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
≥ 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?

Types and Programming Languages

Benjamin C. Pierce
(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 $e \rightsquigarrow e'$ or $e$ is a value.

**Preservation:** If $e : A$ and $e \rightsquigarrow e'$, 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.

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

So there are no more excuses!
A simple motivating example
Interface:

\[ \text{COLOR} = \exists \alpha. \{ \text{red} : \alpha, \text{blue} : \alpha, \text{print} : \alpha \rightarrow \text{String} \} \]

Intended behavior:

\[ \begin{align*}
\text{print red} & \sim \ "\text{red}" \\
\text{print blue} & \sim \ "\text{blue}" \\
\end{align*} \]
One implementation, with $\alpha = \text{Nat}$:

\[
\text{ColorNat} = \text{pack Nat}, \{ \\
\text{red} = 0, \\
\text{blue} = 1, \\
\text{print} = \lambda x. \text{match } x \text{ with} \\
\quad 0 \Rightarrow \text{"red"} \\
\quad | 1 \Rightarrow \text{"blue"} \\
\quad | \_ \Rightarrow \text{"FAIL"}
\} \text{ as COLOR}
\]
A simple motivating example: Enumeration types

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

\[
\begin{align*}
\text{ColorNat} &= \text{pack Nat, } \\
\text{red} &= 0, \\
\text{blue} &= 1, \\
\text{print} &= \lambda x. \text{match } x \text{ with } \\
&\quad |0 \Rightarrow \text{"red"} \\
&\quad |1 \Rightarrow \text{"blue"} \\
&\quad |_\_ \Rightarrow \text{"FAIL"}
\end{align*}
\]

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

```ml
ColorNat = pack Nat,
{ red = 0,
  blue = 1,
  print = \_ \rightarrow match \_ with |
  | 0 \rightarrow "red"
  | 1 \rightarrow "blue"
  | _ \rightarrow "FAIL"
} as COLOR
```

**Goal #1: Enforcing Invariants**

Prove that argument to print must be 0 or 1, and thus it will never return "FAIL".
Another implementation, with $\alpha = \text{Bool}$:

```
ColorBool = pack Bool, {
    red = true,
    blue = false,
    print = \lambda x. match x with
        | true => "red"
        | false => "blue"
} as COLOR
```
Another implementation, with $\alpha = \text{Bool}$:

```
Goal #2: Representation Independence

Prove that the two implementations of Color are contextually equivalent.

} as COLOR
```
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
Suppose our language had the following operator:

\[ \text{eqZero} : \forall \alpha. \alpha \rightarrow \text{Bool} \]

with the semantics:

\[ \text{eqZero } v \sim \begin{cases} 
\text{true} & \text{if } v = 0 \\
\text{false} & \text{otherwise}
\end{cases} \]
A dangerous language extension: Testing for zero!

Suppose our language had the following operator:

\[
\text{eqZero} : \forall \alpha. \alpha \rightarrow \text{Bool}
\]

with the semantics:

\[
\text{eqZero } v \sim \begin{cases} 
\text{true} & \text{if } v = 0 \\
\text{false} & \text{otherwise}
\end{cases}
\]

Observation:

- **eqZero** IS type-safe
Suppose our language had the following operator:

\[ \text{eqZero} : \forall \alpha. \; \alpha \to \text{Bool} \]

with the semantics:

\[ \text{eqZero} \; v \sim \begin{cases} 
\text{true} & \text{if } v = 0 \\
\text{false} & \text{otherwise} 
\end{cases} \]

**Observation:**

\[ \text{eqZero IS type-safe but NOT abstraction-safe!} \]
Consider a client that simply applies eqZero to red:

```
unpack ???????? as [\alpha,\{red,blue,print\}] in eqZero red
```
Consider a client that simply applies \texttt{eqZero} to \texttt{red}:

```plaintext
unpack \texttt{ColorNat} as [\alpha, \{\texttt{red}, \texttt{blue}, \texttt{print}\}] in \texttt{eqZero red}
```
Consider a client that simply applies eqZero to \texttt{red}:

\texttt{eqZero 0}
Consider a client that simply applies eqZero to \texttt{red}:

\begin{verbatim}
true
\end{verbatim}
Consider a client that simply applies eqZero to red:

unpack ColorBool as \([\alpha, \{\text{red, blue, print}\}]\) in

eqZero \text{ red}
Consider a client that simply applies eqZero to red:

\[\text{eqZero true}\]
Consider a client that simply applies `eqZero` to `red`:

```python
false
```
eqZero breaks representation independence!

