## Parametric Polymorphism in Haskell

Ben Deane

30th October 2014

Ben Deane

Parametric Polymorphism in Haskell

30th October 2014 1 / 15

# I suspect there are several people in the audience who know more about this than I do!

This is what I think I know. (Broadly on the topic of Parametric Polymorphism.)

# Parametric Type variables (a, b, etc)

#### Ad-hoc

• Type classes (Eq, Num, etc)

#### Parametric

- Type variables (a, b, etc)
- Universal

#### Ad-hoc

- Type classes (Eq, Num, etc)
- Existential?

#### Parametric

- Type variables (a, b, etc)
- Universal
- Compile-time

#### Ad-hoc

- Type classes (Eq, Num, etc)
- Existential?
- Runtime (also)

#### Parametric

- Type variables (a, b, etc)
- Universal
- Compile-time
- C++ templates, Java generics

#### Ad-hoc

- Type classes (Eq, Num, etc)
- Existential?
- Runtime (also)
- Classical ("normal" OO)

## Polymorphic Datatypes

#### For example...

data Maybe a = Nothing | Just a

data List a = Nil | Cons a (List a)

data Either a b = Left a | Right b

```
1
2
3
4
5
```

Ben Deane

イロト 不得 トイヨト イヨト 二日

## **Polymorphic Functions**

# For example...

fst :: (a,b) -> a

id :: a -> a

Ben Deane

(日)

- Work over all types
- Assume nothing behaviour-wise
- Parametricity
  - Intuitively, all instances act the same way
  - Theorems for free
  - eg.reverse . map f  $\Leftrightarrow$  map f . reverse

# **Totality**

#### Partial functions

1 head :: [a] -> a 2 3 tail :: [a] -> [a]

What happens when a is []?

These are not *total functions*. They are undefined for some inputs.

# *Unification*: the process of solving a system of equations in type variables.

and

# Inference: Why you thought you meant Int -> Int but the compiler knows you really meant Num a => a -> a.

• What can we say about a -> a?

<ロ> <四> <四> <四> <四> <四</p>

- What can we say about a -> a?
  - We know it has to be id.

э

イロト イポト イヨト イヨト

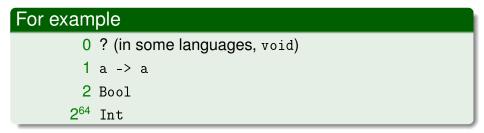
- What can we say about a -> a?
  - We know it has to be id.
- What can we say about Int -> Int?

- What can we say about a -> a?
  - We know it has to be id.
- What can we say about Int -> Int?
  - Almost nothing!

- What can we say about a -> a?
  - We know it has to be id.
- What can we say about Int -> Int?
  - Almost nothing!

• Even though we know more about Int than about a!

- The set of representable values
- How many values inhabit the type
- Characterised by cardinality of the set



# Sum Types and Product Types

#### Sum types represent alternation

# For example $a + b \Leftrightarrow$ Either a b $1 + a \Leftrightarrow$ Maybe a

# Sum Types and Product Types

#### Product types represent composition

For example	
$a * b \Leftrightarrow (a,b)$	
$a * 2 \Leftrightarrow (a, Bool)$	

3

# What about function types?

- They equate to a power function
  - $b^a \Leftrightarrow a \rightarrow b$
- (For concrete types, not type variables)

3

## From CIS194 2014

#### Week 4, Exercise 5

```
-- How many distinct functions inhabit this type?
 1
   ex5 :: Bool -> Bool
 2
   -- Answer: 4
 3
   ex5 = const True
 4
   ex5_2 = const False
 5
   ex5 3 = id
6
   ex5_4 = complement
7
8
   -- Using type algebra:
   -- Bool -> Bool
9
10
  -- => 2 -> 2
11 -- => 2^2 = 4
```

- Theorems for Free Philip Wadler
- The Algebra of Algebraic Data Types Chris Taylor
   London HUG video
- Adventures with Types in Haskell Simon Peyton-Jones