Progress & Preservation Considered Boring!
A Paean to Parametricity

Derek Dreyer

Max Planck Institute for Software Systems (MPI-SWS)
Kaiserslautern and Saarbrücken, Germany

PLMW 2014
San Diego

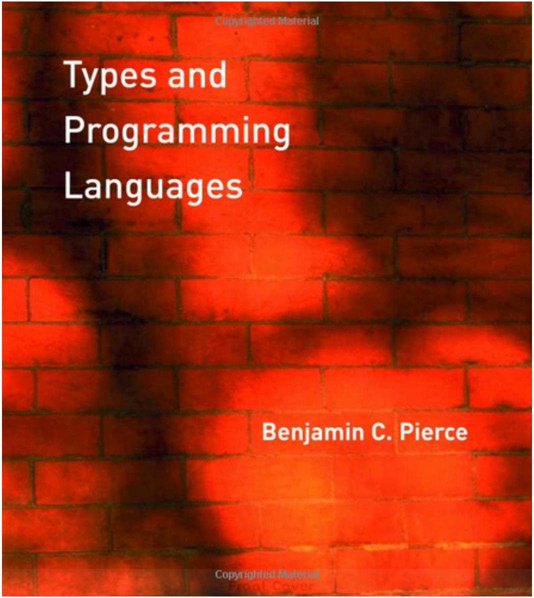
Parametricity at POPL 2014

≥ 5 talks this week related to parametricity and logical relations:

- Birkedal: **Modular reasoning about concurrent higher-order imperative programs**
- Brookes, O'Hearn, Reddy: **The Essence of Reynolds**
- Atkey: **From parametricity to conservation laws, via Noether's theorem**
- Atkey, Ghani, Johann: **A relationally parametric model of dependent type theory**
- Benton, Hofmann, Nigam: **Abstract effects and proof-relevant logical relations**

Parametricity: who needs it?

What are type systems good for?



What are type systems good for?

(1) Detecting a certain class of runtime errors

- *e.g.*, cannot apply an integer as if it were a function
- "Well-typed programs don't get stuck"

This is what syntactic type safety is all about.

Progress: If $e : A$, then $e \leadsto e'$ or e is a value.
Preservation: If $e : A$ and $e \leadsto e'$, then $e' : A$.

What are type systems good for?

(2) Data abstraction: modules, ADTs, classes, etc.

- Enforcing invariants on a module's private data structures
- Representation independence: should be able to change private data representation without affecting clients

Together, these properties are often called abstraction safety.

Points of this talk

1. Type safety does not imply abstraction safety!

2. Parametricity = Type safety + Abstraction safety

3. Logical relations
   = How we formally reason about parametricity

Why do we teach our students progress & preservation rather than parametricity?

Until recently, parametricity was not developed enough to be able to account for ML-like languages, whereas P&P scales easily. . .

- . . . but this is no longer the case.

Parametricity is often presented using “scary” denotational semantics:

- It’s not necessary; one can build logical relations directly over operational semantics

Why do we teach our students progress & preservation rather than parametricity?

Until recently, parametricity was not developed

So there are no more excuses!

- It's not necessary; one can build logical relations directly over operational semantics

A simple motivating example

A simple motivating example: Enumeration types

Interface:

$$\mathsf{COLOR} \;=\; \exists\alpha.\ \{\ \mathsf{red} : \alpha,\ \ \mathsf{blue} : \alpha,\ \ \mathsf{print} : \alpha \to \mathsf{String}\ \}$$

Intended behavior:

$$\begin{aligned} \mathsf{print}\ \mathsf{red} &\ \leadsto\ \texttt{"red"} \\ \mathsf{print}\ \mathsf{blue} &\ \leadsto\ \texttt{"blue"} \end{aligned}$$

A simple motivating example: Enumeration types

One implementation, with $\alpha = \mathsf{Nat}$:

$$
\begin{aligned}
\mathsf{ColorNat} = \mathsf{pack}\ \mathsf{Nat}, \{ & \\
& \mathsf{red} = 0, \\
& \mathsf{blue} = 1, \\
& \mathsf{print} = \lambda x.\ \mathsf{match}\ x\ \mathsf{with} \\
& \qquad\qquad\qquad\quad 0 \Rightarrow \texttt{"red"} \\
& \qquad\qquad\qquad\ \mid 1 \Rightarrow \texttt{"blue"} \\
& \qquad\qquad\qquad\ \mid \_ \Rightarrow \texttt{"FAIL"} \\
\}\ \mathsf{as}\ \mathsf{COLOR} &
\end{aligned}
$$

