Σύντομη Εισαγωγή στην SML/NJ

Φυλλάδιο σημειώσεων για το 1ο εργαστήριο του μαθήματος

1 Εγκατάσταση και πρώτη γνωριμία

Η πιο πρόσφατη έκδοση της SML/NJ τη στιγμή συγγραφής αυτού του κειμένου υπάρχει στο:

`https://www.smlnj.org/dist/working/110.96/index.html`

Windows

- `https://www.smlnj.org/dist/working/110.96/windows.html`
- Start → Programs → SML of New Jersey
- Start → Run, "cmd", "sml"

Unix

- Debian/Ubuntu: `apt-get install smlnj smlnj-doc rlwrap`
- Εκτέλεση από terminal με `sml` ή `rlwrap sml`
- Εναλλακτικά `mlton` (`www.mlton.org`)

2 Χρήση του toplevel

```
Standard ML of New Jersey (64-bit) v110.96 [built: Fri Dec 13 14:59:34 2019]
- print "Hello world!\n";
Hello world!
val it = () : unit
- 13 + 29;
val it = 42 : int
- ~it;
val it = ~42 : int
```

E: Τι είναι το it;

A: Μια ειδική μεταβλητή του toplevel που περιέχει πάντα το αποτέλεσμα της τελευταίας έκφρασης.

E: Πώς θα αλλάξω την τιμή μιας μεταβλητής;

A: Στην ML δεν υπάρχει πολλαπλή ανάθεση. Οι μεταβλητές απλά αντιστοιχούν ονόματα σε τιμές και ο τελεστής = χωρίς χρήση του `val` ελέγχει ισότητα.

```
- val x = 3;
val x = 3 : int
- x = x + 1;
val it = false : bool
```

E: Πώς θα γράψω ένα loop;

A: Πολύ εύκολα, όπως σε όλες τις Turing complete γλώσσες!

```
- fun loop x = loop x;
val loop = fn : 'a -> 'b
```

E: Χα χα, δεν εννοούσα αυτό. Εννοούσα πως μπορώ να γράψω κάτι σαν τα for loops της C;

A: Εξίσου απλά, με χρήση αναδρομής:

```
- fun for 0 = ()
=   | for n = for (n-1);
val for = fn : int -> unit
```

E: Χμμμ... θα μου πάρει λίγο καιρό να συνηθίσω αυτό το στυλ προγραμματισμού. Τι στο καλό είναι αυτό το `unit`;

A: Σημαίνει ότι η συνάρτηση αυτή δεν επιστρέφει κάποια τιμή — είναι κάτι σαν το `void` στη C. Με άλλα λόγια, η συνάρτηση θα εκτελεστεί μόνο για τις παρενέργειές της.

E: Παρενέργειες; Μα η παραπάνω συνάρτηση δεν κάνει πολλά πράγματα...

A: Δίκιο έχεις. Ας γράψουμε λοιπόν μια συνάρτηση που να τυπώνει τα τετράγωνα όλων των φυσικών από το 0 μέχρι το n, το κάθε ένα σε διαφορετική γραμμή, ξεκινώντας από το μεγαλύτερο:

```
- fun fordown n =
=   if n < 0 then ()
=   else (print (Int.toString (n*n)); print "\n"; fordown (n-1));
val fordown = fn : int -> unit
- fordown 4;
16
9
4
1
0
val it = () : unit
```

E: Και αν ήθελα να τυπώσω τα τετράγωνα ξεκινώντας από το 0;

A: Καλή ερώτηση. Αυτό μπορεί να γίνει με δύο τρόπους. Ο ένας είναι να χρησιμοποιήσεις μια επιπλέον μεταβλητή για να αρχίσεις την επανάληψη από το μηδέν και μία για να ξέρεις που πρέπει να σταματήσεις.

```
- fun forup n lim =
=   if n > lim then ()
=   else (print (Int.toString (n*n)); print "\n"; forup (n+1) lim);
val forup = fn : int -> int -> unit
- forup 0 4;
0
1
4
9
16
val it = () : unit
```

