Αφαίρεση στον FP

Πολυμορφισμός

Συναρτήσεις υψηλότερης τάξης

Οκνηρός και Άπληστος Υπολογισμός

Πολυμορφισμός

- Θα χρησιμοποιήσουμε σαν παράδειγμα τη συνάρτηση ταυτότητας I, που ορίζεται ως:

fun I x = x

- Ο ορισμός της I είναι γενικός, δηλαδή η I μπορεί να εφαρμοστεί σε δεδομένα διαφορετικών τύπων:
  - I 5 = 5 Τι τύπος φαίνεται να είναι η I εδώ;
  - I “ab” =
  - I true =
  - I (5, false) =

Μεταβλητές Τύπων

- Επιδιωκόμενη συμπεριφορά για την I:

  όταν εφαρμόζεται σε «αντικείμενα τύπου T», τότε τα απεικονίζει σε «αντικείμενα τύπου T», όπου T μπορεί να είναι οποιοσδήποτε (απλός ή σύνθετος τύπος).

$$I : T \rightarrow T$$

- Χρησιμοποιούμε **μεταβλητές τύπων** α, β, γ, …, για να αναπαραστήσουμε οποιοδήποτε τύπο.

Παραδείγματα Πολυμορφικών Συναρτήσεων

• **fun** I x = x	I : α → α
• **fun** fst (a, b) = a	fst: (α x β) → α
• **fun** snd (a, b) = b	snd: (α x β) → β
• **fun** swapPair (a, b) = (b, a)	swapPair: (α x β) →(β x α)
• **fun** K5 x = 5	K5 : α → int

Πρωτογενείς τύποι και στιγμιότυπα

- Καθώς οι συναρτήσεις συμμετέχουν σε εκφράσεις, έτσι μπορούν στιγμιότυπα τύπων να συσχετίζονται με εκφράσεις.
- Στην έκφραση

3 + fst (5, true)

η συνάρτηση fst συμμετέχει με το **στιγμιότυπο** τύπου

(int x bool) → int του πολυμορφικού (**πρωτογενούς**) τύπου της

α x β → α,

δηλαδή η μεταβλητή τύπου α αντικαθίσταται από τύπο int, και η μεταβλητή τύπου β αντικαθίσταται από τύπο bool.

- Μπορούν να παραχθούν στιγμιότυπα τύπου αντικαθιστώντας μεταβλητές τύπου με άλλες μεταβλητές τύπου.
  - (γ x γ) → γ είναι στιγμιότυπο του (α x β) → α (α / γ, β /γ)

Παράδειγμα στιγμιοτύπων τύπου

- Με τι στιγμιότυπο τύπου συμμετέχει η I : α → α στις παρακάτω εκφράσεις;

I (3 + 5)

I (true, 6)

I (double) όπου double : int → int

Πώς διακρίνουμε πρωτογενείς τύπους από ορισμούς συναρτήσεων;

- Υπάρχει αυτόματος μηχανισμός ενσωματωμένος στις γλώσσες του FP (αργότερα).

- Άτυπα (χειρωνακτικά) «δουλεύοντας» με τον κανόνα υπολογισμού της συνάρτησης.

fun minpair (a, b) = **if** a < b **then** a **else** b

- LHS ορισμού: Η minpair δέχεται όρισμα ένα ζεύγος (a, b), άρα αν a : α, και b : β, τότε ο γενικός πρωτογενής τύπος της, εφόσον είναι συνάρτηση είναι

  $(\alpha \times \beta) \rightarrow \gamma$

- RHS ορισμού: Υπάρχει υποθετική έκφραση και θυμόμαστε ότι τα κλαδιά της (**then**…, **else**…) πρέπει να είναι ίδιου τύπου, άρα συμπεραίνουμε ότι αφού a και b είναι τα αποτελέσματα που επιστρέφουν τα κλαδιά αντίστοιχα, θα πρέπει

