A diagram editor to mechanise categorical proofs

Ambroise Lafont (PARTOUT)

23 November 2023

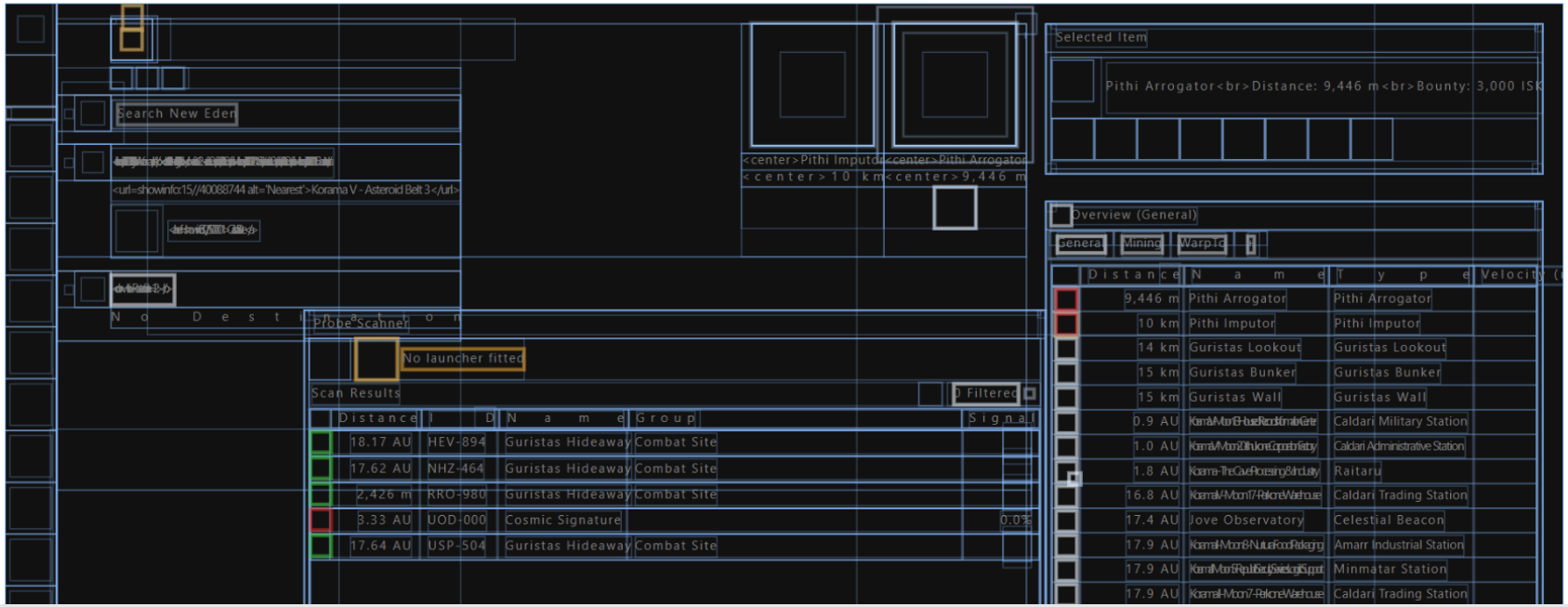
A bit of history

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

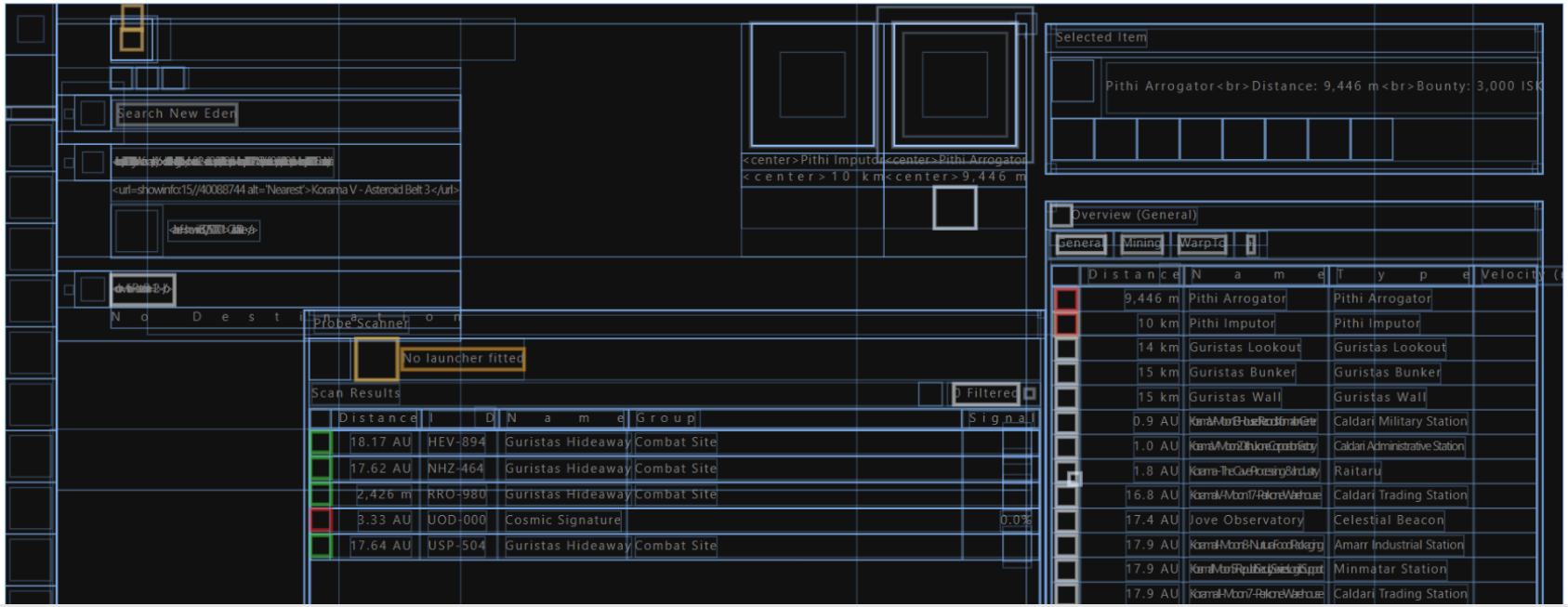
Languages

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

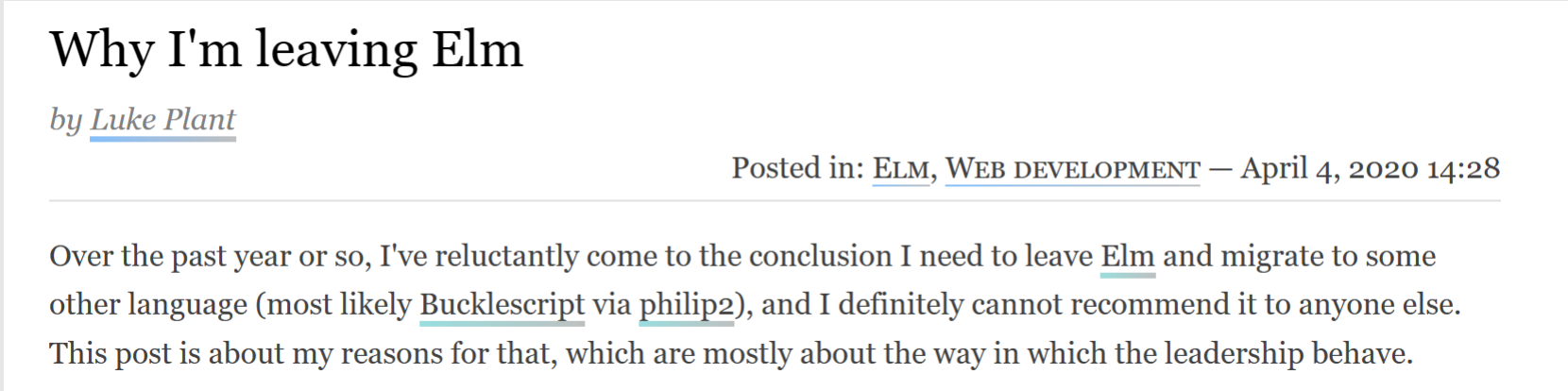
- Is it still a thing?

  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…

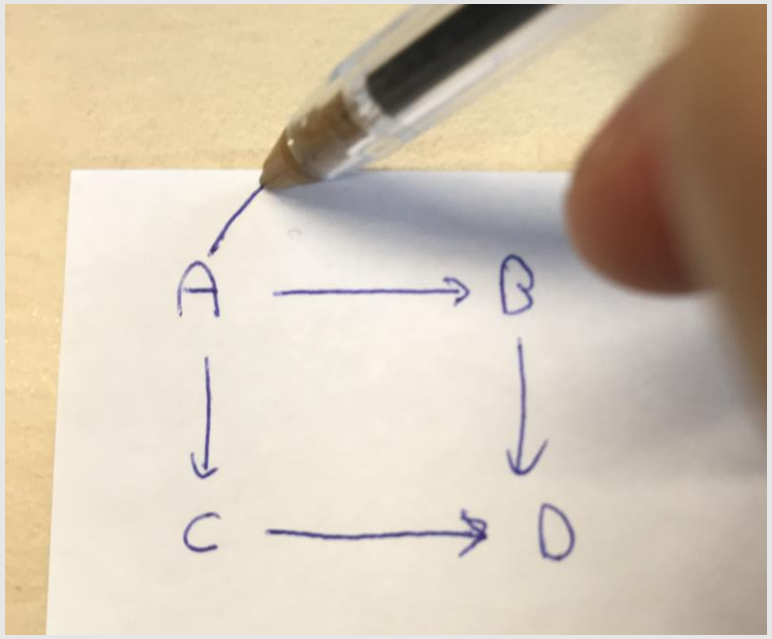
- A categorical diagram editor

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.



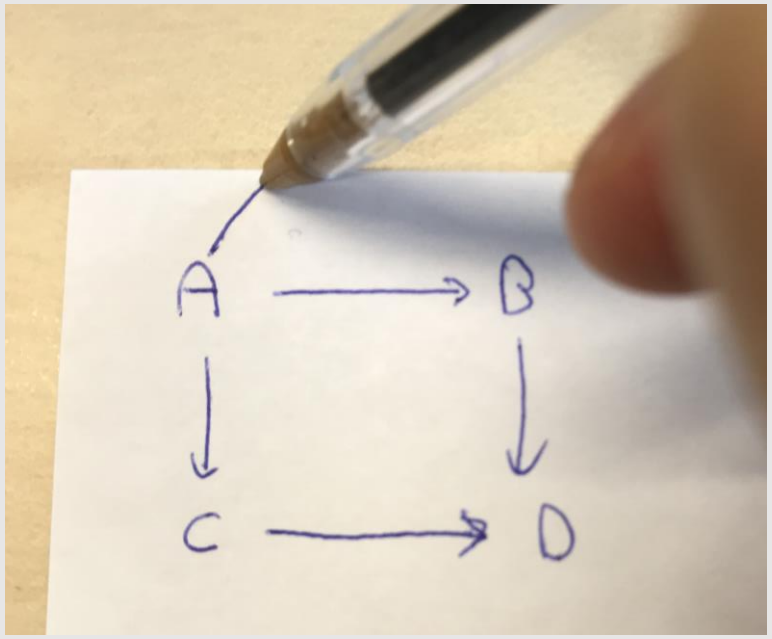
2020: I need a pretext to program in Elm

- 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

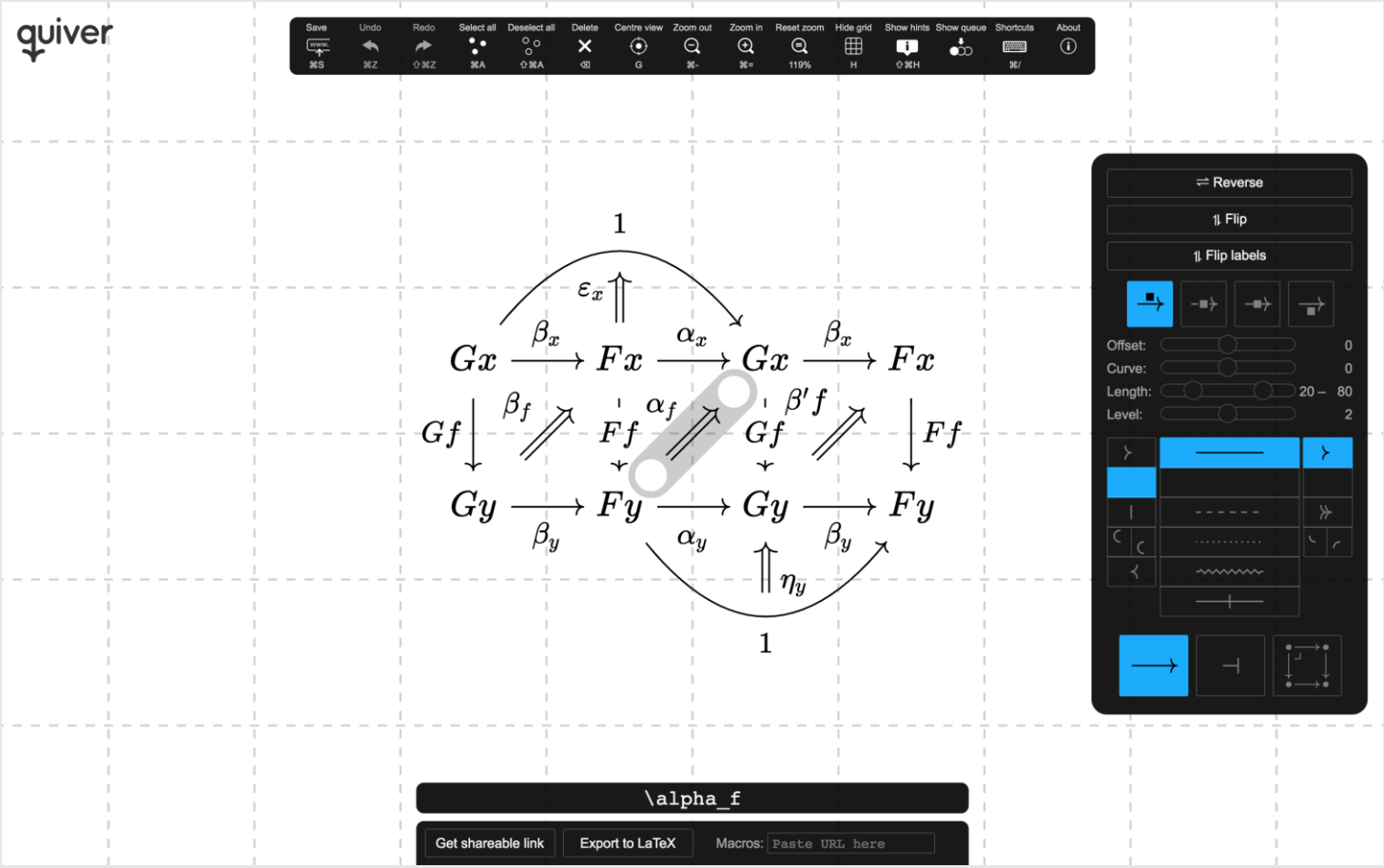
Nov. 2020: An existential crisis

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*."

