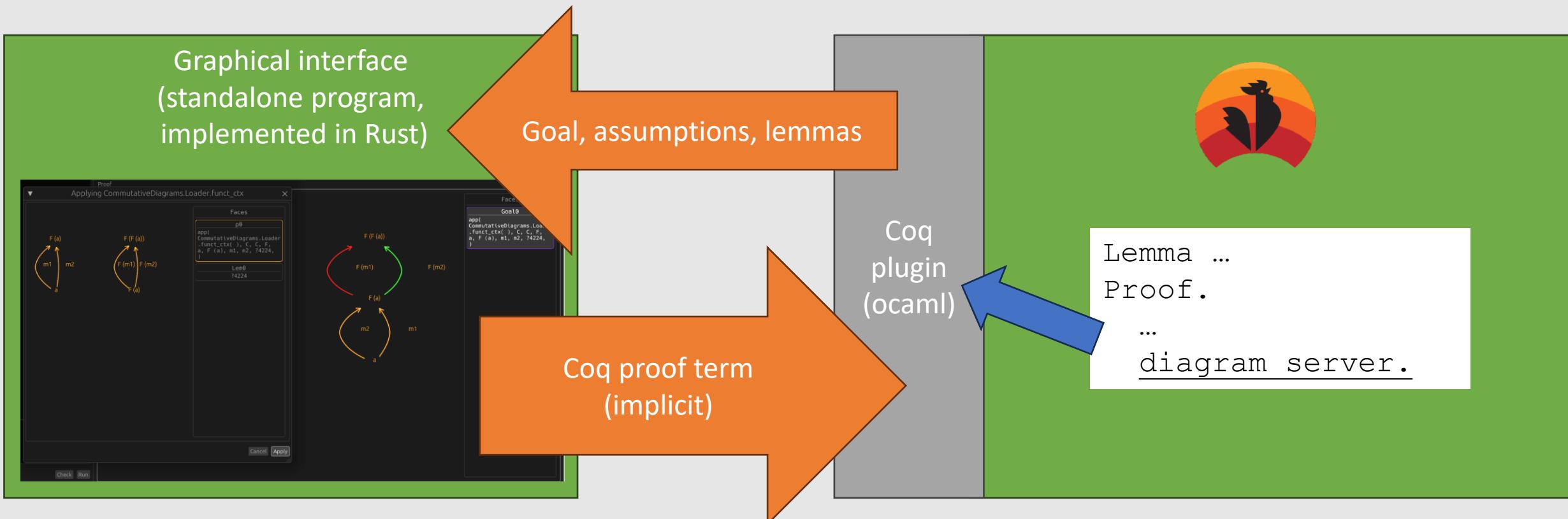


# Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs



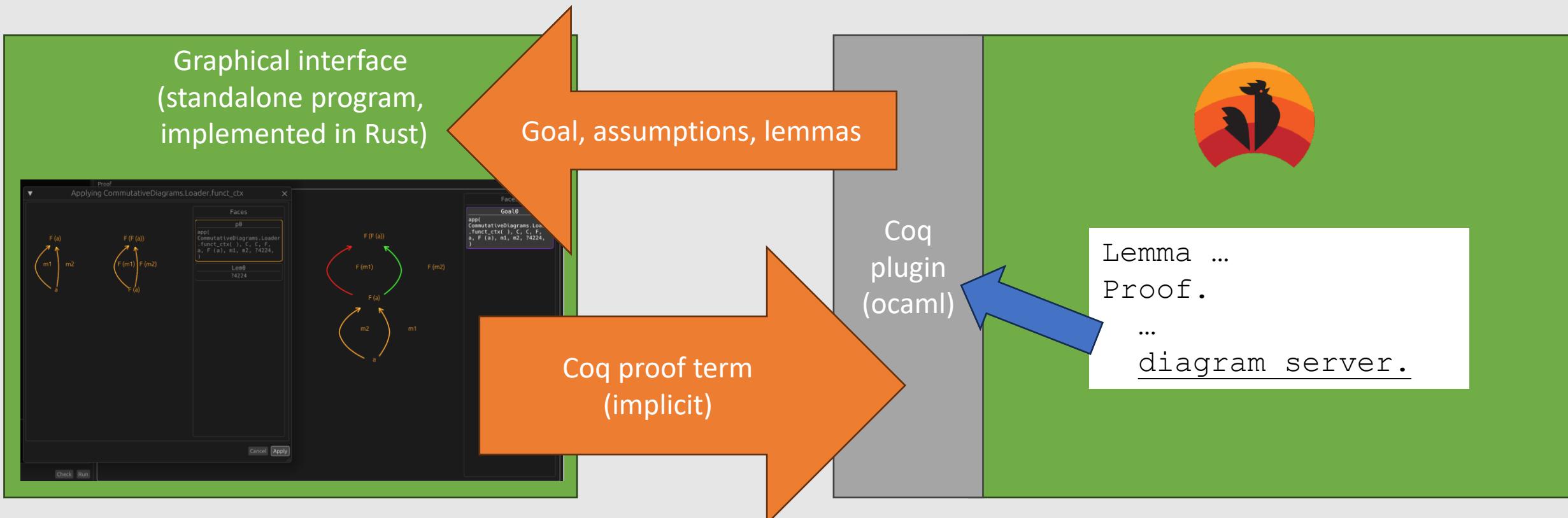
- Automatic layout (limited manual editing)

# Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs



- Automatic layout (limited manual editing)
- Proof tactics (including invoking a lemma)

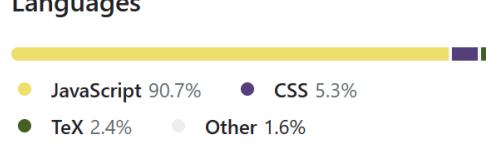
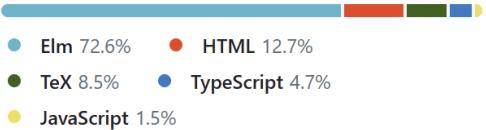
# Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs



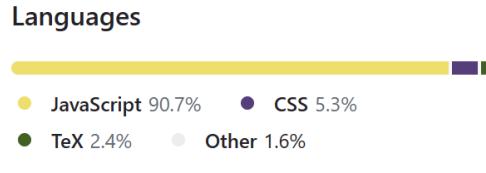
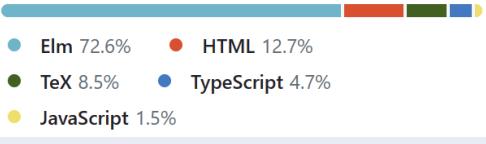
- Automatic layout (limited manual editing)
- Proof tactics (including invoking a lemma)

I plan to use his plugin in the future  
(to apply Coq lemmas directly in the editor)

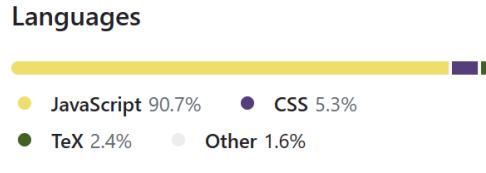
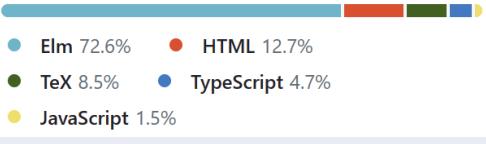
# Comparison with quiver

	Quiver	YADE											
Programming Languages	 <table border="1"> <thead> <tr> <th>Languages</th></tr> </thead> <tbody> <tr> <td>JavaScript 90.7%</td></tr> <tr> <td>CSS 5.3%</td></tr> <tr> <td>TeX 2.4%</td></tr> <tr> <td>Other 1.6%</td></tr> </tbody> </table>	Languages	JavaScript 90.7%	CSS 5.3%	TeX 2.4%	Other 1.6%	 <table border="1"> <thead> <tr> <th>Languages</th></tr> </thead> <tbody> <tr> <td>Elm 72.6%</td></tr> <tr> <td>HTML 12.7%</td></tr> <tr> <td>TeX 8.5%</td></tr> <tr> <td>TypeScript 4.7%</td></tr> <tr> <td>JavaScript 1.5%</td></tr> </tbody> </table>	Languages	Elm 72.6%	HTML 12.7%	TeX 8.5%	TypeScript 4.7%	JavaScript 1.5%
Languages													
JavaScript 90.7%													
CSS 5.3%													
TeX 2.4%													
Other 1.6%													
Languages													
Elm 72.6%													
HTML 12.7%													
TeX 8.5%													
TypeScript 4.7%													
JavaScript 1.5%													
Styling options	+	-											
User-friendly	+	-											
Editing features	-	+ Tabs, Copy & paste, find & replace, expand selection to connected components, ...											
LaTeX export	yes	yes											
A nice name	+	-											
Mechanisation features	-	+											

