Η γλώσσα ML σε βάθος

Joan Miró, *El Carnaval del Arlequín*, 1925

Κωστής Σαγώνας <kostis@cs.ntua.gr>

Τι σημαίνουν οι τύποι συναρτήσεων στην ML

- f : A → B σημαίνει:
  - Για κάθε x ∈ A,

$$f(x) = \begin{cases} \text{για κάποιο στοιχείο } y = f(x) \in B \\ \text{ατέρμονη εκτέλεση} \\ \text{η εκτέλεση τερματίζει εγείροντας κάποια εξαίρεση} \end{cases}$$

- Με λόγια:
  "εάν η αποτίμηση f(x) τερματίσει κανονικά, τότε f(x)∈ B"
- Δηλαδή, η πρόσθεση δε θα εκτελεστεί σε μια έκφραση της μορφής **`f(x)+3`** εάν η **`f(x)`** εγείρει κάποια εξαίρεση

Επισημειώσεις τύπων (type annotations)

```
- fun prod (a,b) = a*b;
val prod = fn : int * int -> int
```

- Γιατί **`int`** και όχι **`real`**;
- Διότι ο προεπιλεγμένος τύπος (default type) του αριθμητικού τελεστή **`*`** (όπως και των **`+`**, **`-`**) είναι

  `int * int -> int`
- Αν θέλουμε να χρησιμοποιήσουμε τη συνάρτηση με ορίσματα τύπου **`real`** μπορούμε να βάλουμε μια υποσημείωση τύπου στα συγκεκριμένα ορίσματα

Παράδειγμα επισημειώσεων τύπων στην ML

```
- fun prod (a:real,b:real):real = a*b;
val prod = fn : real * real -> real
```

- Οι επισημειώσεις τύπων αποτελούνται από μια άνω κάτω τελεία και έναν τύπο και μπορούν να μπουν παντού
- Όλοι τα παρακάτω ορισμοί είναι ισοδύναμοι:

```
fun prod (a,b):real = a * b;
fun prod (a:real,b) = a * b;
fun prod (a,b:real) = a * b;
fun prod (a,b) = (a:real) * b;
fun prod (a,b) = a * b:real;
fun prod (a,b) = (a*b):real;
fun prod ((a,b):real * real) = a*b;
```


Συναρτήσεις μετατροπής τύπων

```
- real 123;
val it = 123.0 : real
- floor 3.6;
val it = 3 : int
- str #"a";
val it = "a" : string
```

Ενσωματωμένες συναρτήσεις μετατροπής τύπων:

- **`real`** (**`int`** → **`real`**),
- **`floor`** (**`real`** → **`int`**), **`ceil`** (**`real`** → **`int`**),
- **`round`** (**`real`** → **`int`**), **`trunc`** (**`real`** → **`int`**),
- **`ord`** (**`char`** → **`int`**),
- **`chr`** (**`int`** → **`char`**),
- **`str`** (**`char`** → **`string`**)

Σύνταξη ταιριάσματος

- Ένας **κανόνας** έχει την παρακάτω σύνταξη στην ML:

  <rule> **`::=`** *<pattern>* **`=>`** *<expression>*
- Ένα **ταίριασμα** αποτελείται από έναν ή περισσότερους κανόνες που διαχωρίζονται μεταξύ τους από '**`|`**':

  <match> **`::=`** *<rule>* **`|`** *<rule>* **`'|'`** *<match>*
- Σε ένα ταίριασμα κάθε κανόνας πρέπει να έχει τον ίδιο τύπο με την έκφραση (*expression*) στο δεξί μέρος του κανόνα
- Ένα ταίριασμα δεν είναι έκφραση από μόνο του, αλλά αποτελεί μέρος διαφόρων εκφράσεων της ML

Εκφράσεις case

```
- case 1+1 of
=    3 => "three" |
=    2 => "two" |
=    _ => "hmmm...";
val it = "two" : string
```

- Έχουν τη σύνταξη:

