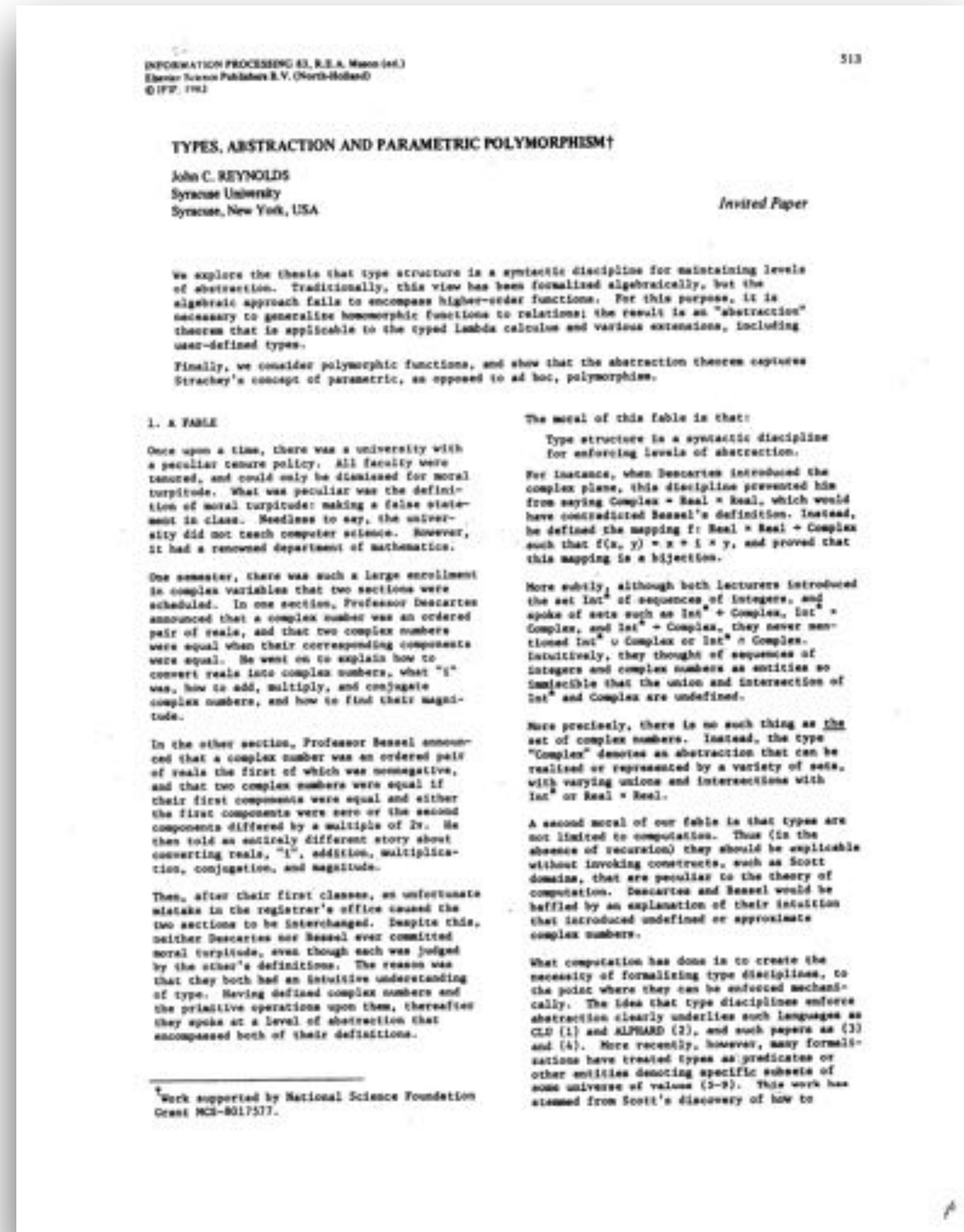
Logical Relations
&
Free Theorems

Tom Schrijvers

Parametricity



INFORMATION PROCESSING 83, R.E.A. Mason (ed.)
Elsevier Science Publishers B.V. (North-Holland)
© IFIP, 1983

513

TYPES, ABSTRACTION AND PARAMETRIC POLYMORPHISM†

John C. REYNOLDS
Syracuse University
Syracuse, New York, USA

Invited Paper

We explore the thesis that type structure is a syntactic discipline for maintaining levels of abstraction. Traditionally, this view has been formalized algebraically, but the algebraic approach fails to encompass higher-order functions. For this purpose, it is necessary to generalize homomorphic functions to relations; the result is an "abstraction" theorem that is applicable to the typed lambda calculus and various extensions, including user-defined types.