# Comparison with quiver

	Quiver	YADE
Programming Languages	 <p>Languages</p> <ul style="list-style-type: none"><li>JavaScript 90.7%</li><li>CSS 5.3%</li><li>TeX 2.4%</li><li>Other 1.6%</li></ul>	 <p>Languages</p> <ul style="list-style-type: none"><li>Elm 72.6%</li><li>HTML 12.7%</li><li>TeX 8.5%</li><li>TypeScript 4.7%</li><li>JavaScript 1.5%</li></ul>
Styling options	+	-
User-friendly	+	-
Editing features	-	+ Tabs, Copy & paste, find & replace, expand selection to connected components, ...
LaTeX export	yes	yes
A nice name	+	-
Mechanisation features	-	+
Popularity	+++ 	

# Comparison with quiver

	Quiver	YADE
Programming Languages	 <p>Languages</p> <ul style="list-style-type: none"><li>JavaScript 90.7%</li><li>CSS 5.3%</li><li>TeX 2.4%</li><li>Other 1.6%</li></ul>	 <p>Languages</p> <ul style="list-style-type: none"><li>Elm 72.6%</li><li>HTML 12.7%</li><li>TeX 8.5%</li><li>TypeScript 4.7%</li><li>JavaScript 1.5%</li></ul>
Styling options	+	-
User-friendly	+	-
Editing features	-	+ Tabs, Copy & paste, find & replace, expand selection to connected components, ...
LaTeX export	yes	yes
A nice name	+	-
Mechanisation features	-	+
Popularity	+++	Two users: me and Tom Hirschowitz

# Some future plans

- Improve the process for building diagrams using Coq + YADE  
(I still find it tedious)

# Some future plans

- Improve the process for building diagrams using Coq + YADE  
(I still find it tedious)
- Formalise categorical results.

# Some future plans

- Improve the process for building diagrams using Coq + YADE  
(I still find it tedious)
- Formalise categorical results.
- A collaborative online version the editor

# Some future plans

- Improve the process for building diagrams using Coq + YADE  
(I still find it tedious)
- Formalise categorical results.
- A collaborative online version the editor  
(requires a lot of engineering)

# Some future plans

- Improve the process for building diagrams using Coq + YADE  
(I still find it tedious)
- Formalise categorical results.
- A collaborative online version the editor  
(requires a lot of engineering)
- Support for mechanisation of 2-categorical diagrams, string diagrams

# How to try Coreact-YADE

Requirements for the full example available on github<sup>1</sup>

<sup>1</sup> <https://github.com/amblafont/vscode-yade-example>

# How to try Coreact-YADE

Requirements for the full example available on github<sup>1</sup>

Packaged as .deb (Debian amd64)  
or .img (MacOS arm64)

Coreact-YADE

<sup>1</sup> <https://github.com/amblafont/vscode-yade-example>

# How to try Coreact-YADE

Requirements for the full example available on github<sup>1</sup>

Coq

Packaged as .deb (Debian amd64)  
or .img (MacOS arm64)

Coreact-YADE

<sup>1</sup> <https://github.com/amblafont/vscode-yade-example>

# How to try Coreact-YADE

Requirements for the full example available on github<sup>1</sup>

Coq

Visual Studio Code

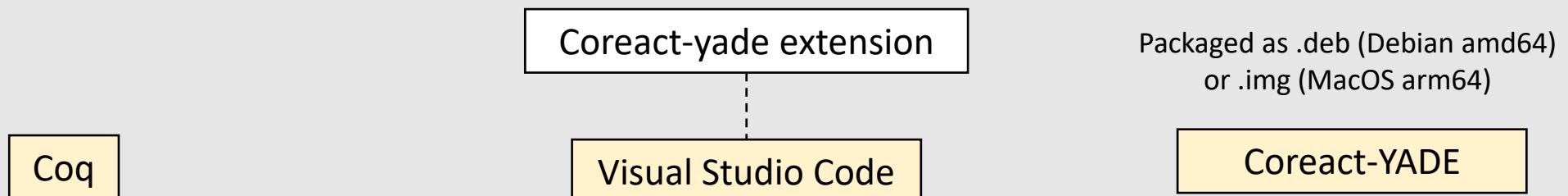
Packaged as .deb (Debian amd64)  
or .img (MacOS arm64)