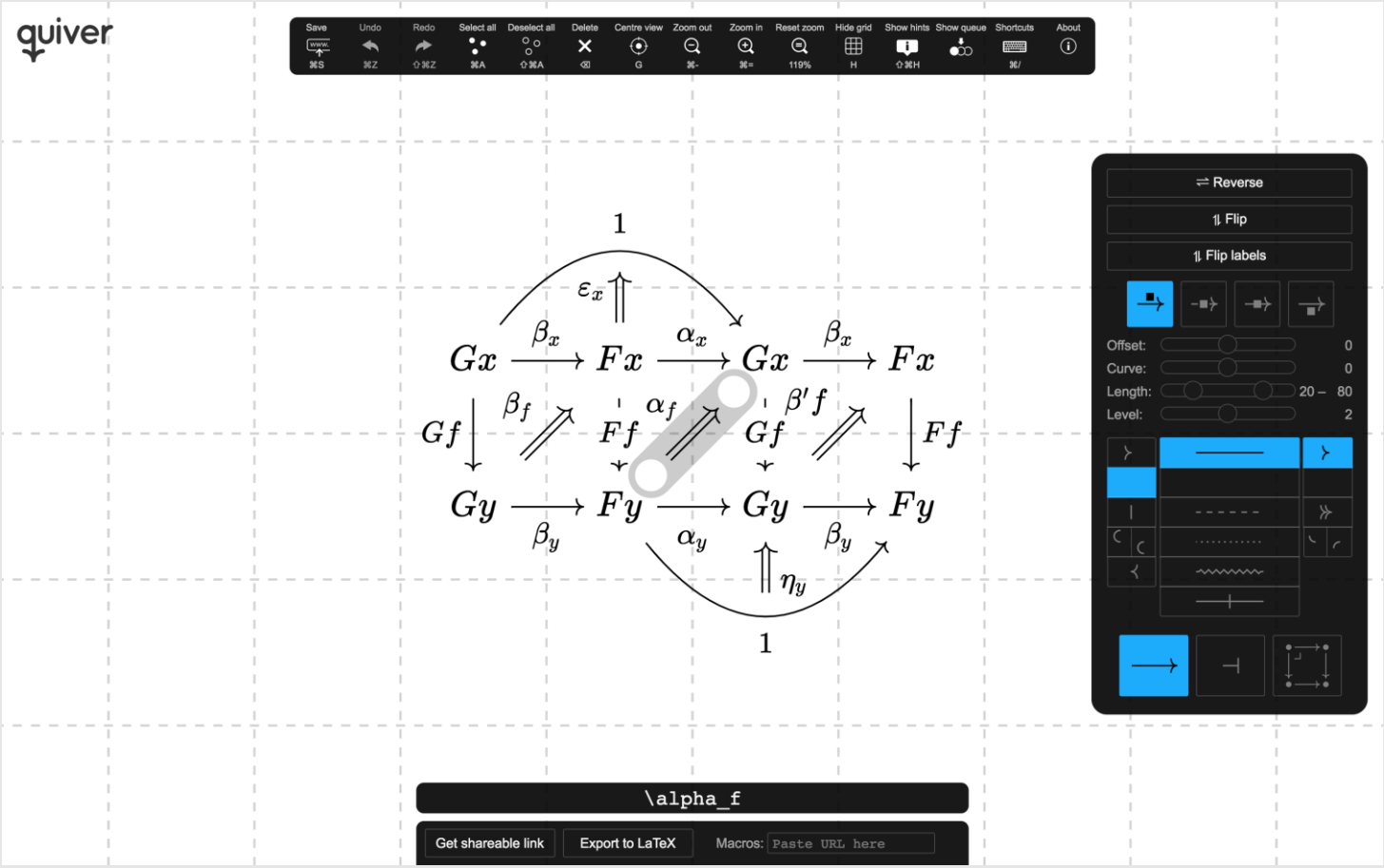
Nov. 2020: An existential crisis

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

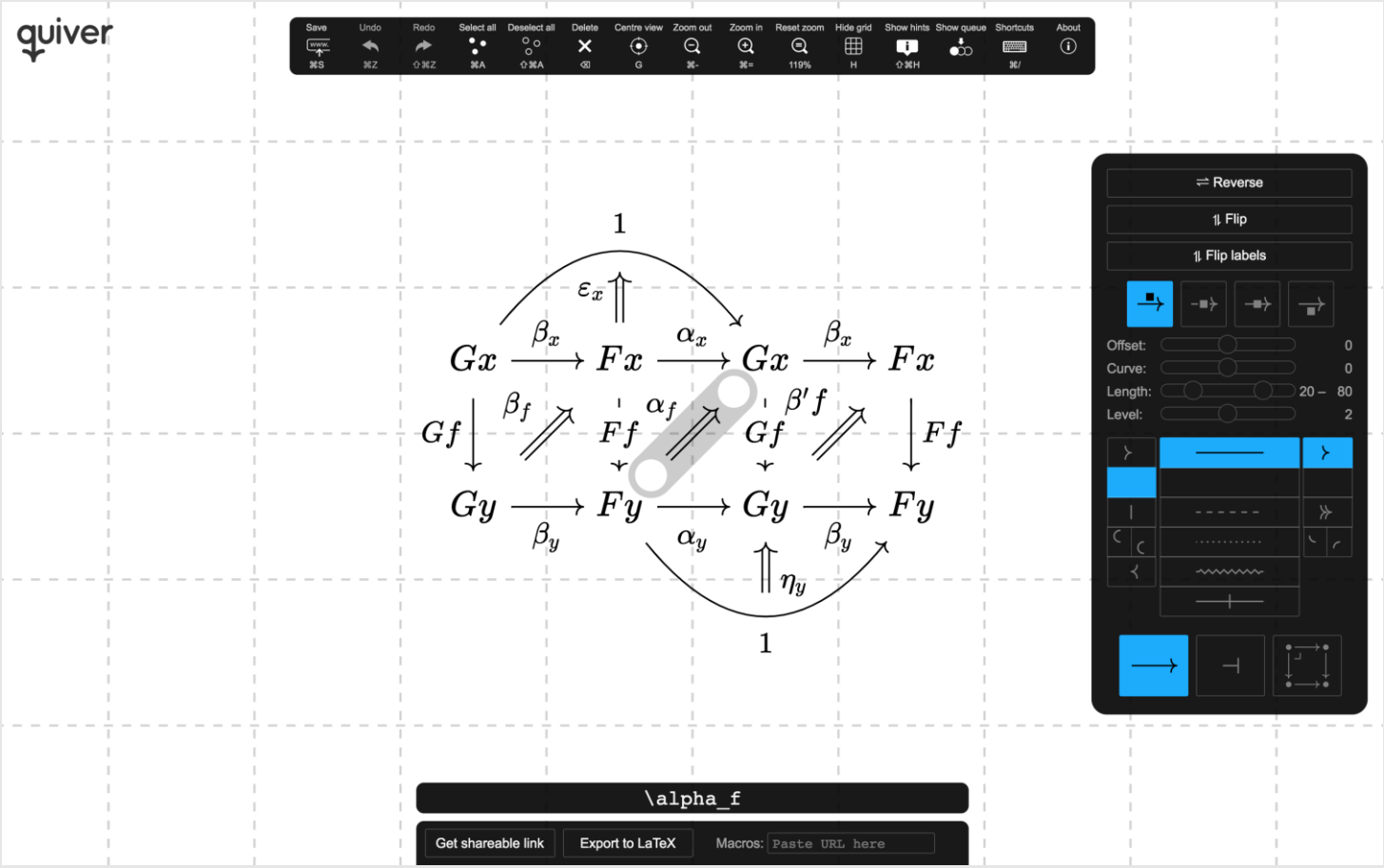
Nathanel Arkhor releases quiver, "*a modern, graphical editor for commutative and pasting diagrams, capable of rendering high-quality diagrams for screen viewing, and exporting to LaTeX via tikz-cd*."



That's what I have been looking for!

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*."

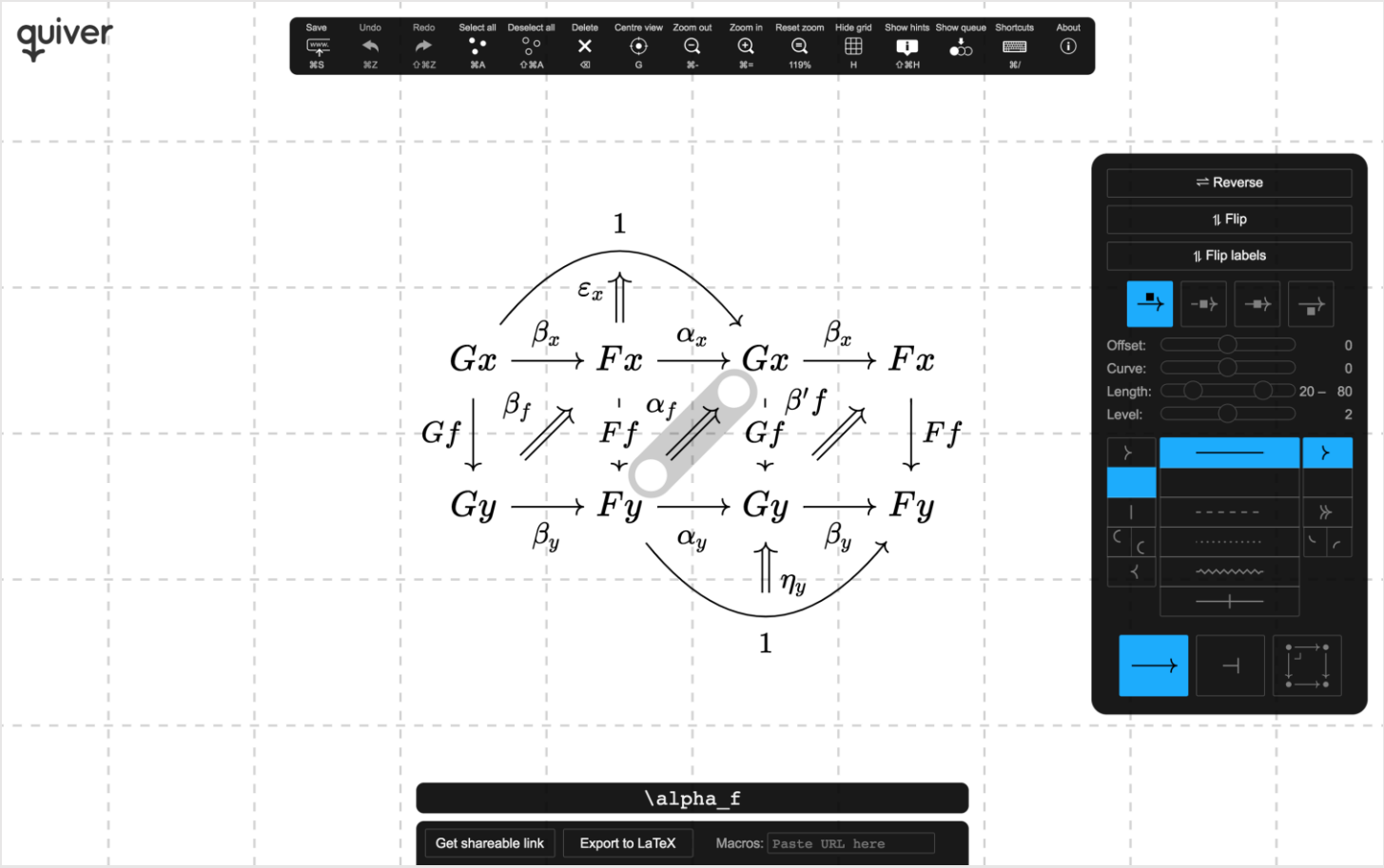


That's what I have been looking for!

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

Nov. 2020: An existential crisis

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.*"



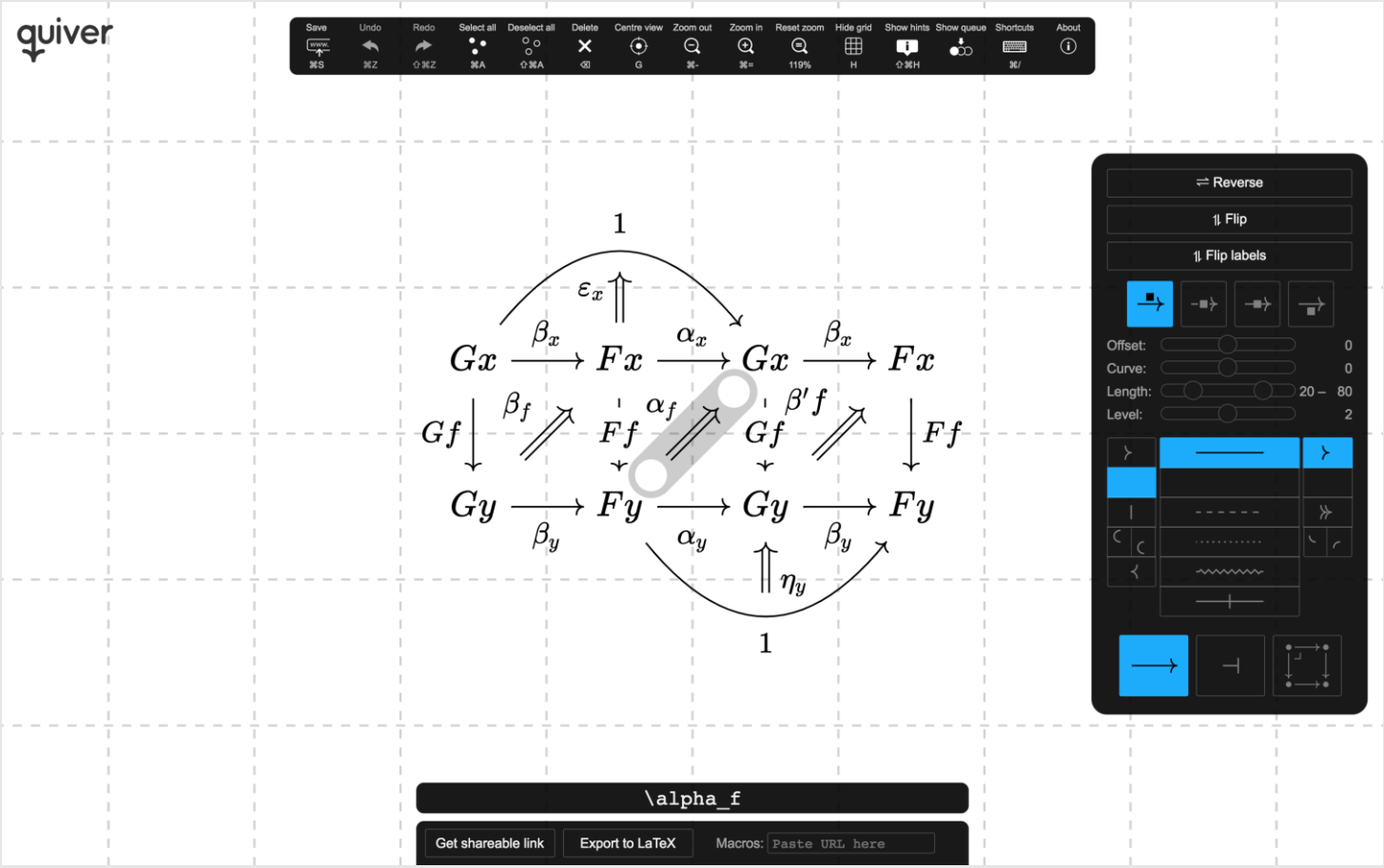
That's what I have been looking for!

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

In principle yes, but…

Nov. 2020: An existential crisis

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.*"



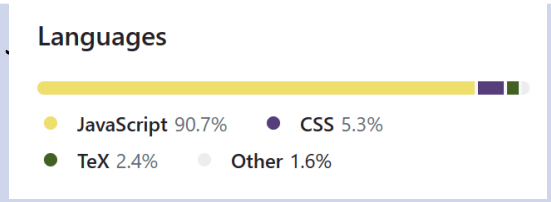
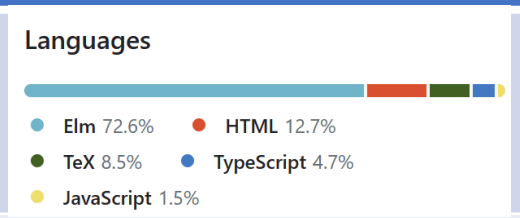
That's what I have been looking for!

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

In principle yes, but…
it is implemented in JavaScript 💀

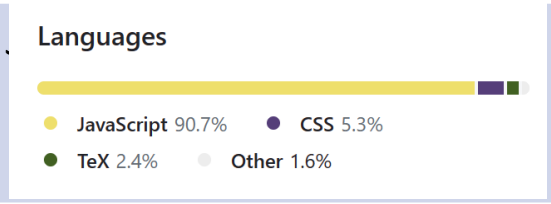
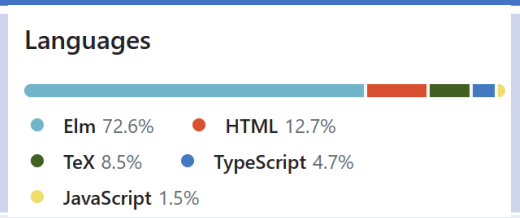
Comparison with quiver

About the same size (around 10k of LoC)

	Quiver	My editor
Programming Languages		
Styling options	+	-
User-friendly	+	-
Editing features	-	+ Tabs, copy & paste, find & replace, expand selection to connected components, …

Comparison with quiver

About the same size (around 10k of LoC)

	Quiver	My editor
Programming Languages		
Styling options	+	-
User-friendly	+	-
Editing features	-	+ Tabs, copy & paste, find & replace, expand selection to connected components, …
LaTeX export	yes	yes[1]

[1] 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

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

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

(1) Remove the generated latex

Easy editing of LaTeX diagrams

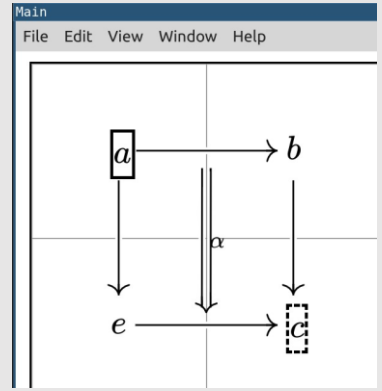
diagram data

standalone version, running in the background

```
% 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

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



Easy editing of LaTeX diagrams

diagram data

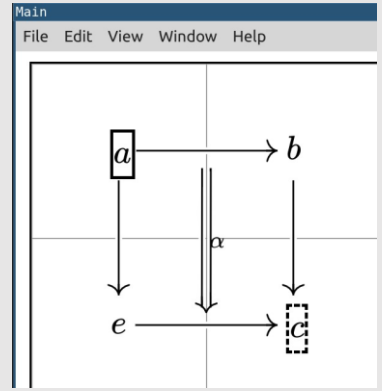
standalone version, running in the background