Finally, we consider polymorphic functions, and show that the abstraction theorem captures Strachey's concept of parametric, as opposed to ad hoc, polymorphism.

1. A FABLE

Once upon a time, there was a university with a peculiar tenure policy. All faculty were tenured, and could only be dismissed for moral turpitude. What was peculiar was the definition of moral turpitude: making a false statement in class. Needless to say, the university did not teach computer science. However, it had a renowned department of mathematics.

One semester, there was such a large enrollment in complex variables that two sections were scheduled. In one section, Professor Descartes announced that a complex number was an ordered pair of reals, and that two complex numbers were equal when their corresponding components were equal. He went on to explain how to convert reals into complex numbers, what "i" was, how to add, multiply, and conjugate complex numbers, and how to find their magnitude.

In the other section, Professor Bessel announced that a complex number was an ordered pair of reals the first of which was nonnegative, and that two complex numbers were equal if their first components were equal and either the first components were zero or the second components differed by a multiple of 2π. He then told an entirely different story about converting reals, "i", addition, multiplication, conjugation, and magnitude.

Then, after their first classes, an unfortunate mistake in the registrar's office caused the two sections to be interchanged. Despite this, neither Descartes nor Bessel ever committed moral turpitude, even though each was judged by the other's definitions. The reason was that they both had an intuitive understanding of type. Having defined complex numbers and the primitive operations upon them, thereafter they spoke at a level of abstraction that encompassed both of their definitions.

The moral of this fable is that:

> Type structure is a syntactic discipline for enforcing levels of abstraction.

For instance, when Descartes introduced the complex plane, this discipline prevented him from saying Complex = Real × Real, which would have contradicted Bessel's definition. Instead, he defined the mapping f: Real × Real → Complex such that f(x, y) = x + i × y, and proved that this mapping is a bijection.

More subtly, although both lecturers introduced the set Int* of sequences of integers, and spoke of sets such as Int* + Complex, Int* × Complex, and Int* → Complex, they never mentioned Int* ∪ Complex or Int* ∩ Complex. Intuitively, they thought of sequences of integers and complex numbers as entities so immiscible that the union and intersection of Int* and Complex are undefined.

More precisely, there is no such thing as <u>the</u> set of complex numbers. Instead, the type "Complex" denotes an abstraction that can be realized or represented by a variety of sets, with varying unions and intersections with Int* or Real × Real.

A second moral of our fable is that types are not limited to computation. Thus (in the absence of recursion) they should be explicable without invoking constructs, such as Scott domains, that are peculiar to the theory of computation. Descartes and Bessel would be baffled by an explanation of their intuition that introduced undefined or approximate complex numbers.

What computation has done is to create the necessity of formalizing type disciplines, to the point where they can be enforced mechanically. The idea that type disciplines enforce abstraction clearly underlies such languages as CLU (1) and ALPHARD (2), and such papers as (3) and (4). More recently, however, many formalizations have treated types as predicates or other entities denoting specific subsets of some universe of values (5-9). This work has stemmed from Scott's discovery of how to

†Work supported by National Science Foundation Grant MCS-8017577.

Reynolds, J.C. (1983).
"Types, abstraction, and parametric polymorphism".
Information Processing. North Holland, Amsterdam. pp. 513–523.