Coreact-YADE

<sup>1</sup> <https://github.com/amblafont/vscode-yade-example>

# How to try Coreact-YADE

Requirements for the full example available on github<sup>1</sup>

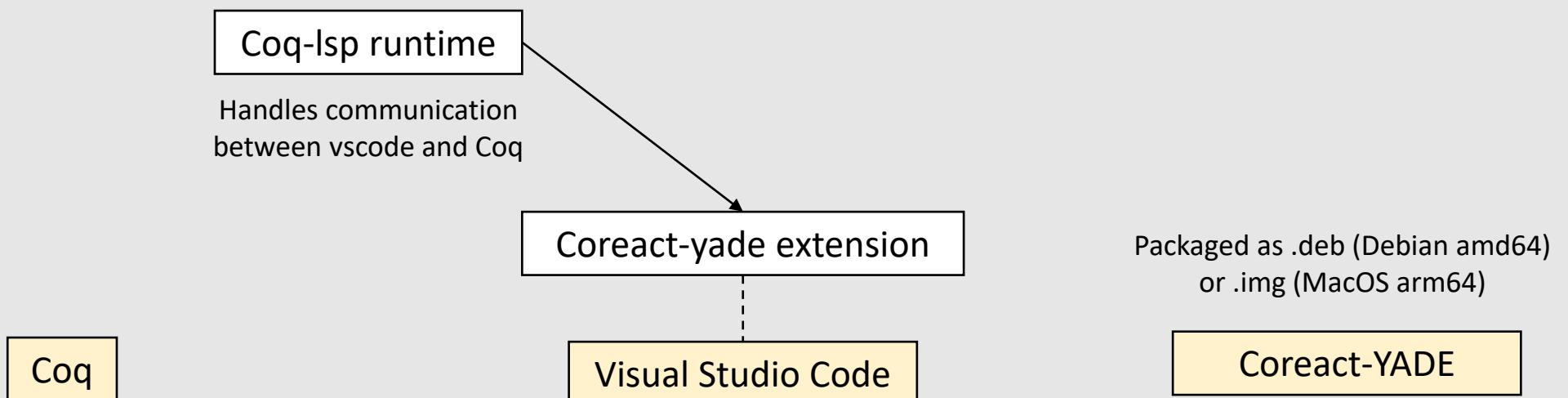


Packaged as .deb (Debian amd64)  
or .img (MacOS arm64)

<sup>1</sup> <https://github.com/amblafont/vscode-yade-example>

# How to try Coreact-YADE

Requirements for the full example available on github<sup>1</sup>



<sup>1</sup> <https://github.com/amblafont/vscode-yade-example>

# How to try Coreact-YADE

Requirements for the full example available on github<sup>1</sup>

## Hierarchy builder

A coq library to help building  
algebraic hierarchies  
(e.g. categories)