Consider a client that simply applies eqZero to red:

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

Bottom Line

Type safety does not guarantee abstraction safety.
Logical relations to the rescue!
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 $\alpha$ 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"
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 \( \alpha \) 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 \( \vdash e_1 \equiv_{\text{ctx}} e_2 : A \).
Returning to our motivating example, let’s show:

\[ \vdash \text{ColorNat} \approx \text{ColorBool} : \text{COLOR} \]
Proof that ColorNat and ColorBool are logically related

\[ \vdash \text{pack Nat, } \{ \text{red} = 0, \text{blue} = 1, \text{print} = \lambda x. \ldots \} \text{ as COLOR} \]

\[ \approx \text{pack Bool, } \{ \text{red} = \text{true}, \text{blue} = \text{false}, \text{print} = \lambda x. \ldots \} \text{ as COLOR} \]

\[ \exists \alpha. \{ \text{red} : \alpha, \text{blue} : \alpha, \text{print} : \alpha \to \text{String} \} \]
Proof that ColorNat and ColorBool are logically related

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

\[
\begin{align*}
\alpha & \mapsto R \\
\left\{ \begin{array}{l}
\text{red} = 0, \\
\text{blue} = 1, \\
\text{print} = \lambda x. \ldots
\end{array} \right. & \approx \\
\left\{ \begin{array}{l}
\text{red} = \text{true}, \\
\text{blue} = \text{false}, \\
\text{print} = \lambda x. \ldots
\end{array} \right.
\end{align*}
\]

\[
\left\{ \begin{array}{l}
\text{red} : \alpha, \\
\text{blue} : \alpha,
\end{array} \right. \\
\text{print} : \alpha \to \text{String} \right\}
\]
Proof that ColorNat and ColorBool are logically related

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

\[
\begin{align*}
\alpha \mapsto R \vdash & \begin{cases}
\text{red} = 0, \\
\text{blue} = 1, \\
\text{print} = \lambda x. \ldots
\end{cases} & \approx & \begin{cases}
\text{red} = \text{true}, \\
\text{blue} = \text{false}, \\
\text{print} = \lambda x. \ldots
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\{ & \text{red} : \alpha, \\
& \text{blue} : \alpha, \\
& \text{print} : \alpha \to \text{String} \}
\end{align*}
\]
Proof that ColorNat and ColorBool are logically related

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

\[
\begin{align*}
\alpha \mapsto R \vdash & \\
\{ & \\
\text{red} = 0, & \\
\text{blue} = 1, & \\
\text{print} = \lambda x. \ldots & \\
\} & \\
\end{align*}
\]

\[
\begin{align*}
\approx & \\
\{ & \\
\text{red} = \text{true}, & \\
\text{blue} = \text{false}, & \\
\text{print} = \lambda x. \ldots & \\
\} & \\
\end{align*}
\]

\[
\begin{align*}
\{ & \\
\text{red} : \alpha, & \\
\text{blue} : \alpha, & \\
\text{print} : \alpha \to \text{String} & \\
\} & \\
\end{align*}
\]
Proof that ColorNat and ColorBool are logically related

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

\[
\alpha \mapsto R \mid \quad \begin{cases}
  \text{red} = 0, \\
  \text{blue} = 1, \\
  \text{print} = \lambda x. \ldots
\end{cases}
\approx \begin{cases}
  \text{red} = \text{true}, \\
  \text{blue} = \text{false}, \\
  \text{print} = \lambda x. \ldots
\end{cases}
\]

\[
\begin{cases}
  \text{red} : \alpha, \\
  \text{blue} : \alpha, \\
  \text{print} : \alpha \to \text{String}
\end{cases}
\]
Proof that ColorNat and ColorBool are logically related

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

$\alpha \leftrightarrow R \vdash$

\[
\lambda x. \text{match } x \text{ with } \\
| 0 \Rightarrow "\text{red}" \\
| 1 \Rightarrow "\text{blue}" \\
| \_ \Rightarrow "\text{FAIL}"
\]

$\simeq$

\[
\lambda x. \text{match } x \text{ with } \\
| \text{true} \Rightarrow "\text{red}" \\
| \text{false} \Rightarrow "\text{blue}"
\]

: $\alpha \rightarrow \text{String}$
Proof that ColorNat and ColorBool are logically related

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

Suppose $\alpha \mapsto R \vdash v_1 \approx v_2 : \alpha$.

\[
\begin{array}{c}
\text{match } v_1 \text{ with } \\
0 \Rightarrow \text{"red"} \\
| 1 \Rightarrow \text{"blue"} \\
| \_ \Rightarrow \text{"FAIL"}
\end{array}
\]  
\[\approx\]  
\begin{array}{c}
\text{match } v_2 \text{ with } \\
\text{true} \Rightarrow \text{"red"} \\
| \text{false} \Rightarrow \text{"blue"}
\end{array}
\]  
: String
Proof that ColorNat and ColorBool are logically related

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

Suppose $(v_1, v_2) \in R$.

$$\alpha \mapsto R \vdash$$

match $v_1$ with

| 0  ⇒ "red" | 1  ⇒ "blue" | _  ⇒ "FAIL"