E: Και ο άλλος τρόπος;

A: Ο δεύτερος τρόπος είναι να χρησιμοποιήσεις το σχήμα της αρχικής συνάρτησης `fordown` αλλά με μια δομή δεδομένων (π.χ. μια λίστα) ως accumulator όπου θα βάζεις τα τετράγωνα των αριθμών εκεί, όπως αυτά παράγονται.

```
- fun forlst n acc =
=   if n < 0 then acc
=   else forlst (n-1) ((n*n)::acc);
val forlst = fn : int -> int list -> int list
- forlst 4 [];
val it = [0,1,4,9,16] : int list
```

E: Ναι αλλά η παραπάνω συνάρτηση απλώς δημιουργεί τη λίστα, δεν την τυπώνει...

A: Ε, κάνε και κάτι μόνος σου: γράψε αυτή τη συνάρτηση για εξάσκηση. Α, μια και θα γράφεις, γράψε και τις παρακάτω συναρτήσεις.

```
fun make_list n =
  let fun loop 0 result = result
        | loop i result = loop (i-1) (i::result)
  in
    loop n []
  end

fun len [] = 0
  | len (h::t) = 1 + len t

(* Tail recursive version of the len function below *)
fun len2 l =
  let fun aux [] i = i
        | aux (h::t) i = aux t (i+1)
  in
    aux l 0
  end
```

E: Πως θα σώσω και θα φορτώσω στον interpreter τον κώδικά μου;

A: Γράφουμε με τον editor της επιλογής μας τον κώδικά μας και τον σώζουμε σε ένα αρχείο. Στη συνέχεια, μέσα στον interpreter χρησιμοποιούμε τη συνάρτηση `use` που παίρνει ως όρισμα ένα string με το όνομα του αρχείου που θέλουμε να φορτώσουμε.

```
- use "myfile.sml";
[opening myfile.sml]
val make_list = fn : int -> int list
val len = fn : 'a list -> int
val len2 = fn : 'a list -> int
val it = () : unit
- len [1,2,3];
val it = 3 : int
```

E: Και αν το αρχείο είναι σε άλλο directory;

A: Σε αυτήν την περίπτωση, είτε δίνουμε το απόλυτο όνομα του αρχείου, είτε χρησιμοποιούμε τις συναρτήσεις `OS.FileSys.getDir()` και `OS.FileSys.chDir("path")`, για να δούμε που βρισκόμαστε και να περιηγηθούμε στους καταλόγους του συστήματος.

E: Και αν βαριέμαι να γράφω `use ...` κάθε φορά που ξεκινάω την `sml`;

A: Μπορείς να φορτώσεις κατ' ευθείαν τον κώδικα στην SML κατά τη στιγμή της εκκίνησης καλώντας την (στο Linux) με την εντολή: `sml myfile.sml`

3 Ταίριασμα προτύπων (pattern matching)

Μπορούμε να κάνουμε pattern matching κατευθείαν στις παραμέτρους της συνάρτησης, όπως φαίνεται στις παρακάτω συναρτήσεις:

```
fun factorial 0 = 1
  | factorial n = n * factorial (n - 1)

fun length [] = 0
  | length (x::xs) = 1 + length xs
```

Επίσης, μπορούμε να κάνουμε pattern matching σε μεταβλητές με χρήση του case:

```
fun factorial n =
  case n of 0 => 1
          | m => m * factorial (m - 1)

fun length l =
  case l of []    => 0
          | x::xs => 1 + length xs
```

Σε κάθε pattern matching που κάνουμε πρέπει, στο μέτρο του δυνατού, να φροντίζουμε να μην εμφανίζεται το μήνυμα `Warning: match nonexhaustive`, όπως συμβαίνει παρακάτω:

```
- fun hd [a] = a
=   | hd (h::t) = h;
stdIn:1.5-4.18 Warning: match nonexhaustive
      a :: nil => ...
      h :: t => ...

val hd = fn : 'a list -> 'a
```

