

A Concrete Parametric Domain-theoretic Model of $PILL_{\gamma}$

Rasmus Lerchedahl Petersen

Department of Theoretical Computer Science
IT University of Copenhagen

August 26th 2005 / Qualification Exam

Outline

- 1 Context
 - Parametricity
 - Fixed Points

Outline

- 1 Context
 - Parametricity
 - Fixed Points
- 2 Concrete Model
 - Stage I: A DILL-model
 - Stage II: A PILL-model
 - Stage III: A Parametric LAPL-structure

Outline

- 1 Context
 - Parametricity
 - Fixed Points
- 2 Concrete Model
 - Stage I: A DILL-model
 - Stage II: A PILL-model
 - Stage III: A Parametric LAPL-structure
- 3 Future Work
 - LILY
 - Expanding the Calculus

Outline

- 1 Context
 - Parametricity
 - Fixed Points
- 2 Concrete Model
 - Stage I: A DILL-model
 - Stage II: A PILL-model
 - Stage III: A Parametric LAPL-structure
- 3 Future Work
 - LILY
 - Expanding the Calculus

Polymorphism

Strict typing discipline has proven to be an efficient tool for developing structured programs

Limitations

However, distinguishing programs with different types, the typing discipline would consider the following two programs different

```
int - reverse : int - list → int - list  
string - reverse : string - list → string - list
```

although they both simply reverse a list, and thus perform essentially the same operation.

Polymorphism

Solution

Polymorphism allows one to define a general reverse function

$$\text{reverse} : \forall \alpha. \alpha - \text{list} \rightarrow \alpha - \text{list}$$

which yields the needed functions upon instantiation:

$$\text{int} - \text{reverse} = \text{reverse}(\text{int})$$

$$\text{string} - \text{reverse} = \text{reverse}(\text{string})$$

`reverse` is then called a *polymorphic* function.



Parametric Polymorphism

For a language with polymorphism we can consider parametricity.

Intuition

Intuitively parametricity is the statement that

“Polymorphic functions behave the same on all type-instantiations”

For example, a function t of type $\forall\alpha.\alpha \rightarrow \alpha$ can not instantiate to the identity on some types and to the successor function on the natural numbers. In fact the program t can only be the polymorphic identity function.

Formalizing Parametricity

We use a formalization due to Reynolds called

Relational Parametricity

Intuitively relational parametricity is the statement that

“Polymorphic functions respect all relations”

Formalizing Parametricity

We use a formalization due to Reynolds called

Relational Parametricity

Intuitively relational parametricity is the statement that

“Polymorphic functions respect all relations”

As an example consider the relation R from `letters` to `numbers` given by

$$a R 1 \quad b R 2 \quad c R 3 \dots$$

Formalizing Parametricity

We use a formalization due to Reynolds called

Relational Parametricity

Intuitively relational parametricity is the statement that

“Polymorphic functions respect all relations”

As an example consider the relation R from `letters` to `numbers` given by

$$a R 1 \quad b R 2 \quad c R 3 \dots$$

This may be lifted to a relation $R - list$ from `letter - lists` to `number - lists` whereby

$$c, e, a, b, d \ R - list \ 3, 5, 1, 2, 4$$



Formalizing Parametricity

If `reverse` is relational parametric we then require

```
reverse(letter - list)(c, e, a, b, d)  
      R - list  
reverse(number - list)(3, 5, 1, 2, 4)
```

Formalizing Parametricity

If `reverse` is relational parametric we then require

```
reverse(letter - list)(c, e, a, b, d)
      R - list
reverse(number - list)(3, 5, 1, 2, 4)
```

which means, that if

$$\text{reverse}(\text{letter} - \text{list})(c, e, a, b, d) = (d, b, a, e, c)$$


Formalizing Parametricity

If `reverse` is relational parametric we then require

$$\begin{array}{c} \text{reverse}(\text{letter} - \text{list})(c, e, a, b, d) \\ R - \text{list} \\ \text{reverse}(\text{number} - \text{list})(3, 5, 1, 2, 4) \end{array}$$

which means, that if

$$\text{reverse}(\text{letter} - \text{list})(c, e, a, b, d) = (d, b, a, e, c)$$

we can be sure that

$$\text{reverse}(\text{number} - \text{list})(3, 5, 1, 2, 4) = (4, 2, 1, 5, 3)$$


Formalizing Parametricity

“In fact the program t can only be the polymorphic identity function.”



Formalizing Parametricity

“In fact the program t can only be the polymorphic identity function.”

Recall t had type $\forall\alpha.\alpha \rightarrow \alpha$.



