Generating morphism types using parametricity and Trocq

Cyril Cohen[1], Assia Mahboubi[2,3], and Vojtěch Štěpančík[2]

[1] Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668 Lyon, France
[2] Nantes Université, École Centrale Nantes, CNRS, Inria, LS2N, UMR 6004 Nantes, France
[3] Vrije Universiteit Amsterdam, The Netherlands

Background Parametricity as described by Reynolds [7] is a property of a type system expressing uniformity of definitions of polymorphic functions. It allows one to reason metatheoretically about their behavior, using relational models, to obtain "theorems for free" [9]. Its extension to dependent type theory [1], where relations are first-class functions into universes, make possible to implement parametricity as a metaprogram, translating terms to terms [6]. Nowadays, this translation is used in practical automation frameworks for e.g. reification [4], data refinement [3] and proof transfer [8, 2].

In this work-in-progress we focus on data generation, rather than theorem generation. We investigate the question: "How can my proof assistant generate the morphisms of my structures?" and later "How can my proof assistant generate the category of my structures?" This question is answered for algebraic structures by rudimentary universal algebra, but we aim to support a more expressive language for defining structures. In particular we treat structures which contain new sorts, e.g. partially ordered sets, and higher-order structures such as topological spaces. We additionally allow the user to provide annotations à la Trocq [2], since there is often not a unique appropriate type of morphisms for a structure and the choice should be surfaced to the user.

Translation The technique we present is based upon binary parametricity translations with Trocq annotations [2]. Given a structure, represented as a Rocq `Record` type `Str` with a single type parameter `A`, compute its parametricity translation, which in turn is a new record type $\texttt{Str}_R$ with type parameters `A` and `A'`, a type-valued relation $\texttt{A}_R$, and structure instances `S : Str A`, `S' : Str A'`. Then reparameterize the translation by replacing $\texttt{A}_R$ with a function argument `f : A → A'`, and substitute the body of the record using $\texttt{A}_R$ `a a' := (f a = a')`. The Trocq annotations play the role of guiding the translation of sorts; we focus on the functional, cofunctional and equivalence cases, that is $(4, 0)$, $(0, 4)$ and $(4, 4)$, respectively. A sort annotated with α translates to $\boxdot^{\alpha}$ (or $\boxdot^{\alpha}_P$ if its propositional), which represents a relation on types (propositions) equipped with data of level α. For example, $\boxdot^{4,0}_P$ `P P'` is a relation on `P` and `P'` equipped with a map `P → P'` and a proof that the relation is equivalent to its graph.

In a univalent metatheory, and assuming all types used as structure parameters are homotopy sets, we can reduce the translated records further. In particular, it is possible to compute the following simplifications, which reduce the worked examples to definitions a user would naturally write by hand. *Contractibility of singletons*: the translation of a variable to $\texttt{a}_R$ `: f a = a'` allows us to remove the arguments `a'` and $\texttt{a}_R$, replacing their occurences with `f a` and `refl`, respectively. By univalence, we can apply a similar contraction of types related by a $\boxdot^{4,4}$ relation, and furthermore replace (co)functional relations by graphs of their underlying maps. *Removing equations in `A`*: the translation of the identity type $\texttt{a} =_{\texttt{A}} \texttt{b}$ is equivalent to a certain type of identifications in `f b = b'`, which has contractible identity types by the assumption of `A` being a homotopy set. *Removing fields eliminating into appropriately annotated propositional sorts*: when a sort is declared as (co)functional, the relation it generates on

its witnesses corresponds to a graph of an implication. Since the sort is propositional, we have that its identity types are contractible.

We give three examples of structures for which the framework can generate the desired type of morphisms. The definition of the structure type and its parametricity translation are presented side-by-side, with the reduced type of morphisms below.