```
% 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

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

(3) Edit & Save the diagram in the editor



Easy editing of LaTeX diagrams

diagram data

standalone version, running in the background

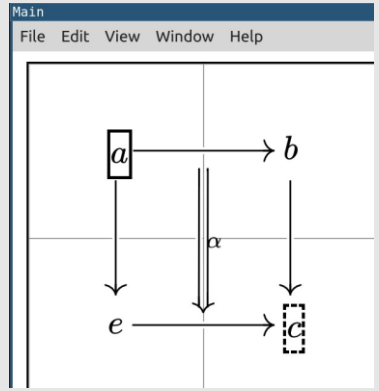
```
% YADE DIAGRAM {"graph":{"edges":[{"from":0,"id":4,"label":{"isPullshout"
% GENERATED LATEX
\begin{tikzpicture}[every node/.style={inner sep=5pt,outer sep=0pt,anchor
\node (0) at (9.476190476190476em, -3.3154761904761907em) {$a$} ;
\node (1) at (19em, -3.3154761904761907em) {$b$} ;
```

example.tex

(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)



Easy editing of LaTeX diagrams

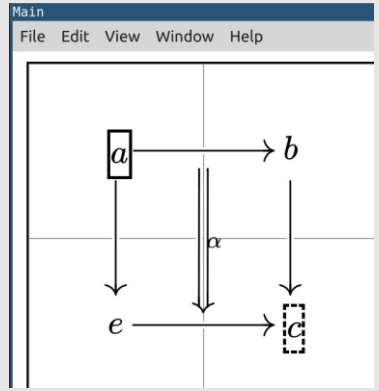
diagram data

standalone version, running in the background

```
% 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

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

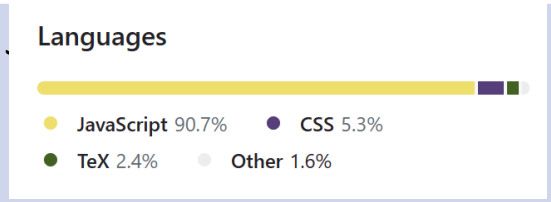
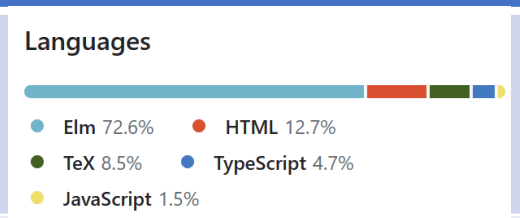


(3) Edit & Save the diagram in the editor

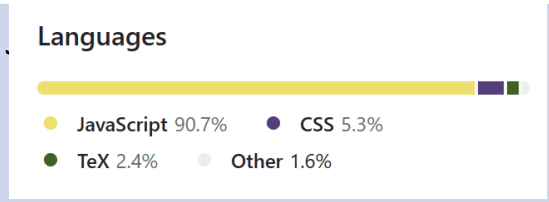
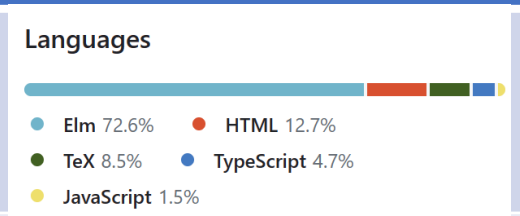
⇒ (4) the editor updates example.tex
(generated LaTeX & diagram data)

DEMO

Comparison with quiver

	Quiver	My editor
Programming Languages		
Styling options	+	-
User-friendly	+	-
Editing features	-	+ Tabs, Copy & paste, find & replace, expand selection to connected components, …
LaTeX export	yes	yes

Comparison with quiver

	Quiver	My editor
Programming Languages		
Styling options	+	-
User-friendly	+	-
Editing features	-	+ Tabs, Copy & paste, find & replace, expand selection to connected components, …
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 😵

README.rst Update information about daily-packages. 10 years ago

README.rst

YADE - Yet Another Dynamic Engine

Extensible open-source framework for discrete numerical models, focused on Discrete Element Method. The computation parts are written in c++ using flexible object model, allowing independent implementation of new algorithms and interfaces. Python is used for rapid and concise scene construction, simulation control, postprocessing and debugging.

Languages

C++ 79.5% Python 18.3%
CMake 2.0% Shell 0.2%

Now I call it Coreact YADE 🤔

Now I call it Coreact YADE 😳

ANR Project[1] (2023 - 2027): Coq-based Rewriting: Towards Executable Applied Category Theory

[1] https://coreact.wiki/

Now I call it Coreact YADE 😳

ANR Project[1] (2023 - 2027): Coq-based Rewriting: Towards Executable Applied Category Theory

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

[1] https://coreact.wiki/

Now I call it Coreact YADE 😵‍💫

ANR Project[1] (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)

[1] https://coreact.wiki/

Now I call it Coreact YADE 😳

ANR Project[1] (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.

[1] https://coreact.wiki/

The Coq proof assistant

A software to write formal proofs interactively

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

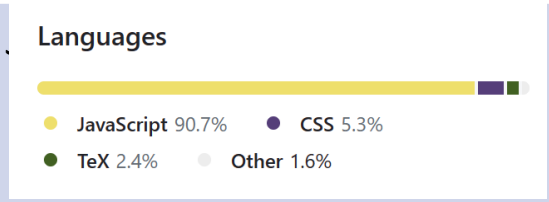
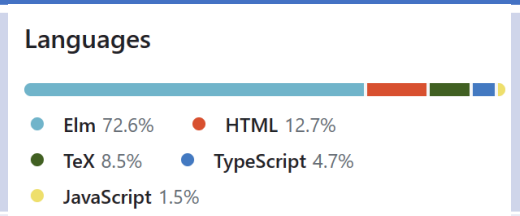
The Coq proof assistant

A software to write formal proofs interactively

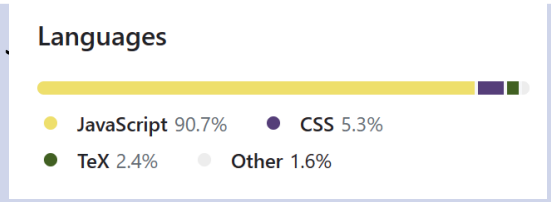
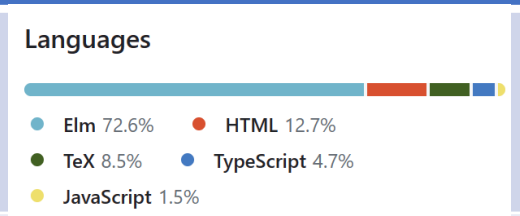
File editor, e.g., Visual Studio Code + coq-lsp extension

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

Comparison with quiver

	Quiver	My editor
Programming Languages		
Styling options	+	-
User-friendly	+	-
Editing features	-	+ Tabs, Copy & paste, find & replace, expand selection to connected components, …
LaTeX export	yes	yes
A nice name	+	-

Comparison with quiver

	Quiver	My editor
Programming Languages		
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	-	+

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 $(\alpha_c: Fc \to Gc)_c$ such that:

Diagrammatic reasoning in category theory

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

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

$$\forall f: c_1 \to c_2, \qquad Ff \circ \alpha_{c_1} = \alpha_{c_2} \circ Gf$$

Diagrammatic reasoning in category theory

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

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

Computer-friendly (text)

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

Diagrammatic reasoning in category theory

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