Formalizing Parametricity

“In fact the program t can only be the polymorphic identity function.”

Recall t had type $\forall\alpha.\alpha \rightarrow \alpha$.

Let σ be a type and consider $x : \sigma$.

We wish to show $t(\sigma)(x) = x$.



Formalizing Parametricity

“In fact the program t can only be the polymorphic identity function.”

Recall t had type $\forall\alpha.\alpha \rightarrow \alpha$.

Let σ be a type and consider $x : \sigma$.

We wish to show $t(\sigma)(x) = x$.

Define R from σ to σ by

$$a R b \iff a = x$$



Formalizing Parametricity

“In fact the program t can only be the polymorphic identity function.”

Recall t had type $\forall\alpha.\alpha \rightarrow \alpha$.

Let σ be a type and consider $x : \sigma$.

We wish to show $t(\sigma)(x) = x$.

Define R from σ to σ by

$$a R b \iff a = x$$

Since $x R x$, we know that $t(\sigma)(x) R t(\sigma)(x)$ which means that $t(\sigma)(x) = x$.



Earlier studies

Lars Birkedal and Rasmus Møgelberg have studied λ_2 , the “simplest possible calculus with polymorphism”:

$$\sigma ::= \alpha \mid \sigma \times \sigma \mid \sigma \rightarrow \sigma \mid \Pi \alpha : \text{Type}.\sigma$$

They used a logic by Abadi and Plotkin to express that terms preserve relations:

$$\begin{aligned} \phi ::= & (M =_{\sigma} N) \mid R(M, N) \mid \perp \mid \top \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \supset \phi \mid \\ & \forall \alpha : \text{Type}.\phi \mid \forall \mathbf{x} : \sigma.\phi \mid \forall R \subset \sigma \times \tau.\phi \mid \\ & \exists \alpha : \text{Type}.\phi \mid \exists \mathbf{x} : \sigma.\phi \mid \exists R \subset \sigma \times \tau.\phi \mid \\ & \sigma[\vec{\rho}] \end{aligned}$$

They then studied models that could contain both λ_2 and the logic.



APL-structures

This was accomplished through extensive use of category theory; in particular fibrations: It is well known, that models of λ_2 are so-called λ_2 -fibrations, while logic is frequently modeled in logic-fibrations:

λ_2 -fibration

Type



Kind

Prop



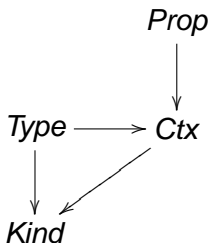
Ctx

Logic fibration



APL-structures

This was accomplished through extensive use of category theory; in particular fibrations: It is well known, that models of λ_2 are so-called λ_2 -fibrations, while logic is frequently modeled in logic-fibrations:



Together they form an APL-structure!



Results

An APL-structure is defined to be parametric if parametricity as formulated in the logic holds in the model.

Results

An APL-structure is defined to be parametric if parametricity as formulated in the logic holds in the model.

This is a good notion in that a parametric APL-structure enjoy the consequences of parametricity.



Results

An APL-structure is defined to be parametric if parametricity as formulated in the logic holds in the model.

This is a good notion in that a parametric APL-structure enjoy the consequences of parametricity.

They have soundness and completeness results, and even describe a parametrization process. . .



Outline

- 1 Context
 - Parametricity
 - **Fixed Points**
- 2 Concrete Model
 - Stage I: A DILL-model
 - Stage II: A PILL-model
 - Stage III: A Parametric LAPL-structure
- 3 Future Work
 - LILY
 - Expanding the Calculus

Turing completeness

λ_2 is strongly normalizing.

Turing completeness

λ_2 is strongly normalizing.

To obtain Turing completeness we would like to add fixed points to the language.

Turing completeness

λ_2 is strongly normalizing.

To obtain Turing completeness we would like to add fixed points to the language.

This suggests domain-theoretic models. . .



Domain Theory

Given a set A (with $|A| > 1$) there are plenty of set-theoretic function $A \rightarrow A$ without fixed points. But...



Domain Theory

Given a set A (with $|A| > 1$) there are plenty of set-theoretic function $A \rightarrow A$ without fixed points. But. . .

In domain theory all endofunctions have fixed points!



Linearity

We cannot just add fixed points. Parametricity would collapse the type theory (Huwig and Poigné).



Linearity

We cannot just add fixed points. Parametricity would collapse the type theory (Huwig and Poigné).

Introducing linearity into the calculus saves the day.



Conclusion

We would like to study parametricity of models of a Polymorphic Intuitionistic/Linear λ -calculus with fixed points.



