An Introduction to Functional Programming

Intermediate Computer Science Pre-College Program
23-27 July 2018
– Module 4 –

Topics for Today

- Algebraic Data Types (eg. enumerations and trees)
- Programming with Trees
- Function Schemes (map, iter, fold) over Trees

A First Example – Days of Week

- Algebraic data types (or unions or variants)
  代数的
- They allow for user-defined types

```
# type dow =
  | Monday
  | Tuesday
  | Wednesday
  | Thursday
  | Friday
  | Saturday
  | Sunday;;
type dow = Monday | Tuesday | Wednesday | Thursday | Friday
    | Saturday | Sunday
```

Days of Week

```
# type dow =
    | Monday
    | Tuesday
    | Wednesday
    | Thursday
    | Friday
    | Saturday
    | Sunday;;
type dow = Monday | Tuesday | Wednesday | Thursday | Friday
     | Saturday | Sunday
# Monday;;
- : dow = Monday
# Sunday;;
- : dow = Sunday
```

Example Function over an ADT

```
1  let is_weekend d =
2    match d with
3    | Saturday -> true
4    | Sunday -> true
5    | _ -> false
6
7  let is_weekend2 d =
8    match d with
9    | Saturday | Sunday -> true
10   | _ -> false
11
12 let is_weekend3 = function
13   | Saturday | Sunday -> true
14   | _ -> false
```

- All three are equivalent

Example Function of an ADT

```
# let next_day = function
   | Monday -> Tuesday
   | Tuesday -> Wednesday
   | Wednesday -> Thursday
   | Thursday -> Friday
   | Friday -> Saturday
   | Saturday -> Sunday
   | Sunday -> Monday;;
val next_day : dow -> dow = <fun>
# next_day Monday;;
- : dow = Tuesday
```

Another Example of a Function on an ADT

```
1  (* next_day : dow -> dow *)
2  let next_day d =
3    match d with
4    | Monday -> Tuesday
5    | Tuesday -> Wednesday
6    | Wednesday -> Thursday
7    | Thursday -> Friday
8    | Friday -> Saturday
9    | Saturday -> Sunday
10   | Sunday -> Monday
```

Another Example of an ADT

```
1 type course =
2   | UGrad of string * int * (dow list)  (* name, enrollment,
      schedule *)
3   | Grad of string * int * (dow list)
```

Represents a course including

- Whether it is an undergrad or grad course
- Its name
- its current enrollment
- Its schedule

```
1 UGrad ("data structures", 23, [Monday; Wednesday])
```

Example: List the names of all the courses in a list of courses

```
1 (* Given a course, returns its name
2    get_name : course -> string  *)
3 let get_name c =
4   match c with
5   | UGrad(name,_,_) -> name
6   | Grad(name,_,_) -> name
```

or just

```
1 let get_name c =
2   match c with
3   | UGrad(name,_,_) | Grad(name,_,_) -> name
```

Example

Determine all the days in which there are courses running

```
1 (* scheduled_days : course * list -> dow * list  *)
2 let rec scheduled_days cs =
3   match cs with
4   | [] -> []
5   | (c:cs') -> (get_schedule c) @ (scheduled_days cs')
```

Exercise 1

- Write a function `max_enrollment` that returns the course with the maximum number of registrants
  - Assume the list of courses is non-empty
- What is the type of this function?

Exercise 2

- Write a function `most_scheduled_day` that returns the day in which most courses are scheduled
- What is the type of this function?

Exercise 3

- Write a function `delete_course` that given a course name and a list of courses returns a new list resulting from dropping that course
- What is the type of this function?

Binary Tree of Integers

Def. *A binary tree of integers is either the empty tree or a node that has a number and two binary trees of integers, namely its left and right subtrees*

- Lets draw some examples of binary trees of numbers on the board

Binary Tree of Integers

```
1 type treei =
2   | Empty
3   | Node of int * treei * treei
```

A binary tree of integers is either the empty tree or a node that has a number and two binary trees of integers, namely its left and right subtrees

Binary Tree of Integers

```
type treei =
  | Empty
  | Node of int * treei * treei
```

A binary tree of integers is either the empty tree or a node that has a number and two binary trees of integers, namely its left and right subtrees

An example tree

```
Node(1,Node(2,Empty,Empty),Node(3,Empty,Empty))
```

Lets draw it on the board

An Example of a Function over Binary Trees of Integers

```
1 let rec size t =
2 match t with
3 | Empty -> 0
4 | Node(i,lt,rt) -> i + size lt + size rt
```

- What is the type of this function?
- Apply it to the following tree:

```
1 Node(1,Node(2,Empty,Empty),Node(3,Empty,Empty))
```

Exercises: Simple Functions on `treei`

- `sum`: adds all the numbers in a `treei`
- `product`: multiplies all the numbers in a `treei`
- `max`: returns the maximum number in a non-empty `treei`
- `min`: returns the minimum number in a non-empty `treei`
- `isEmpty`: returns a boolean indicating whether the argument `treei` is empty or not
- `isLeaf`: returns a boolean indicating whether the argument `treei` is a leaf (what is a leaf?)
- `height`: returns the height of a `treei`

Exercises: Functions on `treei` that Return a List

- preorder

Exercises: Functions on `treei` that Return `treei`

- `increment_tree`
- `double_tree`
- `mirror_tree`

Binary Search Trees

- A `treei` that has an additional condition: in a tree of the form `Node(i,lt,rt)`, `i` is greater than every number in `lt` and smaller than every number in `rt`
- Lets look at some examples

BSTs

- `is_bst : treei -> bool`: hint: use `min` and `max`
- `find_bst : treei -> int -> bool`
- `add_bst : treei -> int -> treei`, fails of number already is in the tree
- `remove_bst : treei -> treei`: fails if number is not in the tree (non-trivial!)

Polymorphic Binary Trees

Binary trees with data of arbitrary type

```
type 'a tree =
  | Empty
  | Node of 'a * 'a tree * 'a tree
```

- A tree of numbers: (type is `int tree`)

```
Node(1,Node(2,Empty,Empty),Node(3,Empty,Empty))
```

- A tree of strings: (type is `string tree`)

```
Node("hello",Node("bye",Empty,Empty),Node("hey",Empty,
    Empty))
```

- A tree of booleans: (type is `bool tree`)

```
Node(true,Node(false,Empty,Empty),Node(true,Empty,Empty
    ))
```

Functions over `tree`

```
1 let rec size_tree t =
2 match t with
3 | Empty -> 0
4 | Node(x,lt,rt) -> 1 + size_tree lt + size_tree rt
```

- What is the type of this function?

Functions Schemes over Trees

- Map, iter and fold over binary trees

Summary

- Algebraic data types
- Trees
- Binary search trees
- Function schemes over trees