A diagram editor to mechanise categorical proofs

Ambroise Lafont

A diagram editor to mechanise categorical proofs

Démonstration de logiciel

Ambroise Lafont[1]

[1]École Polytechnique, Palaiseau, France

Diagrammatic proofs are ubiquitous in certain areas of mathematics, especially in category theory. Mechanising such proofs is a tedious task because proof assistants (such as Coq) are text based. We present a prototypical diagram editor to make this process easier, building upon the vscode extension coq-lsp for the Coq proof assistant and a web application available on the author's personal website. It currently targets the UniMath mathematical library for the Coq proof assistant, but could in principle easily be adapted to other targets.

1 Introduction

Showing that two morphisms between two objects are equal is a ubiquitous task in category theory. Those morphisms are typically themselves compositions of other morphisms.

Example 1. Assuming $d \circ f_1 = u$ and $f_2 \circ d = v$, we deduce that $f_2 \circ u = v \circ f_1$, by rewriting u and v using the two assumed equalities.

Proofs by rewriting, as above, does not pose specific challenge when it comes to mechanisation in proof assistants. However, pen and paper proofs typically adopt a different *diagrammatic* approach: the two composition chains of morphisms that we want to prove equal are first drawn as chains of arrows sharing the same start and end points, thus delineating a shape with two *branches* (the two chains of arrows). Thinking of equality as a "filling", a diagrammatic proof consists in decomposing this shape into smaller juxtaposed *inner* shapes whose fillings are justified.

Example 2. A diagrammatic proof of Example 1 consists in the below right diagram, obtained by splitting the square below left into two juxtaposed "filled" triangles.

Diagrammatic proofs include enough information to construct a mechanisable proof by rewriting. This time-consuming translation is ubiquitous when mechanising categorical

results. We present a prototypical diagram editor that can automatically performs this task: from the right diagram of Example 2, the editor generates a Coq proof script that shows the desired equality using the assumed equalities.

Plan of the paper We quickly describe the technology behind the software in Section 2, before presenting the main features in Section 3. In Section 4, we finally mention some related work.

2 Technology

The software consists of three main components: a diagram editor (about 7000 lines of code), and a vscode extension (about 300 lines of code), and a small Coq library (about 100 lines of code).

2.1 The diagram editor

The diagram editor is mainly implemented in Elm, a functional programming language that compiles to JavaScript. LaTeX labels are rendered using the KaTeX JavaScript library. It is available as a web application that runs in the browser [Laf], as a standalone desktop application (with some additional features) that embeds the web application in an electron runtime.

The diagram editor generates proof scripts relying on the mechanised UniMath mathematical library [VAG+].

2.2 The vscode extension

The vscode extension builds upon coq-lsp [CJGAI], which provides the vscode editor with support for the Coq proof assistant. Our extension interacts with the standalone version of the diagram editor in mainly two ways: to render the Coq goal at the cursor as a diagram (if the Coq goal is indeed an equality between morphisms), and to insert the Coq proof generated from the diagram at the cursor location.

2.3 The Coq library

The Coq library introduces some notations and tactics to convert a Coq goal context into the input format of the editor, where the objects are explicitly mentioned. For example, the top right branch of the left diagram in Example 2 is denoted by $u \cdot f_2$ in UniMath. The vscode extension would call our pretty-printing tactic `norm_graph` to convert it into the string A_1 `--` u `->` A_2 `--` f_2 `->` B_2, which is then sent to the editor for display.

3 Features

In this section, we present the editing capabilities of our software in Section 3.1, then the features related to mechanisation of diagrammatic proof in 3.2, and finally, in Section 3.3, we explain how our diagram editor can be used when writing a LaTeX document with diagrams.

3.1 Diagram editor

Beyond basic editing features, our software offers (among other things) tab management, an optional grid, automatic guessing of labels when completing a (naturality) square, find & replace, z-indices (to handle edge overlaps), quicksaving. A diagram can be exported[1] to LaTeX, json, or svg. A diagram saved in the json format can be reloaded.

[1]We thank Tom Hirschowitz for implementing the LaTeX export feature.

3.2 Mechanisation

Mechanising a diagrammatic proof with our software involves two steps that we detail in this section: construction of the diagrammatic proof, and the generation of the mechanised proof[2].

Construction of a diagrammatic proof Exploiting the basic editing features of the software, the user splits a shape by creating a mediating chain of arrows, as in Example 2, and selects one of the newly created inner shapes. The mediating arrows can be unnamed, leaving their definitions to be inferred later. Then, the editor generates a Coq script that states the equality corresponding to the shape. The unnamed arrows become "holes" that will be guessed later by Coq using unification, for example when applying some known equality between known morphisms. Once the proof is complete, the resulting statement is loaded back into the editor to replace the unnamed arrow with its inferred definition, and the proof script is also saved as a distinguished node sitting inside the shape.

Generation of a mechanised proof The algorithm that generates a mechanised proof assumes that the diagram is planar[3]. The resulting proof script consists of a list of tactics that states the intermediary lemmas corresponding to the inner shapes of the diagram and assemble them to justify equality for the outer shape, introducing associativity steps when required. Each intermediary lemma is provided with a formal proof that was given in the original diagram, as a distinguished node sitting somewhere inside the shape corresponding to the lemma.

3.3 Integration with LaTeX

The standalone version of the diagram editor provides some support to help editing LaTeX files that include diagrams: it periodically scans the LaTeX file to detect embedded diagrams (either inlined as json data, or saved in an external file) in a LaTeX comment. If that comment is not followed by the corresponding generated LaTeX code, then the editor loads the diagram. When the user saves it, the diagram is stored back into the file, as well as the generated LaTeX code corresponding to the diagram. If the user later wants to edit it again, they can simply delete the generated LaTeX code, and the editor will load it again as explained above. The same editing process is implemented for LyX, a WYSIWYG LaTeX editor for LaTeX.

4 Related work

Chabassier's graphical interface [CB] This software consists of a Coq plugin that interacts with a graphical interface implemented in Rust. The latter can render Coq goals as diagrams, with limited editing features. It provides a basic tactic language to make progress on the proof. It can also suggest a list of relevant lemmas that can be applied to the goal, by querying the Coq runtime.

quiver This diagram editor [Ark23] is implemented in JavaScript and runs in the browser. Compared to our software, the styling possibilities are richer, but it misses some helpful features[4] that our software supports, such as find & replace, copy & paste, or selection extension to connected components. Contrary to our editor, the grid is not optional: vertices cannot be created out of it. Finally, it does not offer any feature to help mechanisation.

[2] See `https://github.com/amblafont/vscode-yade-example` for an example.
[3] This constraint induces a canonical choice of the primitive "inner" shapes.
[4] The following examples are mentioned as feature requests on the github repository.

References

[Ark23] Nathanael Arkor : quiver. `https://q.uiver.app/`, novembre 2023.

[CB] Luc Chabassier et Bruno Barras : A graphical interface for diagrammatic proofs in proof assistants. Contributed talks in the 29th International Conference on Types for Proofs and Programs (TYPES 2023).

[CJGAI] Ali Caglayan, Emilio J. Gallego Arias et Shachar Itzhaky : Coq LSP. `https://github.com/ejgallego/coq-lsp`.

[Laf] Ambroise Lafont : A commutative diagram editor. `https://amblafont.github.io/graph-editor/index.html`.

[VAG+] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson *et al.* : Unimath — a computer-checked library of univalent mathematics. available at `http://unimath.org`.