# Implementing parametricity in Rocq-Elpi

Cyril Cohen
cyril.cohen@inria.fr
Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668
Lyon, France

Vojtěch Štěpančík
vojtech.stepancik@inria.fr
Nantes Université, École Centrale Nantes, CNRS, Inria, LS2N, UMR 6004
Nantes, France

## 1 Introduction

A type theory may posses the parametricity property, expressing well-behavedness of polymorphic functions, stemming from a relational interpretation [1, 6, 7]. The property has direct applications to deriving theorems about programs [1, 9] and proof transfer [3, 8].

In dependent type theory, the relational interpretation may be implemented as a meta-program, translating types to relations and terms to proofs of relatedness. This translation is usually presented in a functional style, split between translation of contexts and translation of terms. This approach has the disadvantage that a proper implementation of the term translation needs to handle freshness of variables and thus have access to the context, which is not obvious from the presentation.

In order to give a description more faithful to the implementation we use a sequent calculus, as presented for binary parametricity in Section 2, and in Section 3 we illustrate how it translates to a logic program integrated with Rocq using the Rocq-Elpi library. The translation of fixpoint from the literature [6] is incomplete with regard to Rocq reduction rules, and in Section 4 we describe how to fix it. Furthermore, up to our knowledge there is no version of the parametricity translation treating record types as first-class citizens, which we attempt to rectify by dualizing the inductive-style translation [2] in Section 5. Finally in Section 6 we describe several works depending on or inspired by this contribution.

## 2 A sequent calculus for parametricity

Parametricity translations are usually presented in a functional style: types translate to relations, and terms to their proofs. In several translation cases, such as in the lambda case

$$[\![\ \lambda x : A.\, t\ ]\!] = \lambda (x : A)(x' : A')(x_R : [\![\ A\ ]\!]\ x\ x').\, [\![\ t\ ]\!]$$

there is an implicit $\cdot'$ translation which essentially duplicates variables. To correctly handle freshness of $\cdot'$ and named variables, an implementation needs to use de Bruijn indices or pass around an environment, or both in a locally nameless style, as is the case in the implementation of the ParamCoq plugin [6].

The program $[\![\ \cdot\ ]\!]$ and the implicitly defined $\cdot'$ can be rephrased in the sequent-style presentation from Fig. 1. The latter does not define context translation separately but integrates it to the cases of $\lambda$ and $\Pi$, where freshness of variables can be computed. One may read the judgment operationally: in $t\ \sim\ t'\ \because\ t_R$ the parameter $t$ is the input while $t'$ and $t_R$ are outputs.

In addition to solving the problems from the functional presentation, this presentation is implementation ready in Rocq-Elpi.

## 3 Rocq-Elpi implementation

Rocq-Elpi is a compatibility layer between Rocq and Elpi, the embeddable $\lambda$Prolog interpreter. It provides Rocq with mechanisms to implement commands and tactics in Elpi, using an encoding of

$$\frac{}{\Xi \vdash \square_i\ \sim\ \square_i\ \because\ \lambda (A\, B : \square_i).\, A \to B \to \square_i}$$

$$\frac{(x, x', x_R) \in \Xi \qquad \Xi \vdash}{\Xi \vdash x\ \sim\ x'\ \because\ x_R} \qquad \frac{\Xi \vdash t\ \sim\ t'\ \because\ t_R \quad \Xi \vdash u\ \sim\ u'\ \because\ u_R}{\Xi \vdash t\ u\ \sim\ t'\ u'\ \because\ t_R\ u\ u'\ u_R}$$

$$\frac{\Xi \vdash A\ \sim\ A'\ \because\ A_R \qquad \Xi, x\ \sim\ x'\ \because\ x_R \vdash t\ \sim\ t'\ \because\ t_R}{\Xi \vdash \lambda x : A.\, t\ \sim\ \lambda x' : A'.\, t'\ \because\ \lambda x\, x'\, x_R.\, t_R} \quad x, x' \notin \mathrm{Var}(\Xi)$$

$$\frac{\Xi \vdash A\ \sim\ A'\ \because\ A_R \qquad \Xi, x\ \sim\ x'\ \because\ x_R \vdash B\ \sim\ B'\ \because\ B_R}{\Xi \vdash \Pi x : A.\, B\ \sim\ \Pi x' : A'.\, B'\ \because\ \lambda f\, g.\, \Pi x\, x'\, x_R.\, B_R\ (f\ x)\ (g\ x')} \quad x, x' \notin \mathrm{Var}(\Xi)$$

**Figure 1.** Sequent-style binary parametricity translation for $CC_\omega$.

Rocq terms and commands in higher-order abstract syntax, suitable for logic programming.

