Certified Software Development with Dependent Types in Idris

Lecture 5. Functions over Types

Vitaly Bragilevsky (bravit@sfedu.ru)

I.I. Vorovich Institute of Mathematics, Mechanics and Computer Science
Southern Federal University, Rostov-on-Don, Russia
part of Erasmus+ teaching mobility agreement with University of Twente

What can we do with types?

- Provide type synonyms
- Calculate types from given data

Type Synonyms

```
import Data.Vect

Point : Type
Point = (Double, Double)

Polygon : Nat -> Type
Polygon k = Vect k Point

Triangle : Type
Triangle = Polygon 3
```

syn.idr

Pattern Matching in Type-level Functions

```
data Ty = TyNat | TyBool | TyString

evalType : Ty -> Type

initVal : (ty : Ty) -> evalType ty

toString : (ty : Ty) -> evalType ty -> String
```

pm.idr

We don't need function evalType!

```
data Ty = TyNat | TyBool | TyString

toString : (ty : Ty) -> ?evalType -> String
```

case.idr

Type-level Functions and Totality

```
data Ty = TyNat | TyBool | TyString

infinity : Type
infinity = infinity

nt : (ty : Ty) -> Type
nt TyNat = Nat
nt TyBool = Bool

f : Nat -> infinity

g : (ty : Ty) -> nt ty
```

inf.idr

Functions with Variable Number of Arguments

```
adder  0  10      = 10
adder  1  0  5    = 5
adder  2  0  4  6 = 20
```

Arguments to adder:

- number of additional arguments
- initial value
- additional argument
- ...

Functions with Variable Number of Arguments

```
adder  0  10      = 10
adder  1  0  5    = 5
adder  2  0  4  6 = 20
```

Arguments to adder:

- number of additional arguments
- initial value
- additional argument
- ...

- Type of adder can be calculated:

```
adder 0 : Int -> Int
adder 1 : Int -> Int -> Int
adder 2 : Int -> Int -> Int -> Int
```

adder.idr

Type Safe sprintf Function (1)

Usage

```
sprintf "Hello!"                  = "Hello!"
sprintf "Answer : %d"  2          = "Answer : 42"
sprintf "%s number %d" "Slide" 8 = "Slide number 8"
```

Type Safe sprintf Function (1)

Usage

```
sprintf "Hello!"                  = "Hello!"
sprintf "Answer : %d"  2          = "Answer : 42"
sprintf "%s number %d" "Slide" 8 = "Slide number 8"
```

Components

- Data type describing format
- Function from String to format data type
- Type-level function calculating type of `sprintf`

Type Safe sprintf Function (2)

Types of sprintf

```
sprintf "Hello!"         : String
sprintf "Answer: %d"     : Int -> String
sprintf "%s number %d" : String -> Int -> String
```

Type Safe sprintf Function (2)

Types of sprintf

```
sprintf "Hello!"       : String
sprintf "Answer: %d"   : Int -> String
sprintf "%s number %d" : String -> Int -> String
```

Format data type

```
data Format = Number Format
            | Str Format
            | Lit String Format
            | End
```

Type Safe sprintf Function (2)

Types of sprintf

```
sprintf "Hello!"         : String
sprintf "Answer: %d"     : Int -> String
sprintf "%s number %d" : String -> Int -> String
```

Format data type

```
data Format = Number Format
            | Str Format
            | Lit String Format
            | End
```

`"%s = %d"` $\Longrightarrow$ `Str (Lit " = " (Number End))`

Type Safe sprintf Function (3)

```
SPrintfType : Format -> Type

sprintfFmt : (fmt : Format) ->
             (acc : String) ->
             SPrintfType fmt

toFormat : (xs : List Char) -> Format

sprintf : (fmt : String) ->
          SPrintfType (toFormat (unpack fmt))
```

sprintf.idr

Assignment 2: simple database

- Database is a vector of tuples

Assignment 2: simple database

- Database is a vector of tuples
- Database schema — description of a tuple for storing data in the database — string representation and data type

Assignment 2: simple database

- Database is a vector of tuples
- Database schema — description of a tuple for storing data in the database — string representation and data type
- Building tuple type from given schema

Assignment 2: simple database

- Database is a vector of tuples
- Database schema — description of a tuple for storing data in the database — string representation and data type
- Building tuple type from given schema
- Building tuple value from user input and storing it in the database

Assignment 2: simple database

- Database is a vector of tuples
- Database schema — description of a tuple for storing data in the database — string representation and data type
- Building tuple type from given schema
- Building tuple value from user input and storing it in the database
- Extracting tuple value by its ID and showing it to the user

Assignment 2: simple database

- Database is a vector of tuples
- Database schema — description of a tuple for storing data in the database — string representation and data type
- Building tuple type from given schema
- Building tuple value from user input and storing it in the database
- Extracting tuple value by its ID and showing it to the user
- Processing user commands: setting schema, storing data, extracting data

Bibliography

- Idris Tutorial: Types and Functions
  `http://docs.idris-lang.org/en/latest/tutorial/typesfuns.html`
- Idris Libraries Source Code
  `https://github.com/idris-lang/Idris-dev/tree/master/libs/`
- Edwin Brady, Type-Driven Development with Idris. Manning, MEAP.