Logical Relation

$$
\begin{aligned}
\mathcal{V}^{\rho}_{\textsf{int}} &= \{(n, n) \mid n \in \mathbb{Z}\} \\
\mathcal{V}^{\rho}_{\tau \to \tau'} &= \{(\lambda x\!:\!\rho_1(\tau).\, e_1, \lambda x\!:\!\rho_2(\tau).\, e_2) \mid \forall (v_1, v_2) \in \mathcal{V}^{\rho}_{\tau}.\ (e_1\{v_1/x\}, e_2\{v_2/x\}) \in \mathcal{E}^{\rho}_{\tau'}\} \\
\mathcal{V}^{\rho}_{\forall X.\, \tau} &= \{(\Lambda X.\, e_1, \Lambda X.\, e_2) \mid \forall \tau_1, \tau_2, R \in \mathrm{Rel}[\tau_1, \tau_2].\ (e_1\{\tau_1/X\}, e_2\{\tau_2/X\}) \in \mathcal{E}^{\rho[X \mapsto (\tau_1, \tau_2, R)]}_{\tau}\} \\
\mathcal{V}^{\rho}_{X} &= \{(v_1, v_2) \mid \rho(X) = (\tau_1, \tau_2, R) \text{ and } (v_1, v_2) \in R\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}^{\rho}_{\tau} = \{(e_1, e_2) \mid \quad & \vdash e_1\!:\!\rho_1(\tau) \text{ and } \vdash e_2\!:\!\rho_2(\tau) \text{ and} \\
& \exists v_1, v_2.\ e_1 \longrightarrow^{*} v_1 \text{ and } e_2 \longrightarrow^{*} v_2 \text{ and } (v_1, v_2) \in \mathcal{V}^{\rho}_{\tau}\}
\end{aligned}
$$

Logical Relation

$$
\begin{aligned}
\mathcal{V}^{\rho}_{\textsf{int}} &= \{(n, n) \mid n \in \mathbb{Z}\} \\
\mathcal{V}^{\rho}_{\tau \to \tau'} &= \{(\lambda x\!:\!\rho_1(\tau).\, e_1, \lambda x\!:\!\rho_2(\tau).\, e_2) \mid \forall (v_1, v_2) \in \mathcal{V}^{\rho}_{\tau}.\ (e_1\{v_1/x\}, e_2\{v_2/x\}) \in \mathcal{E}^{\rho}_{\tau'}\} \\
\mathcal{V}^{\rho}_{\forall X.\, \tau} &= \{(\Lambda X.\, e_1, \Lambda X.\, e_2) \mid \forall \tau_1, \tau_2, R \in \mathrm{Rel}[\tau_1, \tau_2].\ (e_1\{\tau_1/X\}, e_2\{\tau_2/X\}) \in \mathcal{E}^{\rho[X \mapsto (\tau_1, \tau_2, R)]}_{\tau}\} \\
\mathcal{V}^{\rho}_{X} &= \{(v_1, v_2) \mid \rho(X) = (\tau_1, \tau_2, R) \text{ and } (v_1, v_2) \in R\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}^{\rho}_{\tau} = \{(e_1, e_2) \mid\ & \vdash e_1 : \rho_1(\tau) \text{ and } \vdash e_2 : \rho_2(\tau) \text{ and} \\
& \exists v_1, v_2.\ e_1 \longrightarrow^{*} v_1 \text{ and } e_2 \longrightarrow^{*} v_2 \text{ and } (v_1, v_2) \in \mathcal{V}^{\rho}_{\tau}\}
\end{aligned}
$$

Very Syntax-Directed Formulation!

Logical Relation

$$
\begin{aligned}
\mathcal{V}^{\rho}_{\textsf{int}} &= \{(n, n) \mid n \in \mathbb{Z}\} \\
\mathcal{V}^{\rho}_{\tau \to \tau'} &= \{(\lambda x\!:\!\rho_1(\tau).\, e_1, \lambda x\!:\!\rho_2(\tau).\, e_2) \mid \forall (v_1, v_2) \in \mathcal{V}^{\rho}_{\tau}.\ (e_1\{v_1/x\}, e_2\{v_2/x\}) \in \mathcal{E}^{\rho}_{\tau'}\} \\
\mathcal{V}^{\rho}_{\forall X.\, \tau} &= \{(\Lambda X.\, e_1, \Lambda X.\, e_2) \mid \forall \tau_1, \tau_2, R \in \mathrm{Rel}[\tau_1, \tau_2].\ (e_1\{\tau_1/X\}, e_2\{\tau_2/X\}) \in \mathcal{E}^{\rho[X \mapsto (\tau_1, \tau_2, R)]}_{\tau}\} \\
\mathcal{V}^{\rho}_{X} &= \{(v_1, v_2) \mid \rho(X) = (\tau_1, \tau_2, R) \text{ and } (v_1, v_2) \in R\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}^{\rho}_{\tau} = \{(e_1, e_2) \mid \quad & \vdash e_1 : \rho_1(\tau) \text{ and } \vdash e_2 : \rho_2(\tau) \text{ and} \\
& \exists v_1, v_2.\ e_1 \longrightarrow^* v_1 \text{ and } e_2 \longrightarrow^* v_2 \text{ and } (v_1, v_2) \in \mathcal{V}^{\rho}_{\tau}\}
\end{aligned}
$$

Very Syntax-Directed Formulation!

