Functional Programming

Second project: Idris

D.C.M. Hoogeveen

July 3, 2019

University of Twente

Table of Contents

Idris in general

Idris characteristics

- **Functional programming** language
- Haskell-inspired syntax
- **Strict** evaluation order
- Installable using Cabal
- Pac-man Complete
- **Compiled** (via e.g. C and Javascript);
    - Optimisations possible (e.g. aggresive erasure and inlining)
- **Dependent type** system

Dependent types: what?

Example
Looking at following type definition, what can we infer?

```
foo : Vect len elem -> Vect len elem
```

Dependent types: what?

Example
Looking at following type definition, what can we infer?

```
foo : Vect len elem -> Vect len elem
```

Inferred
Same length output as input and with the same type element.

Dependent types: what?

Example
Looking at following type definition, what can we infer?

```
foo : Vect len elem -> Vect len elem
```

Inferred
Same length output as input and with the same type element.

Conclusion
Dependent types allow us to define output dependent on the input.

Dependent type: why?

Useful for:

- **Checking** intended properties
- **Guiding** programmer
- Building **generic** libraries

Type Drive Development

Type
Write the type definition

Define
Create a stub implementation; can also be a hole: *?x*

Refine
Improve/complete implementation

Idris in practice

Example
Take first element of a vector:

```
myhead : Vect (S len) elem -> elem
myhead (x :: xs) = x
```

Advantage
Normally, a check is needed to prevent an error when a vector with zero length is used, however this is already defined using (S len)!

Idxsize

Example
Count number of values that are True within a vector recursively

Idxsize

Example
Count number of values that are True within a vector recursively

```
idxsize : Vect n Bool -> Nat
```

Idxsize

Example
Count number of values that are True within a vector recursively

```
idxsize : Vect n Bool -> Nat

idxsize [] = 0
```

Idxsize

Example
Count number of values that are True within a vector recursively

```
idxsize : Vect n Bool -> Nat

idxsize [] = 0

idxsize (True :: xs)  = 1 + idxsize xs
idxsize (False :: xs) = idxsize xs
```

Usage of idxsize as type

Example
Combine two vectors by only returning element of second vector if element of first vector is True

Defined type definition:

```
get : (xs : Vect n Bool) -> Vect n a -> Vect ? a
```

Expected use:

```
get [True, False, True] [1,2,3] == [1,3]
```

What to fill in for '?'?

Usage of idxsize as type

Example

```
get : (xs : Vect n Bool) -> Vect n a -> Vect ? a
```

Expecting a Nat (natural number type) of amount of values in first vector whose value is True.

Usage of idxsize as type

Example

```
get : (xs : Vect n Bool) -> Vect n a -> Vect ? a
```

Expecting a Nat (natural number type) of amount of values in first vector whose value is True.

Recall idxsize

```
idxsize : Vect n Bool -> Nat
```

This is exactly the type needed for '?'!

Usage of idxsize as type

Example

```
get : (xs : Vect n Bool) -> Vect n a -> Vect ? a
```

Expecting a Nat (natural number type) of amount of values in first vector whose value is True.

Recall idxsize

```
idxsize : Vect n Bool -> Nat
```

This is exactly the type needed for '?'!

Solution
? = idxsize xs

Questions