The Refined Calculus of Inductive Construction: Parametricity and Abstraction

Chantal Keller
INRIA Saclay–Île-de-France at École Polytechnique
Email: Chantal.Keller@inria.fr

Marc Lasson
ENS Lyon, Université de Lyon, LIP
UMR 5668 CNRS ENS Lyon UCBL INRIA
Email: marc.lasson@ens-lyon.org

***Abstract*—We present a refinement of the Calculus of Inductive Constructions in which one can easily define a notion of relational parametricity. It provides a new way to automate proofs in an interactive theorem prover like Coq.**

I. Introduction

The Calculus of Inductive Constructions (CIC in short) extends the Calculus of Constructions with inductively defined types. It is the underlying formal language of the Coq interactive theorem prover [1].

In the original presentation, CIC had three kinds of sorts: the impredicative sort of propositions $\texttt{Prop}$, the impredicative sort of basic informative types $\texttt{Set}$, and the hierarchy of universes $\texttt{Type}_0$, $\texttt{Type}_1$, ... This presentation was not compatible with the possibility to add axioms in the system, since it could lead to inconsistencies [2]. Nowadays, there is no impredicative sort of basic informative types, and $\texttt{Set}$ represents $\texttt{Type}_0$.

This does not fit well with one of the major original ideas about CIC: the possibility to perform program extraction. Indeed, since the current version of CIC does not separate informative types from non-informative types, extraction needs to normalize its type to guess whether it should be erased or not, and this makes it very uneasy to prove correct [3].

In this paper, we propose a refinement of CIC which reconciles extraction with the possibility to add axioms to the system: $\mathsf{CIC}_{\mathsf{ref}}$, the Refined Calculus of Inductive Constructions. The idea is to split the $(\texttt{Type}_i)_{i\in\mathbb{N}}$ hierarchy into two hierarchies $(\texttt{Set}_i)_{i\in\mathbb{N}}$ and $(\texttt{Type}_i)_{i\in\mathbb{N}^*}$, one for informative types and one for types without computational content.

This calculus allows us to extend the presentation of parametricity for Pure Types Systems introduced by Bernardy *et al.* [4] to the Calculus of Inductive Constructions. Parametricity is a concept introduced by Reynolds [5] to study the type abstraction of system F, and the *abstraction theorem* expresses the fact that polymorphic programs map related arguments to related results. In $\mathsf{CIC}_{\mathsf{ref}}$, we can define a notion of relational parametricity in which the relations' codomains is the $\texttt{Prop}$ sort of propositions.

II. $\mathsf{CIC}_{\mathsf{REF}}$: The Refined Calculus of Inductive Constructions

The Refined Calculus of Inductive Constructions is a refinement of CIC where terms are generated by the same grammar as CIC:

$$\begin{array}{c} A, B, P, Q, F \ :=\ x \ \mid\ s \ \mid\ \forall x : A.B \ \mid\ \lambda x : A.B \\ \mid\ (A\,B) \ \mid\ I \ \mid\ \texttt{case}_I(A, \overrightarrow{Q}, P, \overrightarrow{F}) \ \mid\ c \ \mid\ \texttt{fix}\,(x : A).B \end{array}$$

where s ranges over the set $\{\texttt{Prop}\} \cup \{\texttt{Set}_i, \texttt{Type}_{i+1} \,|\, i \in \mathbb{N}\}$ of *sorts* and x ranges over the set of *variables*. We write $\texttt{Ind}^p(I : A, \overrightarrow{c : C}^k)$ to state that I is a well-formed inductive definition typed with p parameters, of arity A, with k constructors $c_1, \dots, c_k$ of respective types $C_1, \dots, C_k$.

A context Γ is a list of pairs $x : A$ and the typing rules are the rules of CIC (one can refer to [1] for the complete set of rules), except to type sorts and dependent products. As for CIC, typing fixpoints (for $\texttt{fix}$) and elimination rules (for $\texttt{case}$) is subject to restrictions to ensure coherence. We present only the rules which are specific to our type system. Here are the three typing rules to type sorts:

$$\frac{}{\vdash \texttt{Prop} : \texttt{Type}_1} \qquad \frac{}{\vdash \texttt{Set}_i : \texttt{Type}_{i+1}} \qquad \frac{}{\vdash \texttt{Type}_i : \texttt{Type}_{i+1}}$$

The following three typing rules tell which products are authorized in the system. The level of the product is the maximum level of the domain and the codomain:

$$\frac{\Gamma \vdash A : r_i \qquad \Gamma, x : A \vdash B : s_j}{\Gamma \vdash \forall x : A.B : s_{\max(i,j)}}\ (r, s) \in \{\texttt{Type}, \texttt{Set}\}$$