  $\alpha = \beta$

- Και εφόσον το αποτέλεσμα που επιστρέφει όλη η υποθετική έκφραση είναι ίδιου τύπου με τα κλαδιά, θα πρέπει

  $\gamma = \alpha$

- Επίσης ο τύπος του ελέγχου της υποθετικής έκφρασης (**if…**) θα πρέπει να είναι bool, το οποίο είναι συμβατό με τον τύπο της συνάρτησης < : (int x int) → bool, και επειδή η < εφαρμόζεται στα a, b, συμπεραίνουμε ότι

  $\alpha = \beta = \gamma = \text{int}$

- Επομένως ο τύπος της minpair : (int x int) → int

Τι τύπου είναι η ισότητα; (=)

- Μια εκδοχή είναι, θεωρώντας την ως συνάρτηση, με πολυμορφικό τύπο

$$= : (\alpha \times \alpha) \rightarrow \text{bool}$$

- Μια άλλη εκδοχή είναι να θεωρήσουμε ότι υπάρχει κλάση Τύπων που επιδέχονται εφαρμογή της συνάρτησης ισότητας, και να ορίσουμε τον τύπο της πολυμορφικά, με μεταβλητή τύπου για αυτήν την κλάση

$$= : (\alpha^{=} \times \alpha^{=}) \rightarrow \text{bool}$$

- $\alpha^{=}$ θα μπορούσε να είναι int, bool, string, και όποιοι σύνθετοι μπορούν να σχηματιστούν με τον τελεστή τύπων x, αλλά όχι οι τύποι που σχηματίζονται με τον τελεστή τύπων →

Σύνοψη Πολυμορφικών Τύπων

Σταθερές τύπων: {int, string, bool}
Μεταβλητές τύπων: {α, β, γ, …}
Μεταβλητές τύπων ισότητας: {$\alpha^{=}$, $\beta^{=}$, $\gamma^{=}$, …}

$$\frac{T \text{ είναι σταθερά τύπου}}{\text{T είναι τύπος } (\iota\sigma)}$$

$$\frac{T \text{ είναι μεταβλητή τύπου } (\iota\sigma)}{\text{T είναι τύπος } (\iota\sigma)} \qquad \frac{T \text{ είναι μεταβλητή τύπου ισότητας}}{\text{T είναι τύπος } (\iota\sigma)}$$

$$\frac{T1 \text{ είναι τύπος}, \text{T2 είναι τύπος}}{\text{T1} \rightarrow \text{T2 είναι τύπος}} \qquad \frac{T1 \text{ είναι τύπος } (\iota\sigma), \text{T2 είναι τύπος } (\iota\sigma), ..., \text{Tn είναι τύπος } (\iota\sigma)}{(\text{T1} \times T2 \times ... \times Tn) \text{ είναι τύπος } (\iota\sigma)}$$

Συναρτήσεις Υψηλότερης Τάξης (Higher Order functions – HO συναρτήσεις)

- Θεωρώντας τις συναρτήσεις ως (αφηρημένα) δεδομένα, προκύπτουν:
  - συναρτήσεις που είναι ορίσματα άλλων συναρτήσεων
  - συναρτήσεις που παράγουν συναρτήσεις
  - δομές δεδομένων (πλειάδες, λίστες κλπ) που αποτελούνται από συναρτήσεις. (η δημιουργία δομών επιτυγχάνεται μέσω συναρτήσεων!)

Τύπος συνάρτησης ΗΟ

- Πολυμορφικά (α → β),
  όπου είτε α, είτε β, είτε και τα δύο είναι τύποι συναρτήσεων, δηλαδή είναι της μορφής
  (γ → δ).

- (int → int) → (int x bool)

  είναι ο τύπος μιας συνάρτησης υψηλότερης τάξης (έστω f), η οποία δέχεται ως όρισμα μια συνάρτηση πρώτης τάξης (έστω g), για την οποία ισχύει ο τύπος
  g: int → int,
  και την αντιστοιχίζει σε μια δυάδα με τον σύνθετο τύπο (int x bool).