Coq

Coq-lsp runtime

Handles communication  
between vscode and Coq

Coreact-yade extension

Visual Studio Code

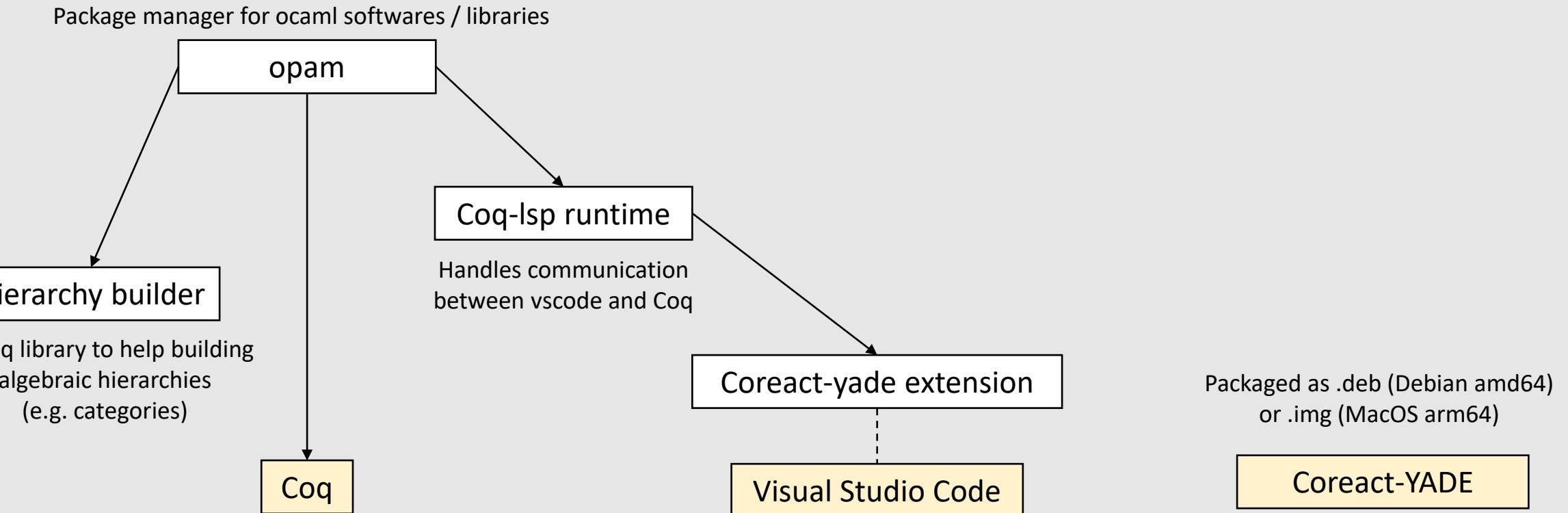
Packaged as .deb (Debian amd64)  
or .img (MacOS arm64)

