Parametricity in an Impredicative Sort

Chantal Keller    Marc Lasson

INRIA – École Polytechnique – École Normale Supérieure de Lyon

October, 26th 2012
Motivation

Explore parametricity for CIC

- theoretical application to proof assistants based on Type Theory? with impredicative sorts (like in Coq)?
- possible implementation? automation?
- theoretical consequences?
- links with realizability and extraction?
Outline

1. Parametricity?
2. Applications in CIC and Coq
3. Refining CC
4. Inductive definitions
5. Conclusion
The slogan

In a typed $\lambda$-calculus with polymorphism (eg. ML, system F):

- “A function behaves uniformly wrt its polymorphic arguments.”
- Idea: it cannot inspect its polymorphic arguments
- Examples:
  - functions of type $\forall \alpha, \alpha \to \alpha$ are identities
  - functions of type $\forall \alpha \beta, \alpha \to \beta \to \alpha$ are projections
  - functions of type $\forall \alpha$, list $\alpha \to$ list $\alpha$ can only rearrange lists

Define logical relations between programs: $M \sim_{\tau} N$

- $f \sim_{A \to B} f' \triangleq \forall (a a' : A), a \sim_A a' \Rightarrow f a \sim_B f' a'$
Parametricity in a nutshell

The slogan

In a typed $\lambda$-calculus with polymorphism (eg. ML, system F):

- “A function behaves uniformly wrt its polymorphic arguments.”
- Idea: it cannot inspect its polymorphic arguments
- Examples:
  - functions of type $\forall \alpha, \alpha \to \alpha$ are identities
  - functions of type $\forall \alpha \beta, \alpha \to \beta \to \alpha$ are projections
  - functions of type $\forall \alpha$, list $\alpha \to$ list $\alpha$ can only rearrange lists

Define logical relations between programs: $M \sim_T N : \text{Prop}$

- $f \sim_{A \to B} f' \triangleq \forall (a a' : A), a \sim_A a' \Rightarrow f a \sim_B f' a'$
Theorems for free!

Abstraction theorem:

- if ⊢ M : A then M ∼_A M

Example:

- have r : ∀α, list α → list α and f : A → B
- then map f ∘ r_A = r_B ∘ map f

Sketch of the proof

- for any relation R, if (l, l') ∈ R, then (r_A l, r_B l') ∈ R
- take R = {(l, l')|map f l = l'}
- then if map f l = l' then map f (r_A l) = r_B l'
- ie. map f (r_A l) = r_B (map f l)
Outline

1. Parametricity?
2. Applications in CIC and Coq
3. Refining CC
4. Inductive definitions
5. Conclusion
Theorems for free

Naturality properties:

- Lots of formal proofs rely on commutation between functions

Example with data types with structure: Finite Group Theory

- \( \mathcal{H} = (\alpha, \cdot, \text{inv}, [\text{axioms}]) \) a group structure
- \( \text{fingrp}_{\mathcal{H}} \) the type of finite subgroups of \( \mathcal{H} \)
- \( Z : \text{fingrp}_{\mathcal{H}} \rightarrow \text{fingrp}_{\mathcal{H}} \) a group constructor
- We can prove: if \( Z \sim Z \) then for any \( G \), \( Z G \) is a characteristic subgroup of \( G \) (ie invariant by automorphism) (requires proof irrelevance)
- The abstraction theorem gives a proof of \( Z \sim Z \) for any concrete implementation of \( Z \) (eg. center, normalizer...)

Independence results

Provably not parametric:

- A type $\tau$ is *provably not parametric* if one can prove that $\forall x : \tau, \neg(x \sim_{\tau} x)$.
- In that case: $\tau$ is not inhabited.

