List Processing in SML

CS251 Programming Languages
Fall 2018
Lyn Turbak
Department of Computer Science
Wellesley College

Solutions

List Processing in SML

Consing Elements into Lists

- val nums = 9 :: 4 :: 7 :: [];
val nums = [9,4,7] : int list
- 5 :: nums;
val it = [5,9,4,7] : int list
- nums;
val it = [9,4,7] : int list (* nums is unchanged *)
- (1+2) :: (3*4) :: (5-6) :: [];
val it = [3,12,1] : int list
- [1+2, 3*4, 5-6];
val it = [3,12,1] : int list
- [1=2, 3 < 4, false];
val it = [false, true, false] : bool list
- ["I", "do", String.substring ("note",0,3), "li" ^ "ke"];
val it = ["I", "do", "not", "like"] : string list
- ["#a", 8, ("#z", 5)];
val it = [[#a",8,(#z",5)] : (char * int) list
- [(7,2,5), [6], 9::[3,4]];
val it = [[7,2,5],[6],[9,3,4]] : int list list

Unlike in Racket & Python, all elements of an SML list must have the same type.
- 1 :: [2,3,4];
val it = [1,2,3,4] : int list
- op:: (1, [2,3,4]); (* op:: is prefix version of infix :: *)
val it = [1,2,3,4] : int list
- op:: ;
val it = fn : 'a * 'a list -> 'a list
- "a" :: [1,2,3];
stdin:1.1-8.3 Error: operator and operand don't agree [literal]
operator domain: string * string list
operand: string * int list
in expression: "a" :: 1 :: 2 :: 3 :: nil
- [1,2] :: [3,4,5];
stdin:9.1-9.17 Error: operator and operand don't agree [literal]
operator domain: int list * int list list
operand: int list * int list
in expression: (1 :: 2 :: nil) :: 3 :: 4 :: 5 :: nil

Tuples vs. Lists

Tuples are heterogeneous fixed-length product types:

- (1+2, 3-4, "foo" ^ "bar", String.substring ("baz", 2));
val it = (3,false,"foo barbar","#z") : int * bool * string * char

List are homogeneous variable-length product types:

- [1, 2+3, 4*5, 6-7, 8 mod 3];
val it = [1,5,20,~1,2] : int list
- [1-2, 3<4];
val it = [false,true] : bool list
- ("foo", "bar" ^ "baz", String.substring ("abcdefg", 2, 3));
val it = ["foo","barbaz","cde"] : string list
- [#"a", String.substring ("baz", 2), chr(100)];
val it = [#"a",#"z",#"d"] : char list
**Some Simple List Operations**

- `List.length [7,3,6,1];`
  ```
  val it = 4 : int
  ```
- `List.hd [7,3,6,1];`
  ```
  val it = 7 : int
  ```
- `List.tl [7,3,6,1];`
  ```
  val it = [3,6,1] : int list
  ```
- `List.take ([7,3,6,1],2);`
  ```
  val it = [7,3] : int list
  ```
- `List.take ([7,3,6,1],3);`
  ```
  val it = [7,3,6] : int list
  ```
- `List.drop ([7,3,6,1],2);`
  ```
  val it = [6,1] : int list
  ```
- `List.drop ([7,3,6,1],3);`
  ```
  val it = [1] : int list
  ```
- `List.nth ([7,3,6,1],1);`
  ```
  val it = 3 : int
  ```
- `List.nth ([7,3,6,1],2);`
  ```
  val it = 6 : int
  ```
- `List.nth ([7,3,6,1],3);`
  ```
  val it = false : bool
  ```
- `List.null [7,3,6,1];`
  ```
  val it = false : bool
  ```
- `List.nul [];
  ```
  val it = true : bool
  ```
- `[[7,8,1,6],[9]] = [];`
  ```
  val it = false : bool
  ```
- `List.rev [7,3,6,1];`
  ```
  val it = [1,6,3,7] : int list
  ```

(*) An API for all SMLNJ List operations can be found at: [http://www.standardml.org/Basis/list.html *] (*)

---

**Appending Lists**

- `[7,2] @ [8,1,6];`
  ```
  val it = [7,2,8,1,6] : int list
  ```
- `[7,2] @ [8,1,6] @ [9] @ [];
  ```
  val it = [7,2,8,1,6,9] : int list
  ```
- `op::; (* prefix cons function *)`
  ```
  val it = fn : "a" * "a" list -> "a" list
  ```
- `op@; (* prefix append function *)`
  ```
  val it = fn : "a" list * "a" list -> "a" list
  ```
- `List.concat all els in a list of lists *)`
  ```
  val it = [[7,2],[8,1,6],[9]] : int list
  ```
- `List.concat;
  ```
  val it = fn : "a" list list -> "a" list
  ```

---

**Pattern Matching on Lists**

```ml
(* matchtest : (int * int) list -> (int * int) list *)

fun matchtest xs =
  case xs of
  [] => []
  | [(a,b)] => [(b,a)]
  | (a,b) :: (c,d) :: zs => (a+c,b*d) :: (c,d) :: zs
```

- `matchtest [];`
  ```
  val it = [] : (int * int) list
  ```
- `matchtest [(1,2)];`
  ```
  val it = [(2,1)] : (int * int) list
  ```
- `matchtest [(1,2),(3,4)];`
  ```
  val it = [(4,8),(3,4)] : (int * int) list
  ```
- `matchtest [(1,2),(3,4),(5,6)];`
  ```
  val it = [(4,8),(5,6)] : (int * int) list
  ```