The heart of the translation is the ternary predicate param/3 on terms, where (param $t\ t'\ t_R$) represents the judgment $t\ \sim\ t'\ \because\ t_R$. The parametricity context $\Xi$ is replaced by a combination of Elpi variables and the logical context of Elpi, which supports querying under assumption that param/3 holds for a triple of terms. This enables us to remove freshness side-conditions and the ambient context from the inference rules.

After the above adjustments, the implementation stays extremely faithful to the form of the inference rules: each inference rule has a corresponding clause of the param/3 predicate. The clause head reflects the conclusion of the inference rule, e.g.

```
param(λ X : A.T)(λ X' : A'.T')
    (λ X : A.x\ λ X' : A'.x'\ λ X_R : (A_R x x').x_R\ T_R x x' x_R) :- ...
```

The capital symbols $X$, $A$, $T$ and their suffixed forms are Elpi variables introduced by the clause head. The lowercase symbols $x$, $x'$ and $x_R$ are Elpi variables introduced by the \ binder[1]. The symbols $\lambda$, : and . together form a shorthand for the constructor fun of the term type in Rocq-Elpi and separating its three arguments.

The body of the clause queries the context for the rule conditions, generates variable names, and mediates between application of Rocq terms and application of Elpi functions. For lambda terms, we first query (param $A\ A'\ A_R$) in the current context. Then we universally quantify over fresh Elpi variables $x, x'$ and $x_R$, and under assumption (param $x\ x'\ x_R$) query (param $(T\ x)(T'\ x')(T_R\ x\ x'\ x_R)$).

[1] In Elpi the name of the variable comes *before* the backslash.

The interactive entrypoint of the library is the dispatch/3 predicate. It expects a global reference representing the name of a top-level declaration, generates the corresponding parametricity translation using param/3, adds it to the Rocq environment, and adds a new clause to param/3. It currently supports translating constants, the inductive-style translation of inductive types, and translating record types as explained in Section 5.

## 4 The fixpoint trick

The translation of $\mathrm{fix}^n f.u$, which defines a fixpoint recursively called $f$ and recursive on the $n^{\text{th}}$ variable (counting from 0) of its body $u$ ought to be as follows

$$\frac{f, f' \notin \mathrm{Var}(\Xi) \qquad \Xi, f \sim f' \because f_R \vdash U \sim U' \because U_R}{\Xi \vdash \mathrm{fix}^n f.U \sim \mathrm{fix}^n f'.U' \because F_R}$$

$$\text{with } F_R := \text{let } f := \mathrm{fix}^n f.u \text{ in let } f' := \mathrm{fix}^n f'.u' \text{ in } \mathrm{fix}^{3n} f_R.U_R$$

**Figure 2.** Naïve parametricity translation of fix expressions.

However the fixpoint equation (e.g. $(\mathrm{fix}^1 f.U)x \equiv U\{f/\mathrm{fix}^1 f.U\}x$ in simple cases) does not hold in general in Rocq, but only when $x$'s head is a constructor. This restriction breaks the abstraction theorem: $F_R$ does not necessarily have type $[\![ \mathrm{ty}(f) ]\!]\ f\ f$. This breakage occurs in both toy examples such as the dummy `Fixpoint f n := 0`, and real examples such as subtraction of natural numbers.

Our solution consists of systematically inserting a match expansion at the top of the fixpoint, thus ensuring that the fixpoints $f$ and $f'$ occurring in recursive calls of $F_R$ are always applied to constructors and thus always expand using their fixpoint equation.