A simple motivating example: Enumeration types

One implementation, with $\alpha = \mathsf{Nat}$:

$$
\begin{aligned}
\mathsf{ColorNat} = \mathsf{pack}\ \mathsf{Nat}, \{\ & \\
& \mathsf{red} = 0, \\
& \mathsf{blue} = 1, \\
& \mathsf{print} = \lambda x.\ \mathsf{match}\ x\ \mathsf{with} \\
& \qquad\qquad\quad 0 \Rightarrow \texttt{"red"} \\
& \qquad\qquad\ \mid 1 \Rightarrow \texttt{"blue"} \\
& \qquad\qquad\ \mid \_ \Rightarrow \boxed{\texttt{"FAIL"}} \\
\}\ \mathsf{as}\ \mathsf{COLOR} &
\end{aligned}
$$

A simple motivating example: Enumeration types

One implementation, with $\alpha = \text{Nat}$:

Goal #1: Enforcing Invariants

Prove that argument to print must be 0 or 1, and thus it will never return "FAIL".

$\mid$ _ $\Rightarrow$ $\boxed{\texttt{"FAIL"}}$

} as COLOR

A simple motivating example: Enumeration types

Another implementation, with $\alpha = \mathsf{Bool}$:

$$
\begin{aligned}
\mathsf{ColorBool} = \mathsf{pack}\ \mathsf{Bool}, \{\ & \\
& \mathsf{red} = \mathsf{true}, \\
& \mathsf{blue} = \mathsf{false}, \\
& \mathsf{print} = \lambda x.\ \mathsf{match}\ x\ \mathsf{with} \\
& \qquad\qquad\qquad\quad \mathsf{true} \Rightarrow \texttt{"red"} \\
& \qquad\qquad\qquad\ \mid \mathsf{false} \Rightarrow \texttt{"blue"} \\
\}\ \mathsf{as}\ \mathsf{COLOR} &
\end{aligned}
$$

Another implementation, with $\alpha = \text{Bool}$:

Goal #2: Representation Independence

Prove that the two implementations of Color are **contextually equivalent**.

$\}$ as COLOR

Representation independence subsumes invariants

If we can prove

$$\text{ColorNat} \equiv_{\text{ctx}} \text{ColorBool} : \text{COLOR},$$

then since ColorBool's print function never returns `"FAIL"`,
that means ColorNat's print function never returns `"FAIL"`.

More generally, Goal #2 subsumes Goal #1.

The trouble with type safety

A dangerous language extension: Testing for zero!

Suppose our language had the following operator:

$$\mathsf{eqZero} \;:\; \forall\alpha.\ \alpha \to \mathsf{Bool}$$

with the semantics:

$$\mathsf{eqZero}\ \mathsf{v} \;\leadsto\; \begin{cases} \mathsf{true} & \text{if } \mathsf{v} = 0 \\ \mathsf{false} & \text{otherwise} \end{cases}$$

A dangerous language extension: Testing for zero!

Suppose our language had the following operator:

$$\mathsf{eqZero} \quad : \quad \forall\alpha.\ \alpha \to \mathsf{Bool}$$

with the semantics:

$$\mathsf{eqZero}\ \mathsf{v} \ \leadsto \ \begin{cases} \mathsf{true} & \text{if } \mathsf{v} = 0 \\ \mathsf{false} & \text{otherwise} \end{cases}$$

Observation:

- eqZero IS type-safe

A dangerous language extension: Testing for zero!

Suppose our language had the following operator:

$$\mathsf{eqZero} \quad : \quad \forall \alpha.\ \alpha \to \mathsf{Bool}$$

with the semantics:

$$\mathsf{eqZero}\ \mathsf{v} \ \rightsquigarrow \ \begin{cases} \mathsf{true} & \text{if } \mathsf{v} = 0 \\ \mathsf{false} & \text{otherwise} \end{cases}$$

Observation:

- eqZero IS type-safe but NOT abstraction-safe!

Consider a client that simply applies eqZero to red:

unpack ??????? as $[\alpha$,{red,blue,print}] in

eqZero red

eqZero breaks representation independence!

Consider a client that simply applies eqZero to red:

unpack ColorNat as $[\alpha, \{$red,blue,print$\}]$ in

eqZero red

eqZero breaks representation independence!

Consider a client that simply applies eqZero to red:

eqZero 0

eqZero breaks representation independence!