---

**Other Pattern-Matching Notations**

```ml
fun matchtest2 xs =
  case xs of
  [] => []
  | [(a,b)] => [(b,a)]
  | (a,b) :: (ys as ((c,d) :: zs)) => (a+c,b*d) :: ys
(* subpatterns can be named with "as" *)
```

```ml
fun matchtest3 [] = []
  | matchtest3 [(a,b)] = [(b,a)]
  | matchtest3 ((a,b) :: (ys as ((c,d) :: zs)))
  (* parens around pattern necessary above *)
    = (a+c,b*d) :: ys
```

---

(* An API for all SMLNJ List operations can be found at: [http://www.standardml.org/Basis/list.html *] *)
List Accumulation

(* Recursively sum a list of integers *)
(* sumListRec : int list -> int *)
fun sumListRec [] = 0
| sumListRec (x::xs) = x + (sumListRec xs)

val it = 0 : int
- sumListRec [5,2,4];
val it = 11 : int

(* Iterative (tail-recursive) summation *)
fun sumListIter xs =
let fun loop [] sum = sum
| loop (y::ys) sum = loop ys (y + sum)
end
-
- sumListIter [5,2,4];
val it = 11 : int

SML’s map

- map (* Same as List.map; available at top-level *)
val it = fn : ('a -> 'b) -> 'a list -> 'b
- map (fn x => x + 1) [5,2,4];
val it = [6,3,5] : int list
- map (fn y => y * 2) [5,2,4];
val it = [10,4,8] : int list
- map (fn z => z > 3) [5,2,4];
val it = [true,false,true] : bool list
- map (fn a => (a, (a mod 2) = 0)) [5,2,4];
val it = [(5,false),(2,true),(4,true)] : (int * bool) list
- map (fn s => s ^ "side") ["in", "out", "under"];
val it = ["inside", "outside", "underside"] : string list
- map (fn xs => 6::xs) [[7,2],[3],[8,4,5]];
val it = [[6,7,2],[6,3],[6,8,4,5]] : int list list

Some Other Higher-Order List Ops

(* List.partition : ('a -> bool) -> 'a list * 'a list * splits a list into two: those elements that satisfy the predicate, and those that don’t *)
- List.partition (fn x => x > 0) [3, ~7, ~6, 8, 5];
val it = ([3,8,5],[-7,-6]) : int list * int list
- List.partition (fn y => (y mod 2) = 0) [5,2,4,1];
val it = ([2,4],[5,1]) : int list * int list
(* List.all : ('a -> bool) -> 'a list -> bool returns true iff the predicate is true for all elements in the list. *)
- List.all (fn x => x > 0) [5,2,4,1];
val it = true : bool
- List.all (fn y => (y mod 2) = 0) [5,2,4,1];
val it = false : bool
(* List.exists : ('a -> bool) -> 'a list -> bool returns true iff the predicate is true for at least one element in the list. *)
- List.exists (fn y => (y mod 2) = 0) [5,2,4,1];
val it = true : bool
- List.exists (fn z => z < 0) [5,2,4,1];
val it = false : bool
Zipping in SML

(* 'a list * 'b list -> ('a * 'b) list *)
(* Note that input is a *tuple* of lists! *)
- ListPair.zip ("a", [1, 2, 3, 4]);
val it = [("a", 1), ("b", 2), ("c", 3)]; (* string * int *)

(* ('a * 'b) list -> 'a list * 'b list *)
- ListPair.unzip [("a", 1), ("b", 2), ("c", 3)];
val it = (["a", "b", "c"], [1, 2, 3]);

(* An API for all SMLNJ ListPair operations can be found at: http://www.standardml.org/Basis/list-pair.html *)

List Processing in SML 13

foldr : The Mother of All List Recursions

- List.foldr (* Same as List.foldr; available at top-level *)
  (* Note that combiner takes *tupled* args! *)
val it = fn : ('a * 'b list) -> 'b -> 'b list
val it = elem : ('a list, 'b) -> 'b list
val it = foldr op+ 0 [5, 2, 4];
val it = foldr op+ 0 [5, 2, 4];
val it = foldr op+ 0 [5, 2, 4];
val it = foldr (fn (x, y) => x * y) 1 [5, 2, 4];
val it = foldr (fn (x, y) => x andalso y) true [true, false, true];
val it = foldr (fn (x, y) => x orelse y) false [true, false, true];
val it = foldr (fn (x, y) => (x > 0) andalso y) true [5, 2, 4];
val it = foldr (fn (x, y) => (x < 0) orelse y) false [5, 2, 4];

Your Turn with SML’s higher-order ops

fun sumSquaresEvens ns = (* use foldr, map, List.filter *)
foldr op+ 0
(map (fn n => n * n)
(List.filter (fn n => n mod 2 = 0)
ns));
val it = 140 : int

fun myReverse xs = foldl op:: [] xs (* use List.foldl *)
myReverse [7, 6, ~2, ~9, 10];
val it = 140 : int

fun consecutiveProds ns = (* use map, List.zip *)
map op* (if List.null ns
then []
else ListPair.zip(ns, (List.tl ns)));
val it = 42, ~12, 18, ~90 : int list

List Processing in SML 15 14 16