  <*case-expr*> ::= **case** <*expression*> **of** <*match*>

- Η έκφραση **case** της ML είναι μια πολύ ισχυρή δομή–και αντίθετα με τις περισσότερες άλλες γλώσσες, μπορεί να κάνει περισσότερα από απλή σύγκριση με σταθερές

Παράδειγμα χρήσης case

```
case list of
  _::_::c::_ => c |
  _::b::_ => b |
  a::_ => a |
  nil => 0
```

- Η τιμή αυτής της έκφρασης είναι:
  - το τρίτο στοιχείο της λίστας **list**, αν η λίστα έχει τουλάχιστον τρία στοιχεία, ή
  - το δεύτερο στοιχείο της λίστας αν έχει μόνο δύο, ή
  - το πρώτο στοιχείο της λίστας **list** εάν έχει μόνο ένα, ή
  - ο ακέραιος **0** εάν η λίστα **list** είναι κενή.

Η έκφραση case είναι μια γενίκευση της if

if exp_1 **then** exp_2 **else** exp_3

case exp_1 **of**
  true => exp_2 **|**
  false => exp_3

Οι δύο παραπάνω εκφράσεις είναι ισοδύναμες

Με άλλα λόγια, ένα **if-then-else** είναι μια ειδική περίπτωση ενός **case**

Αποτίμηση "βραχυκύκλωσης" στην ML

```
- true orelse 1 div 0 = 0;
val it = true : bool
```

- Οι τελεστές **andalso** και **orelse** "βραχυκυκλώνουν" (short-circuit) στην ML:
  - Εάν η έκφραση του πρώτου ορίσματος του **orelse** αποτιμάται ως αληθής (true), η έκφραση του δεύτερου δεν αποτιμάται
  - Παρόμοια, εάν το πρώτο όρισμα του **andalso** είναι ψευδές
- Με βάση το "γράμμα" της θεωρίας, δεν είναι πραγματικοί τελεστές αλλά λέξεις κλειδιά
- Αυτό διότι, σε μια γλώσσα σαν την ML, όλοι οι τελεστές αποτιμούν πλήρως τα ορίσματά τους

Πολυμορφικές συναρτήσεις για λίστες

- Αναδρομική συνάρτηση που υπολογίζει το μήκος μιας λίστας (οπουδήποτε τύπου)

```
- fun length x =
=   if null x then 0
=   else 1 + length (tl x);
val length = fn : 'a list -> int
- length [true,false,true];
val it = 3 : int
- length [4.0,3.0,2.0,1.0];
val it = 4 : int
```

Σημείωση: η συνάρτηση **length** είναι μέρος της ML, οπότε ο παραπάνω ορισμός είναι περιττός

Πολυμορφισμός για τύπους ισότητας

```
- fun length_eq x =
=   if x=[] then 0
=   else 1 + length_eq (tl x);
val length_eq = fn : ''a list -> int
- length_eq [true,false,true];
val it = 3 : int
- length_eq [4.0,3.0,2.0,1.0];
Error: operator and operand don't agree
  [equality type required]
```

- Μεταβλητές τύπων που αρχίζουν με δύο αποστρόφους, όπως ο **''a**, περιορίζονται σε τύπους ισότητας
- Η ML συμπεραίνει αυτόν τον περιορισμό διότι συγκρίναμε τη μεταβλητή **x** για ισότητα με την κενή λίστα. Αυτό δε θα σύμβαινε εάν είχαμε χρησιμοποιήσει τη συνθήκη **null x** αντί για την **x=[]**

Αποδοτικές συναρτήσεις για λίστες

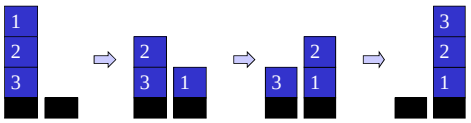
- Αναστροφή μιας λίστας

```
fun reverse nil = nil
  | reverse (x::xs) = (reverse xs) @ [x];
```