Σε αντίθεση με την παραπάνω συνάρτηση, οι συναρτήσεις που ορίζουμε είναι καλό να είναι πλήρως ορισμένες, ούτως ώστε να αποφεύγεται η παραπάνω προειδοποίηση. Για το σκοπό αυτό μπορούμε να χρησιμοποιήσουμε και εξαιρέσεις, όπου κρίνεται απαραίτητο:

```
- exception HdEmpty;
exception HdEmpty
- fun hd []     = raise HdEmpty
=   | hd [a]    = a
=   | hd (h::t) = h;
val hd = fn : 'a list -> 'a
```

Το pattern matching μπορεί, τέλος, να χρησιμοποιηθεί για να επιλέξουμε ένα στοιχείο από μια πλειάδα (tuple):

```
fun fst (x,_) = x;
fun snd (_,x) = x;
- fst (10,20);
val it = 10 : int
- snd (10,20);
val it = 20 : int
```

Για την επιλογή ενός στοιχείου από μια πλειάδα με βάση τη θέση του, η SML παρέχει έναν ειδικό τελεστή:

```
- #1 (10,20);
val it = 10 : int
```

4 Υπερφόρτωση τελεστών

Μερικοί τελεστές της SML είναι υπερφορτωμένοι. Επιτελούν, δηλαδή, διαφορετική λειτουργία ανάλογα με τον τύπο των τελουμένων τους. Ένας τέτοιος τελεστής είναι ο γνωστός μας τελεστής + της πρόσθεσης:

```
- 1 + 2;
val it = 3 : int
- 1.0 + 2.0;
val it = 3.0 : real
```

Όμως, η ML δεν επιτρέπει πρόσθεση μεταξύ αριθμών διαφορετικού τύπου ούτε κάνει αυτόματη μετατροπή από έναν τύπο σε κάποιον άλλο:

```
- 1.0 + 2;
stdIn:1.1-1.8 Error: operator and operand don't agree [literal]
  operator domain: real * real
  operand:         real * int
  in expression:
    1.0 + 2
```

Για όσους δεν κατάλαβαν τις ακριβείς συνέπειες του παραπάνω για το τι επιτρέπεται και τι δεν επιτρέπεται στην ML, δείχνουμε άλλο ένα παράδειγμα:[1]

```
- val x : Int64.int = 1234567890123;
val x = 1234567890123 : Int64.int
- val y = 42;
val y = 42 : int
- x + y;
stdIn:4.1-4.7 Error: operator and operand don't agree [tycon mismatch]
  operator domain: Int64.int * Int64.int
  operand:         Int64.int * int
  in expression:
    x + 42
```

Κατά συνέπεια, πολλές φορές μπορεί να χρειαστεί να χρησιμοποιήσουμε κάποιες από τις *συναρτήσεις μετατροπής τύπων* της βιβλιοθήκης:

```
- val x : Int64.int = 1234567890123;
val x = 1234567890123 : Int64.int
- val y = 42;
val y = 42 : int
- x + (Int64.fromInt y);
val it = 1234567890165 : Int64.int
- 3.14 * real 42;
val it = 131.88 : real
```

Επίσης, όταν ορίζουμε συναρτήσεις που χρησιμοποιούν τους τελεστές οι οποίοι έχουν έναν προεπιλεγμένο default τυπο, ίσως είναι απαραίτητο να δηλώσουμε τον επιθυμητό τύπο των παραμέτρων τους έτσι ώστε να έχουμε το αναμενόμενο αποτέλεσμα:

```
- fun add x y = x+y;
val add = fn : int -> int -> int
- fun add (x:real) y = x+y;
val add = fn : real -> real -> real
```

[1] Η επανάληψη είναι μήτηρ μαθήσεως, ακόμα και στην ML που δεν υποστηρίζει επανάληψη!