Consider a client that simply applies eqZero to red:

true

eqZero breaks representation independence!

Consider a client that simply applies eqZero to red:

unpack ColorBool as $[\alpha, \{$red, blue, print$\}]$ in

eqZero red

eqZero breaks representation independence!

Consider a client that simply applies eqZero to red:

eqZero true

eqZero breaks representation independence!

Consider a client that simply applies eqZero to red:

false

eqZero breaks representation independence!

Bottom Line

Type safety does not guarantee abstraction safety.

Logical relations to the rescue!

Logical relations and representation independence

We say e_1 and e_2 are logically related at $\exists\alpha.A$ (written $e_1 \approx e_2 : \exists\alpha.A$) if:

- There exists a **"simulation relation"** R between their private representations of α that is preserved by their operations (of type A)
- Intuition: $(v_1, v_2) \in R$ means that v_1 and v_2 are two different representations of the same "abstract value"

Logical relations and representation independence

We say e_1 and e_2 are logically related at $\exists\alpha.A$ (written $e_1 \approx e_2 : \exists\alpha.A$) if:

- There exists a **“simulation relation”** R between their private representations of α that is preserved by their operations (of type A)
- Intuition: $(v_1, v_2) \in R$ means that v_1 and v_2 are two different representations of the same “abstract value”

Theorem (Representation Independence)

If $\vdash e_1 \approx e_2 : A$, then $\vdash e_1 \equiv_{\mathrm{ctx}} e_2 : A$.

Proof that ColorNat and ColorBool are logically related

Returning to our motivating example, let's show:

$$\vdash \mathsf{ColorNat} \approx \mathsf{ColorBool} : \mathsf{COLOR}$$

Proof that ColorNat and ColorBool are logically related

$$\vdash \boxed{\begin{array}{l} \text{pack Nat}, \{ \\ \quad \text{red} = 0, \\ \quad \text{blue} = 1, \\ \quad \text{print} = \lambda x.\ \ldots \\ \} \text{ as COLOR} \end{array}} \approx \boxed{\begin{array}{l} \text{pack Bool}, \{ \\ \quad \text{red} = \text{true}, \\ \quad \text{blue} = \text{false}, \\ \quad \text{print} = \lambda x.\ \ldots \\ \} \text{ as COLOR} \end{array}}$$

$$: \boxed{\begin{array}{l} \exists \alpha.\ \{\ \text{red} : \alpha, \\ \qquad\quad \text{blue} : \alpha, \\ \qquad\quad \text{print} : \alpha \to \text{String}\ \} \end{array}}$$

Pick $R = \{(0,\mathsf{true}),(1,\mathsf{false})\}$
as our simulation relation for α.

$$\alpha \mapsto R \vdash \boxed{\begin{array}{l} \{ \\ \quad \mathsf{red} = 0, \\ \quad \mathsf{blue} = 1, \\ \quad \mathsf{print} = \lambda x.\ \ldots \\ \} \end{array}} \approx \boxed{\begin{array}{l} \{ \\ \quad \mathsf{red} = \mathsf{true}, \\ \quad \mathsf{blue} = \mathsf{false}, \\ \quad \mathsf{print} = \lambda x.\ \ldots \\ \} \end{array}}$$

$$: \boxed{\begin{array}{l} \{\ \mathsf{red} : \alpha, \\ \quad \mathsf{blue} : \alpha, \\ \quad \mathsf{print} : \alpha \to \mathsf{String}\ \} \end{array}}$$

Proof that ColorNat and ColorBool are logically related

Pick $R = \{(0,\text{true}),(1,\text{false})\}$
as our simulation relation for α.

$$\alpha \mapsto R \vdash \boxed{\begin{array}{l} \{ \\ \quad \text{red} = 0, \\ \quad \text{blue} = 1, \\ \quad \text{print} = \lambda x.\ \ldots \\ \} \end{array}} \approx \boxed{\begin{array}{l} \{ \\ \quad \text{red} = \text{true}, \\ \quad \text{blue} = \text{false}, \\ \quad \text{print} = \lambda x.\ \ldots \\ \} \end{array}}$$

$$: \boxed{\begin{array}{l} \{\, \text{red} : \alpha, \\ \quad \text{blue} : \alpha, \\ \quad \text{print} : \alpha \to \text{String} \,\} \end{array}}$$

Proof that ColorNat and ColorBool are logically related

Pick $R = \{(0,\mathsf{true}),(1,\mathsf{false})\}$
as our simulation relation for α.