What does it mean?

Warm-Up

$$\forall \alpha \,.\, \alpha \rightarrow \alpha$$

Examples

```
f : forall a. a → a
```

Examples

```
f : forall a. a → a
```

Example 1:

Examples

```
f : forall a. a → a
```

Example 1:

```
f x = x
```

Examples

```
f : forall a. a → a
```

Example 1:

```
f x = x                  id
```

Examples

```
f : forall a. a → a
```

Example 1:

```
f x = x                id
```

Example 2:

Examples

```
f : forall a. a → a
```

Example 1:

```
f x = x                 id
```

Example 2:

can't think of anything

Logical Relations

```
f : forall a. a → a
```

Logical Relations

```
f : forall a. a → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\quad f_A, f_B \in \mathscr{R} \to \mathscr{R}$$

Definition

$$\mathscr{R} \to \mathscr{S}$$

$$=$$

$$\{f, g \mid \forall x, y \in \mathscr{R} : f\,x, g\,y \in \mathscr{S}\}$$

Logical Relations

```
f : forall a. a → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\quad f_A, f_B \in \mathscr{R} \to \mathscr{R}$$

Logical Relations

```
f : forall a. a → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\quad f_A, f_B \in \mathscr{R} \to \mathscr{R}$$

$$\forall A, B, \mathscr{R} : A \times B \,. \forall x, y \in \mathscr{R} \,. f_A\, x, f_B\, y \in \mathscr{R}$$

Logical Relations

```
f : forall a. a → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \in \mathscr{R} \,.\, f_A\, x, f_B\, y \in \mathscr{R}$$

Logical Relations

```
f : forall a. a → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \in \mathscr{R} \,.\, f_A\, x, f_B\, y \in \mathscr{R}$$

functional relation

$$x, y \in \mathscr{R} \Leftrightarrow h\, x = y$$

$$\forall A, B, h : A \to B \,.\, \forall x : A \,.\, h\,(f_A\, x) = f_B\,(h\, x)$$

Logical Relations

```
f : forall a. a → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \in \mathscr{R} \,.\, f_A\, x, f_B\, y \in \mathscr{R}$$

functional relation $\quad x, y \in \mathscr{R} \Leftrightarrow h\,x = y$

$$\forall A, B, h : A \to B \,.\, \forall x : A \,.\, h\,(f_A\, x) = f_B\,(h\,x)$$

pointfree

$$\forall A, B, h : A \to B \,.\, h \circ f_A = f_B \circ h$$

Application

```
f : forall a. a → a
```

free theorem

$$\forall A, B, h : A \to B \,.\, h \circ f_A = f_B \circ h$$

Application

`f : forall a. a → a`

free theorem

$$\forall A, B, h : A \to B \,.\, h \circ f_A = f_B \circ h$$

$A = B$ $\quad$ $h = \textbf{const}\, x$

$$\forall A, x : A \,.\, \textbf{const}\, x \circ f = f \circ \textbf{const}\, x$$

Application

```
f : forall a. a → a
```

free theorem

$$\forall A, B, h : A \to B \,.\, h \circ f_A = f_B \circ h$$

$A = B$ $\quad$ $h = \textbf{const}\, x$

$$\forall A, x : A \,.\, \textbf{const}\, x \circ f = f \circ \textbf{const}\, x$$