- Ερωτήσεις:
  - Πόσο αποδοτική είναι η συνάρτηση **reverse**?
  - Μπορούμε να αναστρέψουμε μια λίστα με ένα μόνο πέρασμα;

Πιο αποδοτική συνάρτηση reverse

```
fun reverse xs
  let
    fun rev (nil, z) = z
      | rev (y::ys, z) = rev (ys, y::z)
  in
    rev (xs, nil)
  end;
```



Συναρτήσεις Υψηλής Τάξης

Η λέξη κλειδί op

```
- op *;
val it = fn : int * int -> int
- quicksort ([1,4,3,2,5], op <);
val it = [1,2,3,4,5] : int list
```

- Οι δυαδικοί τελεστές είναι ειδικές συναρτήσεις
- Όμως μερικές φορές θέλουμε να τις χρησιμοποιήσουμε σαν κοινές συναρτήσεις: για παράδειγμα, να περάσουμε τον τελεστή **<** σαν όρισμα τύπου **int * int -> bool**
- Η λέξη κλειδί **op** πριν από κάποιο τελεστή επιστρέφει τη αντίστοιχη συνάρτηση

Συναρτήσεις υψηλής τάξης

- Κάθε συνάρτηση έχει μία **τάξη** (order):
  - Μια συνάρτηση που δεν παίρνει άλλες συναρτήσεις ως παραμέτρους και δεν επιστρέφει ως αποτέλεσμα μια άλλη συνάρτηση έχει τάξη 1
  - Μια συνάρτηση που παίρνει άλλες συναρτήσεις ως παραμέτρους ή επιστρέφει ως αποτέλεσμα μια άλλη συνάρτηση έχει τάξη n+1, όπου n είναι η μέγιστη τάξη των παραμέτρων της και του αποτελέσματός της
- Η συνάρτηση **quicksort** που μόλις είδαμε είναι συνάρτηση δεύτερης τάξης

Πρακτική εξάσκηση

- Τι τάξεως είναι οι συναρτήσεις της ML με τους παρακάτω τύπους;

```
int * int -> bool
int list * (int * int -> bool) -> int list
int -> int -> int
(int -> int) * (int -> int) -> (int -> int)
int -> bool -> real -> string
```

- Τι μπορούμε να πούμε για την τάξη της συνάρτησης με τον παρακάτω τύπο;

```
('a -> 'b) * ('c -> 'a) -> 'c -> 'b
```

Προκαθορισμένες συναρτήσεις υψηλής τάξης

- Τρεις σημαντικές προκαθορισμένες συναρτήσεις υψηλής τάξης:
  1. **map**
  2. **foldr**
  3. **foldl**
- Η **foldr** και η **foldl** είναι παρόμοιες

Η συνάρτηση map

- Εφαρμόζει μια συνάρτηση σε κάθε στοιχείο μιας λίστας και επιστρέφει τα αποτελέσματα της εφαρμογής σε μια νέα λίστα

```
- map ~ [1,2,3,4];
val it = [~1,~2,~3,~4] : int list
- map (fn x => x+1) [1,2,3,4];
val it = [2,3,4,5] : int list
- map (fn x => x mod 2 = 0) [1,2,3,4];
val it = [false,true,false,true] : bool list
- map (op +) [(1,2),(3,4),(5,6)];
val it = [3,7,11] : int list
- val f = map (op +);
val f = fn : (int * int) list -> int list
- f [(1,2),(3,4)];
val it = [3,7] : int list
```


Η συνάρτηση foldr

- Συνδυάζει, μέσω μιας συνάρτησης, όλα τα στοιχεία μιας λίστας
- Παίρνει ως ορίσματα μια συνάρτηση f, μια αρχική τιμή c, και μια λίστα $x = [x_1, \ldots, x_n]$ και υπολογίζει την τιμή:

$$f\,(x_1,\ f\,(x_2, \cdots\ f\,(x_{n-1},\ f\,(x_n, c))\cdots))$$

- Για παράδειγμα η κλήση:

  foldr (op +) 0 [1,2,3,4]

