Power of Pi

Wouter Swierstra
Joint work with Nicolas Oury

Dependent types make a language more **expressive**.

Cryptol

Cryptol: example

```
x : [32]; -- a 32-bit word
x = 1337;
```

- The type of a word records its *size*.

Cryptol: example

```
swab : [32] -> [32]
swab [a b c d] = [b a c d]
```

- You can eliminate a word of size *n**k* by pattern matching on it as *n* words of size *k*.

Words

```
data Vec (A : Set) : Nat -> Set
  Nil : Vec A 0
  _::_ : A -> Vec A n -> Vec A (S n)

Word : Nat -> Set
Word n = Vec Bit n
```

Views

- Introducing Cryptol-style pattern matching on words entails:
  - Defining a data type `WordView` indexed by a `Word (n * k)`;
  - Defining a function `view` that produces a suitable `WordView xs,` for every `xs : Word (n * k).`

WordView

```
data WordView : Vec A (n * k) -> Set
  Split : (xss : Vec (Vec A k) n)
        -> WordView (concat xss)
```

View

```
chop : (k : Nat) -> Vec A (n * k)
     -> Vec (Vec A k) n


view : (xs : Vec A (n * k))
     ->  WordView xs
view xs = ... Split (chop k xs) ...
```

Example

```
swab : Word 32 -> Word 32
swab xs with view xs
... | Split (a :: b :: c :: d :: Nil)
   = concat (b :: a :: c :: d :: Nil)
```

Haskell

- GHC supports:
  - GADTs;
  - functional dependencies;
  - view patterns.
- Why do we need dependent types?

Bitmaps

The PBM monochrome bitmap format is one way to generate black-and-white images:

```
P1 50 100\n 00110100100010...
```

Haskell & PBM

- A PBM parser must return [[Bit]]...
- Even though there exact size of the bitmap is known once you've inspected the header;
- Many, many binary file formats are structured the same way.

Data, dependently

- In dependently typed languages:
  - you can define a data type of file formats;
  - and get parsers and printers for free.

A small universe

```
data U : Set where
  CHAR : U
  VEC : Nat -> U -> U
  BIT : U ....
el : U -> Set
```

Formats

```
data Format : Set where
  EOF : Format
  Bad : Format
  Read : (u : U)
       -> (el u -> Format)
       -> Format
```

PBM Format

```
PBM : Format
PBM = char ‘P’ $
      char ‘1’ $
      Read NAT $ \n ->
      Read NAT $ \m ->
      Read (VEC n (VEC m) BIT)
char c f = Read CHAR (\c’ -> ...)
```

Format Universe

```
< _ > : Format -> Set
< EOF >         = Unit
< Bad >         = Empty
< Read u f>     = Sigma (el u)
                        (res . f)
```

Read and Show

```
read : (f : Format) -> List Bit
     -> Maybe < f >

show : (f : Format) -> < f >
     -> List Bit
```

Joe Haskell Programmer says:

"Binary data is easy. I'm smart enough to handle it myself – I don't need all those annoying types."

Haskell & Databases

- Haskell has no type safe database interface:
  - use extensible records;
  - use type class tomfoolery;
  - represent everything by a String.
- ... accompanied by a preprocessor.

What's missing?

- A proper interface should:
  - connect to a database to query the type of all the fields;
  - **compute** the type of the database schema;
  - ensure static properties, such as the size of strings.

Bounded Strings

- Who said Haskell was expressive?

```
data N1 = N1 ...

data N255 = N255
```

Bounded Strings

- Who said Haskell was expressive?

```
data N1 = N1 ...
data N255 = N255

class Less a b
instance Less N1 N255
instance Less N2 N255...
```

- Precise data types

- Precise data types
- Views

- Precise data types
- Views
- Universes