Parametricity *or:* How I Learned to Stop Worrying and Love Theorems for Free

Charles Yuan

Hype for Types, Spring 2021

How many functions of this type?

```
'a -> 'a
```

How many functions of this type?

```
'a -> 'a
```

```
fn x => (print "hello"; x)
```

How many functions of this type?

```
'a -> 'a
```

```
let fun f x = f x in f end
```

How many *pure, total* functions of this type?

```
'a -> 'a
```

How many *pure, total* functions of this type?

```
'a -> 'a -> 'a
```

How many *pure, total* functions of this type?

`'a list -> 'a list`

What do they all have in common?

Claim: all functions of type

`'a list -> 'a list`

merely *rearrange* their input (insensitive to contents).

How many *pure, total* functions of this type?

```
('a -> 'b) -> 'a list -> 'b list
```

What do they all have in common?

Claim: all functions of type

```
('a -> 'b) -> 'a list -> 'b list
```

are *close relatives* of `map`.

Claim: all functions of type

```
('a -> 'b) -> 'a list -> 'b list
```

are just `map` composed with a *rearranging* function.

How many *pure, total* functions of this type?

```
'a -> 'b
```

A fun isomorphism

$$A \cong \forall X.(A \to X) \to X$$

A fun isomorphism

$$A \cong \tilde{A} \triangleq \forall X.(A \to X) \to X$$

$$f : A \to \tilde{A}$$

$$f = \lambda(x : A)\ \Lambda(X)\ \lambda(k : A \to X)\ k(x)$$

$$g : \tilde{A} \to A$$

$$g = \lambda(h : \tilde{A})\ h[A](\lambda(x : A)\ x)$$

A fun isomorphism

$$f = \lambda(x : A)\ \Lambda(X)\ \lambda(k : A \to X)\ k(x)$$

$$g = \lambda(h : \tilde{A})\ h[A](\lambda(x : A)\ x)$$

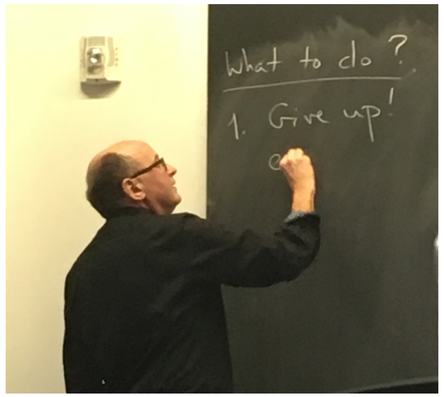
$$\begin{aligned}
& g(f(x)) \\
&= g(\Lambda(X)\ \lambda(k : A \to X)\ k(x)) \\
&= (\lambda(k : A \to A)\ k(x))(\lambda(x : A)\ x) \\
&= (\lambda(x : A)\ x)(x) \\
&= x
\end{aligned}$$

A fun isomorphism

$$f = \lambda(x : A)\ \Lambda(X)\ \lambda(k : A \to X)\ k(x)$$

$$g = \lambda(h : \tilde{A})\ h[A](\lambda(x : A)\ x)$$

$$\begin{aligned} & f(g(x)) \\ = {} & f(x[A](\lambda(x : A)\ x)) \\ = {} & \Lambda(X)\ \lambda(k : A \to X)\ k(x[A](\lambda(x : A)\ x)) \end{aligned}$$



Theorems for free!

Philip Wadler
University of Glasgow*

June 1989

Theorems for free!