Conclusion

We would like to study parametricity of models of a Polymorphic Intuitionistic/Linear λ -calculus with fixed points.

The logic is modified to account for linearity, obtaining LAPL.



Conclusion

We would like to study parametricity of models of a Polymorphic Intuitionistic/Linear λ -calculus with fixed points.

The logic is modified to account for linearity, obtaining LAPL.

The notion of model is now an LAPL-structure.



Conclusion

We would like to study parametricity of models of a Polymorphic Intuitionistic/Linear λ -calculus with fixed points.

The logic is modified to account for linearity, obtaining LAPL.

The notion of model is now an LAPL-structure.

A concrete LAPL-structure should be domain-theoretic.



Outline

- 1 Context
 - Parametricity
 - Fixed Points
- 2 **Concrete Model**
 - **Stage I: A DILL-model**
 - Stage II: A PILL-model
 - Stage III: A Parametric LAPL-structure
- 3 Future Work
 - LILY
 - Expanding the Calculus

Setup

A model of the simply typed λ -calculus is obtained from a model of the untyped λ -calculus. One such consists of:

- A reflexive CPO, D .

Setup

A model of the simply typed λ -calculus is obtained from a model of the untyped λ -calculus. One such consists of:

- A reflexive CPO, D .

D is a pointed chain-complete partial order such that we have

$$\Phi: D \rightarrow [D \rightarrow D] \quad \text{and} \quad \Psi: [D \rightarrow D] \rightarrow D$$

both strict and continuous, satisfying

$$\Phi \circ \Psi = id_{[D \rightarrow D]}$$



Setup

A model of the simply typed λ -calculus is obtained from a model of the untyped λ -calculus. One such consists of:

- A reflexive CPO, D .

D is a pointed chain-complete partial order such that we have

$$\Phi: D \rightarrow [D \rightarrow D] \quad \text{and} \quad \Psi: [D \rightarrow D] \rightarrow D$$

both strict and continuous, satisfying

$$\Phi \circ \Psi = id_{[D \rightarrow D]}$$

- Strict coding of functions



Setup

A model of the simply typed λ -calculus is obtained from a model of the untyped λ -calculus. One such consists of:

- A reflexive CPO, D .
 D is a pointed chain-complete partial order such that we have

$$\Phi: D \rightarrow [D \rightarrow D] \quad \text{and} \quad \Psi: [D \rightarrow D] \rightarrow D$$

both strict and continuous, satisfying

$$\Phi \circ \Psi = id_{[D \rightarrow D]}$$

- Strict coding of functions
- Strict coding of pairing and projections

We think of this as the bit-level.



Admissible Pers

Definition

An *admissible partial equivalence relation* on D is a partial equivalence relation R on D satisfying

strict $\perp_D R \perp_D$

complete For $(x_i)_{i \in I}$ and $(y_i)_{i \in I}$ chains in D :

$$(\forall i \in I. x_i R y_i) \Rightarrow \bigsqcup_{i \in I} x_i R \bigsqcup_{i \in I} y_i$$



Admissible functions

Definition

For R and S admissible pers on D , define the set of functions *admissible from R to S* as

$$\mathcal{F}(R, S) = \{f \in [D \rightarrow D] \mid x R y \Rightarrow f(x) S f(y)\}$$



AdmPer(D)

Consider the category $\text{AdmPer}(D)$ of admissible partial equivalence relations on D and continuous functions:

Objects: are admissible pers.

Morphisms: a morphism $[f]: R \rightarrow S$ is an equivalence class in $\mathcal{F}(R, S) / R \sim_S$. Elements of $[f]$ are called *realizers* for $[f]$.

We think of these as types and typed functions.



AdmPer(D) $_{\perp}$

Linearity is modeled as strictness.

Definition

The set of **strict** functions admissible from R to S is defined as

$$\mathcal{F}(R, S)_{\perp} = \{f \in \mathcal{F}(R, S) \mid f(\perp_D) = \perp_D\}$$

Consider then the category $\text{AdmPer}(D)_{\perp}$ of admissible pers and strict continuous functions:

Objects: same as for $\text{AdmPer}(D)$, admissible pers.

Morphisms: a morphism $[f]: R \rightarrow S$ is an equivalence class in

$$\mathcal{F}(R, S)_{\perp} / R \sim_S.$$

We think of these as types and strict typed functions.