5 Ορισμός datatypes

Στην SML μπορούμε να ορίσουμε νέους τύπους δεδομένων, π.χ. ένα δυαδικό δέντρο με μια απλή δήλωση:

```
datatype btree = Empty | Leaf | Node of btree * btree
```

Στη συνέχεια, μπορούμε να γράψουμε συναρτήσεις που επεξεργάζονται τους συγκεκριμένους τύπους δεδομένων, όπως η παρακάτω συνάρτηση που υπολογίζει το πλήθος των φύλλων ενός δυαδικού δέντρου:

```
- fun cntleaves Empty = 0
=   | cntleaves Leaf  = 1
=   | cntleaves (Node (tree1, tree2)) =
=         (cntleaves tree1) + (cntleaves tree2);
val cntleaves = fn : btree -> int
- val tree = Node (Node (Leaf, Leaf), Leaf);
val tree = Node (Node (Leaf, Leaf), Leaf) : btree
- cntleaves tree;
val it = 3 : int
```

6 If και βραχυκύκλωση

Η αποτίμηση των λογικών τελεστών στην SML γίνεται με την τεχνική της βραχυκύκλωσης. Αυτό σημαίνει ότι η αποτίμηση σταματάει όταν πλέον το αποτέλεσμά της είναι γνωστό, π.χ.:

```
if true  orelse  2 div 0 = 1 then print "ok\n"  else print "bad\n";
if false andalso 2 div 0 = 1 then print "bad\n" else print "ok\n";
```

Στο παραπάνω παράδειγμα αν η αποτίμηση των συνθηκών δε χρησιμοποιούσε την εν λόγω τεχνική, τότε θα εμφανιζόταν το μήνυμα `uncaught exception Div [divide by zero]`.

7 Συνηθισμένα λάθη αρχαρίων

Έλλειψη κάποιου μέρους του if then else Στην ML το else είναι υποχρεωτικό. Το if then else είναι *έκφραση* και επιστρέφει πάντα μια τιμή - σε αυτή την περίπτωση αγνοούμε την τιμή που επιστρέφει, η οποία είναι τύπου unit.

```
(* Wrong way of doing this *)
fun times3 x =
  ( if x > 3 then
      print "Greater than 3\n";
    x * 3 )

(* Right way of doing this *)
fun times3 x =
  ( if x > 3 then
      print "Greater than 3\n"
    else ();
    x * 3 )
```

Συναρτήσεις με παρενέργειες (π.χ., εκτύπωση με αλλαγή γραμμής)

```
(* Wrong way of doing this *)
fun println s = print s; print "\n";

(* Right way *)
fun println s = (print s; print "\n");
fun println s = print (s ^ "\n");
```

Διαφορές μεταξύ λιστών και πλειάδων Στην ML τα στοιχεία μιας λίστας πρέπει να έχουν όλα τον *ίδιο* τύπο. Αντίθετα, οι πλειάδες μπορεί να έχουν στοιχεία *οποιουδήποτε* τύπου σε κάθε θέση τους.

```
(* Right *)
[1, 2, 3, 4]

(* Wrong *)
[1, 2, "Hello"]

(* Right *)
(1, 2, "Hello")
```

Λάθη ταιριάσματος προτύπων (pattern matching errors) Η έκφραση `[hd,tl]` ταιριάζει μόνο με μια λίστα δύο στοιχείων και είναι ισοδύναμη με `(hd::tl::[])`.

```
(* Wrong *)
fun len [hd,tl] = 1 + len tl
  | len [] = 0

(* Right *)
fun len (hd::tl) = 1 + len tl
  | len [] = 0
```

Επίσης, η σειρά των εμφάνισης των προτύπων έχει σημασία. Η ML μας προειδοποιεί όταν κάποιο πρότυπο δεν ταιριάζει ποτέ.