Coreact-YADE

<sup>1</sup> <https://github.com/amblafont/vscode-yade-example>

# How to try Coreact-YADE

Requirements for the full example available on github<sup>1</sup>

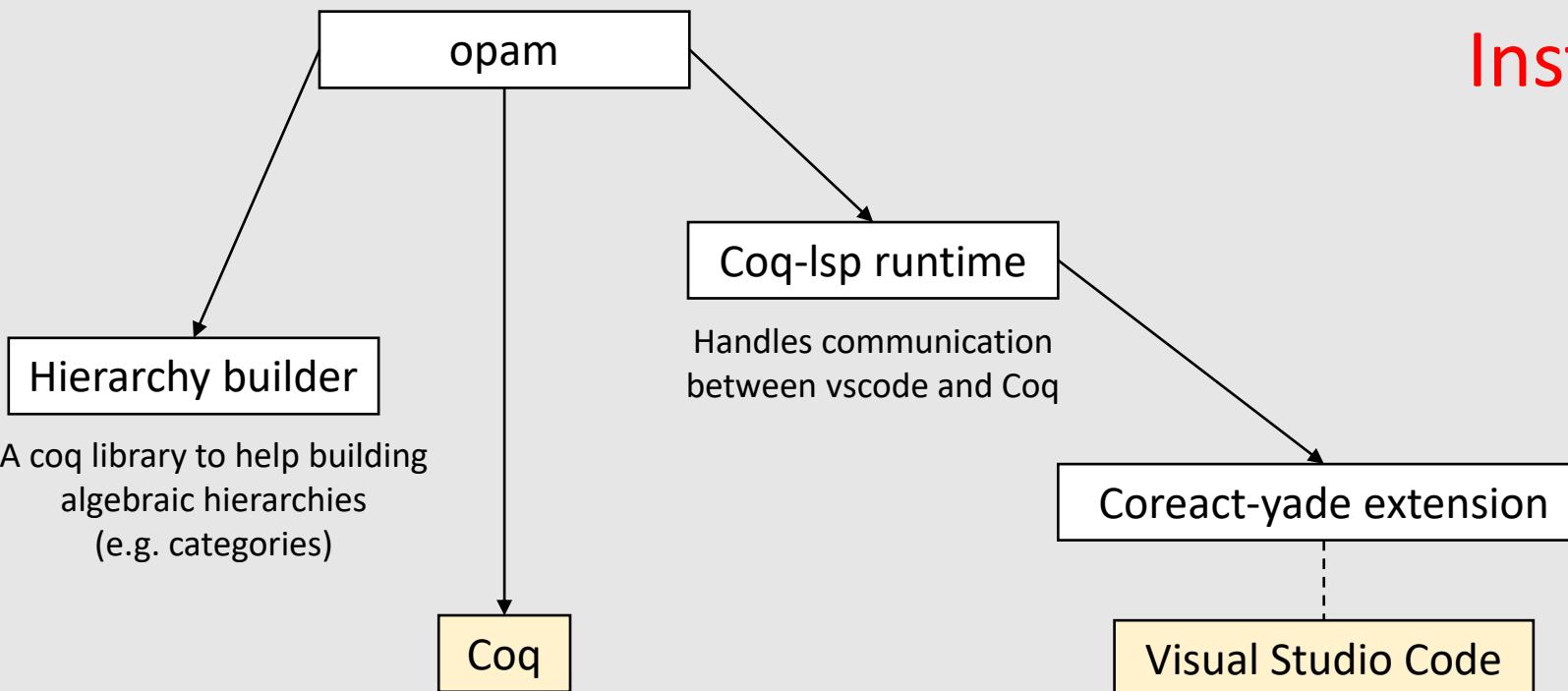


<sup>1</sup> <https://github.com/amblafont/vscode-yade-example>

# How to try Coreact-YADE

Requirements for the full example available on github<sup>1</sup>

Package manager for ocaml softwares / libraries



Installation is a bit tedious.

Packaged as .deb (Debian amd64) or .img (MacOS arm64)

Coreact-YADE

<sup>1</sup> <https://github.com/amblafont/vscode-yade-example>

# How to make YADE easier to install?

**The clean way:**

Embed YADE in (the web version of) vscode

# How to make YADE easier to install?

**The clean way:**

Embed YADE in (the web version of) vscode

✓ No installation:

run vscode + coq + YADE in your browser

# How to make YADE easier to install?

## **The clean way:**

Embed YADE in (the web version of) vscode

✓ No installation:

    run vscode + coq + YADE in your browser

✗ Requires a lot of engineering.

# How to make YADE easier to install?

## **The clean way:**

Embed YADE in (the web version of) vscode

✓ No installation:

run vscode + coq + YADE in your browser

✗ Requires a lot of engineering.

## **The lazy way:**

Distribute an “image” of an OS with everything installed, for some emulation software

# How to make YADE easier to install?

## **The clean way:**

Embed YADE in (the web version of) vscode

✓ No installation:

    run vscode + coq + YADE in your browser

✗ Requires a lot of engineering.

## **The lazy way:**

Distribute an “image” of an OS with everything installed, for some emulation software

✓ No need to change the implementation

# How to make YADE easier to install?

## The clean way:

Embed YADE in (the web version of) vscode

✓ No installation:

    run vscode + coq + YADE in your browser

✗ Requires a lot of engineering.

## The lazy way:

Distribute an “image” of an OS with everything installed, for some emulation software

✓ No need to change the implementation

✗ Still requires to install the emulation software

# How to make YADE easier to install?

## The clean way:

Embed YADE in (the web version of) vscode

✓ No installation:

    run vscode + coq + YADE in your browser

✗ Requires a lot of engineering.

## The lazy way:

Distribute an “image” of an OS with everything installed, for some emulation software

✓ No need to change the implementation

✗ Still requires to install the emulation software

✗ Images can be big (several GBs).

# How to make YADE easier to install?

## The clean way:

Embed YADE in (the web version of) vscode

✓ No installation:

    run vscode + coq + YADE in your browser

✗ Requires a lot of engineering.

## The lazy way:

Distribute an “image” of an OS with everything installed, for some emulation software

✓ No need to change the implementation

✗ Still requires to install the emulation software

✗ Images can be big (several GBs).

## The very lazy way:

Distribute an installation script to build the “image” on the user machine

# How to make YADE easier to install?

## The clean way:

Embed YADE in (the web version of) vscode

✓ No installation:

    run vscode + coq + YADE in your browser

✗ Requires a lot of engineering.

## The lazy way:

Distribute an “image” of an OS with everything installed, for some emulation software

✓ No need to change the implementation

✗ Still requires to install the emulation software

✗ Images can be big (several GBs).

## The very lazy way:

Distribute an installation script to build the “image” on the user machine

✓ No need to host the (big) image

# How to make YADE easier to install?

## The clean way:

Embed YADE in (the web version of) vscode

✓ No installation:

    run vscode + coq + YADE in your browser

✗ Requires a lot of engineering.