Not *free* as in beer, *free* as in monads

Parametricity theorem (Reynolds)

Formalizes intuition: ML polymorphic types are contracts

```
(* given some *) val r : 'a list -> 'a list

(* for all *) val a : s -> t

(* it is true that *) (map a) o (r : s list -> s list)

(* is equivalent to *) (r : t list -> t list) o (map a)
```

```
(map Char.ord) (rev [#"a", #"b", #"c"])

= [99,98,97] : int list

= rev (map Char.ord [#"a", #"b", #”c"])
```

```
(map (fn x => x + 1)) (tl [1, 2, 3])

= [3, 4] : int list

= tl (map (fn x => x + 1) [1, 2, 3])
```

```
(map (fn x => x + 1)) (filter (fn x => x mod 2 = 0) [1, 2, 3])

= [3] : int list

filter (fn x => x mod 2 = 0) (map (fn x => x + 1) [1, 2, 3])
```

```
(map (fn x => x + 1)) (filter (fn x => x mod 2 = 0) [1, 2, 3])

= [3] : int list

filter (fn x => x mod 2 = 0) (map (fn x => x + 1) [1, 2, 3])

= [2, 4] : int list
```

```
(map (fn x => x + 1)) (filter (fn x => x mod 2 = 0) [1, 2, 3])

= [3] : int list

filter (fn x => x mod 2 = 0) (map (fn x => x + 1) [1, 2, 3])

= [2, 4] : int list

filter (fn x => x mod 2 = 0) : int list -> int list
```

Assume $a : A \to A'$ and $b : B \to B'$.

$$head : \forall X.\ X^* \to X$$
$$a \circ head_A = head_{A'} \circ a^*$$

$$tail : \forall X.\ X^* \to X^*$$
$$a^* \circ tail_A = tail_{A'} \circ a^*$$

$$(+\!\!+) : \forall X.\ X^* \to X^* \to X^*$$
$$a^*\ (xs +\!\!+_A ys) = (a^*\ xs) +\!\!+_{A'} (a^*\ ys)$$

$$concat : \forall X.\ X^{**} \to X^*$$
$$a^* \circ concat_A = concat_{A'} \circ a^{**}$$

$$fst : \forall X.\ \forall Y.\ X \times Y \to X$$
$$a \circ fst_{AB} = fst_{A'B'} \circ (a \times b)$$

$$snd : \forall X.\ \forall Y.\ X \times Y \to Y$$
$$b \circ snd_{AB} = snd_{A'B'} \circ (a \times b)$$

$$zip : \forall X.\ \forall Y.\ (X^* \times Y^*) \to (X \times Y)^*$$
$$(a \times b)^* \circ zip_{AB} = zip_{A'B'} \circ (a^* \times b^*)$$

$$filter : \forall X.\ (X \to Bool) \to X^* \to X^*$$
$$a^* \circ filter_A\ (p' \circ a) = filter_{A'}\ p' \circ a^*$$

$$sort : \forall X.\ (X \to X \to Bool) \to X^* \to X^*$$
if for all $x, y \in A,\quad (x < y) = (a\ x <'\ a\ y)$ then
$$a^* \circ sort_A\ (<) = sort_{A'}\ (<') \circ a^*$$

$$fold : \forall X.\ \forall Y.\ (X \to Y \to Y) \to Y \to X^* \to Y$$
if for all $x \in A, y \in B,\quad b\ (x \oplus y) = (a\ x) \otimes (b\ y)$ and $b\ u = u'$ then
$$b \circ fold_{AB}\ (\oplus)\ u = fold_{A'B'}\ (\otimes)\ u' \circ a^*$$

$$I : \forall X.\ X \to X$$
$$a \circ I_A = I_{A'} \circ a$$

$$K : \forall X.\ \forall Y.\ X \to Y \to X$$
$$a\ (K_{AB}\ x\ y) = K_{A'B'}\ (a\ x)\ (b\ y)$$

Suppose that I tell you that I am thinking of a function m with the type

$$m : \forall X.\forall Y.(X \to Y) \to (X^* \to Y^*)$$

You will immediately guess that I am thinking of the map function, $m(f) = f^*$. Of course, I could be thinking of a different function, for instance, one that reverses a list and then applies f^* to it. But intuitively, you know that map is the only interesting function of this type: that all others must be rearranging functions composed with map.

