Parametricity and Relational Reasoning

Derek Dreyer

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

Oregon PL Summer School
June 2014

Parametricity and local relational reasoning

Relational reasoning:

- Proving properties relating behavior of multiple programs, most importantly equivalence and refinement
- Essential for verifying program transformations/compilers + general verification when there is no clear "logical" spec

Local (or modular) relational reasoning:

- Relational reasoning on modules, preserved by linking
- Canonical example: "contextual" equivalence/refinement

Parametricity:

- A powerful local relational reasoning principle that comes "for free" from the abstractions in high-level languages (types, polymorphism, local state, closures, etc.)

Focus of these lectures

1. Parametricity: who needs it?
2. Formalizing parametricity using operational semantics-based logical relations
   - The basic setup: System F
   - + recursive types (step-indexing)
   - + continuations ($\top\top$-closure aka biorthogonality)
   - + first-order state (Kripke logical relations)
3. A brief tour of recent literature
   - Log. rel. for concurrency, bisimulation techniques, relational separation logics, compiler certification

Focus of these lectures

1. Parametricity: who needs it?

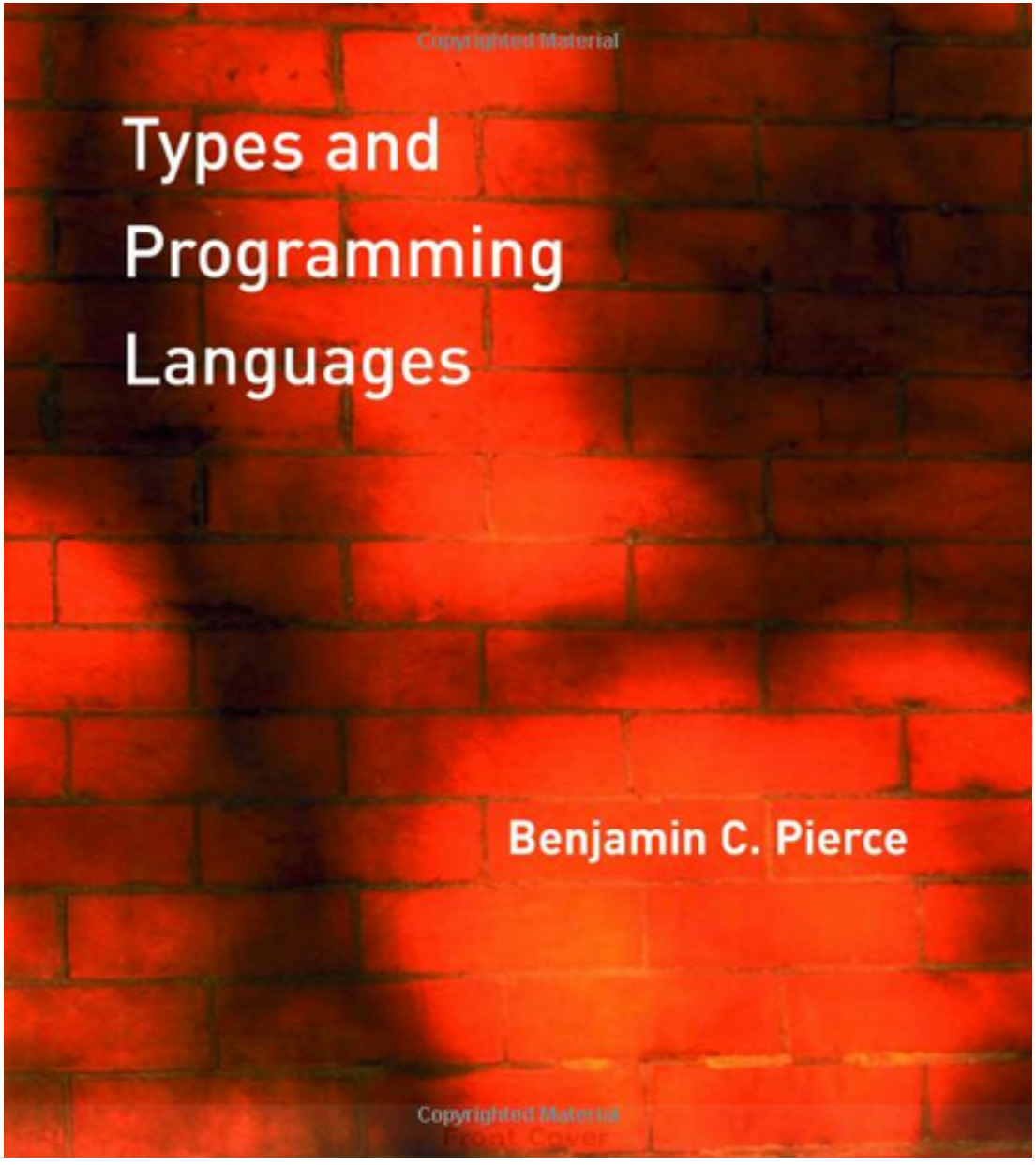
My Strategy for Doing This in 4 Lectures

I explain how to define and use logical relations,
NOT how to prove them sound!

3. A brief tour of recent literature
   - Log. rel. for concurrency, bisimulation techniques, relational separation logics, compiler certification

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 $e \rightsquigarrow e'$ 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.

Key take-away points

1. Type safety does not imply abstraction safety!
2. Parametricity = Type safety + Abstraction safety
3. Parametricity is a relational property.