$$\alpha \mapsto R \vdash \boxed{\begin{array}{l} \{ \\ \quad \mathsf{red} = 0, \\ \quad \mathsf{blue} = 1, \\ \quad \mathsf{print} = \lambda x.\ \ldots \\ \} \end{array}} \approx \boxed{\begin{array}{l} \{ \\ \quad \mathsf{red} = \mathsf{true}, \\ \quad \mathsf{blue} = \mathsf{false}, \\ \quad \mathsf{print} = \lambda x.\ \ldots \\ \} \end{array}}$$

$$: \boxed{\begin{array}{l} \{\ \mathsf{red} : \alpha, \\ \ \ \mathsf{blue} : \alpha, \\ \ \ \mathsf{print} : \alpha \to \mathsf{String}\ \} \end{array}}$$

Proof that ColorNat and ColorBool are logically related

Pick $R = \{(0,\text{true}),(1,\text{false})\}$
as our simulation relation for α.

$$\alpha \mapsto R \vdash \boxed{\begin{array}{l} \{ \\ \quad \text{red} = 0, \\ \quad \text{blue} = 1, \\ \quad \text{print} = \lambda x. \ \ldots \\ \} \end{array}} \approx \boxed{\begin{array}{l} \{ \\ \quad \text{red} = \text{true}, \\ \quad \text{blue} = \text{false}, \\ \quad \text{print} = \lambda x. \ \ldots \\ \} \end{array}}$$

$$: \boxed{\begin{array}{l} \{ \ \text{red} : \alpha, \\ \quad \text{blue} : \alpha, \\ \quad \text{print} : \alpha \to \text{String} \ \} \end{array}}$$

Pick $R = \{(0,\mathsf{true}),(1,\mathsf{false})\}$
as our simulation relation for α.

$$\alpha \mapsto R \vdash \boxed{\begin{array}{l} \lambda x.\ \mathsf{match}\ x\ \mathsf{with} \\ \quad\quad\ \ 0 \Rightarrow \texttt{"red"} \\ \quad\quad \mid 1 \Rightarrow \texttt{"blue"} \\ \quad\quad \mid \_ \Rightarrow \texttt{"FAIL"} \end{array}} \approx \boxed{\begin{array}{l} \lambda x.\ \mathsf{match}\ x\ \mathsf{with} \\ \quad\quad\ \ \mathsf{true} \Rightarrow \texttt{"red"} \\ \quad\quad \mid \mathsf{false} \Rightarrow \texttt{"blue"} \\ \end{array}}$$

$$: \quad \alpha \to \mathsf{String}$$

Proof that ColorNat and ColorBool are logically related

Pick $R = \{(0,\text{true}),(1,\text{false})\}$
as our simulation relation for α.

Suppose $\alpha \mapsto R \vdash \mathsf{v}_1 \approx \mathsf{v}_2 : \alpha$.

$$\alpha \mapsto R \vdash \boxed{\begin{array}{l}\text{match } \mathsf{v}_1 \text{ with} \\ \quad 0 \Rightarrow \texttt{"red"} \\ \mid 1 \Rightarrow \texttt{"blue"} \\ \mid \_ \Rightarrow \texttt{"FAIL"}\end{array}} \approx \boxed{\begin{array}{l}\text{match } \mathsf{v}_2 \text{ with} \\ \quad \text{true} \Rightarrow \texttt{"red"} \\ \mid \text{false} \Rightarrow \texttt{"blue"} \\ \end{array}} \quad : \quad \text{String}$$

Proof that ColorNat and ColorBool are logically related

Pick $R = \{(0,\mathsf{true}),(1,\mathsf{false})\}$
as our simulation relation for α.

Suppose $(\mathsf{v}_1, \mathsf{v}_2) \in R$.

$$\alpha \mapsto R \vdash \boxed{\begin{array}{l}\mathsf{match}\ \mathsf{v}_1\ \mathsf{with}\\ \quad 0 \Rightarrow \texttt{"red"}\\ \mid\ 1 \Rightarrow \texttt{"blue"}\\ \mid\ \_ \Rightarrow \texttt{"FAIL"}\end{array}} \approx \boxed{\begin{array}{l}\mathsf{match}\ \mathsf{v}_2\ \mathsf{with}\\ \quad \mathsf{true} \Rightarrow \texttt{"red"}\\ \mid\ \mathsf{false} \Rightarrow \texttt{"blue"}\\ \\ \end{array}}$$

$$: \quad \mathsf{String}$$

Proof that ColorNat and ColorBool are logically related

Pick $R = \{(0,\text{true}),(1,\text{false})\}$
as our simulation relation for α.