Παράδειγμα συνάρτησης με όρισμα συνάρτηση

- I (double) Τύπος;
- Θυμηθείτε ότι I : α →α και double : int →int
- Ο τύπος μιας συνάρτησης ΗΟ, προκύπτει με μετεγγραφή/αντικατάσταση (όπως η διαδικασία υπολογισμού εκφράσεων!)

Η σύνθεση ως συνάρτηση ΗΟ: εξαγωγή τύπου

- Όπου f, g, συναρτήσεις, (f ° g) x = f (g (x))
- Η σύνθεση ° είναι συνάρτηση άρα ο γενικός τύπος της είναι της μορφής °: Τ1 → Τ2.
- Εφόσον δέχεται δύο ορίσματα, Τ1 = Τ3 x Τ4, και αφού τα ορίσματα είναι συναρτήσεις, Τ3 = Τ5 → Τ6, και Τ4 = Τ7 → Τ8.
- Επιπλέον η ° παράγει μια συνάρτηση άρα Τ2 = Τ9 → Τ10.
- Αρα ° : (Τ5 →Τ6) x (Τ7 → Τ8) → (Τ9 →Τ10).
- Επιπλέον για να μπορούν να συντεθούν οι f και g, πρέπει το πεδίο τιμών της g να είναι ιδίου τύπου με το πεδίο ορισμού της f, δηλαδή Τ8 = Τ5. Το αποτέλεσμα της σύνθεσης θα πρέπει να είναι ιδίου τύπου με το πεδίο τιμών της f, δηλαδή Τ10 = Τ6, και ο τύπος του ορίσματος της σύνθεσης θα πρέπει να είναι ίδιος με αυτόν του ορίσματος της g, δηλαδή Τ7 = Τ9.
- Αν f : α → β και g: γ→δ, με αντικατάσταση/μετεγγραφή στον τύπο της σύνθεσης (Τ5 = α, Τ6 =β, Τ7 = γ και Τ8 = δ) προκύπτει:

- ° : (α →β) x (γ → α) → (γ →β).

Συναρτήσεις γενικού σκοπού

- Το κύριο όφελος από τη χρήση συναρτήσεων HO είναι ότι μπορούμε να ορίζουμε συναρτήσεις **γενικού σκοπού**.
- Παράδειγμα: Μια συνάρτηση άθροισης

```
fun sum g 0 = g 0
  | sum g n = sum g (n-1) + g n
```

- Υπολογίστε τη συμπεριφορά της για g τη συνάρτηση ύψωσης στο τετράγωνο (square), και για g τη συνάρτηση διπλασιασμού (double).
- Εξάγετε τύπο για τη sum.

Η πιο γενική συνάρτηση ΗΟ

fun apply (f, x) = f x

- Υπολογίστε συμπεριφορά για f = double, x = 3
- Μπορούμε να εκμεταλλευτούμε τις συναρτήσεις ΗΟ για να μετατρέψουμε διαδικασιακά προγράμματα σε συναρτησιακά.

Procedural→FP (while loops)

```
PROG:   BEGIN   Z:=X;
                WHILE X=< Y DO
                BEGIN
                Z:=Z*X;
                X:=X+1
                END
        END
```