  αποτιμάται σε **1+(2+(3+(4+0)))=10**

Παραδείγματα χρήσης foldr

```
- foldr (op +) 0 [1,2,3,4];
val it = 10 : int
- foldr (op * ) 1 [1,2,3,4];
val it = 24 : int
- foldr (op ^) "" ["abc","def","ghi"];
val it = "abcdefghi" : string
- foldr (op ::) [5] [1,2,3,4];
val it = [1,2,3,4,5] : int list
- foldr;
val it = fn : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
- foldr (op +);
val it = fn : int -> int list -> int
- foldr (op +) 0;
val it = fn : int list -> int
- val addup = foldr (op +) 0;
val addup = fn : int list -> int
- addup [1,2,3,4,5];
val it = 15 : int
```


Η συνάρτηση foldl

- Συνδυάζει, μέσω μιας συνάρτησης, όλα τα στοιχεία μιας λίστας (όπως η **foldr**)
- Παίρνει ως ορίσματα μια συνάρτηση f, μια αρχική τιμή c, και μια λίστα $x = [x_1, \ldots, x_n]$ και υπολογίζει την τιμή:

$$f\,(x_n,\ f\,(x_{n-1}, \cdots\ f\,(x_2,\ f\,(x_1, c))\cdots))$$

- Για παράδειγμα η κλήση:

  foldr (op +) 0 [1,2,3,4]

  αποτιμάται σε **4+(3+(2+(1+0)))=10**

Σημείωση: Η **foldr** αποτιμήθηκε ως **1+(2+(3+(4+0)))=10**

Παραδείγματα χρήσης foldl

- Η **foldl** αρχίζει από αριστερά, η **foldr** από τα δεξιά
- Φυσικά, δεν υπάρχει κάποια διαφορά όταν η συνάρτηση είναι ανακλαστική και μεταβατική, όπως η **+** και η *****
- Για άλλες συναρτήσεις όμως υπάρχει διαφορά

```
- foldr (op ^) "" ["abc","def","ghi"];
val it = "abcdefghi" : string
- foldl (op ^) "" ["abc","def","ghi"];
val it = "ghidefabc" : string
- foldr (op -) 0 [1,2,3,4];
val it = ~2 : int
- foldl (op -) 0 [1,2,3,4];
val it = 2 : int
```

Δηλώσεις Τύπων Δεδομένων

Ορισμοί τύπων δεδομένων

- Προκαθορισμένος τύπος, αλλά όχι πρωτόγονος στην ML

```
datatype bool = true | false;
```

- Παραμετρικός κατασκευαστής τύπου (parametric type constructor) για λίστες:

```
datatype 'e list = nil
                 | :: of 'e * 'e list
```

- Ορίζεται για την ML στην ML!

Ορισμοί τύπων δεδομένων

- Έχουν τη γενική μορφή

datatype <*name*> **=** <*clause*> **|** … **|** <*clause*>
<*clause*> **::=** <*constructor*> **|** <*constructor*> **of** <*type*>

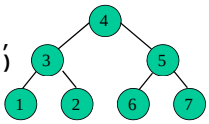
- Παραδείγματα:
  - **datatype color = Red | Yellow | Blue**
    - στοιχεία : **Red**, **Yellow**, και **Blue**
  - **datatype atom = Atm of string | Nmbr of int**
    - στοιχεία : **Atm("a")**, **Atm("b")**, …, **Nmbr(0)**, **Nmbr(1)**, …
  - **datatype list = Nil | Cons of atom * list**
    - στοιχεία : **Nil**, **Cons(Atm("a"),Nil)**, …
      Cons(Nmbr(2),Cons(Atm("ugh"),Nil)), …

Ορισμοί αναδρομικών τύπων δεδομένων

```
datatype 'd tree = Leaf of 'd
                 | Node of 'd * 'd tree * 'd tree;
```