Case: $\mathsf{v}_1 = 0$ and $\mathsf{v}_2 = \text{true}$.

$$\alpha \mapsto R \vdash \boxed{\begin{array}{l}\text{match } 0 \text{ with}\\ \quad 0 \Rightarrow \texttt{"red"}\\ \mid 1 \Rightarrow \texttt{"blue"}\\ \mid \_ \Rightarrow \texttt{"FAIL"}\end{array}} \approx \boxed{\begin{array}{l}\text{match true with}\\ \quad \text{true} \Rightarrow \texttt{"red"}\\ \mid \text{false} \Rightarrow \texttt{"blue"}\\ \\ \end{array}}$$

$$: \quad \text{String}$$

Proof that ColorNat and ColorBool are logically related

Pick $R = \{(0,\mathsf{true}),(1,\mathsf{false})\}$
as our simulation relation for α.

Case: $\mathsf{v}_1 = 1$ and $\mathsf{v}_2 = \mathsf{false}$.

$$\alpha \mapsto R \vdash \boxed{\begin{array}{l} \text{match } 1 \text{ with} \\ \quad 0 \Rightarrow \texttt{"red"} \\ \mid\ 1 \Rightarrow \texttt{"blue"} \\ \mid\ \_ \Rightarrow \texttt{"FAIL"} \end{array}} \approx \boxed{\begin{array}{l} \text{match false with} \\ \quad \text{true} \Rightarrow \texttt{"red"} \\ \mid\ \text{false} \Rightarrow \texttt{"blue"} \\ \end{array}}$$

$$: \quad \text{String}$$

Proof that ColorNat and ColorBool are logically related

Pick $R = \{(0,\text{true}),(1,\text{false})\}$
as our simulation relation for α.

QED!

OK, that was pretty trivial, let's not get too excited. . .

: String

The flip side: Client-side abstraction

In order for representation independence to work, clients must behave “parametrically”.

- We must rule out non-parametric functions like eqZero.

The flip side: Client-side abstraction

In order for representation independence to work, clients must behave "parametrically".

- We must rule out non-parametric functions like eqZero.

Theorem (Abstraction)

If $\vdash e : A$, then $\vdash e \approx e : A$.

This theorem looks weirdly trivial, but it is not!

- The logical relation only relates "well-behaved" terms, *i.e.*, terms that are parametric and don't get stuck.
- Type safety falls out as an easy corollary.

Proof that eqZero is not well-typed

Suppose $\vdash \mathsf{f} : \forall\alpha.\ \alpha \to \mathsf{Bool}$

Proof that eqZero is not well-typed

$$\vdash \ \mathsf{f} \ : \ \forall\alpha.\ \alpha \to \mathsf{Bool}$$

$$\vdash \ \mathsf{f} \ \approx \ \mathsf{f} \ : \ \forall\alpha.\ \alpha \to \mathsf{Bool}$$

Proof that eqZero is not well-typed

Pick $R = \mathsf{Val} \times \mathsf{Val}$
as our simulation relation for α.

$$
\begin{array}{rl}
 & \vdash\ \mathsf{f}\ :\ \forall\alpha.\ \alpha \to \mathsf{Bool} \\
 & \vdash\ \mathsf{f}\ \approx\ \mathsf{f}\ :\ \forall\alpha.\ \alpha \to \mathsf{Bool} \\
\alpha \mapsto R & \vdash\ \mathsf{f}\ \approx\ \mathsf{f}\ :\ \alpha \to \mathsf{Bool}
\end{array}
$$

Proof that eqZero is not well-typed

Pick $R = \mathsf{Val} \times \mathsf{Val}$

as our simulation relation for α.

$$
\begin{array}{rcl}
 & \vdash & \mathsf{f} \;:\; \forall\alpha.\, \alpha \to \mathsf{Bool} \\
 & \vdash & \mathsf{f} \;\approx\; \mathsf{f} \;:\; \forall\alpha.\, \alpha \to \mathsf{Bool} \\
\alpha \mapsto R & \vdash & \mathsf{f} \;\approx\; \mathsf{f} \;:\; \alpha \to \mathsf{Bool} \\
\forall \mathsf{v}_1, \mathsf{v}_2.\;\; \alpha \mapsto R & \vdash & \mathsf{f}(\mathsf{v}_1) \;\approx\; \mathsf{f}(\mathsf{v}_2) \;:\; \mathsf{Bool}
\end{array}
$$

