Truth Tables

Professor William L. Harrison

September 7, 2018

Today

List data type

```
data [a] = [] | a : [a]
```

Discussion

- Two data constructors: empty list, `[]`, and "cons", "`:`"
- E.g., "`[1,2,3]`" is really "`1 : (2 : (3 : []))`"

Patterns are data *destructors*

like data constructors, defined by `data`

- E.g., the data constructors `C1` and `C2` are also used in patterns:

```
data T = C1 A | C2 B
```

Patterns are data *destructors*

like data constructors, defined by `data`

- E.g., the data constructors `C1` and `C2` are also used in patterns:

  ```
  data T = C1 A | C2 B
  ```

- This allows us to define by the structure of the input data:

  ```
  foo :: T -> ...
  foo (C1 a) = ...a...
  foo (C2 b) = ...b...
  ```

  Pattern "peels off the constructor".

Patterns are data *destructors*

like data constructors, defined by `data`

- E.g., the data constructors `C1` and `C2` are also used in patterns:

```
data T = C1 A | C2 B
```

- This allows us to define by the structure of the input data:

```
foo :: T -> ...
foo (C1 a) = ...a...
foo (C2 b) = ...b...
```

Pattern "peels off the constructor".

- E.g., `length`:

```
length []     = 0
length (x:xs) = 1 + length xs
```

Case Expressions

- Equivalent definition of `length`:

```
length l = case l of
             []     -> 0
             (x:xs) -> 1 + length xs
```

Case Expressions

- Equivalent definition of `length`:

```
length l = case l of
             []     -> 0
             (x:xs) -> 1 + length xs
```

- Definitional Extension: The definition on the previous slide is just *syntactic sugar* for this one.

Case Expressions

- Equivalent definition of `length`:

```
length l = case l of
             []     -> 0
             (x:xs) -> 1 + length xs
```

- Definitional Extension: The definition on the previous slide is just *syntactic sugar* for this one.
- Another example:

```
not :: Bool -> Bool
not b = case b of
          True  -> False
          False -> True
```

Case Expressions

- Equivalent definition of `length`:

```
length l = case l of
             []     -> 0
             (x:xs) -> 1 + length xs
```

- Definitional Extension: The definition on the previous slide is just *syntactic sugar* for this one.
- Another example:

```
not :: Bool -> Bool
not b = case b of
          True  -> False
          False -> True
```

- Syntactic Banana Peel: clauses "`True -> False`" and "`False -> True`" need to line up column-wise.

Graceless Error Reporting

```
ghci> :t error
```

Graceless Error Reporting

```
ghci> :t error
  error :: [Char] -> a

ghci> :t undefined
```

Graceless Error Reporting

```
ghci> :t error
  error :: [Char] -> a

ghci> :t undefined
  undefined :: a

ghci> error "whoops"
```

Graceless Error Reporting

```
ghci> :t error
  error :: [Char] -> a

ghci> :t undefined
  undefined :: a

ghci> error "whoops"
  *** Exception: whoops

ghci> 1 + (2 * error "whoops")
```

Graceless Error Reporting

```
ghci> :t error
  error :: [Char] -> a

ghci> :t undefined
  undefined :: a

ghci> error "whoops"
  *** Exception: whoops

ghci> 1 + (2 * error "whoops")
  *** Exception: whoops
```

Local Definitions with `where`

`a` and `b` are local to `ex`

```
-- "(A => (-A => B))"
ex :: Prop
ex = Imply a (Imply (Not a) b)
  where
    a = Atom "A"
    b = Atom "B"
```

Safe lookup

```
ghci> :t lookup
  lookup :: Eq a => a -> [(a, b)] -> Maybe b
```

Safe lookup

```
ghci> :t lookup
  lookup :: Eq a => a -> [(a, b)] -> Maybe b
ghci> lookup "A" [("A",True),("B",False)]
```

Safe lookup

```
ghci> :t lookup
  lookup :: Eq a => a -> [(a, b)] -> Maybe b
ghci> lookup "A" [("A",True),("B",False)]
  Just True
ghci> lookup "B" [("A",True),("B",False)]
```

Safe lookup

```
ghci> :t lookup
  lookup :: Eq a => a -> [(a, b)] -> Maybe b
ghci> lookup "A" [("A",True),("B",False)]
  Just True
ghci> lookup "B" [("A",True),("B",False)]
  Just False
ghci> lookup "C" [("A",True),("B",False)]
```

Safe lookup

```
ghci> :t lookup
  lookup :: Eq a => a -> [(a, b)] -> Maybe b
ghci> lookup "A" [("A",True),("B",False)]
  Just True
ghci> lookup "B" [("A",True),("B",False)]
  Just False
ghci> lookup "C" [("A",True),("B",False)]
  Nothing
```