Chyp: Composing Hypergraphs, Proving Theorems

Aleks Kissinger

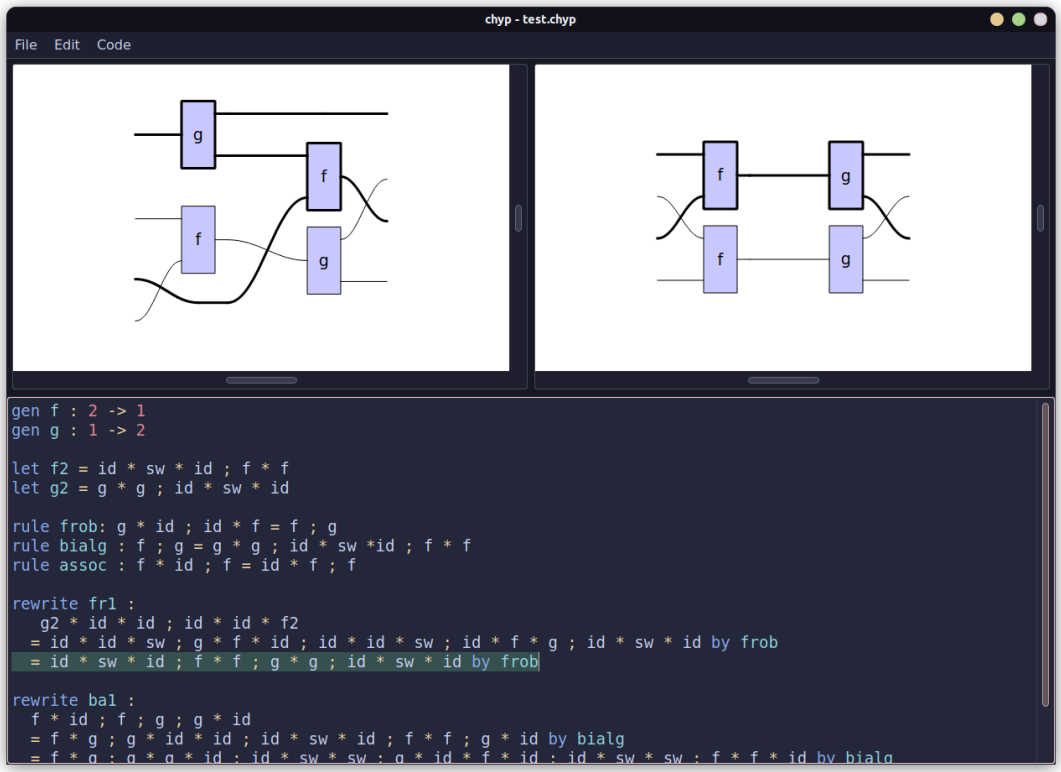
May 2, 2023

Tool presentation: https://github.com/akissinger/chyp

Chyp is a new interactive theorem prover for symmetric monoidal categories (SMCs). SMCs are a handy, generic way to reason about collections of processes and compose sequentially and in parallel. There are essentially two approaches one could take to express morphisms in such a category and to start to do formal, equational reasoning.

1. **The term approach**: inductively construct terms using basic generators, identities, and swaps via operations of sequential $(;)$ and parallel $(\otimes)$ composition. Work with equivalence classes of terms modulo the symmetric monoidal category axioms; for example $(f\ ;\ g) \otimes (h\ ;\ k) \sim (f \otimes h)\ ;(g \otimes k)$.
2. **The diagram approach**: work with *string diagrams*, i.e. diagrams whose basic morphisms are represented by boxes with input and output wires, where composition is represented by "plugging" boxes together.

The advantage of the second approach is that all of the SMC axioms are absorbed in the diagrammatic structure. Namely, two terms are equivalent according to the SMC axioms if and only if they generate isomorphic string diagrams. On the other hand, diagrams want to be drawn, not typed, so they can be unwieldy to input into a software tool or use within existing coding or automated theorem proving paradigms. So, for formalising SMCs, is to better to use terms or diagrams? Chyp aims to break the impasse on this decades-old question: use both!



Chyp is a desktop application implemented in Python using Qt6 for Python (a.k.a. PySide6). Nearly all interaction with Chyp happens in the code editor on the bottom half of the screen. In this editor, the user writes a proof document, getting instant feedback as they type. If the cursor is over something that can be pictured as a string diagram, that diagram appears in the top half of the window and it changes as soon as the user makes changes. The tool produces generally very good-looking string diagrams from terms using a new custom graph layout algorithm based on topological sorting and convex optimisation.

