Gatlab.jl: Symbolic computing with categories using generalized algebraic theories

Owen Lynch
Topos Institute
owen@topos.institute

Kris Brown
Topos Institute
kris@topos.institute

James Fairbanks
University of Florida
fairbanksj@ufl.edu

Evan Patterson
Topos Institute
evan@epatters.org

The framework Catlab.jl seeks to expose the fundamental abstractions of category theory through efficient, reusable software implementations in the Julia programming language. These abstractions can then power domain-specific modeling frameworks and user-facing applications.

Computing with categories can take many forms. In recent years, the Catlab development team has focused on *combinatorial computing*, creating an efficient, in-memory implementation of categorical databases [10, 9], and on *numerical computing*, realizing quantitative functorial semantics through interfaces with numerical solvers [7, 2]. *Symbolic computing* with categories in the style of computer algebra systems, though it has always been part of our vision, has so far been less fully developed. In this demo, we showcase an early but promising effort to integrate symbolic computing into the core of Catlab. We are developing this functionality in a new package called Gatlab.jl.

Gatlab is inspired by Cartmell's generalized algebraic theories (GATs), a logical system combining algebraic theories with dependent types [4, 5]. Unlike ordinary algebraic theories, GATs are expressive enough to formulate the theory of categories, as well as most theories of categories with extra structure, such as monoidal categories and categories with chosen products or limits. In contrast to the equally expressive logical systems of essentially algebraic theories [6], [1, §3.D] and finite limit sketches [3], GATs possess an intuitive syntax in which category-theoretic definitions resemble their textbook forms. As a simple example, the reader will recognize the definition of a category in the GAT below.

```
@theory ThCategory begin
  # Type constructors: objects and morphisms.
  Ob::TYPE
  Hom(dom, codom)::TYPE ⊣ [dom::Ob, codom::Ob]
  # Term constructors: composition and identities.
  (f ⋅ g)::Hom(a,c) ⊣ [(a,b,c)::Ob, f::Hom(a,b), g::Hom(b,c)]
  id(a)::Hom(a,a) ⊣ [a::Ob]
  # Axioms: associativity and unitality.
  assoc := ((f ⋅ g) ⋅ h) == (f ⋅ (g ⋅ h)) ⊣ [(a,b,c,d)::Ob, f::Hom(a,b), g::Hom(b,c), h::Hom(c,d)]
  idl := id(a) ⋅ f == f ⊣ [(a,b)::Ob, f::Hom(a,b)]
  idr := f ⋅ id(b) == f ⊣ [(a,b)::Ob, f::Hom(a,b)]
end
```

In fact, GATs have been implemented in Catlab since nearly the project's inception, where they have played two major roles: as an organizing principle and as a symbolic syntax. In the first role, the categorical structures abounding in applied category theory are systematically recorded as GATs. Specific Julia types and functions are then declared to be *instances* of GATs. For example, nonnegative integers and numerical matrices are an instance of the theory of biproduct categories. In addition, given a GAT, Catlab mechanically generates Julia types for expression trees in the GAT, yielding useful symbolic data structures for objects and morphisms in categories. Both roles can be seen as defining *models* of GATs: instances are models based arbitrary Julia code and symbolic expressions are free or initial models.

Gatlab is a complete rewrite of the GAT system in Catlab, with a new design intended to improve the system's capabilities in both its roles. Gatlab gives a central place to the **category of GATs**, whose objects are GATs and morphisms are (equivalence classes of) interpretations [5, §12]. An **interpretation** of one GAT T in another T' sends type and term constructors in T to type and term *expressions* in T', such that all axioms in T are derivable from axioms in T'. To simplify computations in the category of GATs, Gatlab separates the combinatorial structure of GATs and GAT expressions from their human-readable names, encoding the former using de Bruijn levels and maintaining the latter as auxiliary data. With this design, complex theories can be built up from simpler ones using category-theoretic operations. For example, the theory of symmetric rig categories is the pushout of two copies of the theory of symmetric monoidal categories, augmented by distributivity isomorphisms and coherence axioms.