- Τι φαίνεται να κάνει το πρόγραμμα PROG;

```
PROG:   BEGIN   Z:=X;
                WHILE X=< Y DO
                BEGIN
                Z:=Z*X;
                X:=X+1
                END
        END
```

fun f1(x, y, z) = (x, y, x)

```
PROG:   BEGIN   Z:=X;
                WHILE X=< Y DO
                BEGIN
                Z:=Z*X;
                X:=X+1
                END
        END
```

fun f2 (x, y, z)= (x, y, z*x)

fun f3 (x, y, z) =(x+1, y, z)

val body = f3 . f2 ή ισοδύναμα body = apply (f3, f2)

```
PROG:   BEGIN   Z:=X;
                WHILE X=< Y DO
                BEGIN
                Z:=Z*X;
                X:=X+1
                END
        END
```

fun test (x, y, z) = x =<y

```
PROG:   BEGIN   Z:=X;
                WHILE X=< Y DO
                BEGIN
                Z:=Z*X;
                X:=X+1
                END
        END
```

```
fun loop (x, y, z)= if test (x, y, z)
                    then loop (body (x, y, z))
                    else (x, y, z)
```

```
PROG: BEGIN Z:=X;
            WHILE X=< Y DO
            BEGIN
            Z:=Z*X;
            X:=X+1
            END
      END
```

val prog = loop . f1 ή ισοδύναμα prog = apply (loop, f1).

Ισοδύναμης συμπεριφοράς FP

```
val prog = loop . f1
val body = f3 . f2
fun test (x, y, z) = x =<y
fun f1(x, y, z) = (x, y, x)
fun f2 (x, y, z)= (x, y, z*x)
fun f3 (x, y, z) =(x+1, y, z)
fun loop (x, y, z)= if test (x, y, z)
                    then loop (body (x, y, z))
                    else (x, y, z)
```

Οκνηρός και Άπληστος Υπολογισμός

- Μέχρι τώρα: υπολογισμός εκφράσεων μέσω μετεγγραφής/αντικατάστασης των εφαρμογών συναρτήσεων από τα αποτελέσματά τους, δηλαδή, ως διαδικασία **αναγωγής** με LR κανόνες μετεγγραφής (left to right rewrite rules).

Με LR μετεγγραφή/αντικατάσταση

val x= 3*2

Στην έκφραση **(x + x) – (4 * 2)** με LR μετεγγραφή

- πρώτα υπολογίζεται η τιμή x=6 , οπότε

(6 + 6) – (4 *2)

- μετά υπολογίζεται (6+6)=12

12 – (4 *2)

- μετά (4*2)=8 οπότε

12-8

- τέλος υπολογίζεται (12-8)=4.
- Αλλά αυτή δεν είναι η μόνη σειρά με την οποία μπορούν να πραγματοποιηθούν υπολογισμοί /αντικαταστάσεις!

Με RL μετεγγραφή /αντικατάσταση

val x= 3*2

Στην έκφραση **(x + x) – (4 * 2)** με RL μετεγγραφή

- πρώτα υπολογίζεται η τιμή (4*2)=8

(x + x) – 8

- μετά υπολογίζεται η τιμή x=6

(6+6) – 8

- μετά (6+6) =12 οπότε

12-8

- τέλος υπολογίζεται (12-8)=4.

Με «παραλληλοποίηση»

val x= 3*2

Στην έκφραση **(x + x) – (4 * 2)** με παραλληλοποίηση

- πρώτα παράλληλα : x=6
  (4*2)=8

οπότε (6+6)-8

- μετά (6+6)=12

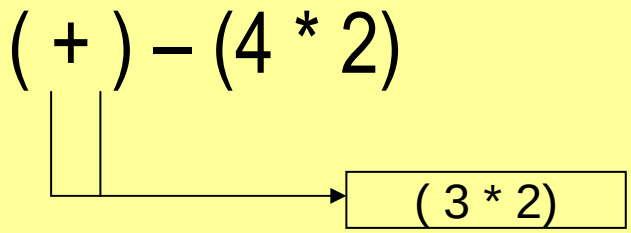
οπότε 12-8

- τέλος (12-8) =4

Αναβολή εφαρμογής της val

- Και στα τρία σενάρια, θα μπορούσε να αντιπαρατεθεί ένα εναλλακτικό στο οποίο δεν υπολογίζεται/αντικαθίσταται η τιμή x, δηλαδή αναβάλλεται η εφαρμογή της val.
- Στο LR (3*2 + 3*2)-(4*2)
- Στο RL (x+ x) - 8 → (3*2 + 3*2) -8
- Στο παράλληλο (3*2 + 3*2) – 8
- Οι περισσότεροι θα το απέφευγαν όμως, γιατί απαιτείται διπλός υπολογισμός (3*2)