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

(* Wrong *)
fun is_leaf Empty = false
  | is_leaf (Node (v, l, r)) = false
  | is_leaf (Node (v, Empty, Empty)) = true

(* Right *)
fun is_leaf Empty = false
  | is_leaf (Node (v, Empty, Empty)) = true
  | is_leaf (Node (v, l, r)) = false
```

8 Συχνές ερωτήσεις

E: Η ML έχει κάποιες βιβλιοθήκες που μπορώ να χρησιμοποιήσω;

A: Βεβαίως. Τη στιγμή συγγραφής αυτού του φυλλαδίου περιγράφονται στις ιστοσελίδες:

- `http://www.standardml.org/Basis/overview.html`
- `http://www.standardml.org/Basis/manpages.html`
- `http://www.smlnj.org/doc/smlnj-lib/Manual/toc.html`

E: Υπάρχει κάποιος τρόπος στην ML, για να δω όλα τα στοιχεία μιας δομής δεδομένων (π.χ. μιας λίστας); Όταν η δομή είναι μεγάλη μου βγάζει τελίτσες από κάποιο σημείο και μετά...

A: Ο ένας τρόπος είναι να υλοποιήσεις τη δική σου συνάρτηση ωραίου τυπώματος για τη δομή δεδομένων που θες να δεις (μάλλον το έχεις κάνει ήδη στην προτροπή που υπάρχει στη σελίδα 3). Ο άλλος τρόπος στο toplevel του SML/NJ interpreter είναι να αναθέσεις σε μια αναφορά με όνομα `Control.Print.printLength` την τιμή που θέλεις.

```
- [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17];
val it = [1,2,3,4,5,6,7,8,9,10,11,12,...] : int list
- Control.Print.printLength := 42;
[autoloading]
[library $smlnj/compiler/current.cm is stable]
...  (* many other lines supressed *)
[autoloading done]
val it = () : unit
- [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17];
val it = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17] : int list
```

Ασκήσεις εξάσκησης

1. ***Ταίριασμα προτύπων***

   Γράψτε πρότυπα που να ταιριάζουν μια μεταβλητή x με την τιμή 0 στις παρακάτω λίστες και πλειάδα.

   (α) `[~2, ~1, 0, 1, 2]`

   (β) `[(1,2), (0,1)]`

   (γ) `((1,2,0), (3,2,4))`

2. ***Εύρεση λαθών***

   Να βρεθούν τα λάθη ή οι προειδοποιήσεις της ML στις παρακάτω περιπτώσεις:

   (α)
   ```
   print 42
   ```

   (β)
   ```
   fun foo = 42
   ```

   (γ)
   ```
   fun inc x = x + 1
   fun f n = inc true
   ```

   (δ)
   ```
   fun divide_by_two x = x / 2
   ```

   (ε)
   ```
   datatype 'a btree = Leaf of 'a | Node of 'a btree * 'a btree
   fun preorder Leaf(v) = [v]
     | preorder Node(l,r) = preorder l @ preorder r
   ```

   (στ)
   ```
   datatype 'a option = NONE | SOME of 'a
   fun filter pred l =
         let fun filterP (x::r, l) =
                   case (pred x) of
                      SOME y => filterP (r, y::l)
                    | NONE   => filterP (r, l)
               | filterP ([], l) = rev l
         in
           filterP (l, [])
         end
   ```

   (ζ)
   ```
   let val e = 5 and f = e + 1 in e + f end
   ```

   (η)
   ```
   val (x, x) = (42, 42)
   ```

(θ)
```
fun f (x, y)    = x + y + 1
  | f (x, y, z) = x + y + z + 2
  | f  _        = 3
```

(ι)
```
fun f _ = "nonzero"
  | f 0 = "zero"
```

(ια)
```
fun h (x::xs) = x + 1
```

(ιβ)
```
val succ = op + 1
```

3. ***Διαχωρισμός λίστας***

   Να γράψετε μία συνάρτηση που να δέχεται μία λίστα και να τη χωρίζει σε δύο υπολίστες εκ των οποίων η πρώτη θα περιέχει το πρώτο ήμισυ και η άλλη το δεύτερο. (Αν η αρχική λίστα έχει περιττό μήκος, επιλέξτε εσείς σε ποιά λίστα θα καταλήξει το μεσαίο στοιχείο.)

4. ***Τέλειοι αριθμοί***

   Ένας *τέλειος* αριθμός ισούται με το άθροισμα των παραγόντων του, συμπεριλαμβανομένης της μονάδας (1) αλλά όχι του εαυτού του. Για παράδειγμα, ο 6 είναι ένας τέλειος αριθμός γιατί $6 = 3 + 2 + 1$. Να γράψετε μία συνάρτηση που να επιστρέφει `true` αν ένας αριθμός είναι τέλειος και `false` αν δεν είναι.

5. ***Περίγραμμα δέντρου***

   Γράψτε μια συνάρτηση που παίρνει ένα δέντρο σαν όρισμα, και επιστρέφει μια λίστα με όλα τα στοιχεία του που είναι φύλλα, από αριστερά προς τα δεξιά, για τον εξής πολυμορφικό ορισμό της δομής των δέντρων:

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

6. ***Δυναμοσύνολο***

   Να ορίσετε μία συνάρτηση στην οποία όταν δίνεται ένα σύνολο με τη μορφή λίστας να επιστρέφει το σύνολο των υποσυνόλων του σε μορφή λίστας.

7. ***Ταξινόμηση***

   Να γράψετε μία συνάρτηση ταξινόμησης για τη μέθοδο Merge Sort χρησιμοποιώντας τη συνάρτηση `halve`:

```
fun halve nil = (nil, nil)
| halve [a] = ([a], nil)
| halve (a::b::cs) =
    let
      val (x, y) = halve cs
    in
      (a::x, b::y)
    end
```

8. ***Οι πύργοι του Hanoi***[2]

Υποθέστε ότι σας δίνονται τρεις ράβδοι και δίσκοι διαφορετικού μεγέθους. Οι δίσκοι μπορούν να στοιβαχθούν στις ράβδους σχηματίζοντας πύργους. Βρίσκονται αρχικά στη ράβδο left με φθίνουσα τάξη μεγέθους. Πρέπει να μεταφερθούν στη ράβδο right ώστε τελικά να βρεθούν τοποθετημένοι με την ίδια σειρά. Αυτό πρέπει να επιτευχθεί με τους παρακάτω περιορισμούς:

- Σε κάθε βήμα μόνο ένας δίσκος μεταφέρεται από μία ράβδο σε άλλη.
- Ένας δίσκος απαγορεύεται να τοποθετηθεί πάνω από μικρότερο δίσκο.
- Η ράβδος middle μπορεί να χρησιμοποιηθεί για προσωρινή τοποθέτηση των δίσκων.

Να γράψετε μία συνάρτηση που να επιτυγχάνει το επιθυμητό αποτέλεσμα.

[2]**Ιστορική σημείωση:** Το παιχνίδι ονομάσθηκε «Πύργος του Ανόι» γιατί το σχήμα του πύργου θυμίζει την αρχιτεκτονική των ναών στις χώρες της Άπω Ανατολής. Το παιχνίδι πρωτοεμφανίστηκε από τον Γάλλο μαθηματικό Édouard Lucas γύρω στα 1883. Η προέλευσή του είναι μάλλον από τις Ινδίες. Υπάρχει για το παιχνίδι ο παρακάτω θρύλος με τίτλο «Πύργος του Βράχμα»:

> Όταν ο Βράχμα δημιούργησε τον κόσμο, έστησε σε ένα ναό στην πόλη Μπενάρες 64 δακτυλίδια χρυσού άνισου μεγέθους, όλα περασμένα σε μία ράβδο. Οι ιερείς του ναού έπρεπε να δουλεύουν μέρα και νύχτα ασταμάτητα για να μεταφέρουν τα δακτυλίδια σε μία άλλη ράβδο, χρησιμοποιώντας μία τρίτη σαν βοηθητική, έτσι ώστε να μην τοποθετήσουν μεγαλύτερο δακτυλίδι πάνω από μικρότερο και μετακινώντας ένα μόνο δακτυλίδι σε κάθε κίνηση. Ο θρύλος λέει πως πριν προλάβουν οι ιερείς να μεταφέρουν όλα τα δακτυλίδια στην άλλη ράβδο, ο ναός θα καταρρεύσει μέσα στην σκόνη και ο κόσμος θα χαθεί μέσα σε τρομακτικό κρότο βροντής.