A more general notion of morphism of GATs is based on contexts and realizations in a GAT [5, §13]. A **morphism of GATs** $T \to T'$ sends each type constructor in T to a context in T' and each term constructor in T to a realization in T', compatibly with the axioms. Morphisms of GATs act contravariantly on models: given a GAT morphism $F : T \to T'$, any model M of the theory T' pulls back to a model $F^*(M)$ of the theory T. A great many categorical constructions arise in this way. To give just one example, for any cartesian monoidal category C, there is a monoidal category $\mathrm{Lens}(\mathsf{C})$ whose objects are pairs of objects in C and whose morphisms are lenses in C. Using Gatlab, this construction can be defined by a GAT morphism from the theory of monoidal categories to the theory of cartesian categories:

```
Lens = @theory_map ThMonoidalCat ThCartesianCat begin
  Ob => [pos::Ob, dir::Ob]
  Hom(A,B) => [expose::Hom(A.pos, B.pos), update::Hom(A.pos × B.dir, A.dir)]
  f ⋅ g => begin
    let A, F = f.dom.pos, g.codom.dir
    expose = f.expose ⋅ g.expose
    update = (copy(A) × id(F)) ⋅ (id(A) × f.expose × id(F)) ⋅ (id(A) × g.update) ⋅ f.update
  end
  f ⊗ g => begin
    let A1, A2, D2 = f.dom.pos, g.dom.pos, g.codom.dir
    expose = f.expose × g.expose
    update = (id(A1) × swap(D1, A2) × id(D2)) ⋅ (f.update × g.update)
  end
  [...]
end
```

Using metaprogramming, we can then perform *correct-by-construction* transfer of Julia instances from one theory to another. In this example, any valid instance of a cartesian category automatically gives a valid monoidal category of lenses.

Turning to the second role (symbolic reasoning), Gatlab will provide automated equational reasoning in GATs by integrating GAT expressions with e-graphs. *E-graphs* extend the classic union-find data structure from equivalence relations on elements of a set to congruence relations on expression trees, efficiently representing many equalities between expressions within a single data structure [8, 11]. As a first example, applying the equality saturation procedure to a finite presentation of a category will enumerate (possibly without termination) the distinct morphisms in the category. We will also use e-graphs to efficiently implement C-sets presented as colimits of representable, complementing the combinatorial implementation of categorical databases with a symbolic one.

Combinatorial, symbolic, and numerical computing are traditionally the purview of distinct software systems, even distinct programming languages. Catlab aims to unify them within a single framework, designed along category-theoretic principles to compute with category-theoretic structures. The work-in-progress demonstrated in Gatlab takes another step toward realizing that vision.

References

[1] Jiří Adámek & Jiří Rosický (1994): *Locally presentable and accessible categories*. Cambridge University Press. doi: 10.1017/CBO9780511600579.

[2] John Baez, Xiaoyan Li, Sophie Libkind, Nathaniel Osgood & Evan Patterson (2022): *Compositional modeling with stock and flow diagrams*. In *Proceedings of the 2022 Applied Category Theory Conference*, arXiv: 2205.08373.

[3] Michael Barr & Charles Wells (1999): *Category theory for computing science*, 3rd edition. Les Publications CRM. Republished in *Reprints in Theory and Applications of Categories*, No. 22 (2012) pp. 1–538. http://www.tac.mta.ca/tac/reprints/articles/22/tr22abs.html.

[4] John Cartmell (1978): *Generalised algebraic theories and contextual categories*, PhD thesis. Oxford University.

[5] John Cartmell (1986): *Generalised algebraic theories and contextual categories*. *Annals of Pure and Applied Logic*. 32, pages 209–243, doi: 10.1016/0168-0072(86)90053-9.

[6] Peter Freyd (1972): *Aspects of topoi*. *Bulletin of the Australian Mathematical Society*. 7(1), pages 1–76, doi: 10.1017/S0004972700044828.

[7] Sophie Libkind, Andrew Baas, Evan Patterson & James Fairbanks (2021): *Operadic modeling of dynamical systems: Mathematics and computation*. In *Proceedings of the 2021 Applied Category Theory Conference*, arXiv: 2105.12282.

[8] Greg Nelson & Derek C. Oppen (1980): *Fast decision procedures based on congruence closure*. *Journal of the ACM*. 27(2), pages 356–364, doi: 10.1145/322186.322198.

[9] Evan Patterson, Owen Lynch & James Fairbanks (2022): *Categorical data structures for technical computing*. *Compositionality*. 4(5), doi: 10.32408/compositionality-4-5, arXiv: 2106.04703.

[10] Patrick Schultz, David I. Spivak, Christina Vasilakopoulou & Ryan Wisnesky (2017): *Algebraic databases*. *Theory and Applications of Categories*. 32(16), pages 547–619, arXiv: 1602.03501. Available at http://www.tac.mta.ca/tac/volumes/32/16/32-16abs.html.

[11] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock & Pavel Panchekha (2021): *egg: Fast and extensible equality saturation*. *Proceedings of the ACM on Programming Languages*. 5(POPL), pages 1–29, doi: 10.1145/3434304.