The monoidal adjunction

We use a coded version of lifting due to Longley and Simpson:

Definition

Define the map $L_0: \mathcal{P}(D) \rightarrow \mathcal{P}(D)$ by

$$L_0(A) = \{d \in D \mid \pi(\Phi(d)(i)) = i \wedge \pi'(\Phi(d)(i)) \in A\}.$$

for $A \subseteq D$. And then the map

$\mathcal{L}_0: \text{SgAdmPer}(D)_0 \rightarrow \text{SgAdmPer}(D)_{\perp 0}$ by

$$\mathcal{L}_0(R) = \{L(K) \mid K \in D/R\} \cup \{\{\perp_D\}\}.$$



Outline

- 1 Context
 - Parametricity
 - Fixed Points
- 2 Concrete Model
 - Stage I: A DILL-model
 - **Stage II: A PILL-model**
 - Stage III: A Parametric LAPL-structure
- 3 Future Work
 - LILY
 - Expanding the Calculus

Adding Polymorphism

Polymorphism is added in the standard way:

- All categories are turned into fibered categories.
- All fibers are copies of the old categories, i.e. a DILL-model.
- Simple Ω -products are by intersection.



Outline

- 1 Context
 - Parametricity
 - Fixed Points
- 2 Concrete Model
 - Stage I: A DILL-model
 - Stage II: A PILL-model
 - Stage III: A Parametric LAPL-structure
- 3 Future Work
 - LILY
 - Expanding the Calculus

Apply Completion Process

To ensure that we end up with a parametric model, we apply the completion process. Roughly it does the following:

- All types are given a relational interpretation.

Apply Completion Process

To ensure that we end up with a parametric model, we apply the completion process. Roughly it does the following:

- All types are given a relational interpretation.
- Types that cannot be given a relational interpretation are discarded.

Apply Completion Process

To ensure that we end up with a parametric model, we apply the completion process. Roughly it does the following:

- All types are given a relational interpretation.
- Types that cannot be given a relational interpretation are discarded.
- Only terms which respect the relational interpretations are allowed.



Apply Completion Process

To ensure that we end up with a parametric model, we apply the completion process. Roughly it does the following:

- All types are given a relational interpretation.
- Types that cannot be given a relational interpretation are discarded.
- Only terms which respect the relational interpretations are allowed.

If what is left still constitutes a PILL-model, it is probably parametric. . .



Apply Completion Process

$$\sigma \mapsto (\sigma, R_\sigma)$$



Logic: Ctx and Prop

To model the logic, we use the category of sets:

- Types are modeled as sets.
- Propositions on types are modeled as subsets.

Relational Interpretation: J

Since we have already applied the completion process the relational interpretation is present in the model.

$$(\sigma, R_\sigma) \mapsto R_\sigma$$



Outline

- 1 Context
 - Parametricity
 - Fixed Points
- 2 Concrete Model
 - Stage I: A DILL-model
 - Stage II: A PILL-model
 - Stage III: A Parametric LAPL-structure
- 3 Future Work
 - LILY
 - Expanding the Calculus

PILL_γ with an operational semantics

Pitts and coworkers have defined an operational semantics for (a language equivalent to) PILL_γ.

PILL_γ with an operational semantics

Pitts and coworkers have defined an operational semantics for (a language equivalent to) PILL_γ.

Thus one might write an interpreter (or even a compiler) and run PILL_γ programs.



PILL_γ with an operational semantics

Pitts and coworkers have defined an operational semantics for (a language equivalent to) PILL_γ.

Thus one might write an interpreter (or even a compiler) and run PILL_γ programs.

This allows us to equate terms based on their operational behavior rather than provability in the logic (as we did previously).



Ongoing work

Building an LAPL-structure from equivalence classes of LILY-terms would prove the consequences of parametricity for LILY.

This work is by now nearly done.



Outline

- 1 Context
 - Parametricity
 - Fixed Points
- 2 Concrete Model
 - Stage I: A DILL-model
 - Stage II: A PILL-model
 - Stage III: A Parametric LAPL-structure
- 3 Future Work
 - LILY
 - Expanding the Calculus

More effects

$PILL_{\gamma}$ can be viewed as a linear calculus with intuitionism as an added effect. It could be interesting to consider other possible effects.

- Continuations
- References
- Concurrency?

Summary

A concrete LAPL-structure has been presented. Through this we have established:

- The LAPL logic is consistent.
- The folklore result has been proved.

Acknowledgments

I would like to thank a number of people today:

Lars Birkedal For being my supervisor and coauthor.

Rasmus Ejlers Møgelberg For co-authoring and many helpful discussions.

Jacob Thamsborg Without whom it is unlikely that I had ended up here. . .

The Theory Department For a nice working environment.

The audience For patience and etiquette (I hope).

Censor For reading through the entire thing.

All the forgotten . . .