## 5 Record translation

In existing treatments of parametricity translations, records are not considered, implicitly elaborated to inductive types with one constructor, or reduced to iterated inductive $\Sigma$ types. The issue with the inductive-style approach is that the result of translating an inductive type is an *indexed* inductive type. As such, the translation cannot be readily represented by a record type. A bespoke treatment of record types is thus necessary if we want the translation to be a record type again, potentially with a definitional eta rule.

We approach the translation dually to the inductive one. Recall that the translation of inductive types ensures that the generated constructors are well-typed as translations of their corresponding original constructors. Contrastingly, the translation of the eliminator needs to be defined in terms of the generated eliminator, since that by itself doesn't have the correct type. For record types, we define a translation which constructs a record whose *projections* are well-typed as translations of the original projections, but whose *constructor* needs to be manipulated to give a translation of the original constructor.

Note that in the case of inductive types in Rocq, we are able to treat eliminators of inductive types uniformly, since those are themselves defined in terms of the syntactic primitive `match`, and we give rules to translate generic `match` expressions. However, constructors of record types are *not* defined in terms of the record literal syntax `{| p := _ |}`. Instead, record literals are syntactic sugar for calling the constructors, and they are elaborated before they are passed to Rocq-Elpi. If the record literal syntax was the primitive for constructing record types, than we suspect it would be possible to treat record construction uniformly as well.

## 6 Applications

***Direct applications.*** The parametricity plugin described here has successfully replaced the original parametricity plugin inside the CoqEAL library [4] and inside a modular proof of correctness of mergesort based on parametricity [5].

***Trocq.*** The proof transfer framework Trocq [3] is based on an enhanced version of parametricity translation. Because of the added complexity of annotations it is implemented using a 4-ary predicate param/4 and does not plug into the presented implementation. The principle is the same, however — the translation is presented in sequent style, which is then expressed as an Elpi predicate.

## Acknowledgments

We thank Marc Lasson for his explanations of the internals of the original parametricity plugin and Enrico Tassi for his support while developing the Rocq-Elpi plugin. We also thank Assia Mahboubi for her inputs on the work and present submission. The present work was partly funded by ERC grant FRESCO 10100199.

## References

[1] Jean-Philippe Bernardy and Marc Lasson. 2011. Realizability and Parametricity in Pure Type Systems. In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6604)*, Martin Hofmann (Ed.). Springer, 108–122. doi:10.1007/978-3-642-19805-2_8

[2] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. 2010. Parametricity and dependent types. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming* (Baltimore, Maryland, USA) *(ICFP '10)*. Association for Computing Machinery, New York, NY, USA, 345–356. doi:10.1145/1863543.1863592

[3] Cyril Cohen, Enzo Crance, and Assia Mahboubi. 2024. Trocq: Proof Transfer for Free, With or Without Univalence. In *Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 14576)*, Stephanie Weirich (Ed.). Springer, 239–268. doi:10.1007/978-3-031-57262-3_10

[4] Cyril Cohen, Maxime Dénès, and Anders Mörtberg. 2013. Refinements for Free!. In *Certified Programs and Proofs*. Melbourne, Australia, 147 – 162. doi:10.1007/978-3-319-03545-1_10

[5] Cyril Cohen and Kazuhiko Sakaguchi. 2025. A Bargain for Mergesorts: How to Prove Your Mergesort Correct and Stable, Almost for Free. *Proc. ACM Program. Lang.* 9, ICFP, Article 236 (Aug. 2025), 29 pages. doi:10.1145/3747505

[6] Chantal Keller and Marc Lasson. 2012. Parametricity in an Impredicative Sort. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France (LIPIcs, Vol. 16)*, Patrick Cégielski and Arnaud Durand (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 381–395. doi:10.4230/LIPIcs.CSL.2012.381

[7] John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19-23, 1983*, R. E. A. Mason (Ed.). North-Holland/IFIP, 513–523.

[8] Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. 2021. The Marriage of Univalence and Parametricity. *J. ACM* 68, 1, Article 5 (jan 2021), 44 pages. doi:10.1145/3429979

[9] Philip Wadler. 1989. Theorems for Free!. In *Proceedings of the fourth international conference on Functional programming languages and computer architecture, FPCA 1989, London, UK, September 11-13, 1989*, Joseph E. Stoy (Ed.). ACM, 347–359. doi:10.1145/99370.99404