$$\forall A, x : A \,.\, x = f\, x$$

Application

```
f : forall a. a → a
```

free theorem

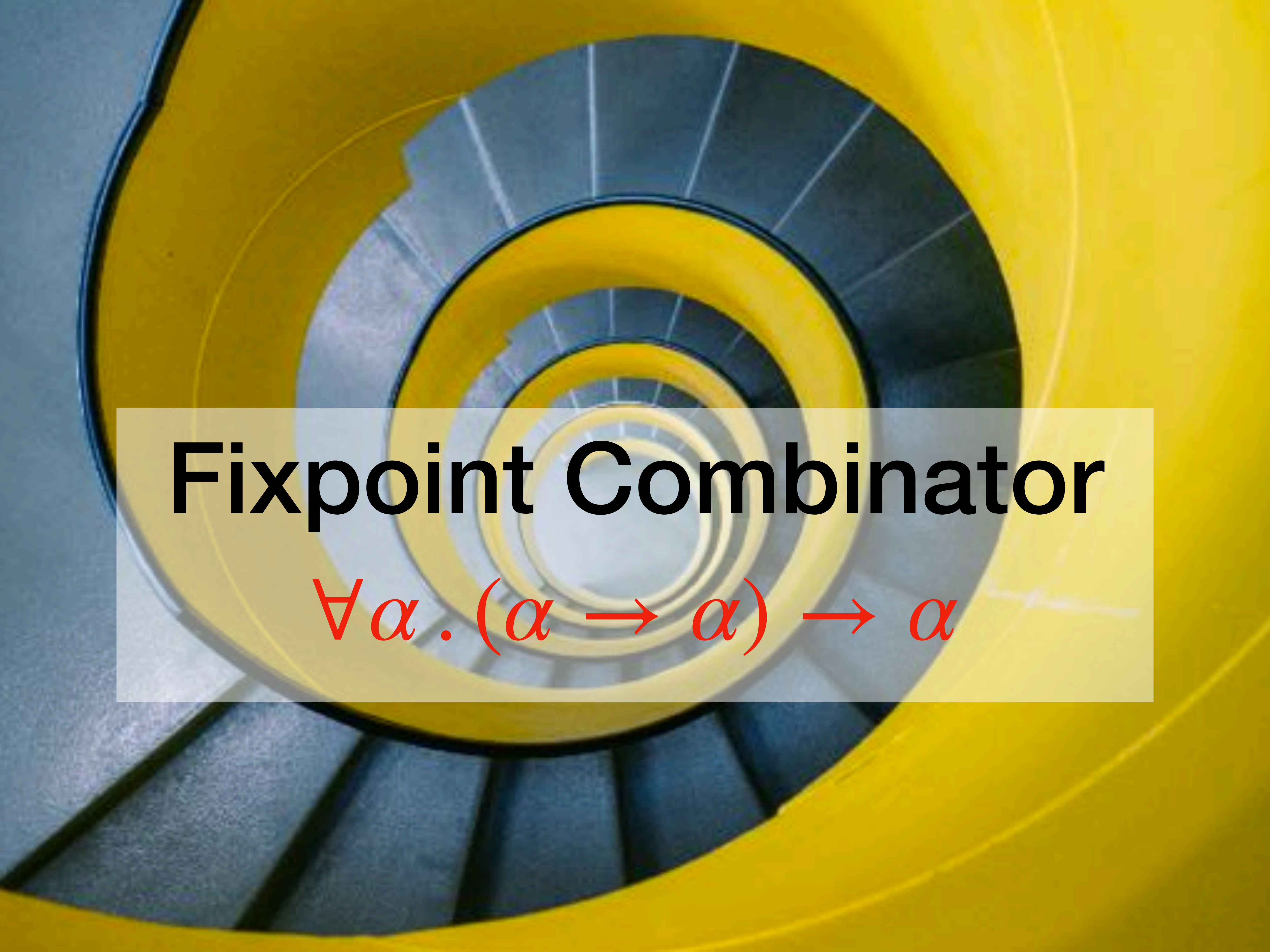
$$\forall A, B, h : A \to B \,.\, h \circ f_A = f_B \circ h$$

$A = B$ $\qquad$ $h = \textbf{const}\, x$

$$\forall A, x : A \,.\, \textbf{const}\, x \circ f = f \circ \textbf{const}\, x$$

$$\forall A, x : A \,.\, x = f\, x$$

$f = \textbf{id}$



Fixpoint Combinator

$$\forall \alpha \,.\, (\alpha \to \alpha) \to \alpha$$

Examples

```
f : forall a. (a → a) → a
```

Examples

```
f : forall a. (a → a) → a
```

Example 1:

```
f x = x (f x)          fix
```

Examples

```
f : forall a. (a → a) → a
```

Example 1:

```
f x = x (f x)          fix
```

Example 2:

Examples

```
f : forall a. (a → a) → a
```

Example 1:

```
f x = x (f x)          fix
```

Example 2:

???

