COMP0020 Functional Programming

Lecture 11

Algebraic Types

Contents

- Motivation : why do we need more types ?
- Type domains revisited
- Define your own types !
- Constructors
- Usage
- Different from type synonyms !

Motivation

- Types seen so far : char, num, bool, *
- Type constructors seen so far : (), [], ->
- Why do we need more types?
  - Readability
  - Built-in validation
- Example : if a type called "dice" has only six legal values (One . . . Six), then :
  - f : : dice -> bool is more informative
  - A value that is **not** one of the legal values (One . . . Six) will be detected as a type error

Type Domains Revisited

- We can **define** a type by listing the legal values permitted for that type
- The type bool has just two legal values : True and False
- The type [char] has potentially infinite legal values : [], ['A'], ['B'], ['h','e','l','l','o'] etc
- Pattern-matching allows equality checks against these legal values

Define your own Types

$mybool ::= Mytrue \mid Myfalse$

MUST use capitals!!!!!

(*the set of legal values for this "algebraic type"*)


```
f :: bool -> num
f True  = 34
f any = 13

g :: mybool -> num
g Mytrue  = 34
g any  = 13
```

Constructors

- The wrong way :
  - wrong_dice := 1|2|3|4|5|6
  - also_wrong_dice := "one" | "two" | "three" | "four" | "five" | "six"
- The correct way :
  - dice : := *One* | *Two* | *Three* | *Four* | *Five* | *Six*
- Constructors start with a capital letter
  - Built-in type bool adheres to this rule
  - Built-in types num and char break the rule (for convenience)

Constructors (2)

- Each legal value must have a constructor
- Each legal value may also have some extra data (if so, we specify its type following the constructor) :

$fluid ::= Gallons\ num \mid Litres\ num$

$tanklevel ::= Emptytank \mid Fulltank \mid Parttank\ fluid$

$x :: tanklevel$

$x = Parttank\ (Gallons\ 3)$

Usage


```
coord3D :== Coord (num, num, num)

origin :: coord3D
origin = Coord (0, 0, 0)

midpoint :: coord3D -> coord3D -> coord3D
midpoint (Coord (x1, y1, z1)) (Coord (x2, y2, z2))
  = Coord ( (x1 + x2)/2, (y1 + y2)/2, (z1 + z2)/2 )
```

Usage (2)

$$studentdata ::= StudentByname\ [char]\ num$$
$$\mid StudentByregid\ num\ \ num$$

$$topmarks :: [studentdata] \to [num]$$
$$topmarks\ [] = []$$
$$topmarks\ ((StudentByname\ any\ x)\ :\ rest) = (x\ :\ (topmarks\ rest)),\ if\ x >= 70$$
$$= topmarks\ rest,\ otherwise$$
$$topmarks\ ((StudentByregid\ id\ x)\ :\ rest) = (x\ :\ (topmarks\ rest)),\ if\ x >= 70$$
$$= topmarks\ rest,\ otherwise$$

Different from type synonyms


```
syncoord == (num, num, num)

x :: syncoord
x = (3, 4, 16)
```



```
coord3D == Coord (num, num, num)
y :: coord3D
y = Coord (3, 4, 16)
```

Summary

- Motivation : why do we need more types ?
- Type domains revisited
- Define your own types !
- Constructors
- Usage
- Different from type synonyms

END OF LECTURE