- Παράδειγμα στιγμιότυπου δένδρου

```
Node(4, Node(3,Leaf(1),Leaf(2)),
        Node(5,Leaf(6),Leaf(7)))
```



- Αναδρομική συνάρτηση χρήσης του τύπου δεδομένων

```
fun sum (Leaf n) = n
  | sum (Node (n,t1,t2)) = n + sum(t1) + sum(t2);
```


Αυστηρό σύστημα τύπων

```
- datatype flip = Heads | Tails;
datatype flip = Heads | Tails
- fun isHeads x = (x = Heads);
val isHeads = fn : flip -> bool
- isHeads Tails;
val it = false : bool
- isHeads Mon;
Error: operator and operand don't agree [tycon mismatch]
  operator domain: flip
  operand:         day
```

- Η ML είναι αυστηρή σε σχέση με τους νέους τύπους, ακριβώς όπως θα περιμέναμε
- Σε αντίθεση π.χ. με τις **enum** δηλώσεις της C, οι λεπτομέρειες της υλοποίησης δεν είναι εμφανείς στον προγραμματιστή

Κατασκευαστές έναντι συναρτήσεων

```
- datatype exint = Value of int | PlusInf | MinusInf;
datatype exint = MinusInf | PlusInf | Value of int
- PlusInf;
val it = PlusInf : exint
- MinusInf;
val it = MinusInf : exint
- Value;
val it = fn : int -> exint
- Value 3;
val it = Value 3 : exint
```

- Ο **Value** είναι ένας κατασκευαστής δεδομένων με μία παράμετρο: την τιμή του ακεραίου **int** που αποθηκεύει
- Δείχνει σα συνάρτηση που παίρνει έναν ακέραιο (**int**) και επιστρέφει έναν **exint** που περιέχει τον ακέραιο

Όμως ένας Value δεν είναι int

```
- val x = Value 5;
val x = Value 5 : exint
- x + x;
Error: overloaded variable not defined at type symbol: +
  type: exint
```

- Ένας **Value 5** είναι ένας **exint**, όχι ένας ακέραιος (**int**), παρότι εμπεριέχει έναν
- Μπορούμε να ανακτήσουμε τις παραμέτρους ενός κατασκευαστή χρησιμοποιώντας ταίριασμα προτύπων
- Κατά συνέπεια, ο κατασκευαστής **Value** δεν είναι συνάρτηση: οι κανονικές συναρτήσεις δε μπορούν να χρησιμοποιηθούν με αυτόν τον τρόπο ως πρότυπα

Κατασκευαστές και ταίριασμα προτύπων

```
- fun square PlusInf = PlusInf
=   | square MinusInf = PlusInf
=   | square (Value x) = Value (x*x);
val square = fn : exint -> exint
- square MinusInf;
val it = PlusInf : exint
- square (Value 3);
val it = Value 9 : exint
```

- Διαχειριζόμαστε νέους τύπους δεδομένων με συναρτήσεις σαν την παραπάνω που ορίζονται μέσω ταιριάσματος προτύπων
- Επειδή ένας **exint** είναι είτε **PlusInf**, ή **MinusInf**, ή **Value**, η παραπάνω συνάρτηση είναι εξαντλητική ως προς το ταίριασμα προτύπων

Χειρισμός εξαιρέσεων στην ML

- Μέσω ταιριάσματος προτύπων μπορούμε επίσης να χειριστούμε εξαιρέσεις

```
- fun square PlusInf = PlusInf
=   | square MinusInf = PlusInf
=   | square (Value x) = Value (x*x)
=       handle Overflow => PlusInf;
val square = fn : exint -> exint
- square (Value 10000);
val it = Value 100000000 : exint
- square (Value 100000);
val it = PlusInf : exint
```

- Θα δούμε περισσότερα για τις εξαιρέσεις στη Java

Ένα ακόμα παράδειγμα: bunch

```
datatype 'x bunch =
   One of 'x |
   Group of 'x list;
```