Quantifying over propositions does not rise the level of the product:

$$\frac{\Gamma \vdash A : \texttt{Prop} \qquad \Gamma, h : A \vdash B : s_i}{\Gamma \vdash \forall h : A.B : s_i}\ s \in \{\texttt{Type}, \texttt{Set}\}$$

And the sort $\texttt{Prop}$ is impredicative, it means that products in $\texttt{Prop}$ may be built by quantifying over objects whose types inhabit any sort:

$$\frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : \texttt{Prop}}{\Gamma \vdash \forall x : A.B : \texttt{Prop}}\ s \in \{\texttt{Type}, \texttt{Set}, \texttt{Prop}\}$$

Finally, as in CIC, the system comes with subtyping rules based on the following inclusion of sorts (where $i < j$):

$$\texttt{Prop} <: \texttt{Set}_1 \qquad \texttt{Set}_i <: \texttt{Set}_j \qquad \texttt{Type}_i <: \texttt{Type}_j$$

One should note that $\mathsf{CIC}_{\mathsf{ref}}$ easily embeds into CIC by mapping any $\texttt{Set}_i$ and $\texttt{Type}_i$ onto the $\texttt{Type}_i$ of CIC. The coherence of CIC thus implies the coherence of $\mathsf{CIC}_{\mathsf{ref}}$.

III. Parametricity

We can define a notion of relational parametricity for $\mathsf{CIC}_{\mathsf{ref}}$.

$$\Theta_I(\overrightarrow{Q}^p, T, \overrightarrow{F}^n) = \lambda\overrightarrow{(x:A)(x':A')(x_R:[\![A]\!]\,x\,x')}^n\,(a : I\,\overrightarrow{Q}^p\,\overrightarrow{x}^n)(a' : I\,\overrightarrow{Q'}^p\,\overrightarrow{x'}^n)(a_R : [\![I]\!]\,\overrightarrow{Q\,Q'\,[\![Q]\!]}^p\,\overrightarrow{x\,x'\,x_R}^n\,a\,a').$$
$$[\![T]\!]\,\overrightarrow{x\,x'\,x_R}^n\,a\,a'\,a_R\,(\texttt{case}_I\,(a, \overrightarrow{Q}^p, T, \overrightarrow{F}^n))\,(\texttt{case}_I\,(a', \overrightarrow{Q'}^p, T', \overrightarrow{F'}^n))$$

Fig. 1. Relation parametricity for inductive types

Definition 1 (Parametricity relation)**.** *For any inductive* $\texttt{Ind}^p(I : A, \overrightarrow{c:C}^k)$, *we define a fresh inductive symbol* $[\![I]\!]$ *and a family* $([\![c_i]\!])_{i=1\ldots k}$ *of fresh constructor names.*

The parametricity translation $[\![\bullet]\!]$ *is defined by induction on the structure of terms and contexts:*

$$\begin{aligned}
[\![\langle\rangle]\!] &= \langle\rangle \\
[\![\Gamma, x:A]\!] &= [\![\Gamma]\!], x:A, x':A', x_R : [\![A]\!]\,x\,x' \\
[\![s]\!] &= \lambda(x:s)(x':s).x \to x' \to \hat{s} \\
[\![x]\!] &= x_R \\
[\![\forall x{:}A.B]\!] &= \lambda(f:\forall x:A.B)(f':\forall x':A'.B'). \\
&\quad \forall(x:A)(x':A')(x_R:[\![A]\!]\,x\,x'). \\
&\quad [\![B]\!]\,(f\,x)\,(f'\,x') \\
[\![\lambda x:A.B]\!] &= \lambda(x:A)(x':A')(x_R:[\![A]\!]\,x\,x').[\![B]\!] \\
[\![(A\,B)]\!] &= ([\![A]\!]\,B\,B'\,[\![B]\!]) \\
[\![\texttt{fix}(x:A).B]\!] &= (\texttt{fix}(x_R:[\![A]\!]\,x\,x').[\![B]\!]) \\
&\quad [\texttt{fix}(x:A).B/x][\texttt{fix}(x':A').B'/x'] \\
[\![\texttt{case}_I(M, \overrightarrow{Q}^p, T, \overrightarrow{F}^n)]\!] &= \texttt{case}_{[\![I]\!]}([\![M]\!], \overrightarrow{Q, Q', [\![Q]\!]}^p, \\
&\quad \Theta_I(\overrightarrow{Q}^p, T, \overrightarrow{F}^n), \overrightarrow{[\![F]\!]}^n)
\end{aligned}$$

where $\hat{\texttt{Prop}} = \hat{\texttt{Set}}_i = \texttt{Prop}$ *and* $\hat{\texttt{Type}}_i = \texttt{Type}_i$ *and where* A' *denotes the term* A *in which we have replaced each variable* x *by a fresh variable* x'. *The definition of* Θ_I *is in* Fig. 1.