We can formalise this intuition as follows. Let m be a function with the type above. Then

$$m_{AB}(f) = f^* \circ m_{AA}(I_A) = m_{BB}(I_B) \circ f^*$$

where I_A is the identity function on A. The function $m_{AA}(I_A)$ is a rearranging function, as discussed in the preceding section. Thus, every function m of the above type can be expressed as a rearranging function composed with map, or equivalently, as map composed with a rearranging function.

The proof is simple. As we have already seen, the parametricity condition for m is that

$$\text{if } f' \circ a = b \circ f \text{ then } m_{A'B'}(f') \circ a^* = b^* \circ m_{AB}(f)$$

Taking $A' = B' = B$, $b = f' = I_B$, $a = f$ satisfies the hypotheses, giving as the conclusion

$$m_{BB}(I_B) \circ f^* = (I_B)^* \circ m_{AB}(f)$$

which gives us the second equality above, since $(I_B)^* = I_{B^*}$. The first equality may be derived by commuting the permuting function with map; or may be derived directly by a different substitution.

Polymorphic Troubles

```
(op =) : 'a * 'a -> bool
```

Polymorphic Troubles

```
(op =) : 'a * 'a -> bool

fun f [a, b] = if a = b then [] else [a, b] | f x = x (* oops *)

  : 'a list -> 'a list
```

Polymorphic Troubles

```
(op =) : ''a * ''a -> bool

val f : ''a list -> ''a list
```

Polymorphic Troubles

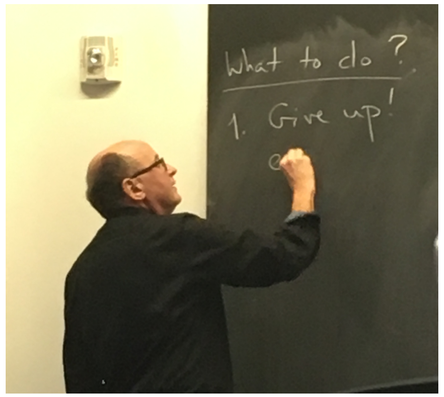
```
$ ocaml

        OCaml version 4.10.0

# (<);;

- : 'a -> 'a -> bool = <fun>
```

A fun isomorphism

$$f = \lambda(x : A)\ \Lambda(X)\ \lambda(k : A \to X)\ k(x)$$

$$g = \lambda(h : \tilde{A})\ h[A](\lambda(x : A)\ x)$$

$$\begin{aligned} & f(g(x)) \\ = {} & f(x[A](\lambda(x : A)\ x)) \\ = {} & \Lambda(X)\ \lambda(k : A \to X)\ k(x[A](\lambda(x : A)\ x)) \end{aligned}$$



A fun isomorphism

$$f = \lambda(x : A)\ \Lambda(X)\ \lambda(k : A \to X)\ k(x)$$
$$g = \lambda(h : \tilde{A})\ h[A](\lambda(x : A)\ x)$$

$$\begin{aligned} & f(g(x)) \\ = {} & f(x[A](\lambda(x : A)\ x)) \\ = {} & \Lambda(X)\ \lambda(k : A \to X)\ k(x[A](\lambda(x : A)\ x)) \end{aligned}$$

If $h : \forall X.(A \to X) \to X$, then
for all $b : B \to B'$, $f : A \to B$,
$$b(h[B]f) = h[B'](b \circ f)$$

A fun isomorphism

$$f = \lambda(x : A)\ \Lambda(X)\ \lambda(k : A \to X)\ k(x)$$

$$g = \lambda(h : \tilde{A})\ h[A](\lambda(x : A)\ x)$$

$$\begin{aligned}
& f(g(x)) \\
&= f(x[A](\lambda(x : A)\ x)) \\
&= \Lambda(X)\ \lambda(k : A \to X)\ k(x[A](\lambda(x : A)\ x)) \\
&= \Lambda(X)\ \lambda(k : A \to X)\ x[X](k \circ (\lambda(x : A)\ x)) \\
&= \Lambda(X)\ \lambda(k : A \to X)\ x[X](k) \\
&= x
\end{aligned}$$