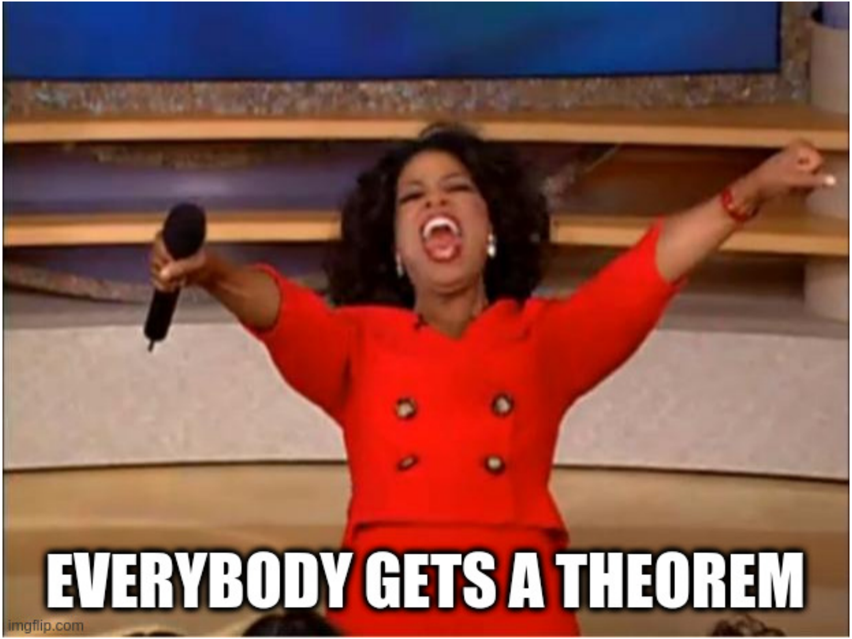
Theorems for free!

Lars Hupel
MuniHac
2020-09-11

INNOQ


EVERYBODY GETS A THEOREM
imgflip.com


Types in Haskell

Type basics

- type variables are lower case
- all types are erased

(ignoring classes for now)

What Haskell sees:

```
id :: a -> a
```

What Haskell sees:

```
id :: a -> a
```

What the runtime sees:

```
id :: Word -> Word
```

Folklore says:

The more type variables, the merrier!

```haskell
data Lens s a = Lens
  { getter :: s -> a
  , setter :: a -> s -> s }
```

```haskell
type Lens s t a b =
  Functor f =>
    (a -> f b) ->
    s -> f t
```

More type variables!

… but why?

We can reason about types!

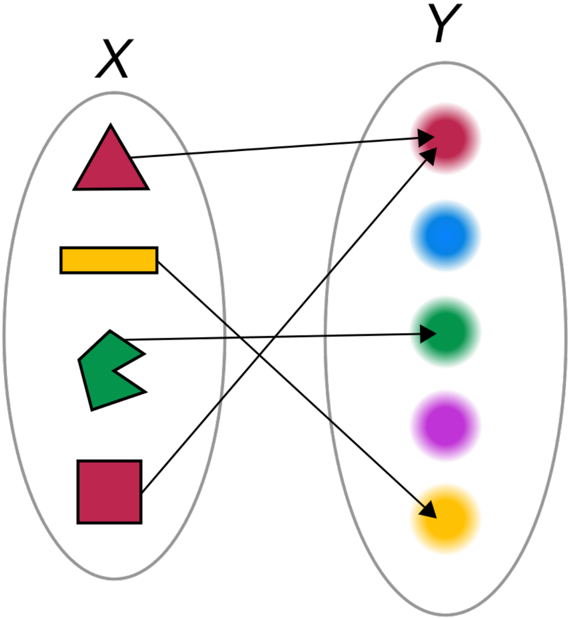
… but how?

Sets in mathematics

In set theory, everything[1] is a set.

For example: $\mathbb{N} = \{0, 1, 2, \ldots\}$

[1] almost


X
Y

Functions are sets

$$f = \{(\blacktriangle, \bullet), (\blacksquare, \bullet), \ldots\}$$

Types are sets

$$\begin{aligned}
[\![\texttt{Bool}]\!] &= \{\texttt{True}, \texttt{False}\} \\
[\![\texttt{Integer}]\!] &= \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(a, b)]\!] &= [\![a]\!] \times [\![b]\!] \\
[\![a \to b]\!] &= \text{the set of all functions from } [\![a]\!] \text{ to } [\![b]\!]
\end{aligned}$$

Key insight:

There are many different interpretations of types.

Side note

Wadler's paper uses A^* instead of `[a]`. Any idea why?

Relations

Relation R between A and B: $R \subseteq A \times B$


always have been
wait, types are relations too?
imgflip.com

Types are relations

We can assign every type t a relation rel_t.

Types are relations

We can assign every type t a relation rel_t.