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

Computer-friendly (text)

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

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 $(\alpha_c: Fc \to Gc)_c$ such that:

Computer-friendly (text)

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

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 $(\alpha_c: Fc \to Gc)_c$ such that:

Computer-friendly (text)

Diagrammatically,

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

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 $(\alpha_c: Fc \to Gc)_c$ such that:

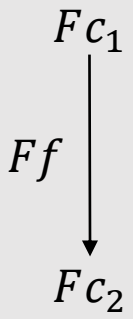
Computer-friendly (text)

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

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 $(\alpha_c: Fc \to Gc)_c$ such that:

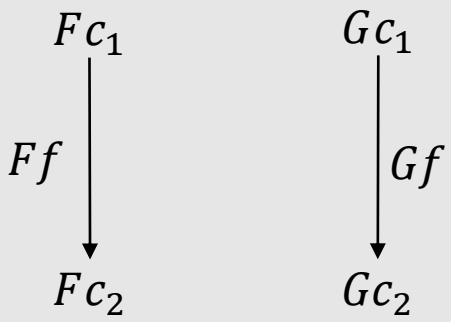
Computer-friendly (text)

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

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 $(\alpha_c: Fc \to Gc)_c$ such that:

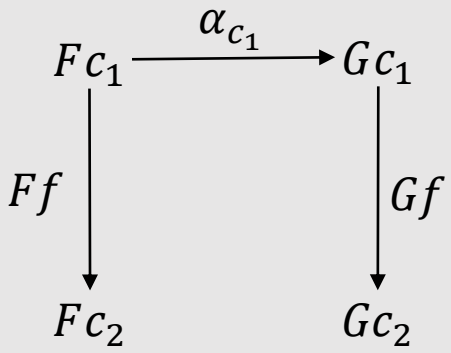
Computer-friendly (text)

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

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 $(\alpha_c: Fc \to Gc)_c$ such that:

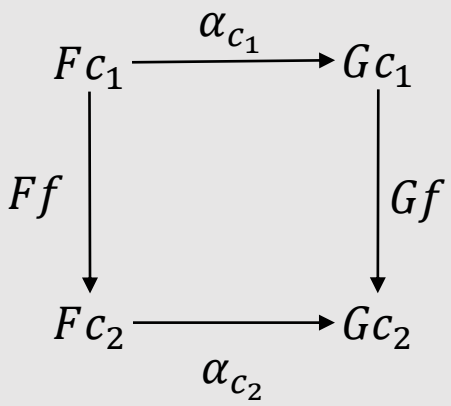
Computer-friendly (text)

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

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 $(\alpha_c: Fc \to Gc)_c$ such that:

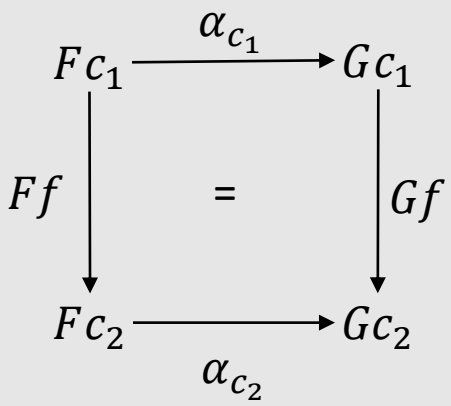
Computer-friendly (text)

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

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 $(\alpha_c: Fc \to Gc)_c$ such that:

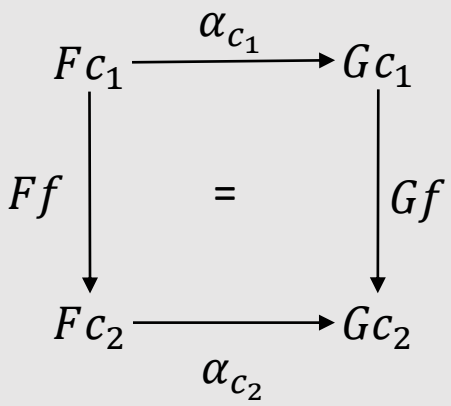
Computer-friendly (text)

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

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,



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

Diagrammatic reasoning in category theory

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

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

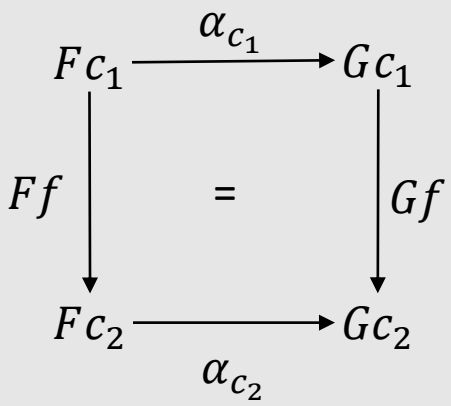
Computer-friendly (text)

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

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

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

Diagrammatically,



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

Diagrammatic reasoning in category theory

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

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

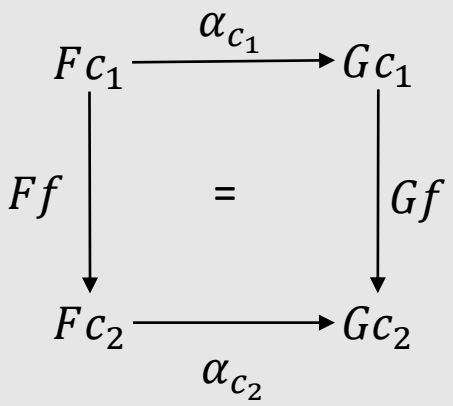
Computer-friendly (text)

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

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,



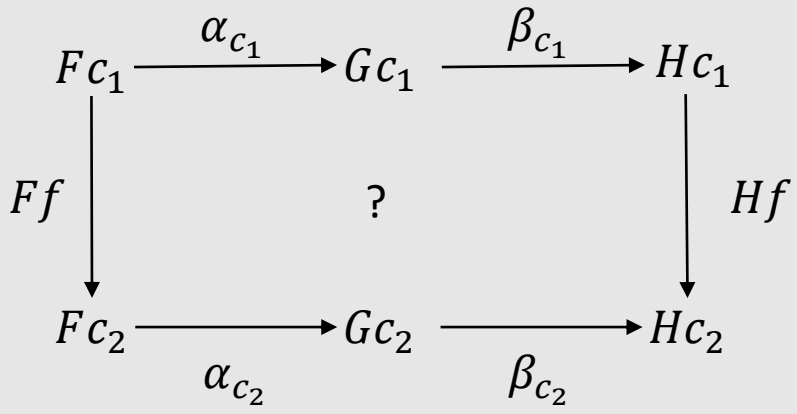
$Gf \circ \alpha_{c_1} = \alpha_{c_2} \circ Ff$

…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 $(\beta_c \circ \alpha_c: Fc \to Hc)_c$

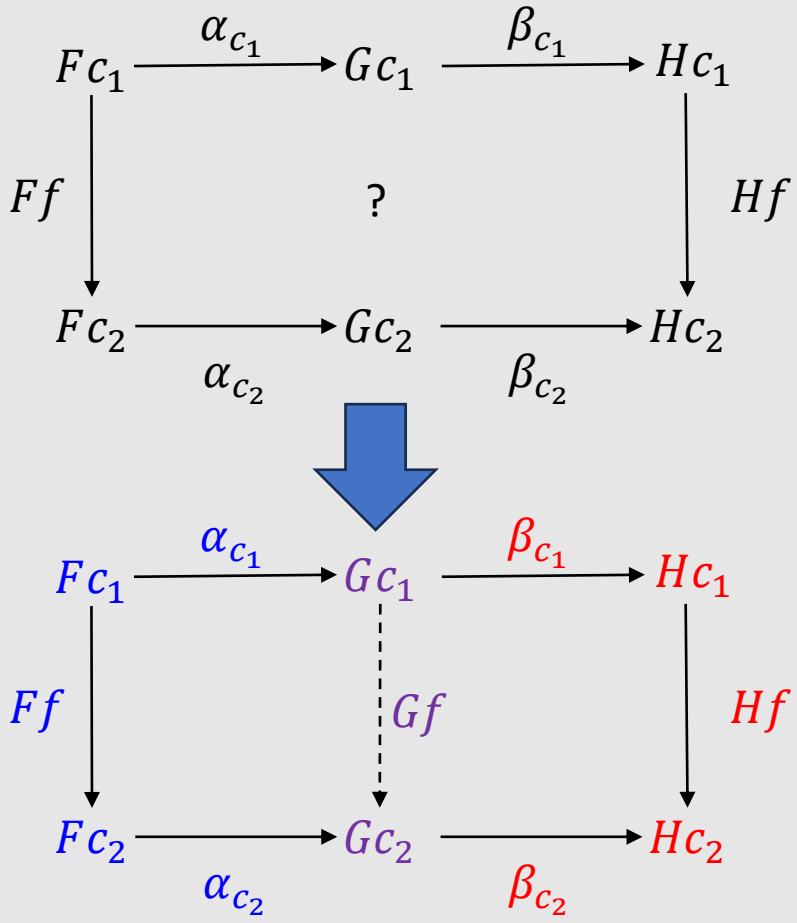
Diagrammatic proof



Natural transformations compose:

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

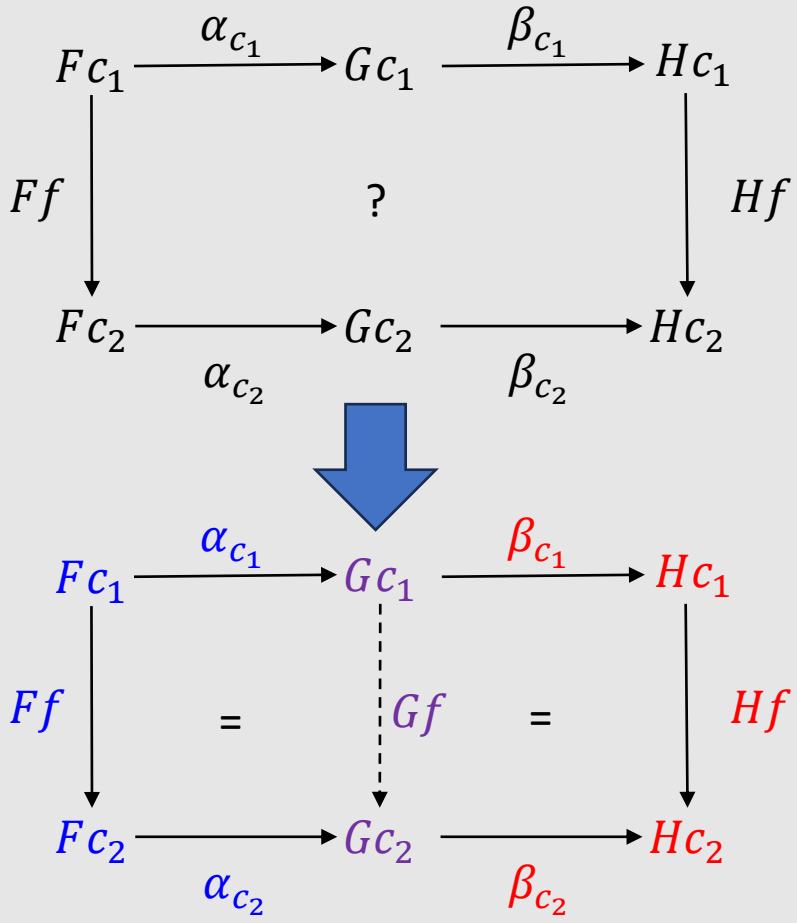
Diagrammatic proof



Natural transformations compose:

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

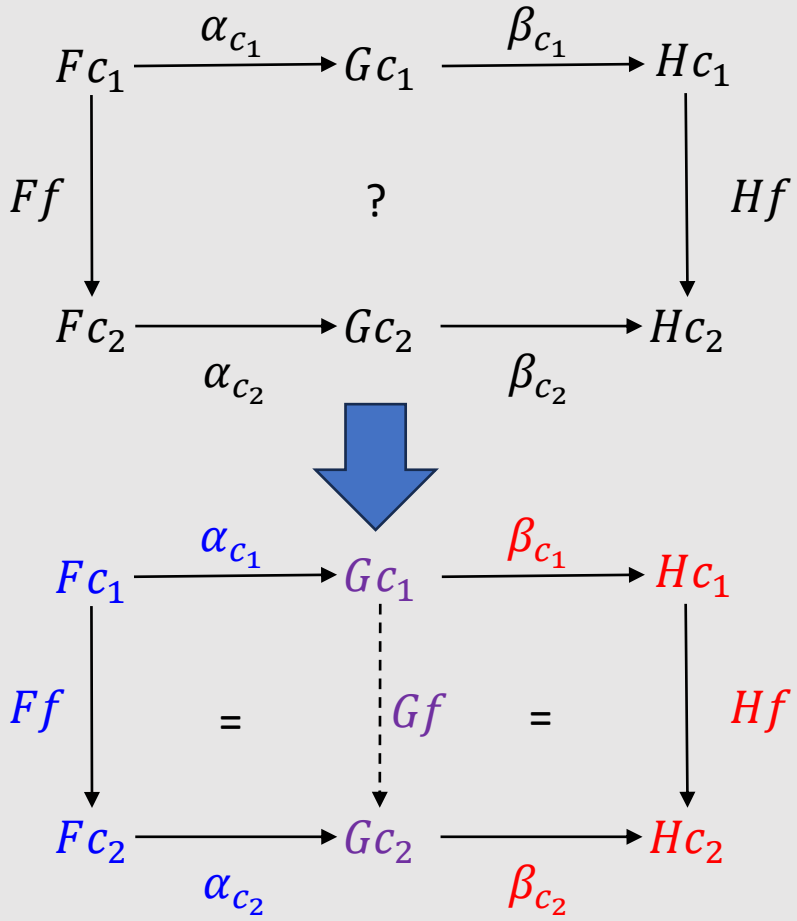
Diagrammatic proof



Natural transformations compose:

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

Diagrammatic proof



Computer-friendly proof

$$Hf \circ \beta_{c_1} \circ \alpha_{c_1}$$

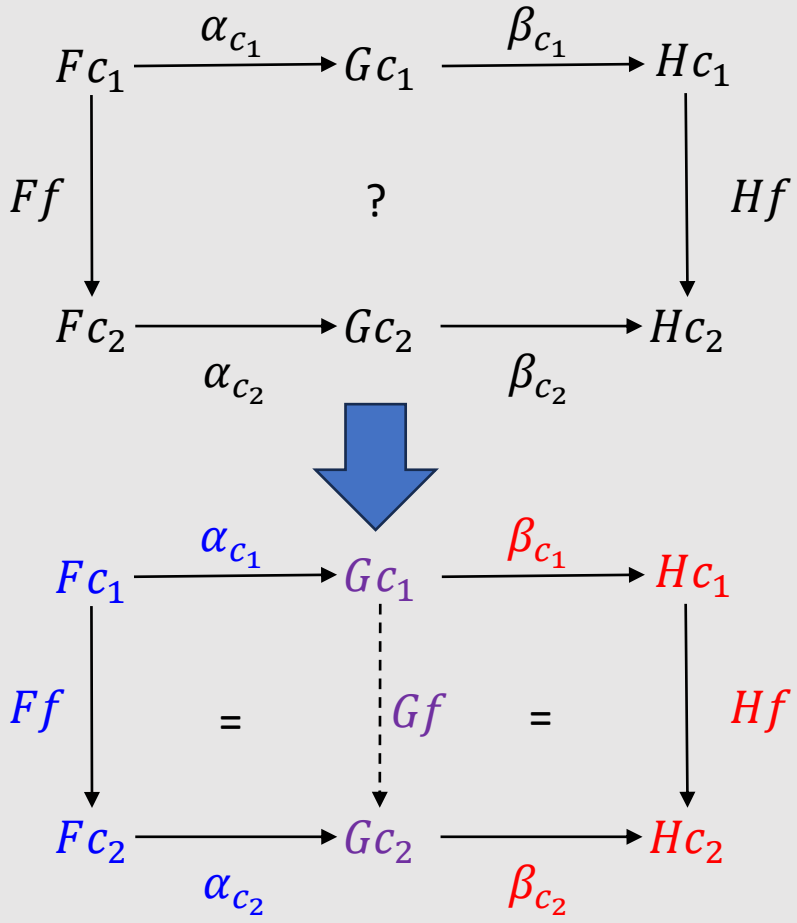
$$= \beta_{c_2} \circ Gf \circ \alpha_{c_1}$$

$$= \beta_{c_2} \circ \alpha_{c_2} \circ Ff$$

Natural transformations compose:

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

Diagrammatic proof



Automatic generation?

Computer-friendly proof

$$Hf \circ \beta_{c_1} \circ \alpha_{c_1}$$

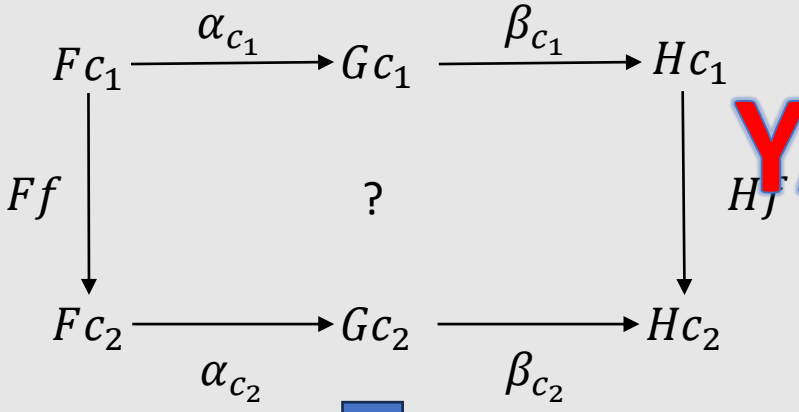
$$= \beta_{c_2} \circ Gf \circ \alpha_{c_1}$$

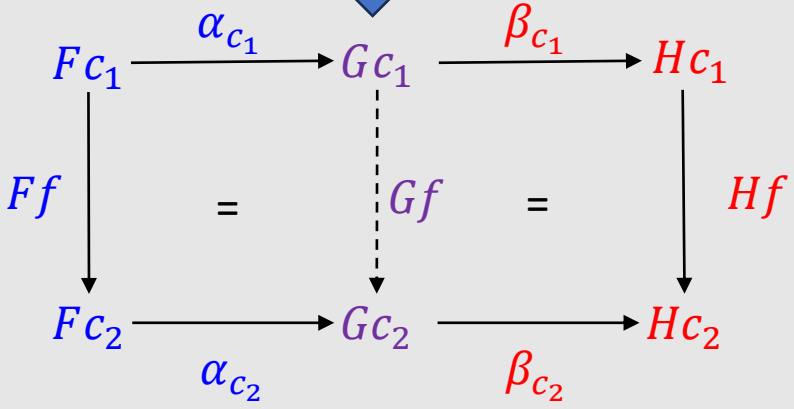
$$= \beta_{c_2} \circ \alpha_{c_2} \circ Ff$$

Natural transformations compose:

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

Diagrammatic proof



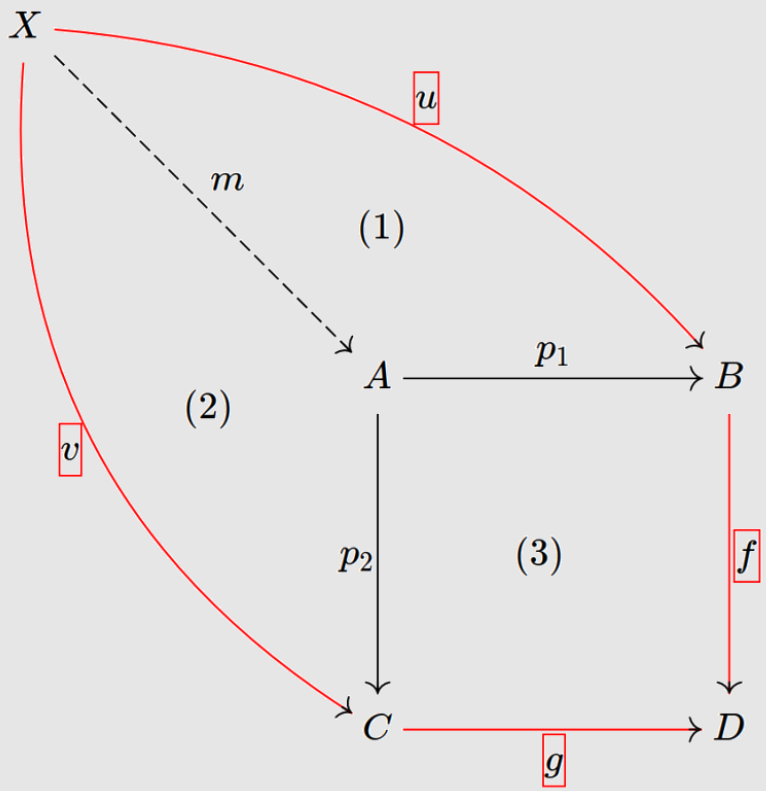


Automatic generation?

Computer-friendly proof

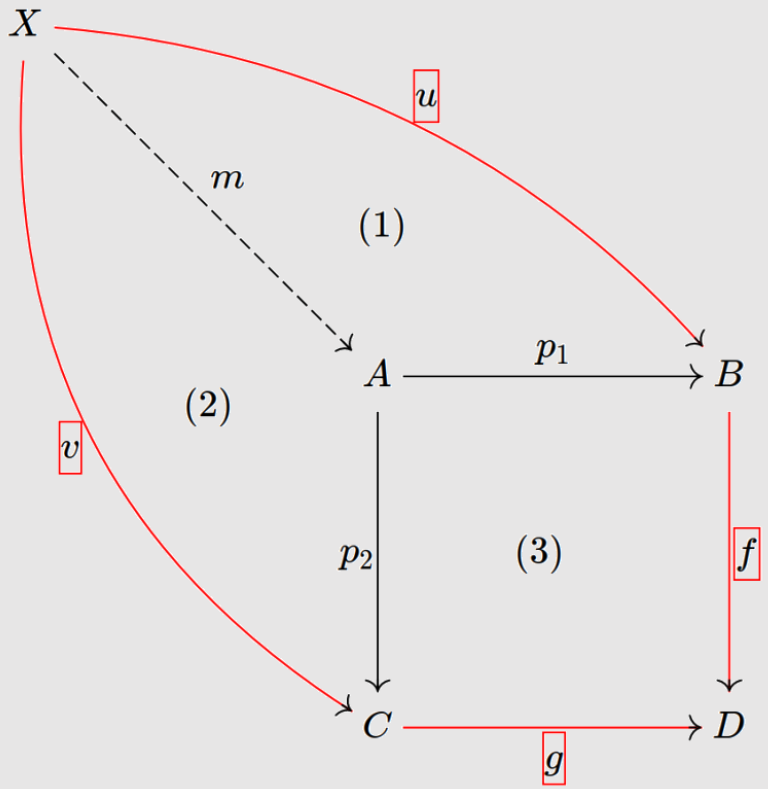
$$Hf \circ \beta_{c_1} \circ \alpha_{c_1}$$
$$= \beta_{c_2} \circ Gf \circ \alpha_{c_1}$$
$$= \beta_{c_2} \circ \alpha_{c_2} \circ Ff$$

Proof generation: sketch of the algorithm



Proof generation: sketch of the algorithm