$\approx$

match $v_2$ with

| true  ⇒ "red" | false ⇒ "blue"

:\ String
Proof that ColorNat and ColorBool are logically related

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

Case: $v_1 = 0$ and $v_2 = \text{true}$.

\[
\begin{align*}
\alpha \leftrightarrow R \vdash & \text{match 0 with} \\
& 0 \Rightarrow "\text{red}" \\
& 1 \Rightarrow "\text{blue}" \\
& _{} \Rightarrow "\text{FAIL}"
\end{align*}
\]

\[
\begin{align*}
\text{match true with} \\
& \text{true} \Rightarrow "\text{red}" \\
& \text{false} \Rightarrow "\text{blue}"
\end{align*}
\]

: String
Proof that ColorNat and ColorBool are logically related

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

Case: $v_1 = 1$ and $v_2 = \text{false}$.

$\alpha \mapsto R \vdash$

<table>
<thead>
<tr>
<th>match 1 with</th>
<th>match false with</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $\Rightarrow$ &quot;red&quot;</td>
<td>true $\Rightarrow$ &quot;red&quot;</td>
</tr>
<tr>
<td>1 $\Rightarrow$ &quot;blue&quot;</td>
<td>false $\Rightarrow$ &quot;blue&quot;</td>
</tr>
<tr>
<td>_ $\Rightarrow$ &quot;FAIL&quot;</td>
<td></td>
</tr>
</tbody>
</table>

: String

QED!
Proof that ColorNat and ColorBool are logically related

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

\[ \alpha \mapsto R \vdash \text{match 1 with} \]
\[ |0 \Rightarrow \text{"red"} | 1 \Rightarrow \text{"blue"} | \_ \Rightarrow \text{"FAIL"} \]

\[ \approx \text{match false with} \]
\[ |\text{true} \Rightarrow \text{"red"} | \text{false} \Rightarrow \text{"blue"} \]

QED!

OK, that was pretty trivial, let’s not get too excited…
In order for representation independence to work, clients must behave “parametrically”.

- We must rule out non-parametric functions like eqZero.
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 \( \vdash f : \forall \alpha. \alpha \rightarrow \text{Bool} \)

This is an example of a free theorem (Wadler, 1989).
Proof that eqZero is not well-typed

⊢ \ f \ : \ \forall \alpha. \alpha \to \text{Bool}

⊢ f \approx f : \forall \alpha. \alpha \to \text{Bool}

This is an example of a free theorem (Wadler, 1989).
Proof that eqZero is not well-typed

Pick $R = \text{Val} \times \text{Val}$
as our simulation relation for $\alpha$.

\[
\begin{align*}
\vdash f : \forall \alpha. \alpha \rightarrow \text{Bool} \\
\vdash f \approx f : \forall \alpha. \alpha \rightarrow \text{Bool} \\
\alpha \mapsto R \vdash f \approx f : \alpha \rightarrow \text{Bool}
\end{align*}
\]
Proof that eqZero is not well-typed

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

as our simulation relation for $\alpha$.

\[
\begin{align*}
\vdash f & : \forall \alpha. \alpha \to \text{Bool} \\
\vdash f \approx f & : \forall \alpha. \alpha \to \text{Bool} \\
\alpha \mapsto R & \vdash f \approx f : \alpha \to \text{Bool} \\
\forall v_1, v_2. \alpha \mapsto R & \vdash f(v_1) \approx f(v_2) : \text{Bool}
\end{align*}
\]

This is an example of a free theorem (Wadler, 1989).
Proof that eqZero is not well-typed

Pick $R = \text{Val} \times \text{Val}$ as our simulation relation for $\alpha$.

$$
\vdash f : \forall \alpha. \alpha \rightarrow \text{Bool} \\
\vdash f \approx f : \forall \alpha. \alpha \rightarrow \text{Bool} \\
\alpha \mapsto R \vdash f \approx f : \alpha \rightarrow \text{Bool} \\
\forall v_1, v_2. \quad \alpha \mapsto R \vdash f(v_1) \approx f(v_2) : \text{Bool}
$$

So $f$ is a constant function, and cannot be eqZero!
Proof that eqZero is not well-typed

Pick \( R = \text{Val} \times \text{Val} \)

as our simulation relation for \( \alpha \).

This is an example of a

free theorem (Wadler, 1989).

So \( f \) is a constant function, and cannot be eqZero!
Theorem (Representation Independence)
If $\vdash e_1 \approx e_2 : A$, then $\vdash e_1 \equiv_{ctx} e_2 : A$.

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
Theorem (Representation Independence)

If \( \vdash e_1 \approx e_2 : A \), then \( \vdash e_1 \equiv \text{ctx } e_2 : A \).

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
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
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)
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
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 ($\mu$, $\forall$, $\exists$, 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