Midterm 1 Review Sheet

Sunday, September 22, 2019 11:53 PM

Primitive Data types:

Int:

- +, -, *, /
- mod (remainder function) 5 mod 2 = 1
- string_of_int

Bool:

- not (negation)
- &&, || (and / or)

String

- "\n" (new line)
- ^ (concatenation)

Function Calls:

$\underbrace{\text{let}}_{\text{let key word}}\ \underbrace{\text{attendees}}_{\text{function name}}\ \underbrace{\text{(price:int)}}_{\text{inputs with type}} =$

Identifier:

$\underbrace{\text{let}}_{\text{let key word}}\ \underbrace{\text{x : int}}_{\text{name with type}} = \underbrace{\text{attendees 500}}_{\text{function call}}$

Colon notation identifies a type

Pattern matching:

$\mid \underbrace{\text{\_\_}}_{\text{case}} \to \underbrace{\text{\_\_}}_{\text{action}}$

✱ Use wild card if value doesn't matter

Test cases

;; Open Assert

let test () : bool =

(2+3) = 5

;; run_test "2+3=5 test" test

(or run_failing_test if test should encounter a failwith statement)

Lists:

[ ] = empty or nil

head :: tail (head followed by a tail)

Arbitrary length, single type

Tuples:

(3, true) Type = int * bool

Fixed length with different data types

(3, true) Type= int * bool

fixed lengths with different data types

Defining a data type:

keyword (type Day = name of type

| Sunday
| Monday } Pattern match w/ options

type week= day * day * day ....

Trees:

- user defined type
- node or leaf

Binary Tree

- either empty or node w/ 2 children
- leaf is a node where both children are empty
- don't need to be balanced

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

Binary Search Tree (BST)

- must be a binary tree
- all of subtrees must be BSTs
- values on left must be < than current node, values on right must be > than current node

Insert

- Is tree empty or not?
  - if its empty, add a node
  - if not
    - does x=n -> return same tree (element already in tree)
    - is x<n -> node (insert x left, n, right)
    - is x>n -> node (left, n, insert x right)

} Must preserve invariance

- is x > n → node (left, n, insert x right)

[illegible]

Delete

- Is tree empty or not?
  - yes return tree
  - no?
    - is x < n → Node (delete x left, n, right)
    - is x > n → Node (left, n, delete x right)
    - is x = n
      - are n's children empty empty → Empty
      - are n's children left empty → left
      - are n's children empty right → right
      - are n's children left right:
        find max in left (y) → Node (delete y left, y, right)

Generic Types

- 'a (tic a)
  can be any lower case letter
- 'a, 'a list, 'a * int ... etc
- in a function, every call of 'a has to be the same type
- 'a and 'b could be different types but they don't have to be
- better, reusable code

First Class Functions

- $t_{in} \rightarrow t_{out}$
- int → int → int (int to int to int)

  ≠

  (int → int) → int

- partial applications
  - sum (x:int) (y:int) = x+y   (int -> int -> int)

let z = sum 3 in (int -> int)

- anonymous functions fun (y:int) -> y+3
  function w/out name
- transform
  - fun } key word
  - take everything in a list and perform some operation on it
    'a list -> 'b list

- First class functions
  - fold
    - takes everything in a list and performs an action on it
      'a list -> 'b

Modularity / Abstraction

- Sets
  - like a list but order doesn't matter
  - no duplicate values

- Abstract type
  - -> Interface -> defines types / operations
  - -> Properties -> defines how operations act w/ eachother

- Abstract because interface reveals nothing about implementation

- Signature   ↙ interface declaration

module type Set = sig

↳ module provides a few Data types and their associated operations

- struct → for an implementation of the interface .9