CSE130 Discussion Section: Recursive data types

4.24.20

Algebraic data types

Sum type: (One Of)

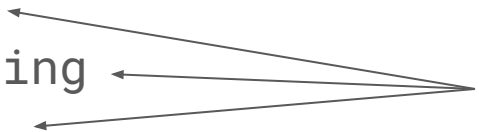
```
data OneOf
  = Left Int
  | Center String
  | Right Bool
```

```
Left 5          :: OneOf
Center "Hello"  :: OneOf
Right False     :: OneOf
```

Algebraic data types

Sum type: (One Of)

```
data OneOf
  = Left Int
  | Center String
  | Right Bool
```



Each of these is a function!

Quiz

Given this definition:

```
data OneOf
  = Left Int
  | Center String
  | Right Bool
```

What would GHCI say about, or what is the type of:

```
ghci> :t Center
```

A. `OneOf`
B. `Int -> OneOf`
C. `String -> OneOf`
D. `String`
E. `It has no type`

Quiz

Given this definition:

```
data OneOf
  = Left Int
  | Center String
  | Right Bool
```

What would GHCI say about, or what is the type of:

```
ghci> :t Center
```

A. `OneOf`
B. `Int -> OneOf`
C. `String -> OneOf`
D. `String`
E. `It has no type`

Algebraic data types

Product type (all-of):

```
data Prod
  = C T1 T2 T3

data Prod = C {
  field1 :: T1,
  field2 :: T2,
  field3 :: T3
}

field1 :: Prod -> T1
```

This is equivalent to a 3-tuple:

```
toProd :: (T1, T2, T3) -> Prod
toProd (t1, t2, t3) = C f1 f2 f3

fromProd :: Prod -> (T1, T2, T3)
fromProd (C t1 t2 t3) = (t1, t2, t3)
```

Quiz

Product type (all-of):

```
data Prod
  = C T1 T2 T3

data Prod = C {
  field1 :: T1,
  field2 :: T2,
  field3 :: T3
}

field1 :: Prod -> T1
```

What is the type of `C` ?
What does GHCI say about:

```
ghci> :t C
```

```
A.  T1 -> T2 -> T3 -> Prod
B.  (T1, T2, T3) -> Prod
C.  Prod
D.  T1 -> Prod
E.  Error
```

Quiz

Product type (all-of):

```
data Prod
  = C T1 T2 T3

data Prod = C {
  field1 :: T1,
  field2 :: T2,
  field3 :: T3
}

field1 :: Prod -> T1
```

What is the type of C ?
What does GHCI say about:

```
ghci> :t C
```

A. T1 -> T2 -> T3 -> Prod
B. (T1, T2, T3) -> Prod
C. Prod
D. T1 -> Prod
E. Error

Algebraic data types

```
data Color = Red | Blue | Green

data Bool = True | False

data Foo = Color | Bool

data Bar = Bar { color :: Color, bool :: Bool }
```

How many values of type `Foo` are there?
How many values of type `Bar`?

Recursive ADTs

Product types can reference themselves!

```
data List
  = Nil
  | Cons Int List

data Tree
  = Empty
  | Node Int Tree Tree
```

This works like a linked list:

```
struct Node {
    int data;
    struct Node* next;
};
```

(`NULL` next pointer corresponds to `Nil`)