Independence of Excluded Middle:

- Peirce’s law is provably not parametric, so uninhabited
- Its negation is also uninhabited (counter-model)
- So it is independent
Possibility to add axioms

**Provably parametric:**

- A type $\tau$ is *provably parametric* if one can prove that
  $\forall x : \tau, x \sim_\tau x$.
- In that case: adding $\tau$ to the system does not break parametricity

**Example:**

- Proof irrelevance is provably parametric
Conclusion of the first two parts

What to remember of this talk:

- parametricity talks about properties shared by all the terms of a same type
- when you define a new Coq function, you could get naturality properties automatically!
- other results for type theoreticians to play with
- a very small patch of Coq with nice properties (beyond parametricity): to come
## Outline

1. Parametricity?
2. Applications in CIC and Coq
3. Refining CC
4. Inductive definitions
5. Conclusion
The Calculus of Constructions

The sort hierarchy of Coq (before)

\[
\begin{align*}
\text{nat} & \quad \in \text{Set} \\
\text{list} & \quad \in \text{Set} \\
\forall \alpha, \alpha \to \alpha & \quad \in \text{Type}_1 \\
P \land Q & \quad \in \text{Type}_2 \\
x = y & \quad \in \text{Type}_3 \\
\forall X, X \to X & \quad \in \ldots
\end{align*}
\]

Impredicative \textbf{Set} and \textbf{Prop}

\[
\begin{align*}
(\forall \alpha : \text{Set}.\alpha \to \alpha) & : \text{Set} \\
(\forall X : \text{Prop}.X \to X) & : \text{Prop}
\end{align*}
\]

Predicative \textbf{Type}

\[
\begin{align*}
(\forall \alpha : \text{Type}_{i}.\alpha \to \alpha) & : \text{Type}_{i+1}
\end{align*}
\]
The Calculus of Constructions

The sort hierarchy of Coq (before)

\[
\begin{align*}
\text{nat} \quad \text{list} \quad \forall \alpha, \alpha \rightarrow \alpha \\
P \land Q \\
x = y \\
\forall X, X \rightarrow X
\end{align*}
\]

\[\in \text{Set} \quad \in \text{Type}_1 \quad \in \text{Type}_2 \quad \in \text{Type}_3 \quad \in \ldots\]

- Impredicativity increases the expressive power of the system

The need for a refinement
The Calculus of Constructions

The sort hierarchy of Coq (before)

\[
\begin{align*}
\text{nat} & \in \text{Set} \\
\text{list} & \in \text{Set} \\
\forall \alpha, \alpha \to \alpha & \in \text{Type}_1 \\
P \land Q & \in \text{Type}_2 \\
x = y & \in \text{Type}_3 \\
\forall X, X \to X & \in \text{Type}_4 \\
\end{align*}
\]

- Impredicativity increases the expressive power of the system
- \textbf{Set} impredicative + classical axioms lead inconsistency
The Calculus of Constructions

The sort hierarchy of Coq (now)

\[
\begin{align*}
\text{nat} & \in \text{Set} \\
\text{list} & \in \text{Set} \\
\forall \alpha, \alpha \to \alpha & \in \text{Type}_1 \\
P \land Q & \in \text{Type}_2 \\
x = y & \in \text{Type}_3 \\
\forall X, X \to X & \in \ldots
\end{align*}
\]

- Impredicativity increases the expressive power of the system
- **Set** impredicative + classical axioms lead inconsistency
- \(\hookrightarrow\) get rid of **Set**
The Calculus of Constructions

The sort hierarchy of Coq (now)

\[
\begin{align*}
\forall \alpha, \alpha & \rightarrow \alpha \\
P \land Q & \\
x = y & \\
\forall X, X \rightarrow X \\
\end{align*}
\]

\[
\begin{cases}
nat \\
list \\
\end{cases} \in \text{Prop}
\]

\[
\begin{cases}
\in \text{Set} \\
\in \text{Type}_2 \in \text{Type}_3 \in \ldots
\end{cases}
\]

- Impredicativity increases the expressive power of the system
- Set impredicative + classical axioms lead inconsistency
- \rightarrow get rid of Set
The Calculus of Constructions

The sort hierarchy of Coq (now)

\[
\begin{align*}
\text{nat} & \rightarrow \\
\text{list} & \\
\forall \alpha, \alpha \rightarrow \alpha & \\
P \land Q & \\
x = y & \\
\forall X, X \rightarrow X & \\
\end{align*}
\]

\[\in \text{Set} \quad \in \text{Type}_2 \quad \in \text{Type}_3 \quad \in \ldots \]

