Parametricity in an Impredicative Sort

Chantal Keller  Marc Lasson

INRIA – École Polytechnique – École Normale Supérieure de Lyon

September, 6th 2012
Motivation

Parametricity recently established for Pure Type Systems

- 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

Slogan:

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

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

- metatheoretical (Reynolds)
The slogan

Slogan:

■ “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 $\forall \alpha, \text{list} \alpha \to \text{list} \alpha$ can only rearrange lists

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

■ metatheoretical (Reynolds)
■ in a Second-Order logic (Abadi-Plotkin)
The slogan

Slogan:

- “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: $M \sim \tau N : \text{Type}$

- metatheoretical (Reynolds)
- in a Second-Order logic (Abadi-Plotkin)
- a type of the system (Bernardy et al. 2010)
The slogan

Slogan:

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

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

- metatheoretical (Reynolds)
- in a Second-Order logic (Abadi-Plotkin)
- a type of the system (Bernardy et al. 2010)
- a proposition of the system (our system)
Abstraction and applications

The abstraction theorem:

- If $\vdash M : \tau$ then $M \sim_\tau M$
Abstraction and applications

The abstraction theorem:

- If $\vdash M : \tau$ then $\vdash [M] : M \sim_{\tau} M$
Abstraction and applications

The abstraction theorem:

- If $\vdash M : \tau$ then $\vdash [M] : M \sim \tau M$

Theorems for free!

- Given $r : \forall \alpha, \text{list} \alpha \rightarrow \text{list} \alpha$
- and $f : \tau \rightarrow \sigma$
- then $\text{map } f \circ r_\sigma = r_\tau \circ \text{map } f$

Sketch of the proof

- Abstraction gives: $r \sim \forall \alpha, \text{list} \alpha \rightarrow \text{list} \alpha r$
- Given a relation between $\tau$ and $\sigma$, $r_\tau$ and $r_\sigma$ are pointwise related $\leftrightarrow$ take the graph of $f$
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...)

Parametricity in an Impredicative Sort

Chantal Keller
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
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 2006)

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

**Impredicative Set and Prop**

\[
(\forall \alpha : \text{Set}. \alpha \to \alpha) : \text{Set} \\
(\forall X : \text{Prop}. X \to X) : \text{Prop}
\]

**Predicative Type**

\[
(\forall \alpha : \text{Type}_i. \alpha \to \alpha) : \text{Type}_{i+1}
\]
The Calculus of Constructions

The sort hierarchy of Coq (before 2006)

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

- Impredicativity increases the expressive power of the system
The Calculus of Constructions

The sort hierarchy of Coq (before 2006)

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

- Impredicativity increases the expressive power of the system
- Set impredicative + classical axioms lead inconsistency
The need for a refinement

The Calculus of Constructions

The sort hierarchy of Coq

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

- 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

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

\{ \in \text{Type}_1 \in \text{Type}_2 \in \text{Type}_3 \in \ldots \}

- 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

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

\[\in \text{Prop} \]

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

**Impredicative Prop**

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

**Predicative Type**

\[(\forall \alpha : \text{Type}_i.\alpha \to \alpha) : \text{Type}_{i+1}\]
The Refined Calculus of Constructions: CC_r

Reintroducing \textit{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 \textit{Set}
- Informative terms are inhabitants of informative types
- Extraction: prune non informative subterms (look at the types)
The need for a refinement

Important rules

Axioms:

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

Other rules:

- Like in CC
- Dependent products such that Prop is impredicative
- Easily embeds into CC (collapse Set and Type) \(\leftrightarrow\) coherent


Presentation

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
- 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:

- $\lbrack \text{Prop} \rbrack = \lambda (PQ : \text{Prop}). P \to Q \to \text{Prop}$
- $\lbrack \text{Set} \rbrack = \lambda (PQ : \text{Set}). P \to Q \to \text{Prop}$
- $\lbrack \text{Type} \rbrack = \lambda (PQ : \text{Type}). P \to Q \to \text{Type}$
Translation of sorts

The translation of sorts defines the nature of parametricity relations:

- \([\text{Prop}] = \lambda(PQ: \text{Prop}).P \rightarrow Q \rightarrow \text{Prop}\)
- \([\text{Set}] = \lambda(PQ: \text{Set}).P \rightarrow Q \rightarrow \text{Prop}\)
- \([\text{Type}] = \lambda(PQ: \text{Type}).P \rightarrow Q \rightarrow \text{Type}\)
Towards an integration in a proof assistant

**Easy part:**

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

**Difficult part:**

- Automatically prove “theorems for free”
- Example: if \( Z \sim Z \) then for any \( G \), \( ZG \) is a characteristic subgroup of \( G \)
- The difficulty is to instantiate the abstraction theorem with well chosen relations.
<table>
<thead>
<tr>
<th>#</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parametricity?</td>
</tr>
<tr>
<td>2</td>
<td>Applications in CIC and Coq</td>
</tr>
<tr>
<td>3</td>
<td>Refining CC</td>
</tr>
<tr>
<td>4</td>
<td>Inductive definitions</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Translation of inductive definitions

Example:

\[
\text{Inductive } \text{list} \ (A : \text{Set}) : \text{Set} := \\
| \text{nil} : \text{list} \ A \\
| \text{cons} : A \rightarrow \text{list} \ A \rightarrow \text{list} \ A.
\]

Translated into:

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

We destruct \( \ell : s \) to build \( A : \_ : r \)

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

Consider:

\[
\text{Fixpoint } \text{length} (l : \text{list } A) : \text{nat} := \text{match } l \text{ with}
\begin{align*}
&\mid \text{nil} \Rightarrow 0 \\
&\mid \text{cons } _l ' \Rightarrow S (\text{length } l ')
\end{align*}
\text{end}.
\]

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

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

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

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

**Definition** `setify (l : list A) : Set := match l with
| nil ⇒ unit
| cons _ _ ⇒ nat
end`.

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

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

that is to say:

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

- We have: `⟦list⟧ l l' : Prop
- But: `(setify l) \rightarrow (setify l') \rightarrow 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
Thanks

Thanks for your attention!

Any questions?