Fixpoint Combinator

```
f : forall a. (a → a) → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B\,. \quad f_A, f_B \in (\mathscr{R} \to \mathscr{R}) \to \mathscr{R}$$

Fixpoint Combinator

```
f : forall a. (a → a) → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\quad f_A, f_B \in (\mathscr{R} \to \mathscr{R}) \to \mathscr{R}$$

$$\forall A, B, \mathscr{R} : A \times B \,. \forall x, y \in \mathscr{R} \to \mathscr{R} \,. f_A\, x, f_B\, y \in \mathscr{R}$$

Fixpoint Combinator

```
f : forall a. (a → a) → a
```

$$\Big\Downarrow \quad \textbf{free theorem}$$

$$\forall A, B, \mathscr{R} : A \times B \,.\quad f_A, f_B \in (\mathscr{R} \to \mathscr{R}) \to \mathscr{R}$$

$$\Big\Updownarrow$$

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \in \mathscr{R} \to \mathscr{R} \,.\, f_A\, x, f_B\, y \in \mathscr{R}$$

$$\Big\Updownarrow$$

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \,.\, (\forall u, v \in \mathscr{R} \,.\, x\, u, y\, v \in \mathscr{R}) \Rightarrow f_A\, x, f_B\, y \in \mathscr{R}$$

Fixpoint Combinator

```
f : forall a. (a → a) → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \,.\, (\forall u, v \in \mathscr{R} \,.\, x\, u, y\, v \in \mathscr{R}) \Rightarrow f_A\, x, f_B\, y \in \mathscr{R}$$

Fixpoint Combinator

```
f : forall a. (a → a) → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \,.\, (\forall u, v \in \mathscr{R} \,.\, x\,u, y\,v \in \mathscr{R}) \Rightarrow f_A\,x, f_B\,y \in \mathscr{R}$$

functional relation

$$\forall A, B, h : A \to B \,.\, \forall x, y \,.\, (\forall u \,.\, h\,(x\,u) = y\,(h\,u)) \Rightarrow h\,(f_A\,x) = f_B\,y$$

Fixpoint Combinator

```
f : forall a. (a → a) → a
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \,.\, (\forall u, v \in \mathscr{R} \,.\, x\, u, y\, v \in \mathscr{R}) \Rightarrow f_A\, x, f_B\, y \in \mathscr{R}$$

functional relation

$$\forall A, B, h : A \to B \,.\, \forall x, y \,.\, (\forall u \,.\, h\,(x\, u) = y\,(h\, u)) \Rightarrow h\,(f_A\, x) = f_B\, y$$

pointfree

$$\forall A, B, h : A \to B \,.\, \forall x, y \,.\, (h \circ x = y \circ h) \Rightarrow h\,(f_A\, x) = f_B\, y$$

Fixpoint Fusion

```
fix :: forall a. (a → a)
fix f = f (fix f)
```

```
h . f = g . h ⇒ h (fix f) = fix g
```

Worker/Wrapper Transformation

Naive Reverse

```
rev :: [a] → [a]
rev = fix rev’ where
    rev’ f []       = []
    rev’ f (x:xs) = f xs ++ x
```

Difference Lists
aka Hughes Lists
aka Cayley Lists

```
type H a = [a] → [a]
```

Difference Lists
aka Hughes Lists
aka Cayley Lists

```
type H a = [a] → [a]

rep :: [a] → H a
rep xs = \ys → xs ++ ys
```

Difference Lists
aka Hughes Lists
aka Cayley Lists

```
type H a = [a] → [a]

rep :: [a] → H a
rep xs = \ys → xs ++ ys

abs :: H a → [a]
abs h  =  []
```

Difference Lists
aka Hughes Lists
aka Cayley Lists

```
type H a = [a] → [a]

rep :: [a] → H a
rep xs = \ys → xs ++ ys

abs :: H a → [a]
abs h  =  []
```

$$\mathbf{abs} \circ \mathbf{rep} = \mathbf{id}$$

Transforming