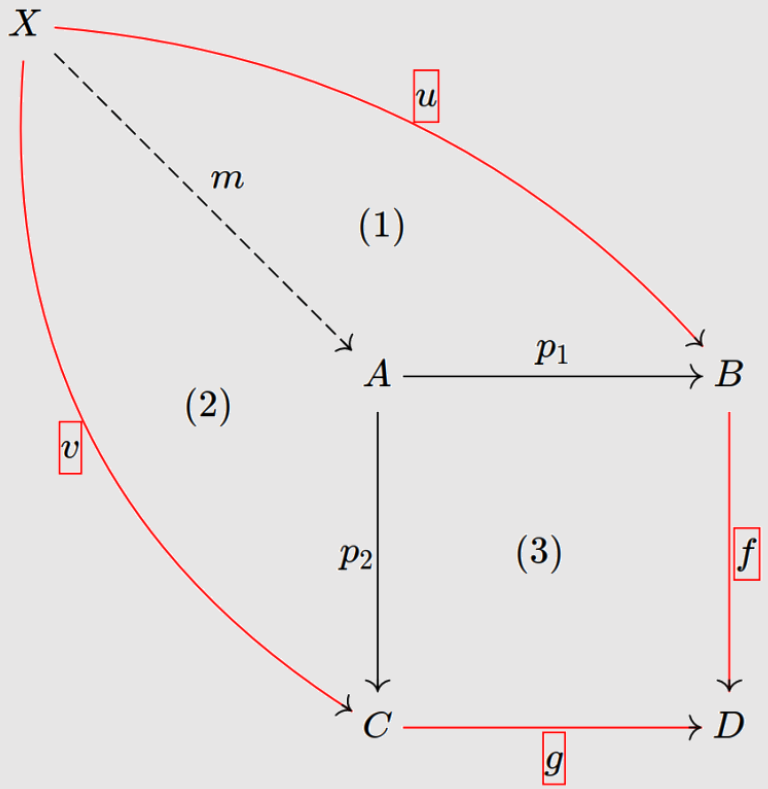
1) Save all subdiagrams as rewrite rules: “top right branch” → “bottom left branch”



Proof generation: sketch of the algorithm

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

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

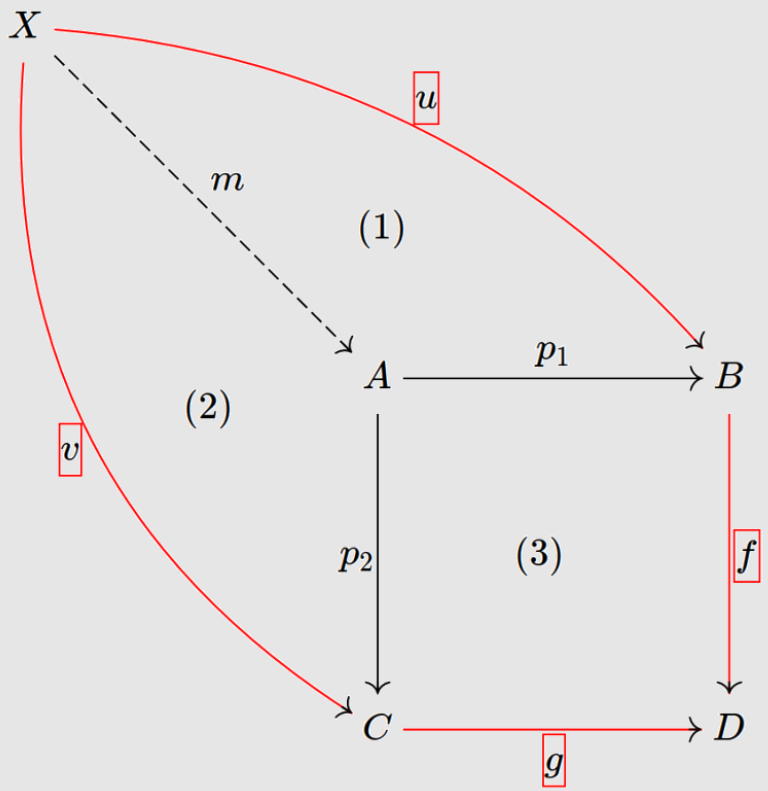


Proof generation: sketch of the algorithm

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

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

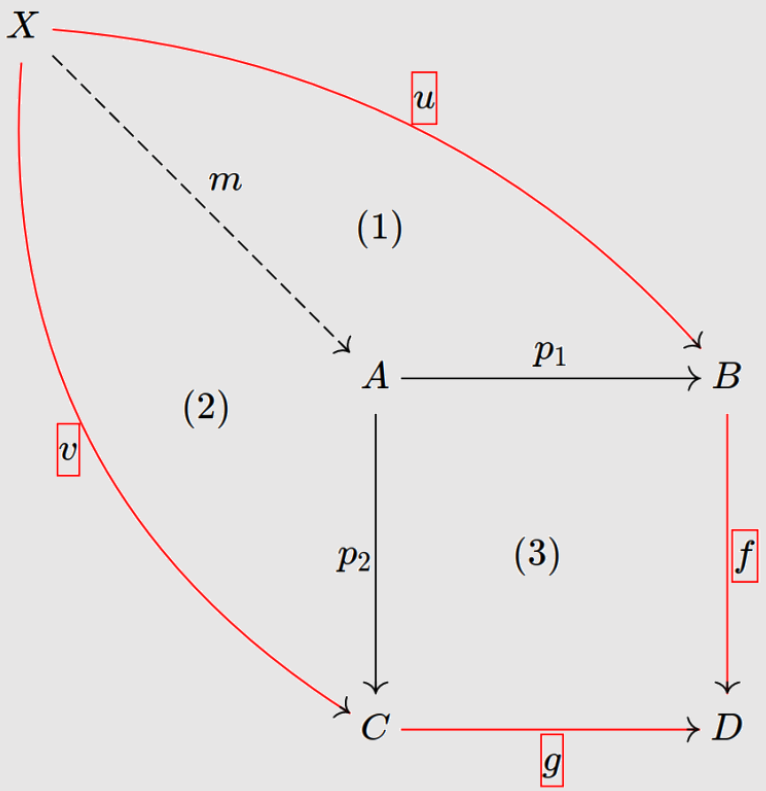
$(2)\ p_2 \circ m \to v$



Proof generation: sketch of the algorithm

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

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

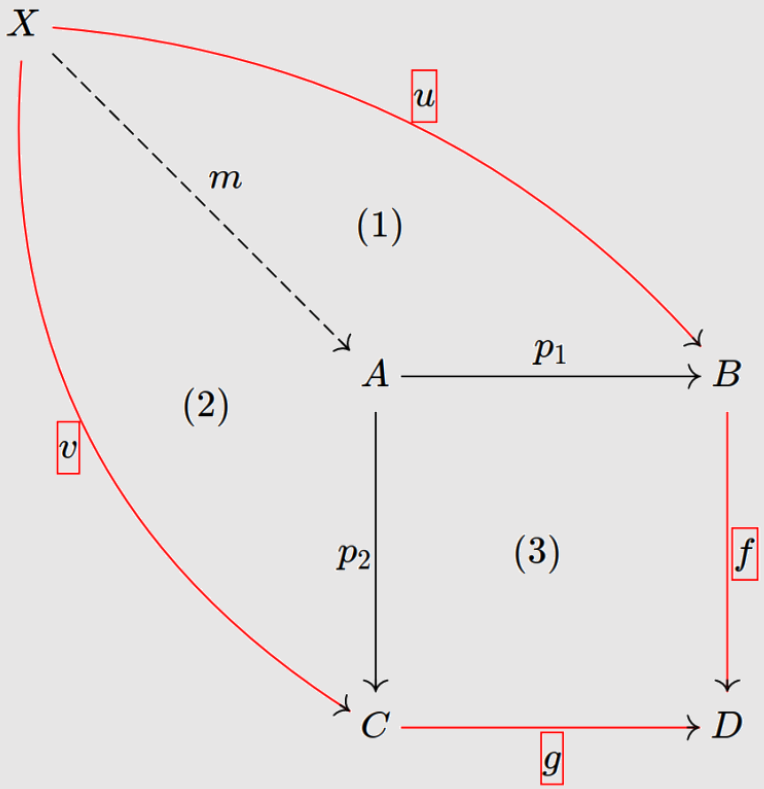


Proof generation: sketch of the algorithm

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

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

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



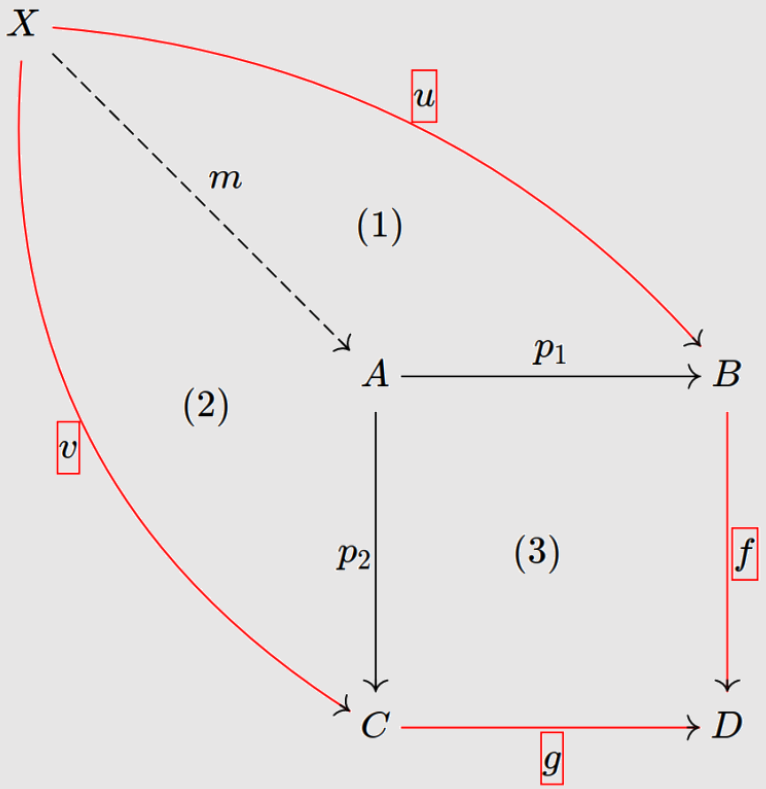
Proof generation: sketch of the algorithm

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

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

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

$$f \circ u$$



Proof generation: sketch of the algorithm

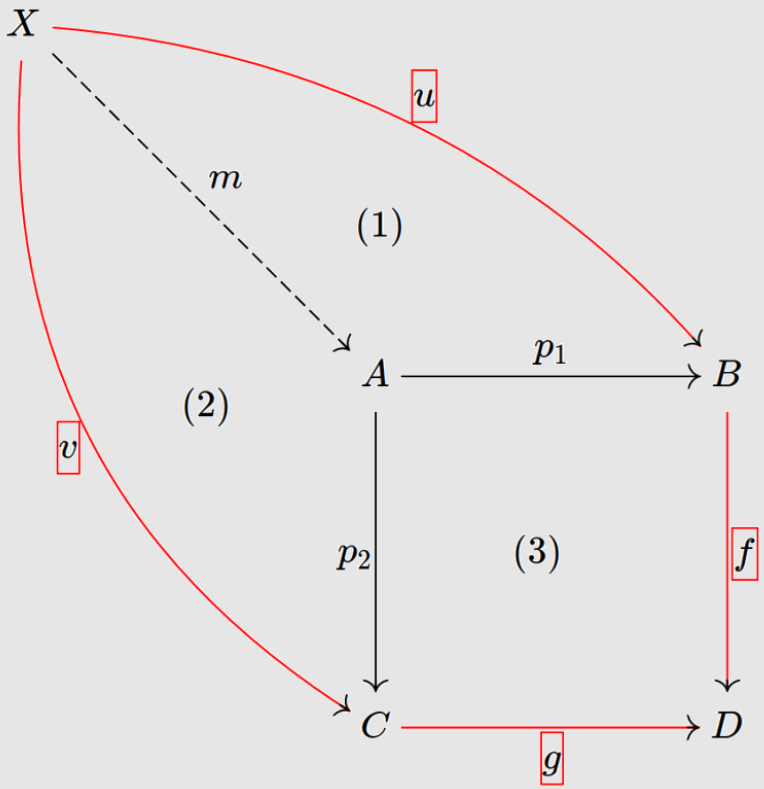
1) Save all subdiagrams as rewrite rules: “top right branch” → “bottom left branch”

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

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” → “bottom left branch”

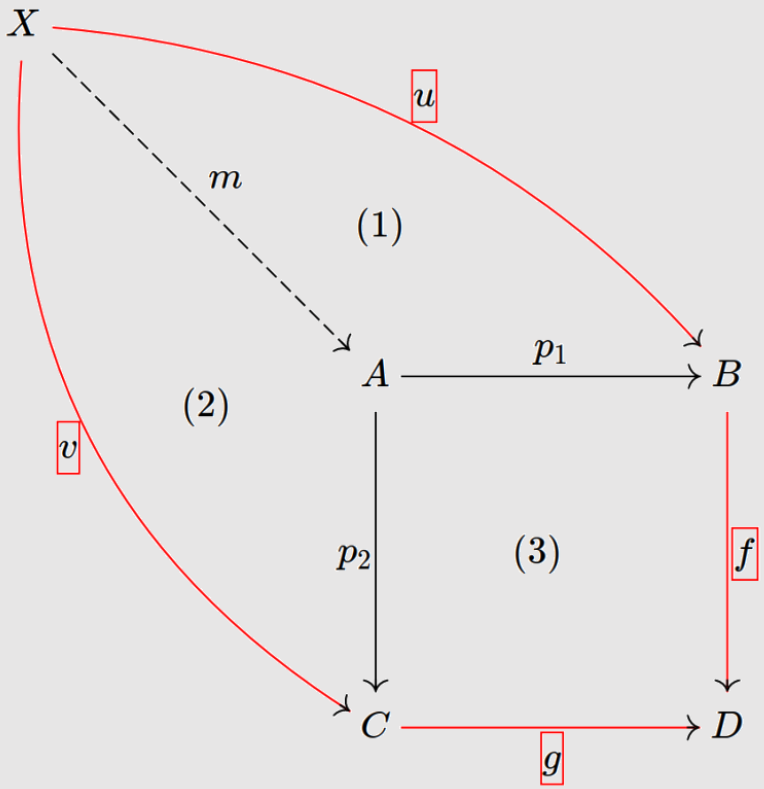
$$(1)\ u \to p_1 \circ m$$
$$(2)\ p_2 \circ m \to v$$
$$(3)\ f \circ p_1 \to g \circ p_2$$

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

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



Proof generation: sketch of the algorithm

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

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

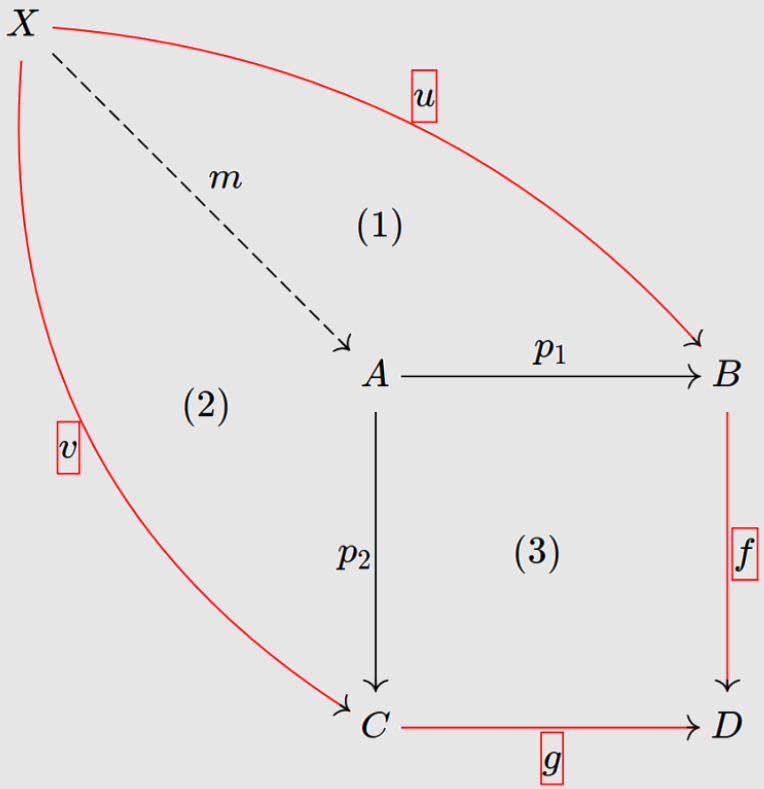
2) Identify the top right branch of the outer diagram.

$$f \circ u$$

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

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

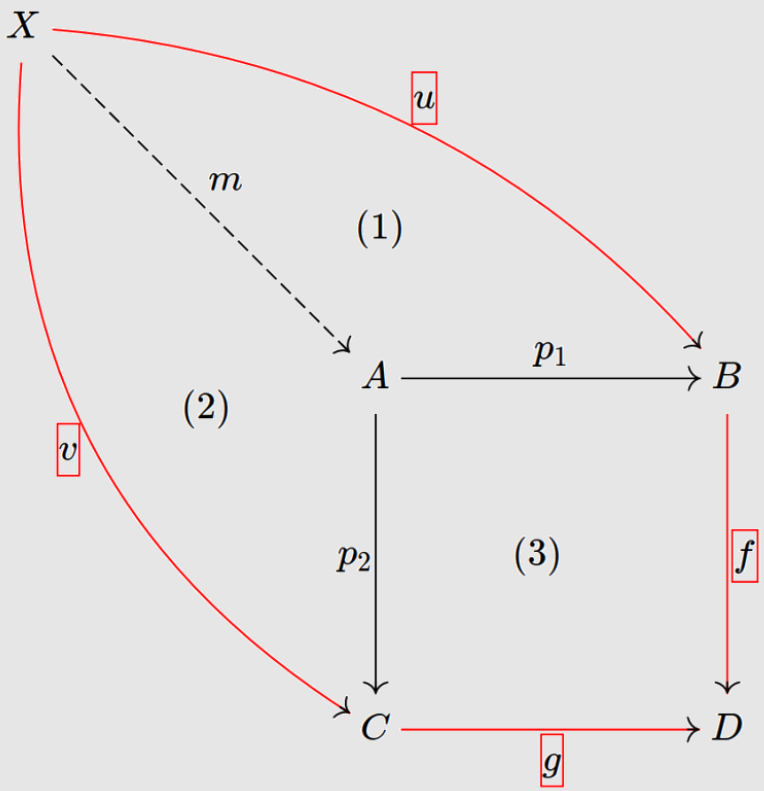
4) Remember the rewrite steps and generate the coq proof script accordingly.



Proof generation: sketch of the algorithm

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

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



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

$$f \circ u$$

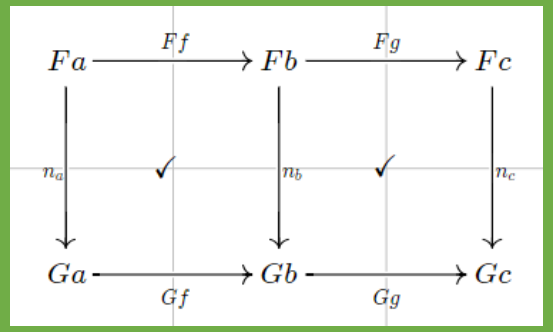
3) Use greedily the rewrite rules until reaching the bottom left branch

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

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

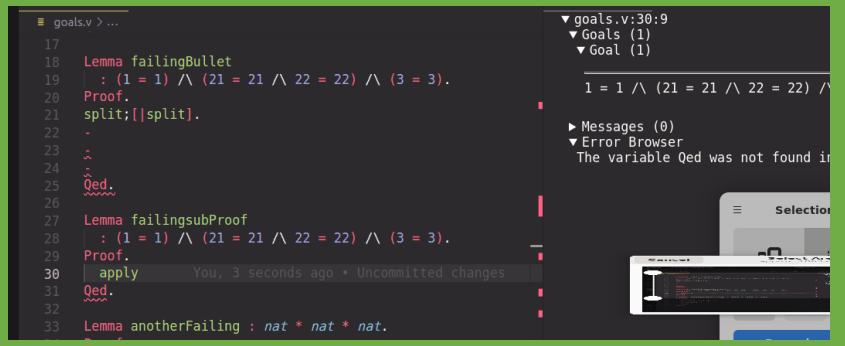
Diagram editor
(standalone version)



Generate proofs

Custom vscode extension (building upon coq-lsp)

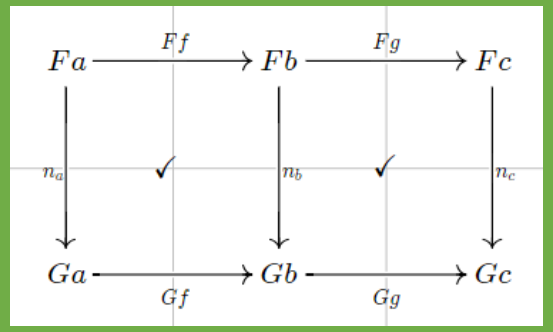
Visual Studio Code



(+ Coq library for custom notations)

Architecture

Diagram editor
(standalone version)

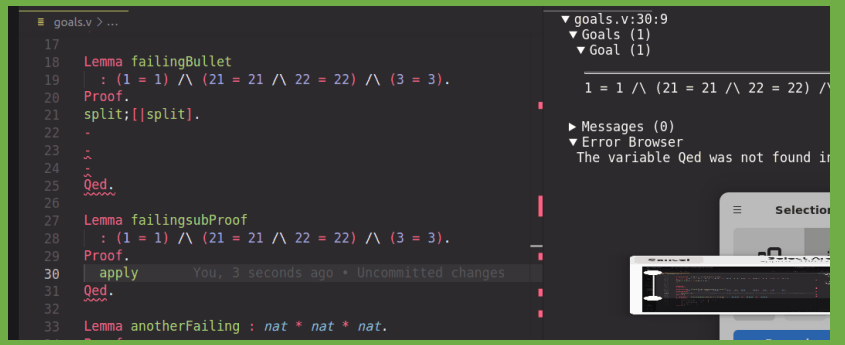


Generate proofs

Show diagram under cursor

Custom vscode extension (building upon coq-lsp)

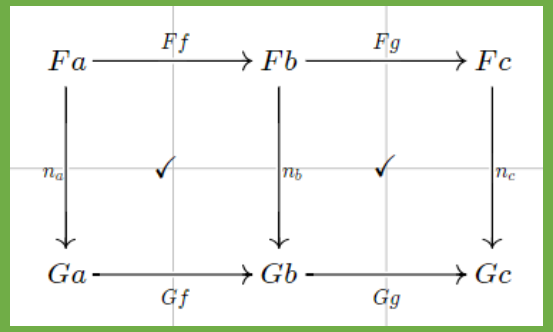
Visual Studio Code



(+ Coq library for custom notations)

Architecture

Diagram editor
(standalone version)



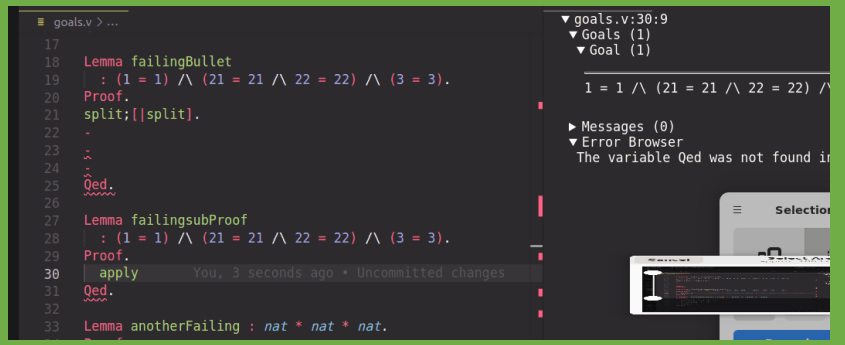
Generate proofs

Show diagram under cursor

Construct diagrams step by step

Custom vscode extension (building upon coq-lsp)

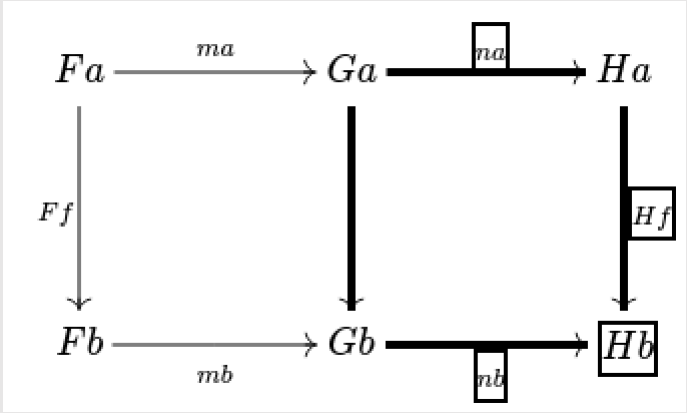
Visual Studio Code



(+ Coq library for custom notations)

Construction of a diagrammatic proof with Coq + YADE

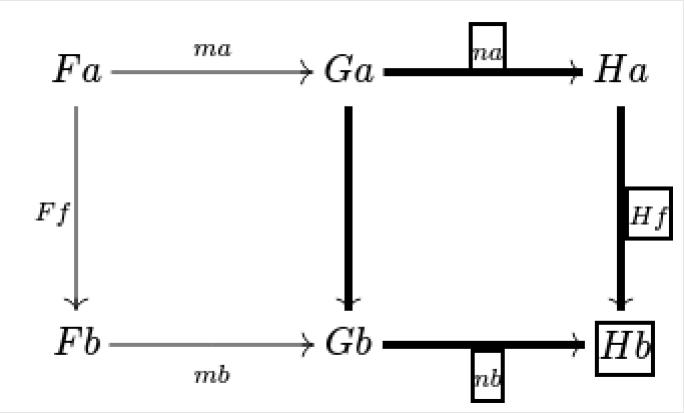
1) Select a subdiagram



Construction of a diagrammatic proof with Coq + YADE

1) Select a subdiagram

2) Execute the export command in the diagram editor

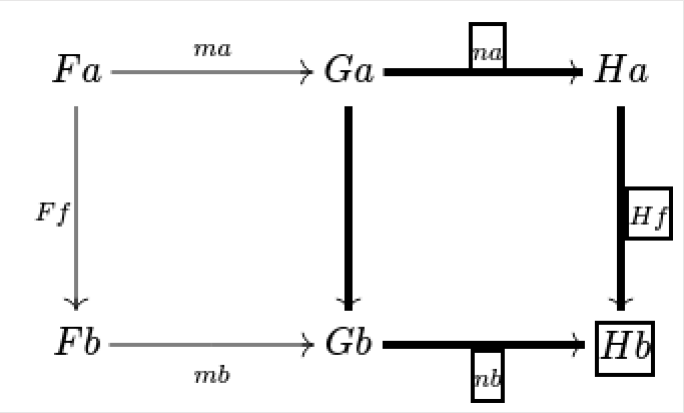


Construction of a diagrammatic proof with Coq + YADE

1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality



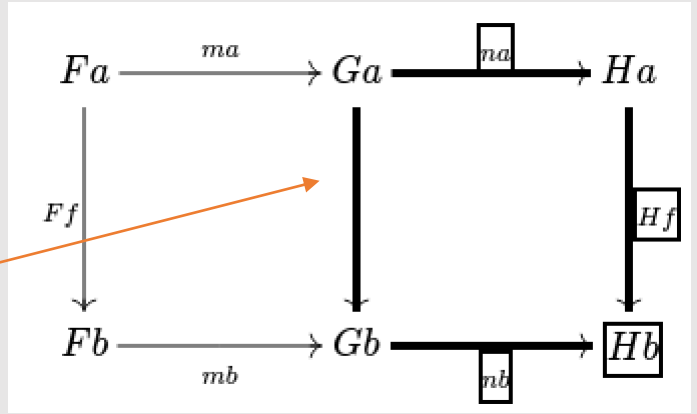
Construction of a diagrammatic proof with Coq + YADE

1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality

unnamed arrow




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

Construction of a diagrammatic proof with Coq + YADE

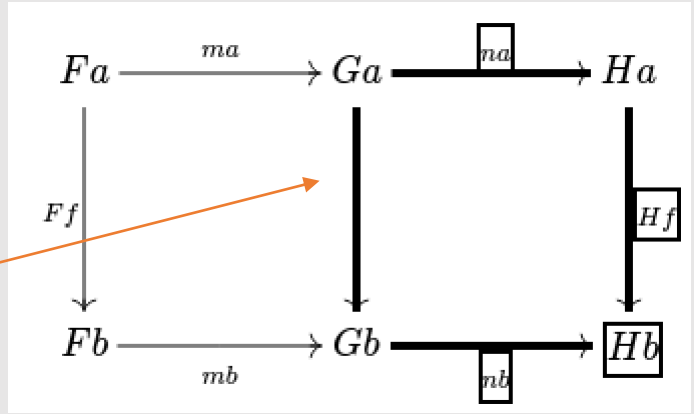
1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality

unnamed arrow

