A categorical diagram editor to help formalising commutation proofs

Ambroise Lafont

University of Cambridge (postdoc)

GReTA-ExACT online workgroup, March 2022

Motivation

- **Diagrammatic reasoning** is useful

$$\begin{tikzcd}
\Sigma(X) \otimes X \arrow[r, "\Sigma(X) \otimes \eta^S_X"] \arrow[dr, equal] \arrow[dd, "a \otimes X"'] & \Sigma(X) \otimes S(X) \arrow[r, "st_{X,(X,v_X)}"] \arrow[d, "\Sigma(X) \otimes \bar{a}"'] & \Sigma(X \otimes SX) \arrow[r, "in_2"] \arrow[d, "\Sigma(X \otimes \bar{a})"'] & \Sigma^+(X \otimes SX) \arrow[r, "\eta_{\Sigma^+, in_1}"] & S(X \otimes SX + X + X) \arrow[d, "S(X \otimes \bar{a} + X + X)"'] \\
 & \Sigma(X) \otimes X \arrow[r, "st_{X,(X,v)}"'] & \Sigma(X \otimes X) \arrow[r, "in_2"'] \arrow[d, "\Sigma(s)"] & \Sigma^+(X \otimes X) \arrow[r, "\eta_{\Sigma^+, in_1}"'] \arrow[d, "\Sigma^+(s)"] & S(X \otimes X + X + X) \arrow[d, "S[s,X,X]"'] \\
X \otimes X \arrow[drr, "s"'] & & \Sigma(X) \arrow[r, "in_2"] \arrow[d, "a"] & \Sigma^+(X) \arrow[r, "\eta_{\Sigma^+, X}"] & SX \arrow[dll, "\bar{a}"] \\
 & & X & &
\end{tikzcd}$$

$\times$ Theorem provers (e.g., Coq) are **text-based**

$\Rightarrow$ need for a device to bridge the gap

YADE

Yet Another Diagram Editor

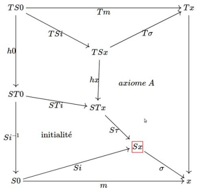
Proof assistant
(Coq)

```
Lemma commutes :
  n_y ∘ F f = G f ∘ n_x.
```

Proof / statement display

Proof generation

Diagram editor
(YADE)



$$\begin{tikzcd}
TS0 \arrow[rrr, "Tm"'] \arrow[dr, "TSi"'] \arrow[dd, "h0"'] & & & Tx \arrow[dddd] \\
& TSx \arrow[urr, "T\sigma"] \arrow[d, "hx"'] & axiome\ A & \\
ST0 \arrow[r, "STi"'] \arrow[dd, "Si^{-1}"'] & STx \arrow[dr, "S\tau"'] & & \\
& \text{initialité} & \boxed{Sx} \arrow[dr, "\sigma"] & \\
S0 \arrow[urr, "Si"'] \arrow[rrr, "m"'] & & & x
\end{tikzcd}$$

About YADE

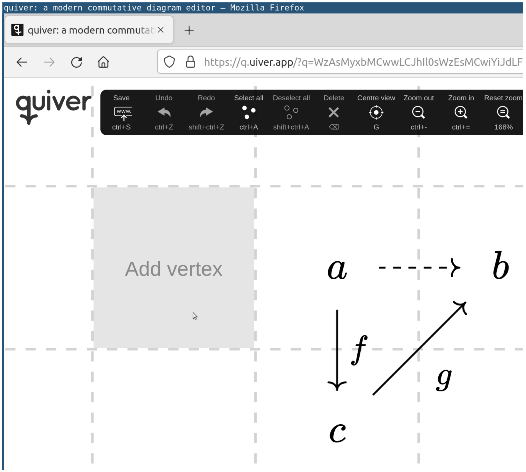
- Available online[1]
- Written in Elm (∼6000 LoC)
- Diagram editor with **Coq proof script generation**
- Export to quiver

quiver is 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.

Why not as an extension of quiver?

[1] `https://amblafont.github.io/graph-editor/index.html`

Why not as an extension of Quiver?


quiver: a modern commutative diagram editor — Mozilla Firefox
quiver: a modern commutat
https://q.uiver.app/?q=WzAsMyxbMCwwLCJhIl0sWzEsMCwiYiJdLF
quiver
Save ctrl+S
Undo ctrl+Z
Redo shift+ctrl+Z
Select all ctrl+A
Deselect all shift+ctrl+A
Delete ⌫
Centre view G
Zoom out ctrl+-
Zoom in ctrl+=
Reset zoom 168%
Add vertex

$$\begin{tikzcd} a \arrow[r, dashed] \arrow[d, "f"] & b \\ c \arrow[ur, "g"'] & \end{tikzcd}$$

No strong reason, but

- This project started before quiver was out.
- I want to be able to draw out of a grid.
- I wanted to experiment with Elm.