**Impredicative Prop**

\[(\forall X : \text{Prop}.X \rightarrow X) : \text{Prop}\]

**Predicative Type**

\[(\forall \alpha : \text{Type}_i.\alpha \rightarrow \alpha) : \text{Type}_{i+1}\]
The Refined Calculus of Constructions: $\text{CC}_r$

Reintroducing $\text{Set}$ as a predicative hierarchy:

- We still have: $\text{Prop} \in \text{Type}_1 \in \text{Type}_2 \in \text{Type}_3 \in \ldots$
- We add: $\text{Set}_0 \subset \text{Set}_1 \subset \text{Set}_2 \subset \ldots$
- Such that:
  - $\text{Set}_0 \in \text{Type}_1$
  - $\text{Set}_1 \in \text{Type}_2$
  - $\text{Set}_2 \in \text{Type}_3$
  - $\ldots$

We know where computation appears:

- Informative types are inhabitants of $\text{Set}$
- Informative terms are inhabitants of informative types
- Extraction: prune non informative subterms (look at the types)
Important rules

Axioms:

\[
\begin{align*}
\vdash \text{Prop} : \text{Type}_1 \\
\vdash \text{Set}_i : \text{Type}_{i+1} \\
\vdash \text{Type}_i : \text{Type}_{i+1}
\end{align*}
\]

Other rules:

- Like in CC
- Dependent products such that Prop is impredicative
- Easily embeds into CC (collapse Set and Type) $\hookrightarrow$ coherent
Main idea:

- Define a translation $\llbracket \bullet \rrbracket$ from terms to terms
- The translation of a “type” (a term inhabiting a sort) is a relation on this type: if $\vdash B : \text{Set}$ then $\vdash \llbracket B \rrbracket : B \to B \to \text{Prop}$
- The translation of other terms are proofs that these relations hold
- It gives the abstraction theorem: if $\vdash A : B$ then $\vdash \llbracket A \rrbracket : \llbracket B \rrbracket A A$
Translation of sorts

The translation of sorts defines the nature of parametricity relations:

- \([\text{Prop}] = \lambda(PQ : \text{Prop}).P \rightarrow Q \rightarrow \text{Prop}\)
- \([\text{Set}] = \lambda(PQ : \text{Set}).P \rightarrow Q \rightarrow \text{Prop}\)
- \([\text{Type}] = \lambda(PQ : \text{Type}).P \rightarrow Q \rightarrow \text{Type}\)
Translation of sorts

The translation of sorts defines the nature of parametricity relations:

- $\mathbb{P}\text{prop} = \lambda(PQ : \text{Prop}).P \rightarrow Q \rightarrow \text{Prop}$
- $\mathbb{R}\text{set} = \lambda(PQ : \text{Set}).P \rightarrow Q \rightarrow \text{Prop}$
- $\mathbb{R}\text{typte} = \lambda(PQ : \text{Type}).P \rightarrow Q \rightarrow \text{Type}$
Towards an integration in a proof assistant

Easy part:

- We recall the abstraction theorem: if $\vdash A : B$ then $\vdash [A] : [B] A A$
- Given a term $A$ of type $B$, internally compute $[A]$, and check its type is $[B] A A$
- Kind of computational reflection

Difficult part:

- Automatically prove “theorems for free”
- Example: if $Z \sim Z$ then for any $G$, $Z G$ is a characteristic subgroup of $G$
- The difficulty is to instantiate the abstraction theorem with well chosen relations.
# Outline

1. Parametricity?
2. Applications in CIC and Coq
3. Refining CC
4. Inductive definitions
5. Conclusion
Translation of inductive definitions

Example:

\textit{Inductive} \ \texttt{list} \ (A: \texttt{Set}) : \ \texttt{Set} := \\
| \ \texttt{nil} : \ \texttt{list} \ A \\
| \ \texttt{cons} : A \rightarrow \ \texttt{list} \ A \rightarrow \ \texttt{list} \ A.

