# Introduction to Type-Level and Generic Programming in Haskell

CUFP 2016

Andres Löh

22 September 2016

wtlogotext.pdf

## Datatype-generic programming

Express algorithms that make use of the structure of datatypes

wtlogo.pdf

## Datatype-generic programming

Express algorithms that make use of the structure of datatypes

$\mathtt{eq}_{\mathtt{A}}$ `:: A -> A -> Bool`

wtlogo.pdf

## A class

```haskell
class Generic a where
  type Rep a
  from :: a -> Rep a
  to   :: Rep a -> a
```

where `from` and `to` are inverses.

wtlogo.pdf

## A class

```
class Generic a where
  type Rep a
  from :: a -> Rep a
  to   :: Rep a -> a
```

where `from` and `to` are inverses.

```
geq :: Generic a => Rep a -> Rep a -> Bool
```

## A class

```
class Generic a where
  type Rep a
  from :: a -> Rep a
  to   :: Rep a -> a
```

where `from` and `to` are inverses.

```
geq :: Generic a => Rep a -> Rep a -> Bool
```

```
eq :: Generic a => a -> a -> Bool
eq x y = geq (from x) (from y)
```

wtlogo.pdf

## Choices

Much flexibility in the details, in particular the definition of `Rep`.

wtlogo.pdf

## Choices

Much flexibility in the details, in particular the definition of `Rep`.

The choice of `Rep` determines expressive power and flavour of generic programs.

wtlogo.pdf

## Choices

Much flexibility in the details, in particular the definition of `Rep`.

The choice of `Rep` determines expressive power and flavour of generic programs.

In this tutorial: generics-sop.

wtlogo.pdf

## Applications

- (De-)serialization
- Data generation
- Data traversals
- Data navigation
- …

wtlogo.pdf

## The generics-sop view on data, informally

## Sample datatypes

```
data Maybe a    = Nothing | Just a
data Either a b = Left a | Right b
data Group      = Group Char Bool Int
data Expr       = NumL Int
                | BoolL Bool
                | Add Expr Expr
                | If Expr Expr Expr
```

wtlogo.pdf

## Sample datatypes

```
data Maybe a    = Nothing | Just a
data Either a b = Left a | Right b
data Group      = Group Char Bool Int
data Expr       = NumL Int
                 | BoolL Bool
                 | Add Expr Expr
                 | If Expr Expr Expr
```

- Choice between constructors,
- each with a sequence of arguments.

wtlogo.pdf

## Sample datatypes

```haskell
data Maybe a    = Nothing | Just a
data Either a b = Left a | Right b
data Group      = Group Char Bool Int
data Expr       = NumL Int
                | BoolL Bool
                | Add Expr Expr
                | If Expr Expr Expr
```

- Choice between constructors,
- each with a sequence of arguments.

$$C_i \; x_0 \ldots x_{n_i - 1}$$

## The plan

$\texttt{C}_{\texttt{i}}\ \texttt{x}_{\texttt{0}}\texttt{...x}_{\texttt{n}_{\texttt{i}}\texttt{-1}}$

- Choice between constructors modelled as an *n*-ary sum.
- Sequence of fields modelled as an *n*-ary product.

wtlogo.pdf

## The plan

```
Cᵢ x₀...xₙᵢ₋₁
```

$C_i\ x_0 \ldots x_{n_i-1}$

- Choice between constructors modelled as an *n*-ary sum.
- Sequence of fields modelled as an *n*-ary product.

$\mathtt{SOP\ (S...Z\ (I}\ x_0\ \mathtt{:*\ ...\ :*\ I}\ x_{n_i-1}\ \mathtt{:*\ Nil))}$

wtlogo.pdf

## Extensions, extensions

```
DataKinds
PolyKinds
ConstraintKinds
GADTs
TypeFamilies
MultiParamTypeClasses
FlexibleInstances
FlexibleContexts
UndecidableInstances
RankNTypes
DefaultSignatures
StandaloneDeriving
TypeOperators
ScopedTypeVariables
```

wtlogo.pdf

## Overall plan

- Learn about *n*-ary products and *n*-ary sums.
- Example generic functions.
- Explain internals, discuss type-level programming features.
- More generic functions.
- Handling metadata.

wtlogo.pdf