Plan

Plan

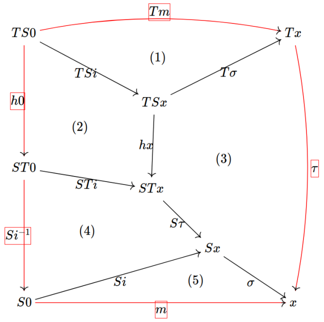
Diagrams are proofs

$$\begin{tikzcd}
TS0 \arrow[rrr, bend left, "\boxed{Tm}"] \arrow[dr, "TSi"] \arrow[dd, "\boxed{h0}"'] & & (1) & Tx \arrow[ddddd, bend left, "\boxed{\tau}"] \\
(2) & TSx \arrow[urr, "T\sigma"] \arrow[dd, "hx"] & & \\
ST0 \arrow[dr, "STi"] \arrow[ddd, "\boxed{Si^{-1}}"'] & & (3) & \\
 & STx \arrow[dr, "S\tau"] & & \\
 & (4) & Sx \arrow[dr, "\sigma"] & \\
S0 \arrow[urr, "Si"] \arrow[rrr, "\boxed{m}"'] & & (5) & x
\end{tikzcd}$$

Generated formalised statement

If inner subdiagrams (1)-(5) commute,
then the outer diagram commutes.

Algorithm for proof generation



$$\begin{tikzcd}
TS0 \arrow[rrrr, bend left, "\boxed{Tm}"] \arrow[drr, "TSi"] \arrow[dd, swap, "\boxed{h0}"] & & (1) & & Tx \arrow[dddd, bend left, "\boxed{\tau}"] \\
& (2) & TSx \arrow[urr, swap, "T\sigma"] \arrow[d, "hx"] & & \\
ST0 \arrow[rr, "STi"] \arrow[dd, swap, "\boxed{Si^{-1}}"] & & STx \arrow[dr, "S\tau"] & (3) & \\
& (4) & & Sx \arrow[dr, "\sigma"] & \\
S0 \arrow[urrr, "Si"] \arrow[rrrr, swap, "\boxed{m}"] & & (5) & & x
\end{tikzcd}$$

1. Identify all subdiagrams and outer diagrams on the canvas
2. For each outer diagram,
   - start from one branch
   - repeatedly “apply” subdiagrams to progress, until reaching the other branch of the outer diagram.

$\Rightarrow$ Internal representation of a commutation proof

Plan

The UniMath mathematical library

Why targeting this library?

- I use UniMath in my formalisation projects.
- Could be easily adapted to another target.

UniMath (2014-)

- built upon Voevodsky's repository Foundations
- Large Coq mathematical library (∼300 000 lines)
  - ∼ two thirds on (bi)category theory
  - Verbose style

Commutation proofs in UniMath

$$\begin{tikzcd}
TS0 \arrow[rrr, bend left, "Tm"] \arrow[dr] \arrow[dd, "\boxed{h0}"'] & & & Tx \arrow[dddd, bend left, "\tau"] \\
& TSx \arrow[urr] \arrow[d] & & \\
ST0 \arrow[r] \arrow[dd, "\boxed{Si^{-1}}"'] & STx \arrow[dr] & & \\
& & Sx \arrow[drr, "\boxed{\sigma}"] \arrow[d, phantom, "\boxed{(5)}"] & \\
S0 \arrow[urr, "\boxed{Si}"] \arrow[rrr, "\boxed{m}"'] & & & x
\end{tikzcd}$$

Applying[1] (5) to the bottom left branch

$$(h_0 \cdot Si^{-1}) \cdot m$$
rewrite assoc'
$$h_0 \cdot (Si^{-1} \cdot m)$$
apply cancel_precomposition
$$Si^{-1} \cdot m$$
apply cancel_precomposition
$$m$$
apply (5)
$$Si \cdot \sigma$$

[1] Directly rewriting the subdiagram sometimes fail.

Generated Coq script

$$\begin{tikzcd}
TS0 \arrow[rrrr, bend left, "\boxed{Tm}"] \arrow[drr, "TSi"] \arrow[dd, "\boxed{h0}"'] & & (1) & & Tx \arrow[dddd, bend left, "\boxed{\tau}"] \\
& (2) & TSx \arrow[urr, "T\sigma"'] \arrow[d, "hx"] & & \\
ST0 \arrow[rr, "STi"] \arrow[dd, "\boxed{Si^{-1}}"'] & & STx \arrow[dr, "S\tau"] & (3) & \\
& (4) & & Sx \arrow[dr, "\sigma"] & \\
S0 \arrow[urrr, "Si"] \arrow[rrrr, "\boxed{m}"'] & & (5) & & x
\end{tikzcd}$$

$\Longrightarrow$

```
Goal { Tm · τ  = h₀ · Si ⁻¹ · m }.

assert(eq : { Tm = TSi · Tσ }).
{ admit. }
etrans.
{
  apply cancel_postcomposition.
  apply eq.
}
clear eq.
assert(eq : { Tσ · τ  = hx · Sτ · σ }).
{ admit. }
etrans.
{
  repeat rewrite assoc'.
  apply cancel_precomposition.
  repeat rewrite assoc.
  apply eq.
}
repeat rewrite assoc.
clear eq.
assert(eq : { TSi · hx = h₀ · STi }).
{ admit. }
etrans.
{
  do 2 apply cancel_postcomposition.
  apply eq.
}
...
```

- Subdiagrams are explicitly asserted and admitted.
- Lots of boilerplate code (focusing)

Plan

Future work

- More display options (colours, label placement, ...)
- Attach properties to subdiagrams (e.g., pullback)
- Higher category theory (higher cells are already there)
- More helpers to build diagrams
  (e.g., use one diagram to complete another one)
- Tighter connection with (js)Coq

Plan

Demo

- Label guessing (naturality)
- Proof generation
    - parse a Coq goal (“prettified” by a custom tactic)
    - generate proof script
    - partial fill