This relation will relate values of $[\![t]\!]$: $\mathrm{rel}_t \subseteq [\![t]\!] \times [\![t]\!]$

Ground types

… are identity relations

$$\begin{aligned} \mathsf{rel}_{\texttt{Bool}} &= \{(\texttt{True}, \texttt{True}), (\texttt{False}, \texttt{False})\} \\ \mathsf{rel}_{\texttt{Integer}} &= \{(n, n) \mid n \in \mathbb{Z}\} \end{aligned}$$

Lists

We have a relation for a.
We want to check if $\mathrm{xs}, ys : [a]$ are related.

Lists

We have a relation for a.
We want to check if $\text{xs}, \text{ys} : [a]$ are related.

$\longrightarrow$ xs and ys need to be the same length and pairwise related

Lists: example

Let $\text{rel}_a\ x\ y = (y = 2 \cdot x)$

$[]$	$[]$	✓
$[1, 2]$	$[2, 4]$	✓
$[1, 2]$	$[2, 4, 6]$	✗
$[1, 2]$	$[0, 1]$	✗

Functions

When are two functions related?

When they send related inputs to related outputs.

Functions

$f : a \to b$ and $g : a \to b$ are related if:

$$\forall x, y \in [\![a]\!].\ (x, y) \in \mathrm{rel}_a \implies (f\ x, g\ y) \in \mathrm{rel}_b$$

Parametricity

The parametricity theorem

If t is a closed term of type T, then $(t, t) \in \mathrm{rel}_T$.

The parametricity theorem

If t is a closed term of type T, then $(t, t) \in \mathrm{rel}_T$.

In other words: every term is related to itself

Let's say we have a function on maps.

```
frobnicate :: [a] -> [a]
```

Let's say we have a function on maps.

```
frobnicate :: [a] -> [a]
```

Parametricity states:

$$(\texttt{frobnicate}, \texttt{frobnicate}) \in$$ 

Let's say we have a function on maps.

```
frobnicate :: [a] -> [a]
```

Parametricity states:

$$(\texttt{frobnicate}, \texttt{frobnicate}) \in$$ 

We can prove:

$$\texttt{frobnicate}\ (\texttt{map}\ g\ xs) = \texttt{map}\ g\ (\texttt{frobnicate}\ xs)$$

Now what?


BEFORE

AFTER

Reasoning about types

Motto: Functions with type variables ...

- don't know anything
- can't do much

In practise

The second `Functor` law is redundant.

It is sufficient to prove that `fmap id = id`.

Free Theorems!

Please enter a (polymorphic) type, e.g. "(a -> Bool) -> [a] -> [a]":

(a -> Bool) -> [a] -> [a]

Please choose a sublanguage of Haskell:

no bottoms (hence no general recursion and no selective strictness)

inequational theorems (only relevant in a language with bottoms)

hide type instantiations in the theorem presentation

The Free Theorem

```
forall t1,t2 in TYPES, R in REL(t1,t2).
 forall p :: t1 -> Bool.
  forall q :: t2 -> Bool.
   (forall (x, y) in R. p x = q y)
   ==> (forall (z, v) in lift{[]}(R). (f p z, f q v) in lift{[]}(R))
```

The Free Theorem

with all permissable relation variables reduced to functions

```
forall t1,t2 in TYPES, g :: t1 -> t2.
 forall p :: t1 -> Bool.
  forall q :: t2 -> Bool.
   (forall x :: t1. p x = q (g x))
   ==> (forall y :: [t1]. map g (f p y) = f q (map g y))
```

Another free theorem

A function with type `(a -> b) -> [a] -> [b]` is either

1. `map`, or
2. `map` with rearrangements

Restrictions

$\bot$ destroys everything[2]

---

[2] not everything

Extensions

We have ignored classes (so far) because they complicate things.

Extensions

We have ignored classes (so far) because they complicate things.

Classes can be modelled as dictionaries with (potentially) rank-2 types

Q & A

INNOQ
www.innoq.com

Lars Hupel

lars.hupel@innoq.com

@larsr_h

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim a. Rh.
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany

Ludwigstr. 180 E
63067 Offenbach
Germany

Kreuzstr. 16
80331 München
Germany

c/o WeWork
Hermannstrasse 13
20095 Hamburg
Germany

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 01 11

Albulastr. 55
8048 Zürich
Switzerland

LARS HUPEL

Consultant
innoQ Deutschland GmbH

Lars enjoys programming in a variety of languages, including Scala, Haskell, and Rust. He is known as a frequent conference speaker and one of the founders of the Typelevel initiative which is dedicated to providing principled, type-driven Scala libraries.

Credits

- John C. Reynolds: `https://commons.wikimedia.org/w/index.php?title=File:Reynolds_John_small.jpg&oldid=452226049`, Andrej Bauer, CC-BY-SA 2.5
- Philip Wadler: `https://commons.wikimedia.org/w/index.php?title=File:Wadler2.JPG&oldid=262214892`, Clq, CC-BY 3.0
- Function: `https://commons.wikimedia.org/w/index.php?title=File:Function_color_example_3.svg&oldid=321533277`, Wvbailey, CC-BY-SA 3.0
- Free Theorems: `https://free-theorems.nomeata.de/`, Joachim Breitner et al.