What is new with respect to previous works is the fact that relations over objects of type `Prop` or $\texttt{Set}_i$ have their codomain in `Prop` instead of higher universes. We also formally define parametricity for inductive types.

Unfortunately, in order to prove the abstraction theorem below, we need to restrict the strong elimination: we have to disallow the `case` destructions used to build objects whose types are of sort `Type` when the destructed inductive definition is not *small* (*small inductive definitions* are inductive definitions which constructors only have arguments of type `Prop` or `Set`, see [6]). We write $\vdash_*$ for the derivability where strong elimination is authorized only over small inductive definitions.

Theorem 1 (Abstraction theorem)**.** *If* $\Gamma \vdash_* A : B$ *then* $[\![\Gamma]\!] \vdash_* A : B$, $[\![\Gamma]\!] \vdash_* A' : B'$, *and* $[\![\Gamma]\!] \vdash_* [\![A]\!] : [\![B]\!]\,A\,A'$.

IV. Applications

A lot of so-called "free theorems" are consequences of the abstraction theorem and our framework is expressive enough to implement most examples that can be found in the literature (see for instance [4], [7]).

Here we propose a new example inspired by François Garillot's thesis [8], in which he remarks that polymorphic functions operating on groups can only compose elements using the laws given by the group's structure, and thus cannot create new elements.

In our system, we may actually use parametricity theory to translate this uniformity property. We take an arbitrary group structure $\mathcal{H}$ defined by its carrier $\alpha : \texttt{Set}_0$, a unit element, a composition law, an inverse and the standard axioms stating that $\mathcal{H}$ is a group. We define `fingrp` the type of all the finite subgroups of $\mathcal{H}$ consisting of a list plus stability axioms. Now consider any term $Z : \texttt{fingrp} \to \texttt{fingrp}$ (examples of such terms abound: e.g. the center, the normalizer, the derived subgroup...). The abstraction theorem states that for any $R : \alpha \to \alpha \to \texttt{Prop}$ compatible with the laws of $\mathcal{H}$ and for any $G\,G' : \texttt{fingrp}$, $[\![\texttt{fingrp}]\!]_R\,G\,G' \to [\![\texttt{fingrp}]\!]_R\,(Z\,G)\,(Z\,G')$ where $[\![\texttt{fingrp}]\!]_R$ is the relation on subgroups induced by R. Given this, we can prove the following properties:

- for any G, $Z\,G \subset G$ (if we take $R : x\,y \mapsto x \in G$);
- for any G, for any ϕ a morphism of $\mathcal{H}$, $\phi(Z\,G) = Z\,\phi(G)$ (if we take $R : x\,y \mapsto y = \phi(x)$). It entails that $Z\,G$ is a *characteristic subgroup* of $\mathcal{H}$.

For a complete Coq formalization of this, please refer to the online source code [9].

V. Conclusion

The system presented here allows to distinguish clearly via typing which expressions will be computationally meaningful after extraction. It allows us to define a notion of parametricity for which relations lie in the sort of propositions. We set here the theoretical foundation for an implementation of a Coq tactic that constructs proof terms by parametricity. A first prototype of such a tactic can be found online [9].

References

[1] The Coq Development Team, "The Coq Proof Assistant: Reference Manual," *Rapport technique - INRIA*, 2011.

[2] T. Coquand, "An Analysis of Girard's Paradox," in *LICS*. IEEE Computer Society, 1986, pp. 227–236.

[3] P. Letouzey, "Extraction in Coq: An Overview," in *CiE*, ser. Lecture Notes in Computer Science, A. Beckmann, C. Dimitracopoulos, and B. Löwe, Eds., vol. 5028. Springer, 2008, pp. 359–369.

[4] J.-P. Bernardy, P. Jansson, and R. Paterson, "Parametricity and dependent types," in *ICFP*, P. Hudak and S. Weirich, Eds. ACM, 2010.

[5] J. C. Reynolds, "Types, Abstraction and Parametric Polymorphism," in *IFIP Congress*, 1983, pp. 513–523.

[6] C. Paulin-Mohring, "Inductive definitions in the system coq rules and properties," in *Typed Lambda Calculi and Applications*, ser. Lecture Notes in Computer Science. Springer, 1993, vol. 664, pp. 328–345.

[7] P. Wadler, "Theorems for free!" in *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, ser. FPCA '89. New York, NY, USA: ACM, 1989.

[8] F. Garillot, "Generic Proof Tools and Finite Group Theory," Ph.D. dissertation, École Polytechnique, 2011.

[9] "Preliminary implementation of a Coq tactic," http://www.lix.polytechnique.fr/~keller/Recherche/coqparam.html.