Monoids consist of two operations and three equations. We are allowed to drop equations and contract singletons, which gives us the type of monoid homomorphisms $\texttt{Monoid}_H$.

```
Record Monoid (A : □) := {                Record Monoid_R (A A' A_R)
                                             (M : Monoid A) (M' : Monoid A') := {
  e : A;                                    e_R : A_R M.e M'.e;
  m : A → A → A;                            m_R : ∀ (x x' x_R : A_R x x')
                                                    (y y' y_R : A_R y y'),
                                                  A_R (M.m x y) (M'.m x' y');
  (* ...and laws *) }.                      (* ...and translations of laws *) }.
Record Monoid_H (A A' f) (M : Monoid A) (M' : Monoid A') := {
  e_H : f M.e = M'.e;
  m_H : ∀ x y, f (M.m x y) = M'.m (f x) (f y); }.
```

Partially ordered types bundle a relation on its elements. Annotating the relation with $(4,0)$ generates order preserving maps, while $(0,4)$ generates order reflecting maps. The reflexivity and transitivity laws eliminate into the relation, and antisymmetry eliminates into equality on `A`, so we may drop them.

```
Record Order (A : □) := {                 Record Order_R (A A' A_R)
                                             (O : Order A) (O' : Order A') := {
  le : A → A → Prop^{4,0};                  le_R : ∀ (x x' x_R : A_R x x')
                                                    (y y' y_R : A_R y y'),
                                                  ⊡_P^{4,0} (O.lt x y) (O'.lt x' y');
  (* ...and laws *) }.                      (* ...and translations of laws *) }.
Record Order_H (A A' f) (O : Order A) (O' : Order A') := {
  le_H : ∀ x y, O.le x y → O'.le (f x) (f y) }.
```

Topological spaces are not algebraic structures. By annotating the inner `Prop` with $(4,4)$ we enable contracting away `P` to `P ∘ f`, and annotating the outer `Prop` with $(0,4)$ gives openness reflecting maps, i.e. continuous maps. Changing the annotation to $(4,0)$ gives open maps instead. All laws eliminate into witnesses of openness, so they may be removed.

```
Record Space (A : □) := {                 Record Space_R (A A' A_R)
                                             (S : Space A) (S' : Space A') := {
  open : (A → Prop^{4,4}) → Prop^{0,4};     open_R : ∀ (P P' P_R : ∀ a a' a_R,
                                                        ⊡_P^{4,4} (P a) (P' a')),
                                                  ⊡_P^{0,4} (open P) (open' P');
  (* ...and laws *) }.                      (* ...and translations of laws *) }.
Record Space_H (A A' f) (S : Space A) (S' : Space A') := {
  open_H : ∀ P', S'.open P' → S.open (P ∘ f); }.
```

Related and future work The case of generating morphisms of generalized algebraic theories (GATs) has been written down by Kaposi et al [5]; however they only consider morphisms covariant in all sorts. With Trocq annotations, we allow the user to specify per-sort variances.

The end goal of this work is to implement a metaprogram which generates the morphism types as sketched above, and a proof that the resulting data forms a category. This is however not expected to hold for all instances of user-provided annotations.

References

[1] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and dependent types. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP '10, page 345–356, New York, NY, USA, 2010. Association for Computing Machinery.

[2] Cyril Cohen, Enzo Crance, and Assia Mahboubi. Trocq: Proof transfer for free, beyond equivalence and univalence. *ACM Trans. Program. Lang. Syst.*, 47(3), September 2025.

[3] Cyril Cohen, Maxime Dénès, and Anders Mörtberg. Refinements for Free! In *Certified Programs and Proofs*, pages 147 – 162, Melbourne, Australia, December 2013.

[4] Jason Gross, Andres Erbsen, and Adam Chlipala. Reification by parametricity - fast setup for proof by reflection, in two lines of ltac. In Jeremy Avigad and Assia Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 289–305. Springer, 2018.

[5] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, 2019.

[6] Chantal Keller and Marc Lasson. Parametricity in an impredicative sort. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 381–395. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[7] John C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19-23, 1983*, pages 513–523. North-Holland/IFIP, 1983.

[8] Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. Equivalences for free: univalent parametricity for effective transport. *Proc. ACM Program. Lang.*, 2(ICFP), jul 2018.

[9] Philip Wadler. Theorems for free! In Joseph E. Stoy, editor, *Proceedings of the fourth international conference on Functional programming languages and computer architecture, FPCA 1989, London, UK, September 11-13, 1989*, pages 347–359. ACM, 1989.