```
  fix rev'
=
  id . fix rev'
=
  abs . rep . fix rev'
=  {- out f g = f . g -}
  abs . out rep (fix rev')
=  {- fixpoint fusion -}
  abs . fix rev2'
```

```
out rep . rev = rev' . out rep
```

Fusion Condition

```
out rep . rev’ = rev2’ . out rep
```

```
rep (rev’ f xs)
=
rev2’ (\ys → rep (f ys)) xs
```

Base Case

```
  rep (rev' f [])
=
  rep []
=
  \ys → [] ++ ys
=
  \ys → ys
=
  rev2' (\ys → rep (f ys)) []
```

```
rev2' g []
  = \ys → ys
```

Inductive Case

```
  rep (rev' f (x:xs))
=
  rep (f xs ++ [x])
=
  rep (f xs) . (x:)
=
  rev2' (\ys → rep (f ys)) (x:xs)
```

```
rev2' g (x:xs)
  = g xs . (x:)
```

Fast Reverse

```
rev :: [a] → [a]
rev = abs . fix rev2’ where
   rev2’ g []     = \ys → ys
   rev2’ g (x:xs) = \ys → g xs (x : ys)
```

```
rev :: [a] → [a]
rev xs = rev2 xs [] where
   rev2 []     ys = ys
   rev2 (x:xs) ys = rev2 xs (x : ys)
```

Constant Types

$$\forall \alpha \,.\, \alpha \rightarrow \textbf{Int}$$

Examples

```
f : forall a. a → Int
```

Example 1:

```
f x = 5
```

Example 2:

```
f x = 35
```

Logical Relations

```
f : forall a. a → Int
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\quad f_A, f_B \in \mathscr{R} \to \mathbf{Int}$$

$$\forall A, B, \mathscr{R} : A \times B \,. \forall x, y \in \mathscr{R} \,. f_A\, x, f_B\, y \in \mathbf{Int}$$

Definition

Int

$$=$$

$$\{x, x \mid x : \textbf{Int}\}$$

Logical Relations

```
f : forall a. a → Int
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \in \mathscr{R} \,.\, f_A\, x, f_B\, y \in \textbf{Int}$$

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \in \mathscr{R} \,.\, f_A\, x = f_B\, y$$

Application

```
f : forall a. a → Int
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \in \mathscr{R} \,.\, f_A\, x = f_B\, y$$

$\mathscr{R} = A \times B$

$$\forall A, B \,.\, \forall x, y \,.\, f_A\, x = f_B\, y$$

Type Constructors

$$\forall \alpha \,.\, \alpha \rightarrow [\alpha]$$

Examples

```
f : forall a. a → [a]
```

Example 1:

```
f x = [x]
```

Example 2:

```
f x = [x,x,x]
```

Logical Relations

```
f : forall a. a → [a]
```

$$\Big\Downarrow \quad \text{free theorem}$$

$$\forall A, B, \mathscr{R} : A \times B . \quad f_A, f_B \in \mathscr{R} \to [\mathscr{R}]$$

$$\Big\Updownarrow$$

$$\forall A, B, \mathscr{R} : A \times B . \forall x, y \in \mathscr{R} . f_A\, x, f_B\, y \in [\mathscr{R}]$$

Definition

$$[\mathscr{R}]$$

$$=$$

$$\{([],[])\} \cup \{(x : xs, y : ys) \,|\, x, y \in \mathscr{R} \wedge xs, ys \in [\mathscr{R}]\}$$

Functional Restriction

$$[f]$$

$$=$$

$$\{([],[])\} \cup \{(x : xs, y : ys) \,|\, f\,x = y \wedge xs, ys \in [f]\}$$

$$=$$

$$\{xs, ys \,|\, ys = \mathbf{fmap}\, f\, xs\}$$

Functional Restriction

```
f : forall a. a → [a]
```

free theorem

$$\forall A, B, \mathscr{R} : A \times B \,.\, \forall x, y \in \mathscr{R} \,.\, f_A\, x, f_B\, y \in [\mathscr{R}]$$

$$\forall A, B, g : A \to B \,.\, \forall x \,.\, \mathbf{fmap}\, g\, (f_A\, x) = f_B\, (g\, x)$$