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

\[
\text{swab : } [32] \rightarrow [32]
\]
\[
\text{swab } [a \ b \ c \ d] = [b \ a \ c \ d]
\]

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

\[
\text{data } \text{Vec } (A : \text{Set}) : \text{Nat} \rightarrow \text{Set} \\
\quad \text{Nil} : \text{Vec} A 0 \\
\quad _:::_ : A \rightarrow \text{Vec} A n \rightarrow \text{Vec} A (S n)
\]

\[
\text{Word} : \text{Nat} \rightarrow \text{Set} \\
\text{Word} n = \text{Vec} \text{Bit} n
\]
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)
```
chop : (k : Nat) \rightarrow Vec A (n \times k)
\rightarrow Vec (Vec A k) n

view : (xs : Vec A (n \times k))
\rightarrow WordView xs

view xs = \ldots \ Split (chop k xs) \ldots
Example

\[
\text{swab} \; : \; \text{Word 32} \rightarrow \text{Word 32} \\
\text{swab} \; \text{xs with view xs} \\
\ldots \mid \text{Split} \; (a :: b :: c :: d :: \text{Nil}) \\
= \text{concat} \; (b :: a :: c :: d :: \text{Nil})
\]
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
00110100100010...
```
Haskell & PBM

- A PBM parser must return \[[\text{Bit}]\]...
- 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
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 \rightarrow$

Read NAT $ \m \rightarrow$

Read (VEC n (VEC m) BIT)

char c f = Read CHAR (c' \rightarrow \ldots)
Format Universe

\[
\begin{align*}
\langle _\_ \rangle &: \text{Format} \rightarrow \text{Set} \\
\langle \text{EOF} \rangle &= \text{Unit} \\
\langle \text{Bad} \rangle &= \text{Empty} \\
\langle \text{Read } u \ f \rangle &= \text{Sigma} \ (\text{el } u) \\
& \quad \quad \quad (\text{res } . \ f)
\end{align*}
\]
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?

```haskell
data N1 = N1 ...
data N255 = N255
```
Bounded Strings

• Who said Haskell was expressive?

```haskell
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