The Refined Calculus of Inductive Construction: Parametricity and Abstraction
Chantal Keller, Marc Lasson

To cite this version:

HAL Id: hal-00757620
https://inria.hal.science/hal-00757620
Submitted on 27 Nov 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 parametericity. 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 Prop, the impredicative sort of basic informative types Set, and the hierarchy of universes Type_0, 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 Set represents 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: CIC_ref, the Refined Calculus of Inductive Constructions. The idea is to split the (Type_i)_{i∈N} hierarchy into two hierarchies (Set_i)_{i∈N} and (Type_i)_{i∈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 CIC_ref, we can define a notion of relational parametricity in which the relations’ codomains is the Prop sort of propositions.

II. CIC_ref: THE RAINED 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:

\[ A, B, P, Q, F := x | s | ∀ x : A . B | λ x : A . B \]

where \( s \) ranges over the set \{Prop\} \cup \{Set_i, Type_{i+1} | i ∈ N\} of sorts and \( x \) ranges over the set of variables. We write \( \text{Ind}^p(I : A, c : C^k) \) to state that \( I \) is a well-formed inductive definition typed with \( p \) parameters, of arity \( A \), with \( k \) constructors \( c_1, \ldots, c_k \) of respective types \( C_1, \ldots, C_k \).

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 \( \text{fix} \)) and elimination rules (for 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:

\[ Γ \vdash \text{Prop} : Type_1 \quad Γ \vdash \text{Set}_i : Type_{i+1} \quad Γ \vdash \text{Type}_i : Type_{i+1} \]

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:

\[ Γ \vdash A : r_i \quad Γ, x : A \vdash B : s_j \]

\[ (r, s) ∈ \{\text{Type}, \text{Set}\} \]

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

\[ Γ \vdash A : \text{Prop} \quad Γ, h : A \vdash B : s_i \quad s ∈ \{\text{Type}, \text{Set}\} \]

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

\[ Γ \vdash A : s \quad Γ, x : A \vdash B : \text{Prop} \quad Γ \vdash ∀ x : A . B : \text{Prop} \quad s ∈ \{\text{Type}, \text{Set}, \text{Prop}\} \]

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

\[ \text{Prop} ⊆ \text{Set}_i \quad \text{Set}_i ⊆ \text{Set}_j \quad \text{Type}_i ⊆ \text{Type}_j \]

One should note that CIC_ref easily embeds into CIC by mapping any Set_i and Type_i onto the Type_i of CIC. The coherence of CIC thus implies the coherence of CIC_ref.

III. PARAMETRICITY

We can define a notion of relational parametricity for CIC_ref.
Definition 1 (Parametricity relation). For any inductive
\[ \text{Ind}^p(I, A, c; C^k) \]
and a family \((\llbracket c(x) \rrbracket)_{x=1..k}\) of fresh constructor names.

The parametricity translation \([\cdot]\) is defined by induction on the structure of terms and contexts:

- \([\Gamma, x : A] = [\Gamma']
  - [s] = \lambda(x : s)(x' : s).x \to x' \to s
  - [x] = x_R
  - \[\forall x : A.B] = \lambda(f : \forall x : A.B)(f' : \forall x' : A'.B'). \forall(x : A)(x' : A')(x_R : [A] x x').
    \[B]\, (f x) (f' x')
  - \[\lambda x : A.B] = \lambda(x : A)(x' : A')(x_R : [A] x x').
    \[B]\, \[([A] B) B'[B']\]
  - \[\text{fix}(x : A).B\] = \[\text{fix}(x_R : [A] x x').B]\]
  - \[\text{case}_I(M, Q^p, T, F^T)\] = \[\text{case}_I([M, Q, Q', [Q^p]], \Theta_I(Q^p, T, F^T), F^T)\]

where \(\text{Prop} = \text{Set}_1\) = Prop and \(\text{Type}_1 = \text{Type}\), and where \(A'\) denotes the term \(A\) in which we have replaced each variable \(x\) by a fresh variable \(x'\). The definition of \(\Theta_I\) is in Fig. 1.

What is new with respect to previous works is the fact that relations over objects of type Prop or Set, 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 \(\vdash_s\) for the derivability where strong elimination is authorized only over small inductive definitions.

Theorem 1 (Abstraction theorem). If \(\Gamma \vdash_s A : B\) then \(\llbracket \Gamma \rrbracket \vdash_s A : B, [\Gamma'] \vdash_s A' : B', and [\Gamma'] \vdash_s [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

\[ \Theta_I(Q^p, T, F^T) = \lambda(x : A)(x' : A')(x_R : [A] x x') \quad (a : I Q^p x^x)(a' : I Q^p x^x)(a_R : [I] Q Q' Q^p x x' x_R a a'). \]

\[ \llbracket T \rrbracket x x' x_R a a' a_R (\text{case}_I(a, Q, T, F^T)) (\text{case}_I(a', Q', T', F^{T'})) \]

Fig. 1. Relation parametricity for inductive types

REFERENCES