Διαγραμματική απεικόνιση έκφρασης



- Δηλαδή το προτιμότερο θα ήταν να αναβληθεί η εφαρμογή της val για όσο χρειάζεται ώστε να πραγματοποιηθεί μόνο μια φορά.

Έχει σημασία η σειρά υπολογισμού;

- Βασική ιδιότητα του FP είναι ότι τα αποτελέσματα, εφόσον υπάρχουν, είναι μοναδικά, και δεν εξαρτώνται από την σειρά υπολογισμού (αντικατάστασης/μετεγγραφής).

- Αλλά η σειρά υπολογισμού μπορεί να επηρεάσει το αν ένα αποτέλεσμα θα παραχθεί (και να επηρεάσει και το χρόνο κατά τον οποίο θα παραχθεί), δηλαδή την **πληρότητα** και την **απόδοση** του υπολογισμού. Για παράδειγμα, τι θα συνέβαινε αν επεκτείναμε συνεχώς έναν αναδρομικό ορισμό, αναβάλλοντας διαρκώς αντικατάσταση με αποτέλεσμα;

- Υποθέστε ότι **n div 0** δεν είναι ορισμένο για οποιονδήποτε ακέραιο n.
- Έστω η έκφραση E= **(3= 3) or ((5 div 0)=4)**
- Με αναβολή της εφαρμογής της div θα υπολογίσουμε E= (true) or ((5 div 0)=4)

  και μετά, εξακολουθώντας να αναβάλουμε την div, από τον ορισμό της **or** (true or x = true) θα υπολογίσουμε τελικό αποτέλεσμα E= true.
- Χωρίς αναβολή της div θα είχε αποτύχει ο υπολογισμός της E!

Χειρισμός Λαθών

- Ο οκνηρός/άπληστος υπολογισμός απαιτεί επομένως και απόφαση σχετικά με το χειρισμό λαθών/μη ορισμένων συναρτήσεων.
- Οι συναρτήσεις που παράγουν αποτέλεσμα ακόμα και όταν η είσοδός τους δεν έχει οριστεί λέγονται **μη αυστηρές**, ενώ όσες δεν είναι ορισμένες η είσοδός τους δεν έχει οριστεί λέγονται **αυστηρές.**
- Αυτό σημαίνει ότι δεδομένου ενός ορισμού μιας συνάρτησης, πρέπει να επιλέξει κανείς αυστηρή ή μη αυστηρή ερμηνεία.

Μη αυστηρές γλώσσες FP

- Miranda, Ponder, Lazy ML, etc.
- Υιοθετούν οκνηρό υπολογισμό
- Πιο ευέλικτες για το χειρισμό λειτουργικότητας εισόδου/εξόδου
- Υποστηρίζουν απλούστερες περιγραφές συναρτήσεων
- Η αποδοτικότητά τους σε χρόνο και μνήμη δεν είναι επαρκώς κατανοητή, ούτε ο τρόπος με τον οποίο θα μπορούσε ο προγραμματιστής να ελέγξει θέματα αποδοτικότητας.

Αυστηρές γλώσσες FP

- Standard ML etc.
- Συνήθως υιοθετούν άπληστο υπολογισμό, και τα ορίσματα των συναρτήσεων υπολογίζονται πλήρως πριν την εφαρμογή της συνάρτησης.
- Έχουν παρόμοια λειτουργικά χαρακτηριστικά με τις διαδικασιακές γλώσσες προγραμματισμού.