## The lazy way:

Distribute an “image” of an OS with everything installed, for some emulation software

✓ No need to change the implementation

✗ Still requires to install the emulation software

✗ Images can be big (several GBs).

## The very lazy way:

Distribute an installation script to build the “image” on the user machine

✓ No need to host the (big) image

✗ Less robust

# How to make YADE easier to install?

## The clean way:

Embed YADE in (the web version of) vscode

✓ No installation:

    run vscode + coq + YADE in your browser

✗ Requires a lot of engineering.

What I did →

## The lazy way:

Distribute an “image” of an OS with everything installed, for some emulation software

✓ No need to change the implementation

✗ Still requires to install the emulation software

✗ Images can be big (several GBs).

## The very lazy way:

Distribute an installation script to build the “image” on the user machine

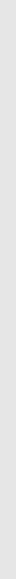
✓ No need to host the (big) image

✗ Less robust

# Testing YADE in an emulated environment

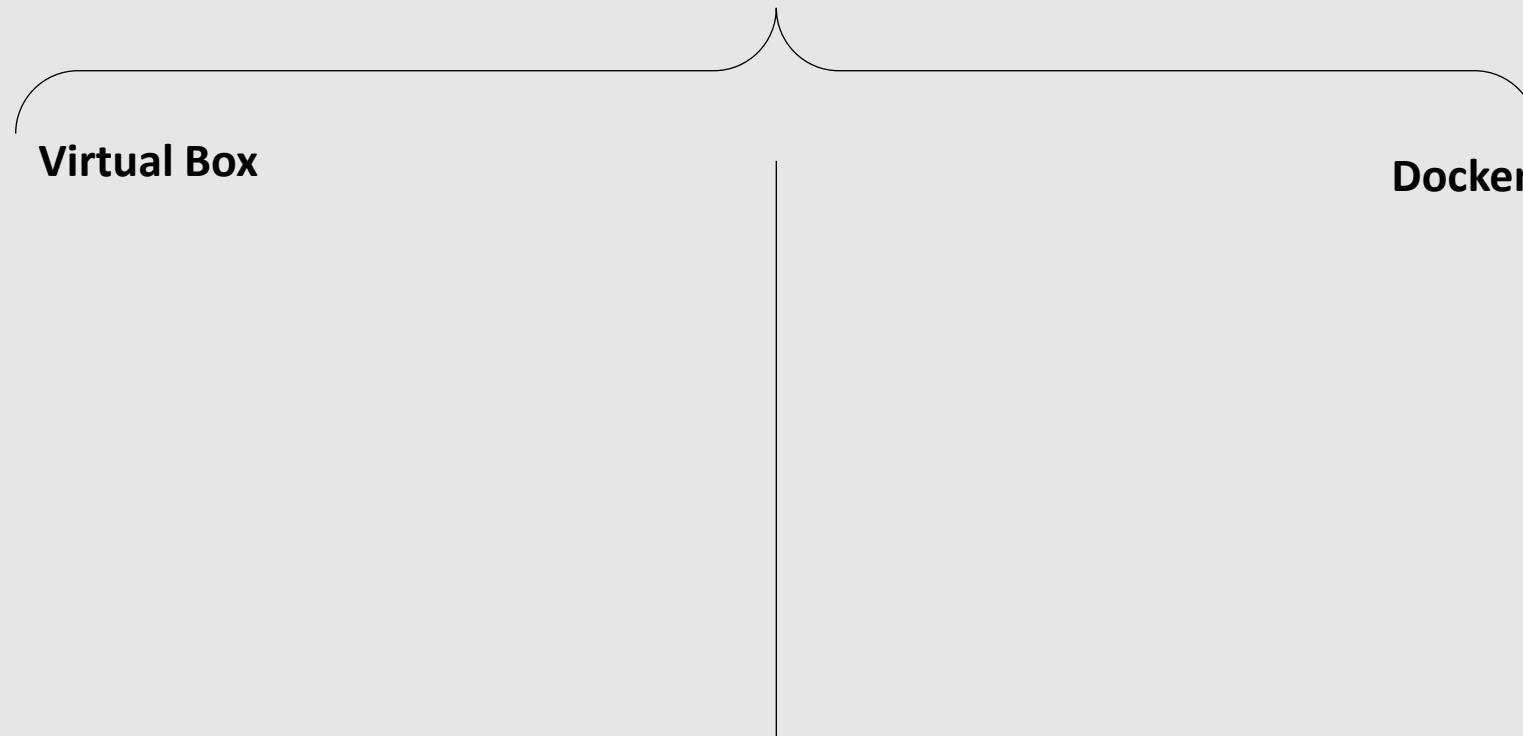
**Virtual Box**

**Docker**



# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)



# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

**Virtual Box**

- ✓ Easy to install
- ✗ Big image size

**Docker**

# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

**Virtual Box**

- ✓ Easy to install
- ✗ Big image size

**Docker**

**Installation script**

A shell script to be run in a fresh  
Debian image<sup>1</sup>

<sup>1</sup><https://www.linuxvmimages.com/>

# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

## Virtual Box

- ✓ Easy to install
- ✗ Big image size

## Docker

- ✓ Better performance (probably not very relevant for YADE)

### Installation script

A shell script to be run in a fresh  
Debian image<sup>1</sup>

<sup>1</sup><https://www.linuxvmimages.com/>

# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

## Virtual Box

- ✓ Easy to install
- ✗ Big image size

## Docker

- ✓ Better performance (probably not very relevant for YADE)
- ✓ A modular system to build new images

### Installation script

A shell script to be run in a fresh  
Debian image<sup>1</sup>

<sup>1</sup><https://www.linuxvmimages.com/>

# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

## Virtual Box

- ✓ Easy to install
- ✗ Big image size

## Docker

- ✓ Better performance (probably not very relevant for YADE)
- ✓ A modular system to build new images
  - recipes are called *dockerfiles*

### Installation script

A shell script to be run in a fresh  
Debian image<sup>1</sup>

<sup>1</sup><https://www.linuxvmimages.com/>

# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

## Virtual Box

- ✓ Easy to install
- ✗ Big image size

## Docker

- ✓ Better performance (probably not very relevant for YADE)
- ✓ A modular system to build new images
  - recipes are called *dockerfiles*
  - a dockerfile can extend another one

## Installation script

A shell script to be run in a fresh  
Debian image<sup>1</sup>

<sup>1</sup><https://www.linuxvmimages.com/>

# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

## Virtual Box

- ✓ Easy to install
- ✗ Big image size

## Docker

- ✓ Better performance (probably not very relevant for YADE)
- ✓ A modular system to build new images
  - recipes are called *dockerfiles*
  - a dockerfile can extend another one
- ✗ Not readily cross-platform

## Installation script

A shell script to be run in a fresh  
Debian image<sup>1</sup>

<sup>1</sup><https://www.linuxvmimages.com/>

# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

## Virtual Box

- ✓ Easy to install
- ✗ Big image size

## Docker

- ✓ Better performance (probably not very relevant for YADE)
- ✓ A modular system to build new images
  - recipes are called *dockerfiles*
  - a dockerfile can extend another one
- ✗ Not readily cross-platform
- ✗ Configuration is complex (especially for GUI images)

## Installation script

A shell script to be run in a fresh  
Debian image<sup>1</sup>

<sup>1</sup><https://www.linuxvmimages.com/>

# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

## Virtual Box

- ✓ Easy to install
- ✗ Big image size

### Installation script

A shell script to be run in a fresh  
Debian image<sup>1</sup>

## Docker

- ✓ Better performance (probably not very relevant for YADE)
- ✓ A modular system to build new images
  - recipes are called *dockerfiles*
  - a dockerfile can extend another one
- ✗ Not readily cross-platform
- ✗ Configuration is complex (especially for GUI images)

### Installation script

A dockerfile (extending that of coq)

<sup>1</sup><https://www.linuxvmimages.com/>

# Testing YADE in an emulated environment

can run an image of a full OS (e.g., Debian) on any host operating system (Windows / Mac / Linux)

## Virtual Box

- ✓ Easy to install
- ✗ Big image size

### Installation script

A shell script to be run in a fresh  
Debian image<sup>1</sup>

## Docker

- ✓ Better performance (probably not very relevant for YADE)
- ✓ A modular system to build new images
  - recipes are called *dockerfiles*
  - a dockerfile can extend another one
- ✗ Not readily cross-platform
- ✗ Configuration is complex (especially for GUI images)

### Installation script

A dockerfile (extending that of coq)

Check <https://github.com/amblafont/vscode-yade-example> for instructions!

<sup>1</sup><https://www.linuxvmimages.com/>