A simple motivating example

A simple motivating example: Enumeration types

Interface:

$$\begin{aligned} \text{COLOR} \;=\; \exists\alpha.\, \{\; & \text{red} : \alpha, \\ & \text{blue} : \alpha, \\ & \text{print} : \alpha \to \text{String} \;\} \end{aligned}$$

Intended behavior:

$$\begin{aligned} \text{print red} &\rightsquigarrow \texttt{"red"} \\ \text{print blue} &\rightsquigarrow \texttt{"blue"} \end{aligned}$$

A simple motivating example: Enumeration types

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

$$
\begin{array}{l}
\mathsf{ColorInt} = \mathsf{pack}\ \mathsf{Int}, \{ \\
\quad \mathsf{red} = 0, \\
\quad \mathsf{blue} = 1, \\
\quad \mathsf{print} = \lambda x.\ \mathsf{match}\ x\ \mathsf{with} \\
\qquad\qquad 0 \Rightarrow \texttt{"red"} \\
\qquad\quad \mid 1 \Rightarrow \texttt{"blue"} \\
\qquad\quad \mid \_ \Rightarrow \texttt{"FAIL"} \\
\} \ \mathsf{as}\ \mathsf{COLOR}
\end{array}
$$

A simple motivating example: Enumeration types

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

$$
\begin{array}{l}
\mathsf{ColorInt} = \mathsf{pack}\ \mathsf{Int}, \{ \\
\quad \mathsf{red} = 0, \\
\quad \mathsf{blue} = 1, \\
\quad \mathsf{print} = \lambda x.\ \mathsf{match}\ x\ \mathsf{with} \\
\qquad\qquad 0 \Rightarrow \texttt{"red"} \\
\qquad\quad \mid\ 1 \Rightarrow \texttt{"blue"} \\
\qquad\quad \mid\ \_ \Rightarrow \boxed{\texttt{"FAIL"}} \\
\} \ \mathsf{as}\ \mathsf{COLOR}
\end{array}
$$

A simple motivating example: Enumeration types

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

Goal #1: Enforcing Invariants

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

```
  | _ ⇒ "FAIL"

} as COLOR
```

A simple motivating example: Enumeration types

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

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

A simple motivating example: Enumeration types

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

$$\mathsf{ColorInt} \equiv_{\mathrm{ctx}} \mathsf{ColorBool} : \mathsf{COLOR},$$

then since ColorBool's print function never returns `"FAIL"`,
that means ColorInt'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:

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

with the semantics:

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

A dangerous language extension: Testing for zero!

Suppose our language had the following operator:

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

with the semantics:

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

Observation:

- eqZero IS type-safe

A dangerous language extension: Testing for zero!

Suppose our language had the following operator:

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

with the semantics:

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

Observation:

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

eqZero breaks representation independence!

Consider a client that simply applies eqZero to red:

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

eqZero red

eqZero breaks representation independence!

Consider a client that simply applies eqZero to red:

unpack ColorInt as $[\alpha,\{\text{red},\text{blue},\text{print}\}]$ 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 $[\alpha,\{\text{red},\text{blue},\text{print}\}]$ 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 ColorInt and ColorBool are logically related

Returning to our motivating example, let's show:

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

Proof that ColorInt and ColorBool are logically related

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

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

Proof that ColorInt and ColorBool are logically related

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

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

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

Proof that ColorInt and ColorBool are logically related

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

$$\alpha \mapsto R \vdash \{\ \text{red} = 0,\ \text{blue} = 1,\ \text{print} = \lambda x.\ \ldots\ \} \approx \{\ \text{red} = \text{true},\ \text{blue} = \text{false},\ \text{print} = \lambda x.\ \ldots\ \}$$

$$: \{\ \text{red} : \alpha,\ \text{blue} : \alpha,\ \text{print} : \alpha \to \text{String}\ \}$$

Proof that ColorInt and ColorBool are logically related

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

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

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

Proof that ColorInt and ColorBool are logically related

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

$$\alpha \mapsto R \vdash \{\ \text{red} = 0,\ \text{blue} = 1,\ \text{print} = \lambda x.\ \ldots\ \} \approx \{\ \text{red} = \text{true},\ \text{blue} = \text{false},\ \text{print} = \lambda x.\ \ldots\ \}$$

$$: \{\ \text{red} : \alpha,\ \text{blue} : \alpha,\ \text{print} : \alpha \to \text{String}\ \}$$

Proof that ColorInt and ColorBool are logically related

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

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

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

Proof that ColorInt and ColorBool are logically related

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

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

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

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

Proof that ColorInt and ColorBool are logically related

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

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

$\alpha \mapsto R \vdash$

```
match v1 with
  0 ⇒ "red"
| 1 ⇒ "blue"
| _ ⇒ "FAIL"
```

$\approx$

```
match v2 with
  true ⇒ "red"
| false ⇒ "blue"
```

$:$ String

Proof that ColorInt and ColorBool are logically related

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

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

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

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

Proof that ColorInt and ColorBool are logically related

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

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

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

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

Proof that ColorInt 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.

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

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

$$\alpha \mapsto R \ \vdash \ \mathsf{f} \ \approx \ \mathsf{f} \ : \ \alpha \to \mathsf{Bool}$$

Proof that eqZero is not well-typed

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

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

Proof that eqZero is not well-typed

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

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

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 $\vdash e \approx e : A$.

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