```
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)

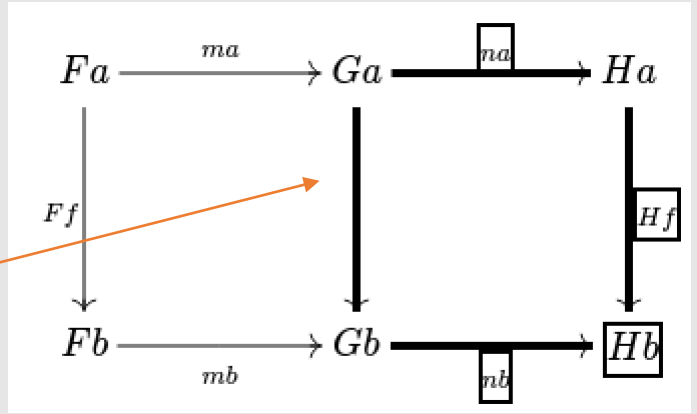
Construction of a diagrammatic proof with Coq + YADE

1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality

unnamed arrow




```
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)

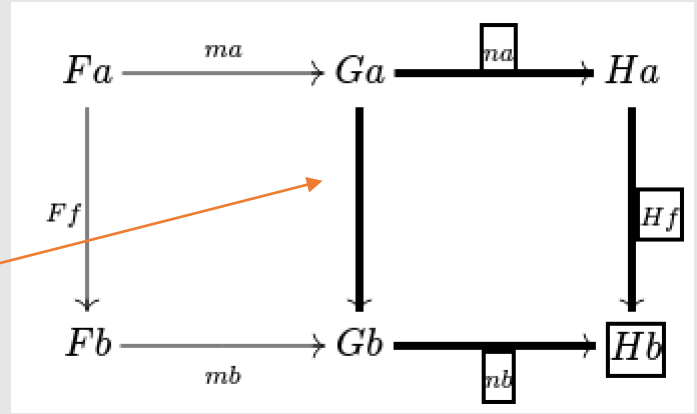
5) Execute the export command in vscode

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



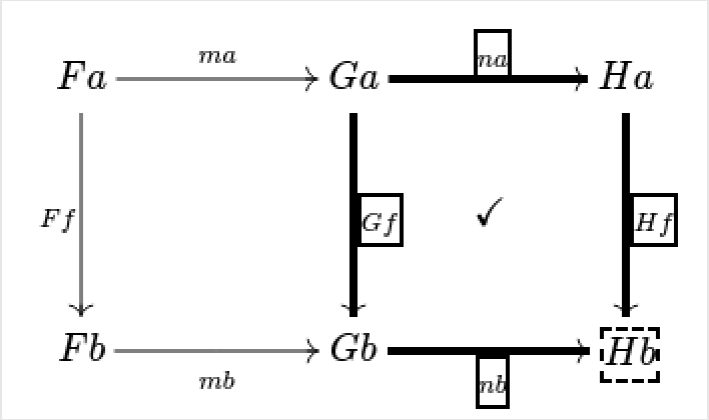
unnamed arrow

```
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:

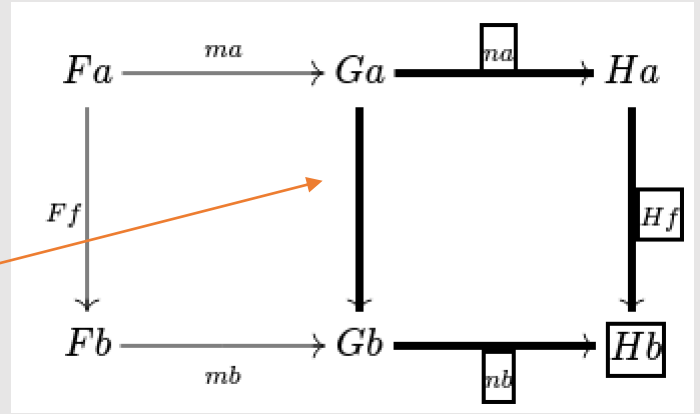


Construction of a diagrammatic proof with Coq + YADE

1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality



unnamed arrow

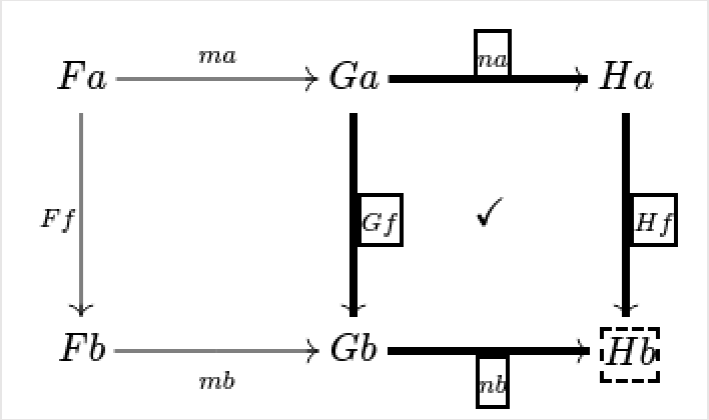
```
eassert (yade : { {n a} · H f = {_} · {n b} }).
{
(* Coq proof to be written here by the user *)
}
```

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

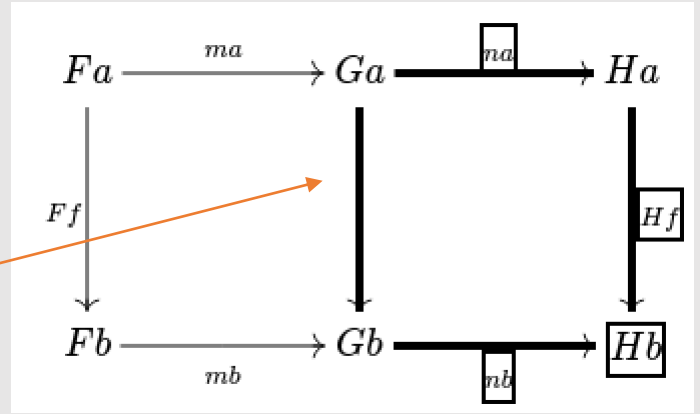


Construction of a diagrammatic proof with Coq + YADE

1) Select a subdiagram

2) Execute the export command in the diagram editor

⇒ vscode asks for a Coq proof of equality



unnamed arrow

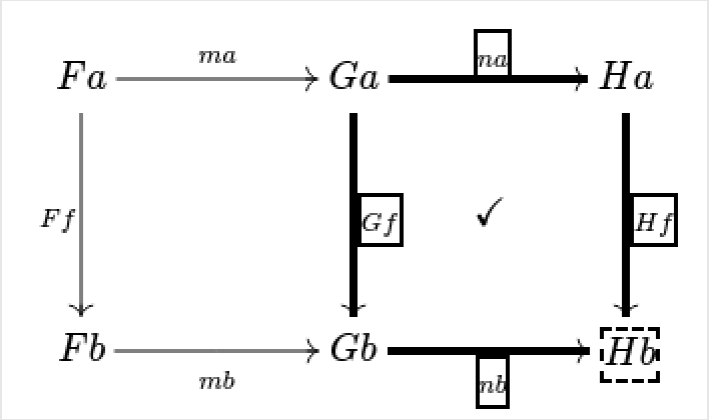
```
eassert (yade : { {n a} · H f = {_} · {n b} }).
{
(* Coq proof to be written here by the user *)
}
```

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

5) Execute the export command in vscode

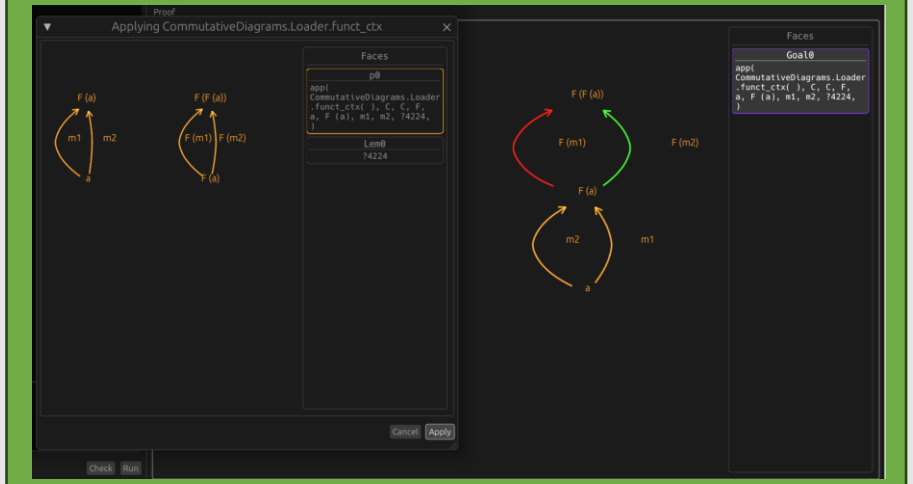
⇒ The diagram gets completed in the editor window:

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



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

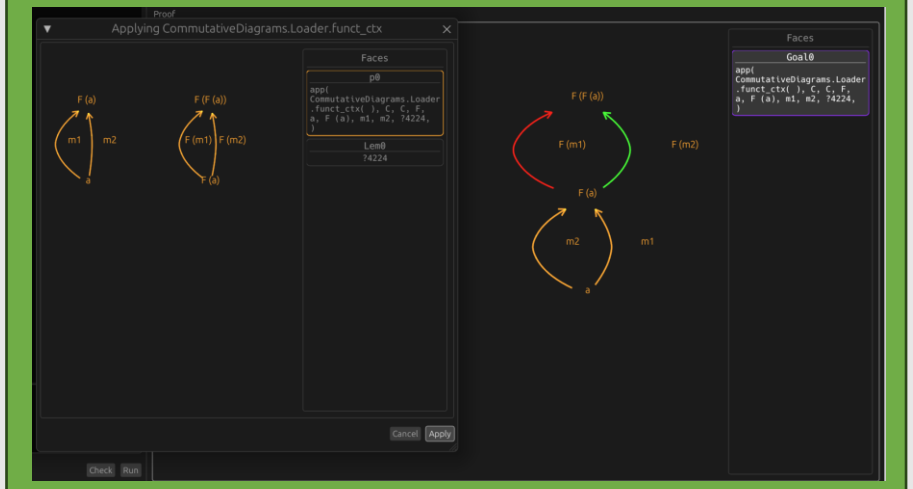
Graphical interface
(standalone program,
implemented in Rust)



Coq
plugin
(ocaml)

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

Graphical interface
(standalone program,
implemented in Rust)

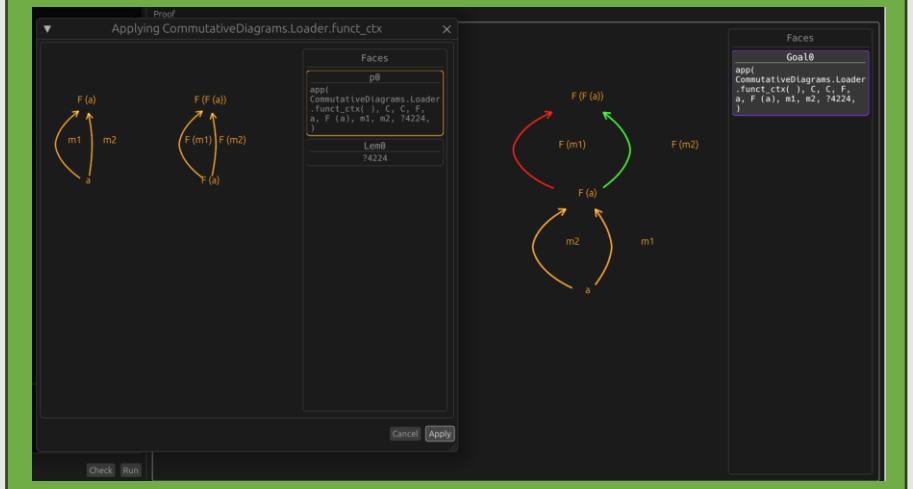


Coq
plugin
(ocaml)

```
Lemma …
Proof.
  …
  diagram server.
```

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

Graphical interface
(standalone program,
implemented in Rust)

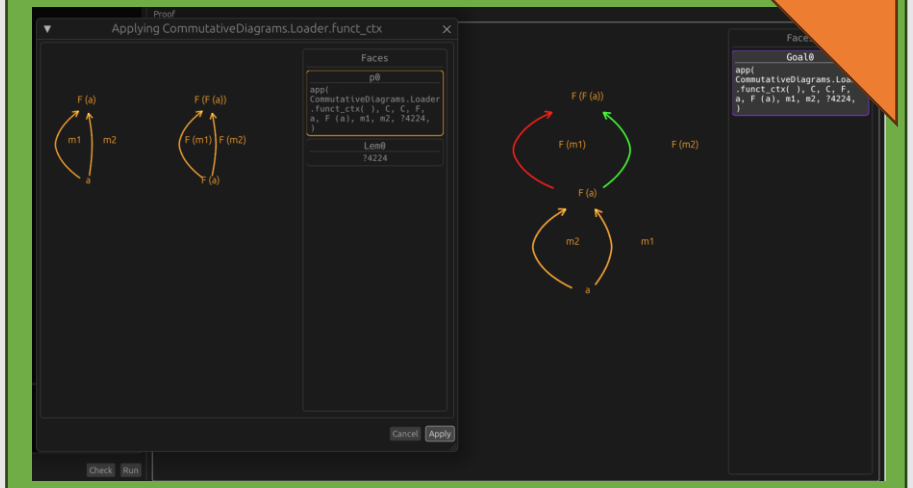


Coq
plugin
(ocaml)

```
Lemma …
Proof.
  …
  diagram server.
```

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

Graphical interface (standalone program, implemented in Rust)

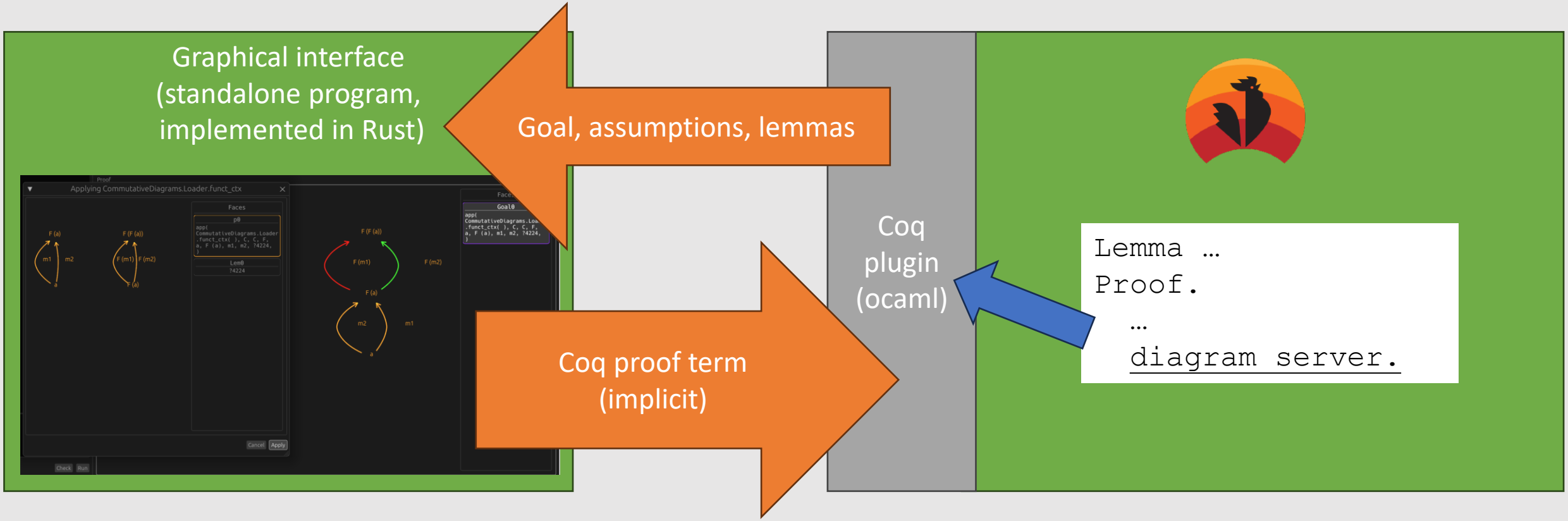


Goal, assumptions, lemmas

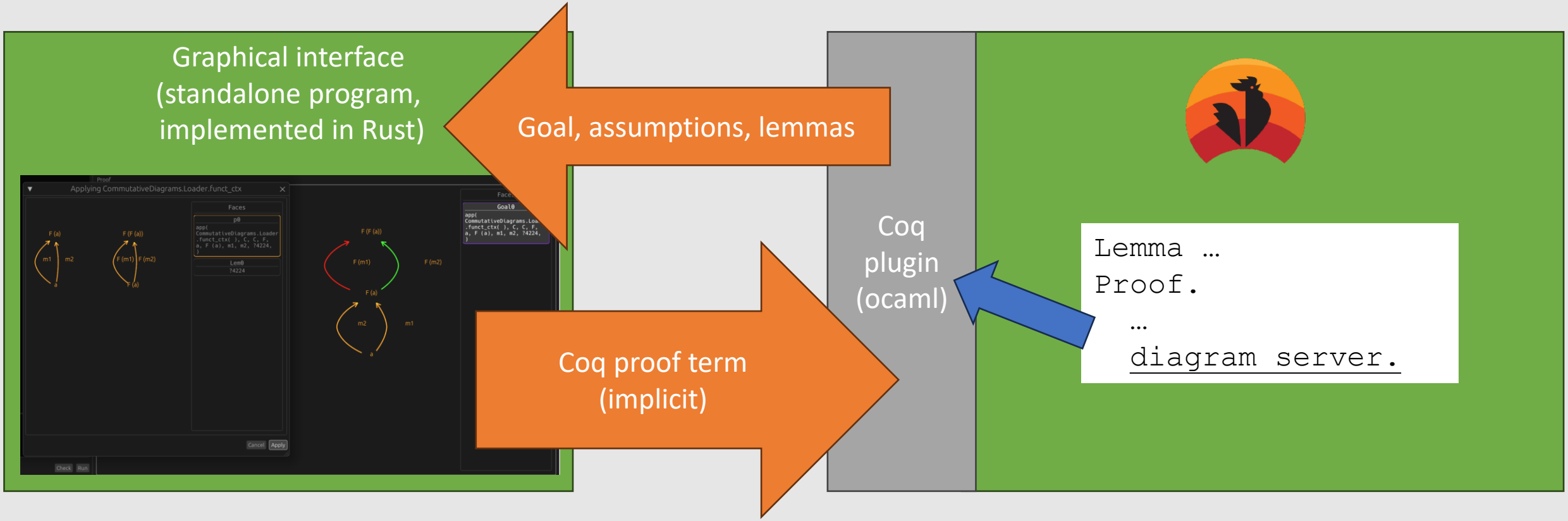
Coq plugin (ocaml)

```
Lemma …
Proof.
  …
  diagram server.
```

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

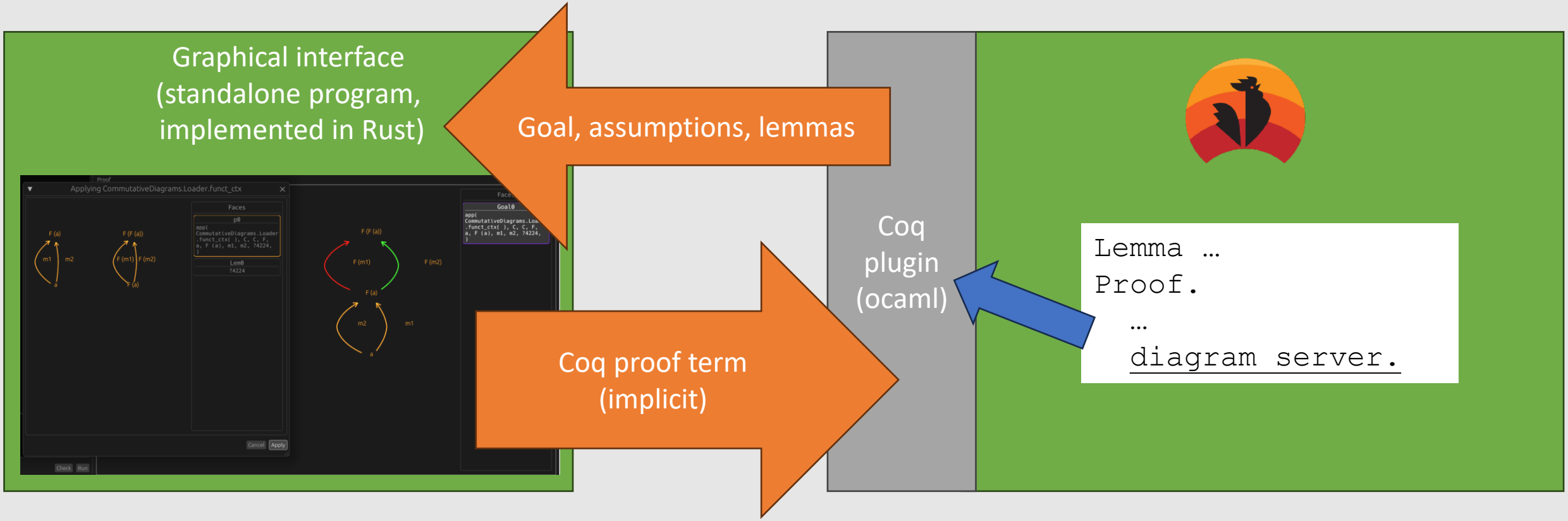


Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs



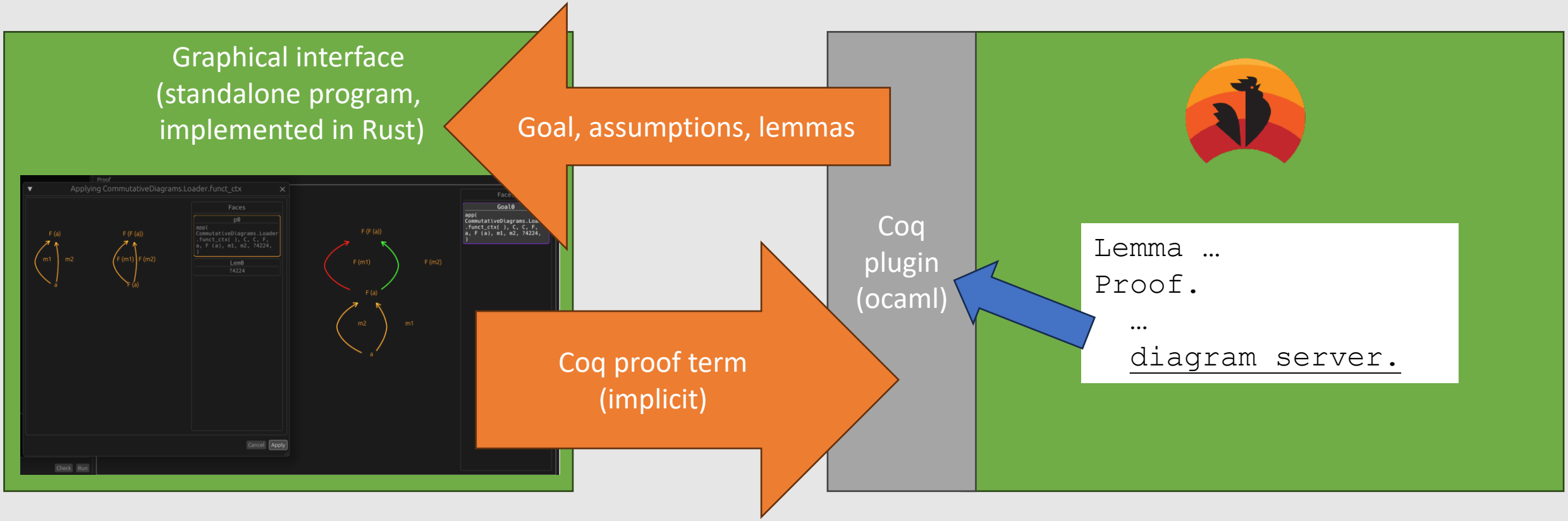
- Automatic layout (limited manual editing)

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



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

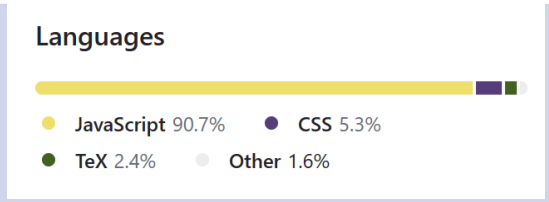
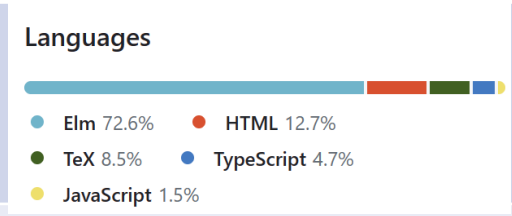
Another related software for mechanisation: Luc Chabassier's interface for diagrammatic proofs



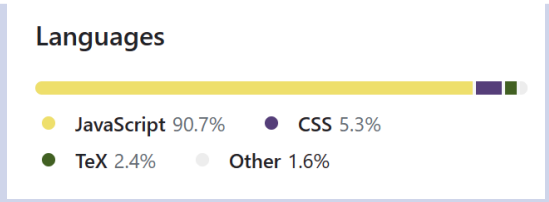
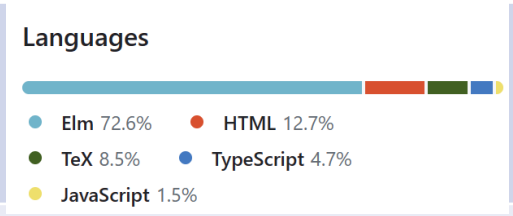
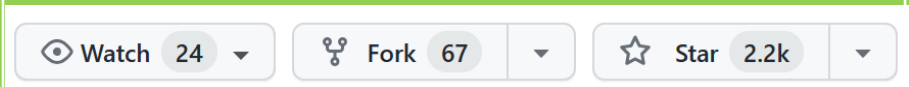
- Automatic layout (limited manual editing)
- Proof tactics (including invoking a lemma)