- Ένα **'x bunch** είναι είτε ένα πράγμα τύπου **'x**, είτε μια λίστα από πράγματα τύπου **'x**
- Όπως συνήθως, η ML συμπεραίνει τύπους αυτόματα:

```
- One 1.0;
val it = One 1.0 : real bunch
- Group [true,false];
val it = Group [true,false] : bool bunch
```


Παράδειγμα: Πολυμορφικός Συμπερασμός

- Η ML μπορεί να συμπεράνει πολυμορφικούς **bunch** τύπους, αλλά δεν χρειάζεται πάντα να τους επιλύσει πλήρως, όπως για παράδειγμα συμβαίνει όταν σε αυτούς περιλαμβάνονται λίστες

```
- fun size (One _) = 1
=   | size (Group x) = length x;
val size = fn : 'a bunch -> int
- size (One 1.0);
val it = 1 : int
- size (Group [true,false]);
val it = 2 : int
```


Παράδειγμα: Μη Πολυμορφικός Συμπερασμός

```
- fun sum (One x) = x
=   | sum (Group xlist) = foldr op + 0 xlist;
val sum = fn : int bunch -> int
- sum (One 5);
val it = 5 : int
- sum (Group [1,2,3]);
val it = 6 : int
```

- Χρησιμοποιήσαμε τον τελεστή **+** (ως όρισμα της **foldr**) στα στοιχεία της λίστας
- Κατά συνέπεια, η ML μπορεί να συμπεράνει ότι ο τύπος της παραμέτρου της συνάρτησης **sum** είναι **int bunch**

Αυτή ήταν η ML

- … ή τουλάχιστον, όλη η ML που θα δούμε στο μάθημα
- Φυσικά, υπάρχουν κάποια μέρη ακόμα:
  - Εγγραφές (records) που είναι σαν τις πλειάδες αλλά έχουν πεδία με ονόματα
    - π.χ. `{name="Arnold", age=42} : {name : string, age : int}`
  - Πίνακες (arrays) με στοιχεία που μπορούν να τροποποιηθούν
  - Αναφορές (references) για τιμές που μπορούν να τροποποιηθούν
  - Χειρισμός εξαιρέσεων (exception handling)
  - Υποστήριξη encapsulation και απόκρυψης δεδομένων:
    - structures: συλλογές από τύπους δεδομένων + συναρτήσεων
    - signatures: interfaces για τα structures
    - functors: κάτι σα συναρτήσεις για structures, που όμως επιτρέπουν μεταβλητές τύπων και την ανάθεση τιμών (instantiation) στις παραμέτρους των structures

Κάποια άλλα μέρη της ML

- API: the standard basis
  - Προκαθορισμένες συναρτήσεις, τύποι, κ.λπ.
  - Κάποιες από αυτές είναι σε structures: `Int.maxInt`, `Real.Math.sqrt`, `List.nth`, κ.λπ.
- eXene: μια βιβλιοθήκη της ML για εφαρμογές σε γραφικό περιβάλλον X windows
- O Compilation Manager για διαχείριση μεγαλύτερων projects

- Άλλες διάλεκτοι της ML
  - O'Caml
  - Η επέκταση της ML για ταυτοχρονισμό (Concurrent ML - CML)

Συμπερασματικά για τις συναρτησιακές γλώσσες

- Η ML είναι η μόνη γλώσσα που θα εξετάσουμε από τις συναρτησιακές γλώσσες προγραμματισμού
- Σε αυτό το είδος προγραμματισμού, η εκτέλεση γίνεται μέσω αποτίμησης εκφράσεων και ταιριάσματος προτύπων
- Εάν σας αρέσει αυτό το στυλ προγραμματισμού, υπάρχουν και άλλες γλώσσες για εξερεύνηση, όπως η Lisp, η Scheme, η Haskell, η Clean και η Erlang