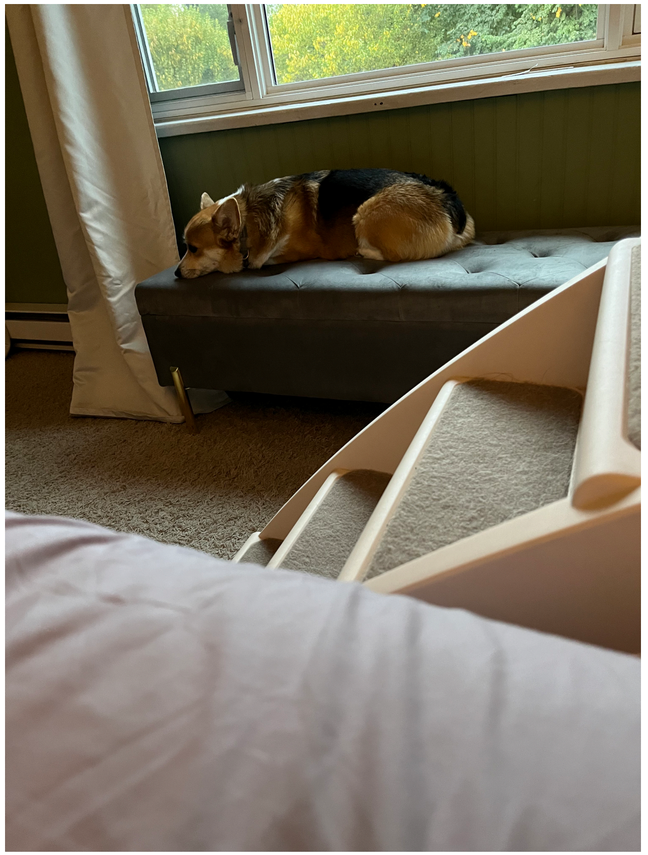
CMPT 383

Lecture 3: Haskell Types



Anders Miltner

- Describe Haskell types
- Describe the relationship between terms and structural typing
- Go into type and term syntax

Basic Types

- Tuples (products)
- Disjoint Unions (sums or coproducts)
- Arrows (functions)
- Polymorphism (foralls)

(T1 , T2)

- Tuples!
- Basic building block in Haskell
- Fixed length, with fixed types for each element of the tuple
- Examples: Go to code

Building tuples

```
tuples_builder :: a -> b -> (a,b)
tuples_builder x y = (x,y)
```

Extracting Information from tuples

```
let tuple = (x,y) in
let x’ = fst (x,y) in
let y’ = snd (x,y) in
let (x’’,y’’) = tuple in
```

Can build arbitrary-length tuples

```
big_tuple_builder :: a -> b -> c -> (a,b,c)
tuples_builder x y z = (x,y,z)
```

```
let tuple = (x,y,z) in
--let x' = fst (x,y) in
--let y' = snd (x,y) in
let (x'',y'',z'') = tuple in
```

Data Types

- This is where functional programming really starts to shine
- Some languages have *very recently* started adopting them
- Think of them like enums with extra information

Enums by example

```
data Bool =
    True
  | False
```

Enums by example

```
data Bool =
    True
  | False
```

Looks like they are just enums right?

Enums by Example

```
data MaybeInt =
    Nothing
  | Just Int
```

```
divide :: Int -> Int -> MaybeInt
divide i j =
  if j == 0 then
    Nothing
  else
    Just (i/j)
```

Enums by Example

```
show_result :: MaybeInt -> String
show_result Nothing = “No Int”
show_result Just i = “Int of ” ++ (show i)
```

```
show_result :: MaybeInt -> String
show_result x =
  case x of
    Nothing -> “No Int”
    Just I -> “Int of ” ++ (show i)
```

Just like lists!

Lists as Sum Types?

```
data List a =
    Nil
  | Cons (a,List a)
```

Trees as Sum Types

```
data Tree a =
    Leaf
  | Node (Tree a,a,Tree a)
```

Rose Trees

```
data RoseTree a =
    Leaf
  | Node (Int,List (RoseTree a))

data List a
    Nil
  | Cons (a,List a)
```

Types

```
data Types =
    Base Int
  | Product (List Types)
  | Sum (List (String,Type))
  | Arrow (Type,Type)
  | Poly (String,Type)
  | Variable String
```

How would we do this in OO language

- Enums?
  - No
- Objects?
  - Not well
- Untyped Languages?
  - Yes, but then you lose typing