COMP3141

Curtis Millar
CSE, UNSW (and Data61)

22 July 2020

Exercise 5

- Parse a serie
- Stack push a
- Evaluate a se
- Calculate a string.

Recall: GADTs

Generalised Algebraic Datatypes (*GADTs*) is an extension to Haskell that, among other things, allows dat

```
{-# LANGUAGE GA
-- Unary natura
data Nat = Z | S Nat
-- is the same as
data Nat :: * where
  Z :: Nat
  S :: Nat -> Nat
```

The `printf` function

Consider the well known C function `printf`:

```
pri
```

In C, the type (and number) of parameters passed to this functio

the first parameter (the format string).

The printf function

To do a similar thing in Haskell, we would like to use a fancier type that allows us to define a function whose subsequent parameter is determined by the first.

```
data Format :: * -> * w
  End :: Format (
  Str :: Format t -
  Dec :: Format t -> Format (Int, t)      -- %d
  L :: String -> Format t -> Format t
  deriving instance Show (Format ts)
  -- just like deriving (Show) for normal data types.
```

Our format strings are indexed by a tuple type containing all of the types of the %directives used.

The printf function

is written:

```
L "Hello" $ Str $ L " You are "
     $ Dec $ L " years old!" End
```

The printf function

```
printf :: Format ts -> ts -> IO ()
printf End () =
    pure ()
printf (Str fmt
    do putStr s
printf (Dec fmt) (i,ts) =
    do putStr (show i); printf fmt ts
printf (L s fmt) ts =
    do putStr s; printf fmt ts
```

Vectors

Define a natural number kind to use on the type level:

```
data Nat = Z | S Nat
```

Our length-inde

```
data Vec (a :: *) :: Na
  Nil  :: Vec a Z
  Cons :: a -> Vec a n -> Vec a (S n)
```

The functions hd and tl can be total:

```
hd :: Vec a (S n) -> a
hd (Cons x xs) = x
tl :: Vec a (S n) -> Vec a n
tl (Cons x xs) = xs
```

Vectors, continued

Our map for vectors is as follows:

```
mapVec :: (a -> b) -> Vec a n -> Vec b n
mapVec f Nil = Nil
mapVec f (Cons x xs
```

Properties

Using this type, it's impossible to write a mapVe [illegible] gth of the vector.

Properties are verified by the compiler!

Appending Vectors

Example (Problem)

```
appendV :: Vec a m -> Vec a n -> Vec a ???
```

We want to write for kind Nat.

We can define a nor

```
plus :: Nat -> Nat -> Nat
plus Z y = y
plus (S x) y = S (plus x y)
```

This function is not applicable to type-level Nats, though.

⇒ we need a type level function.

Type Families

Type level functions, also called *type families*, are defined in Haskell like so:

```
{-# LANGUAGE Ty
type family Plu
  Plus Z     y = y
  Plus (S x) y = S (Plu
```

We can use our type family to define appendV:

```
appendV :: Vec a m -> Vec a n -> Vec a (Plus m n)
appendV Nil         ys = ys
appendV (Cons x xs) ys = Cons x (appendV xs ys)
```

Concatenating Vectors

Example (Problem)

```
concatV :: Vec (Vec a m) n -> Vec a ???
```

We want to write `m * n` in the `???` above, but we do not have times defined for kind `Nat`.

```
{-# LANGUAGE Ty[illegible]
type family Times (a :: Nat) (b :: Nat) :: Nat where
  Times Z n = Z
  Times (S m) n = Plus n (Times m n)
```

We can use our type family to define `concatV`:

```
concatV :: Vec (Vec a m) n -> Vec a (Times n m)
concatV Nil = Nil
concatV (Cons v vs) = v `appendV` concatV vs
```

Filtering Vectors

Example (Prob

```
filterV :: (a -> Bo
```

What is the size of the result of filter?

Filtering Vectors

Example (Problem)

```
filterV :: (a -> Bo
```

We do not know the s

We can use our type

```
filterV :: (a -> Bool) -> Vec a n -> [a]
filterV p Nil = []
filterV p (Cons x xs)
                  | p x  = x : filterV p xs
                  | otherwise = filterV p xs
```

Homework

1. Assignme).
2. Week 7's qui
3. The sixth pr
4. This week's quiz is also up, it's due Friday week (in 9 days).

Consultations

- Consultations will be made on request. Ask on piazza or email `cs3141@c`
- If there is a con number for

- Will be in the Thursday lecture slot, 9am to 11am on Blackb
- Make sure to join the queue on Hopper. Be ready to share you (`ghci` or `stack repl`) and editor set up.