Concept of Programming Languages (CS320)
Lecture 3

By Zhiqiang Ren (Alex)
aren@cs.bu.edu
Content

- Correction
- Operations of List
- Reference X Value X Object
- Assignment
- Graph algorithm (list implementation)
- Quiz
Correction

- Compound expression

```
// all exp except the last one must be of type void
begin exp1; exp2; ...... ; expn end
```

```
// all exp except the last one must be of type void
(exp1; exp2; ...... ; expn)
```
Operations of List

- Think of list as an abstraction / interface.
- Operations ($ATSHOME/prelude/SATS/listo.sats)

```ocaml
fun{a:t@ype} list0_head_exn (xs: list0 a): a
fun{a:t@ype} list0_length (xs: list0 a):<> int
fun{a:t@ype} list0_nth_exn (xs: list0 a, i: int): a
fun{a:t@ype} list0_reverse (xs: list0 a): list0 a
fun{a:t@ype} list0_reverse_append(xs: list0 a, ys: list0 a): list0 a
fun{a:t@ype} list0_tail_exn (xs: list0 a): list0 a
// take the first n
fun{a:t@ype} list0_take_exn (xs: list0 a, n: int): list0 a
// drop the first n
fun{a:t@ype} list0_drop_exn (xs: list0 a, n: int): list0 a
```
Reference X Value X Object

- What is value in ATS?
  - numbers
  - Objects of flat type
  - references
    - Object of boxed type
- Bind a name to a value

```scala
val x: int = 3
val tup1: '(int, int) = '(3, 4)
val tup2: @(int, int) = @(1, 2)
val y = x
val t = tup2
```
Reference X Value X Object

- Value is copied beyond the scope of function

val x:int = 3
val t1: '(int, int) = '(3, 4)
val t2: @(int, int) = @(1, 2)
fun call_by_value (x': int, t1': '(int, int), t2': @(int, int)): void = let
val ret = x + t1.0 + t2.0
in
ret
end
val z = call_by_value (x, t1, t2)
Assignment

- Assignment 1
- Assignment 2

```latex
datatype mylist (a: t@ype) =  
  | Nil (a)  
  | Cons (a) of (a, mylist (a))  
  | Append (a) of (mylist (a), mylist (a))  
  | Reverse (a) of mylist (a)

fun{a:t@ype} mylist_cons (x: a, xs: mylist a) = Cons (x, xs)

fun{a:t@ype} mylist_nil () = Nil ()

fun{a:t@ype} mylist_append (xs: mylist a, ys: mylist a) = Append (xs, ys)

fun{a:t@ype} mylist_reverse (xs: mylist a) = Reverse xs
```
Assignment

• mylist_length
  • \( \text{len (append (xs, ys)) equals to len (xs) + len (ys)} \)
  • \( \text{len (rev(xs)) equals to len (xs)} \)
• mylist_is_empty
• mylist_nth (index starts from 0)
  • \( xs[n] = (\text{rev(xs)})[\text{len(xs)} - n - 1] \)
• mylist2list0
Graph algorithm (list implementation)

- Representation of graph by list of pairs
- ("a", "b") :: ("a", "c") :: ("a", "d") :: ("b", "e") :: ("c", "f") :: ("d", "e") :: ("e", "f") :: ("e", "g") :: nil
Graph algorithm (list implementation)

- Depth First Search
- To remember the visited nodes
  - Mark the node (not feasible in functional programming)
  - Extra booking
    - record the node
    - check whether a node has been recorded
Graph algorithm (list implementation)

```plaintext
staload "prelude/DATS/list.dats"
staload "prelude/DATS/list0.dats"

#define :: list0_cons
#define nil list0_nil

typedef node = string
typedef edge = (node, node)
typedef graph = list0 edge

abstype set
extern fun set_new (): set
extern fun set_contains (s: set, n: node): bool
extern fun set_add (s: set, n: node): set

implement main () = let
  val g = ("a", "b") ::
            ("a", "c") :: ("a", "d") ::
            ("b", "e") :: ("c", "f") ::
            ("d", "e") :: ("e", "f") ::
            ("e", "g") :: nil
  in
    depth ("a", g)
end

extern fun depth (n: node, g: graph): void
```
Quiz

- Divide $r^2$ into $x^2 + y^2$
- Find all the possible pairs
- `fun factor (r: int): list0 (int, int)`
- Algorithm (Dijkstra 1976)
  - $(x, y)$  $x$ goes down from $r$, $y$ goes up from 0
    - $x^2 + y^2 < r^2$ then increment $y$ by 1, and move on
    - $x^2 + y^2 = r^2$ then record it, and move on (change $x$ and $y$)
    - $x^2 + y^2 > r^2$ then decrement $x$ by 1, and move on
    - $x < y$ then stop