This is a source file written in a declarative language that resembles (a small fragment of) the kinds of language used in a traditional interactive theorem prover like Isabelle or Coq. Statements exist for declaring generators, named terms, and equations (a.k.a. rules) that hold between terms with the same numbers of input/output wires. Proofs can be constructed as transitive chains of term equalities, where each step is justified by a named equation arising from a basic rule or an earlier proof, e.g.

```
rewrite lemma1 :
  f * h ; f ; g
  = id * id * h ; id * f ; f ; g by assoc
  = id * id * h ; id * f ; g * g ; id * sw * id ; f * f by bialg
```

This is somewhat reminiscent of the `calc` syntax[1] used by the Lean theorem prover [2]. If the cursor is over a step in a `rewrite` proof, that step turns green if it can be validated by the prover and red if it cannot. In the case of a correct rule application, the LHS and RHS are also highlighted in the diagram above to show exactly where the rule was applied.

Unlike a term-based theorem prover–or other tools based on more generalised notions of string diagrams like DisCoPy[2] [7] or Homotopy.io [1]–Chyp focuses specifically of SMCs, which enables it to use hypergraphs and hypergraph rewriting under the hood. Cospans of hypergraphs, a.k.a. *hypergraphs with boundary*, can be used to faithfully represent morphisms in a free symmetric monoidal category, and equational reasoning can be captured by double-pushout graph rewriting. There is a lot of theory behind this, which is laid out in detail in a three-part series of papers on *String Diagram Rewrite Theory* [4, 5, 6]. **TLDR:** Chyp automatically understands terms up to the SMC axioms, or equivalently up to isomorphism of the associated string diagrams. So, a proof step is validated if and only if it follows from a single application of the given rule, up to the axioms of an SMC. This enables the user to ignore structural axioms completely in the construction of proofs.

Terms in a Chyp proof can get quite large, so it is not always convenient for the user to have to explicitly compute and write the result of each rewrite step. In fact, one of the biggest advantages of tool support is to let a proof assistant let you quickly explore the consequences of rules without necessarily having a full proof in mind beforehand. This kind of automated rewriting is supported in Chyp via an Agda-like [3] syntactic feature called *holes*. For example, we can let Chyp finish `lemma1` from the example above automatically. We start by writing something like this:

```
rewrite lemma1 :
  f * h ; f ; g
  = ? by assoc
```

The "`?`" indicates a hole, which we would like Chyp to fill for us using the rule provided. If we place the cursor over this step, it will turn red. Pressing `CTRL+N` will ask Chyp to find the next way to fill this hole. To do so, it will find the first application for the rule `assoc` in the LHS, apply it, and replace "`?`" with the result. Of course there might be more than one way one could apply a given rule, yielding different results. Pressing `CTRL+N` multiple times will cycle through them. For repeated applications of the same rule, pressing `CTRL+SHIFT+N` will insert a new line "`= ? by RULE`" where `RULE` is the rule used on the current line, and automatically expand the first application.

This tool is a work in progress, but is already usable for defining rules and doing calculations in an SMC. Many more features are planned, such as LaTeX exporting of proofs, rewriting modulo Frobenius structure, automated proofs via tactics, induction/recursion, and a more structured proof language. Currently the codebase is small and relatively simple ($\sim$ 2500 lines of Python code, open-source Apache 2 licensed), so this could be a great starting point for many new projects.

[1] https://leanprover-community.github.io/extras/calc.html

[2] DisCoPy implements several different representations for morphisms in many types of monoidal categories, including one using cospans of hypergraphs. However, it currently does not support hypergraph rewriting.

References

[1] Homotopy.io. https://homotopy.io/.

[2] Lean Theorem Prover. https://leanprover.github.io/.

[3] The Agda Wiki. https://wiki.portal.chalmers.se/agda/.

[4] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory I: rewriting with Frobenius structure. *Journal of the ACM (JACM)*, 69(2):1–58, 2022.

[5] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory II: Rewriting with symmetric monoidal structure. *Mathematical Structures in Computer Science*, 32(4):511–541, 2022.

[6] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. String diagram rewrite theory III: Confluence with and without Frobenius. *Mathematical Structures in Computer Science*, pages 1–41, 2022.

[7] Giovanni de Felice, Alexis Toumi, and Bob Coecke. Discopy: Monoidal categories in python. *arXiv preprint arXiv:2005.02975*, 2020.