Translated into:

\textit{Inductive} \ [\texttt{list}] \ (A A': \texttt{Set}) \ (R:A \rightarrow A' \rightarrow \texttt{Prop}) : \\
\texttt{Set} \rightarrow \texttt{Set} \rightarrow \texttt{Prop} := \\
| [\texttt{nil}] : [\texttt{list}] A A' R (\texttt{nil} A) (\texttt{nil} A') \\
| [\texttt{cons}] : \forall a a', R a a' \rightarrow \\
\forall l l', [\texttt{list}] A A' R l l' \rightarrow \\
[\texttt{list}] A A' R (\texttt{cons} a l) (\texttt{cons} a' l')
Elimination schemes

We destruct $l : s$ to build $A : \_ : r$

<table>
<thead>
<tr>
<th>$s$</th>
<th>$r$</th>
<th>Prop</th>
<th>Set</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prop</td>
<td>small</td>
<td>small (restricted)</td>
<td>large (restricted)</td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td>small</td>
<td>small</td>
<td>large</td>
<td></td>
</tr>
</tbody>
</table>
Translation of small eliminations

Consider:

\begin{align*}
\textbf{Fixpoint} & \quad \text{length} \ (l : \text{list } A) : \text{nat} := \text{match} \ l \ \text{with} \\
& \quad | \quad \text{n} \text{il} \Rightarrow 0 \\
& \quad | \quad \text{cons} \ _\ l' \Rightarrow \text{S} \ (\text{length} \ l')
\end{align*}

For \text{length} to be parametric, we must provide a proof that:

\begin{align*}
\text{forall} \ (l \ l' : \text{list } A), \ & \quad \text{list} \ l \ l' \rightarrow \\
& \quad \text{nat} \ (\text{length} \ l) \ (\text{length} \ l')
\end{align*}

- We have: \quad \text{list} \ l \ l' : \text{Prop}
- And: \quad \text{nat} \ (\text{length} \ l) \ (\text{length} \ l') : \text{Prop}

\rightarrow \text{authorized elimination}
And large eliminations?

**Definition** setify \( (l : \text{list } A) : \text{Set} := \text{match } l \text{ with} \)

\[
\begin{align*}
| \text{nil} & \Rightarrow \text{unit} \\
| \text{cons } _{-} _{-} & \Rightarrow \text{nat}
\end{align*}
\]

end.

For setify to be parametric, we must provide a proof that:

\[
\forall (l \ l') : \text{list } A), \ [\text{list}] l \ l' \rightarrow [\text{Set}] (\text{setify } l) (\text{setify } l')
\]

that is to say:

\[
\forall (l \ l') : \text{list } A), \ [\text{list}] l \ l' \rightarrow (\text{setify } l) \rightarrow (\text{setify } l') \rightarrow \text{Prop}
\]

- We have: \([\text{list}] l \ l' : \text{Prop}\)
- But: \((\text{setify } l) \rightarrow (\text{setify } l') \rightarrow \text{Prop : Type}\)

← unauthorized elimination
Summary

We have parametricity:

- for inductive definitions
- for small eliminations
- but not for large eliminations
- but we have a workaround for many of them (namely, large eliminations over small inductive definitions, containing usual data types)
Outline

1. Parametricity?
2. Applications in CIC and Coq
3. Refining CC
4. Inductive definitions
5. Conclusion
Conclusion

\( CIC_r \), a type system close to \( CIC \):

- that distinguishes computationally meaningful expressions
- with the possibility to add classical axioms
- in which we have a notion of parametricity
- that gives theoretical and practical applications
- like an original way to prove properties in algebra

Perspectives:

- build Coq tactics (in progress)
- define the Refined Coq (in progress)
- extraction of \( CIC_r \), links between extraction and parametricity
- realizability in \( CIC_r \)
Credo

This talk was about parametricity, but what is important is $CIC_r$. 
Thanks

Thanks for your attention!

Any questions?