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

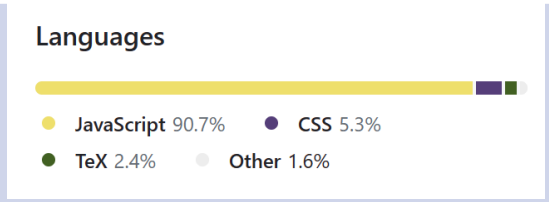
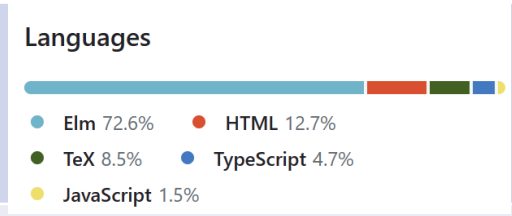
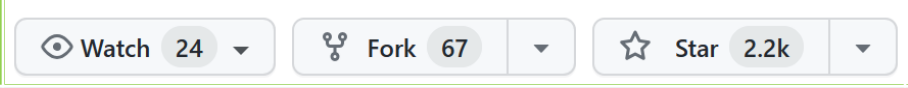
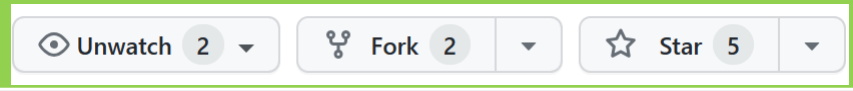
Comparison with quiver

	Quiver	YADE
Programming Languages		
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		
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		
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[1]

[1] https://github.com/amblafont/vscode-yade-example

How to try Coreact-YADE

Requirements for the full example available on github[1]

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

Coreact-YADE

[1] https://github.com/amblafont/vscode-yade-example

How to try Coreact-YADE

Requirements for the full example available on github[1]

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

Coreact-YADE

Coq

[1] https://github.com/amblafont/vscode-yade-example

How to try Coreact-YADE

Requirements for the full example available on github[1]

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

Coq

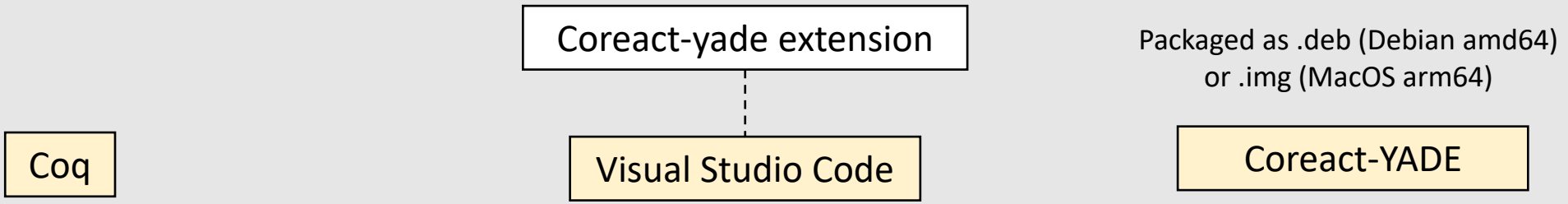
Visual Studio Code

Coreact-YADE

[1] https://github.com/amblafont/vscode-yade-example

How to try Coreact-YADE

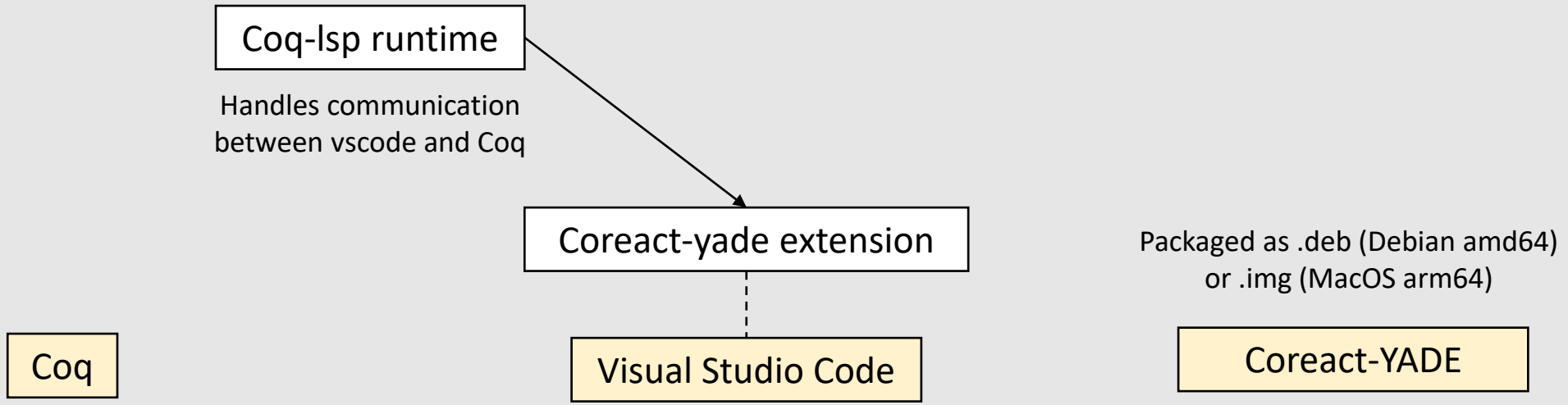
Requirements for the full example available on github[1]



[1] https://github.com/amblafont/vscode-yade-example

How to try Coreact-YADE

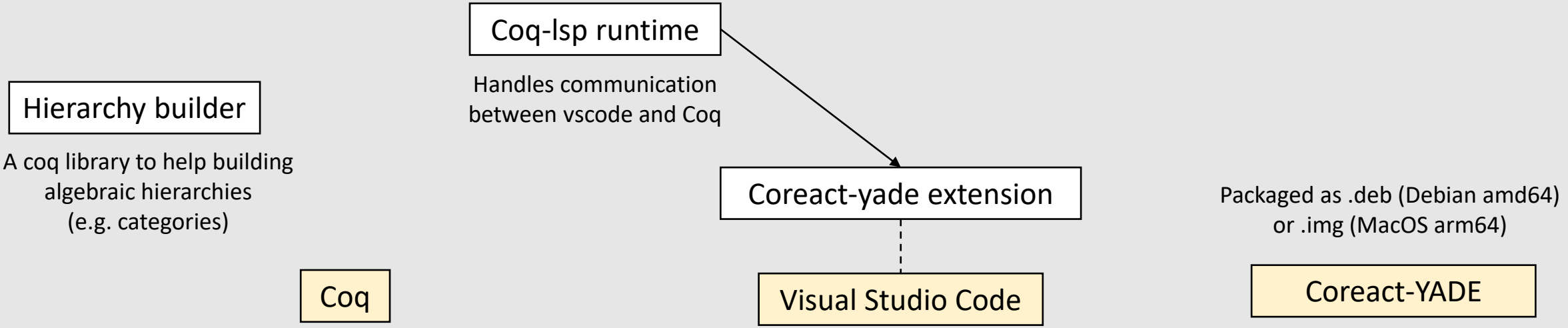
Requirements for the full example available on github[1]



[1] https://github.com/amblafont/vscode-yade-example

How to try Coreact-YADE

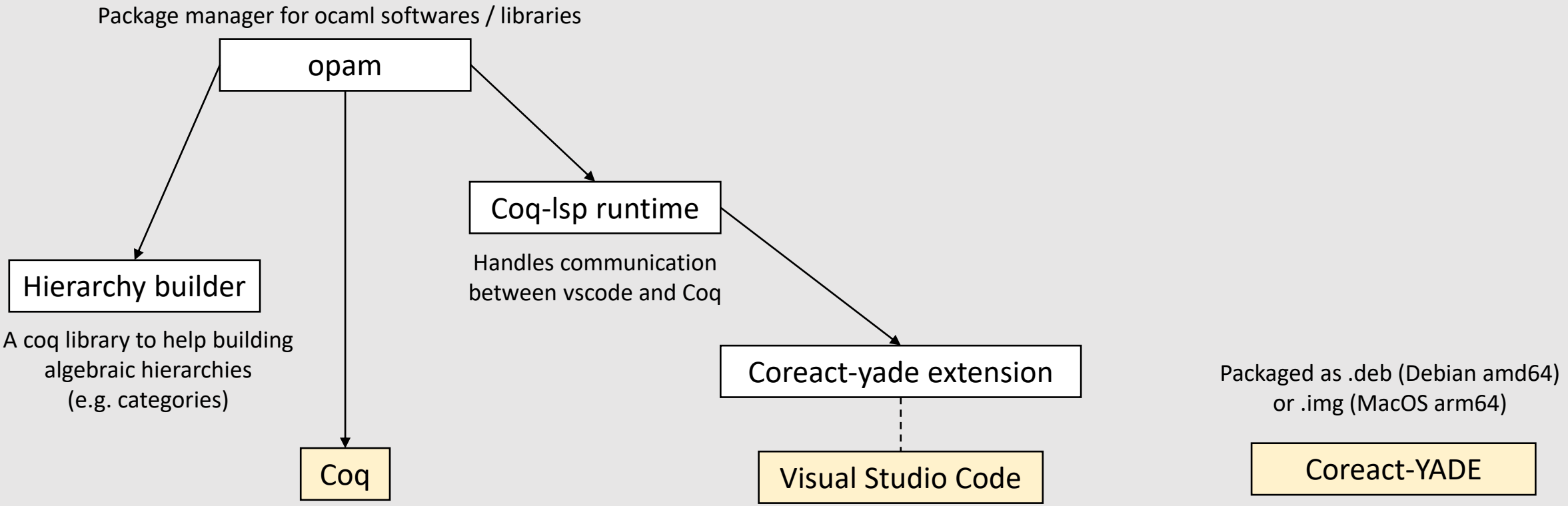
Requirements for the full example available on github[1]



[1] https://github.com/amblafont/vscode-yade-example

How to try Coreact-YADE

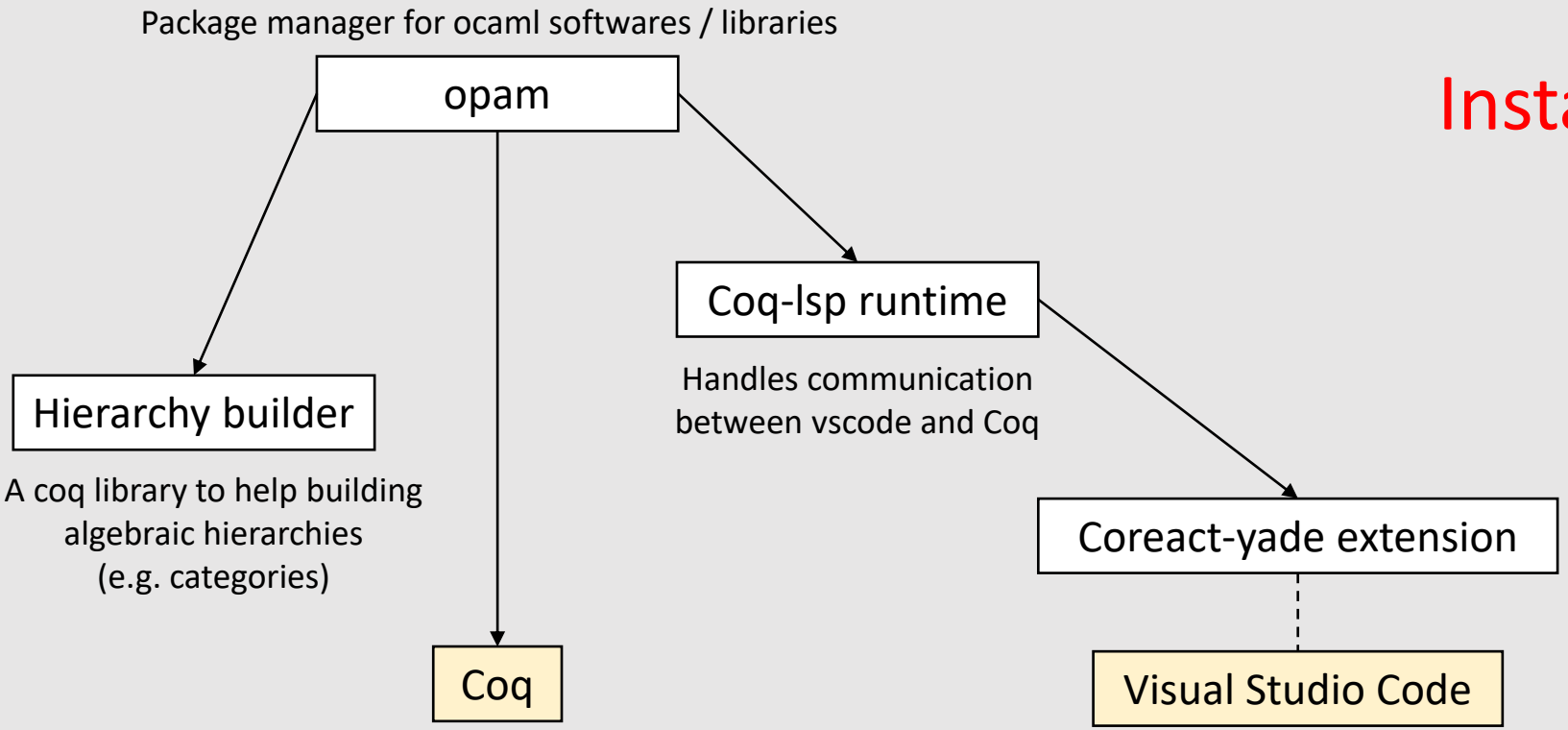
Requirements for the full example available on github[1]



[1] https://github.com/amblafont/vscode-yade-example

How to try Coreact-YADE

Requirements for the full example available on github[1]



Installation is a bit tedious.

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

Coreact-YADE

[1] 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.

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).

What I did →

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

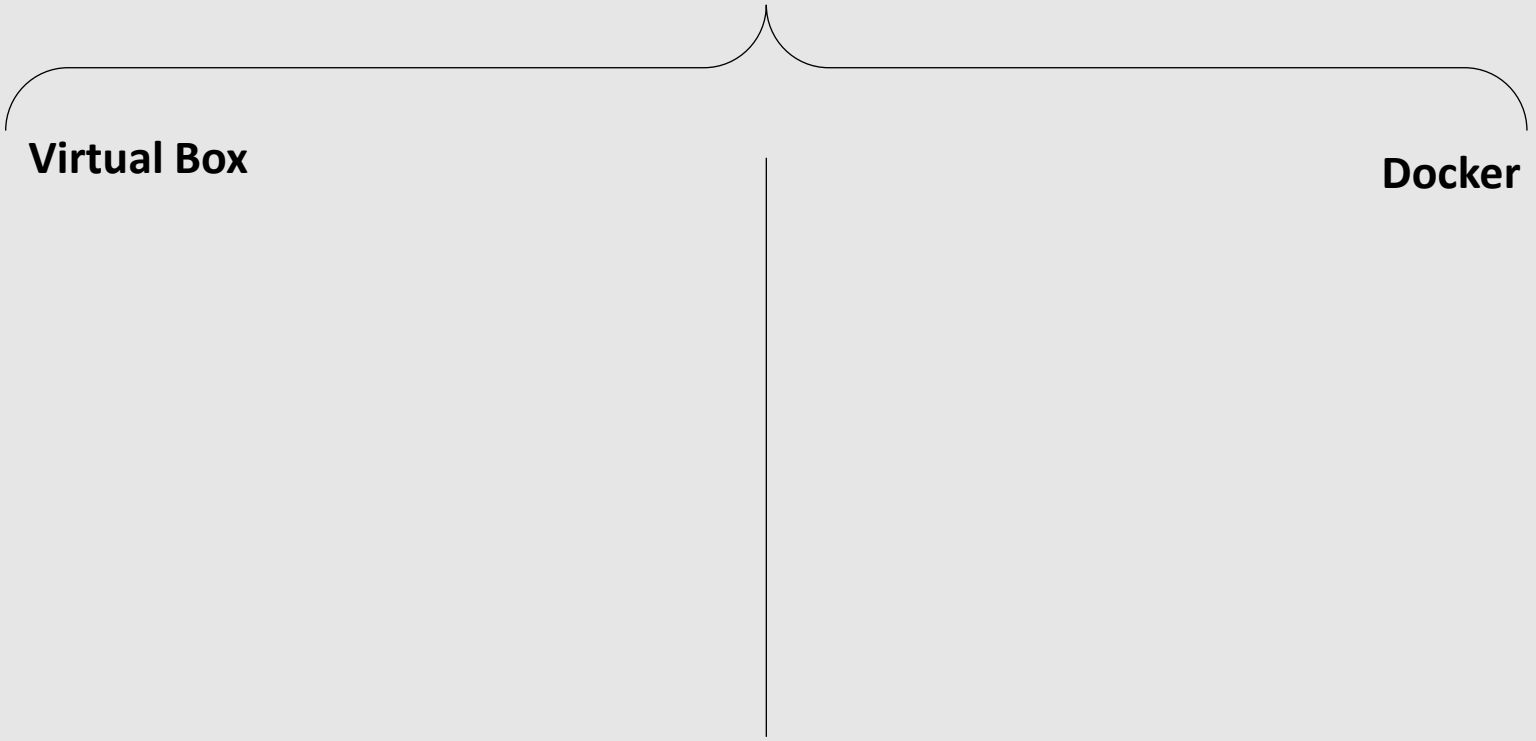
Testing YADE in an emulated environment

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

Installation script

A shell script to be run in a fresh Debian image[1]

Docker

[1]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[1]

[1]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[1]

[1]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[1]

[1]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[1]

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

[1]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[1]

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

[1]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[1]

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)

[1]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[1]

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)

[1]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[1]

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!

[1]https://www.linuxvmimages.com/