Proof that eqZero is not well-typed

Pick $R = \mathsf{Val} \times \mathsf{Val}$
as our simulation relation for α.

$$\begin{array}{rcl} & \vdash & \mathsf{f} : \forall\alpha.\ \alpha \to \mathsf{Bool} \\ & \vdash & \mathsf{f} \approx \mathsf{f} : \forall\alpha.\ \alpha \to \mathsf{Bool} \\ \alpha \mapsto R & \vdash & \mathsf{f} \approx \mathsf{f} : \alpha \to \mathsf{Bool} \\ \forall \mathsf{v}_1, \mathsf{v}_2.\ \ \alpha \mapsto R & \vdash & \mathsf{f}(\mathsf{v}_1) \approx \mathsf{f}(\mathsf{v}_2) : \mathsf{Bool} \end{array}$$

So f is a constant function, and cannot be eqZero!

Proof that eqZero is not well-typed

Pick $R = \mathsf{Val} \times \mathsf{Val}$
as our simulation relation for α.

This is an example of a
free theorem (Wadler, 1989).

So f is a constant function, and cannot be eqZero!

Summary

Theorem (Representation Independence)

If $\vdash e_1 \approx e_2 : A$, then $\vdash e_1 \equiv_{\mathrm{ctx}} e_2 : A$.

Theorem (Abstraction)

If $\vdash e : A$, then $\vdash e \approx e : A$.

> "Type structure is a syntactic discipline for enforcing levels of abstraction."
>
> – John Reynolds

Classic papers on parametricity

Reynolds (1983):

- **Types, abstraction and parametric polymorphism**
- Introduces parametricity and the abstraction theorem: one of the most important papers in PL history

Mitchell (1986):

- **Representation independence and data abstraction**
- Applies parametricity in order to prove representation independence for existential types

Wadler (1989):

- **Theorems for free!**
- Applies parametricity in order to prove many interesting "free theorems" about universal types

Research on parametricity (a very rough picture)

Going beyond System F

- Expanding the theory of parametricity to encompass more sophisticated and/or realistic language features

Universalism

- Exploring properties that hold of **all** terms of a certain (usually universal) type, cf. Wadler's free theorems
- Do these theorems still hold in languages with effects?
- What interesting free theorems do “sexy” types have?

Existentialism

- Exploring the theory of representation independence in languages with state, continuations, concurrency, etc.
- Applications to verification (*e.g.*, certified compilers)

Recommended universalist reading

Kennedy (1997):

- **Relational parametricity and units of measure**
- Presents types for units of measure (now in F♯), and explains their benefits in terms of free theorems

Johann, Voigtländer (2004):

- **Free theorems in the presence of seq**
- Shows that free theorems are not so free, even in a pure language like Haskell, due to the strictness operator *seq*

Atkey (2012):

- **Relational parametricity for higher kinds**
- Extends parametricity to higher kinds using "reflexive graphs", but without explicit category theory

Recommended existentialist reading (if you tire of Camus)

Pitts, Stark (1998):

- **Operational reasoning for functions with local state**
- Presents "Kripke logical relation" for representation independence in simplified ML-like language

Appel, McAllester (2001):

- **An indexed model of recursive types for foundational proof-carrying code**
- Proposes the "step-indexed" logical-relations model, now an essential tool in scaling parametricity to real languages

Ahmed, Dreyer, Rossberg (2009):

- **State-dependent representation independence**
- First paper to scale parametricity & rep. ind. to a full-blown ML-like language (μ, $\forall$, $\exists$, higher-order state)

A little advice...

Don't be afraid of working on an "**old, hard**" problem!

The problem may not be as hard as it seems

- Just because famous researchers X, Y and Z couldn't solve it doesn't mean **you** can't!
- It might not require superhuman technical abilities to make progress, just a fresh perspective and the "right" set of abstractions.

It can be a gold mine

- Deep problems lead to other deep problems, thus guaranteeing you won't run out of things to work on.
- *e.g.*, I would never have guessed when we wrote our POPL'09 paper that our ideas would be relevant to verifying lock-free concurrent data structures, or compiler correctness, or security, or...

Many of the world's experts on parametricity are here. Talk to them!

Here's a starting point:

http://www.mpi-sws.org/~dreyer/parametric