



# Γλώσσες Προγραμματισμού II

<http://courses.softlab.ntua.gr/p12/>

Κωστής Σαγώνας

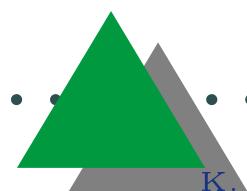
kostis@cs.ntua.gr

Νίκος Παπασπύρου

nickie@softlab.ntua.gr



Εθνικό Μετσόβιο Πολυτεχνείο  
Σχολή Ηλεκτρολόγων Μηχ. και Μηχ. Υπολογιστών  
Εργαστήριο Τεχνολογίας Λογισμικού  
Πολυτεχνειούπολη, 15780 Ζωγράφου.

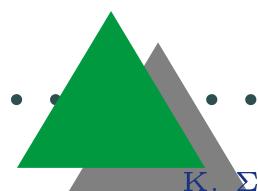


# Εξαγωγή τύπων

(i)

(Type inference)

- Μετασχηματίζει εκφράσεις χωρίς τύπους ή με ελλιπείς τύπους σε εκφράσεις με σωστούς τύπους, συμπληρώνοντας τις πληροφορίες τύπων που λείπουν
- Ενδιαφέροντα θεωρητικά ζητήματα αλλά και πολύ σημαντικές πρακτικές εφαρμογές
- Ιδιαίτερα χρήσιμη σε γλώσσες που υποστηρίζουν (παραμετρικό) πολυμορφισμό



# Εξαγωγή τύπων

(ii)

Μία βιαστική εισαγωγή στα συστήματα τύπων

- Γλώσσα εκφράσεων  $e$  και γλώσσα τύπων  $\tau$
- Σχέση αντιστοίχισης τύπων  $\Gamma \vdash e : \tau$
- Περιβάλλον τύπων  $\Gamma$ :  
απεικόνιση μεταβλητών  $x$  σε τύπους  $\tau$ , π.χ.

$$\Gamma = \{ i : int, z : real, f : int \rightarrow int \}$$

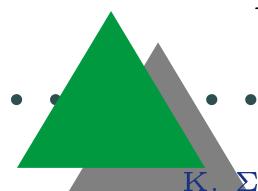
- Κανόνες τύπων

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 + e_2 : int}$$

# Εξαγωγή τύπων

(iii)

- Έστω  $L_T$  μια γλώσσα με δηλώσεις τύπων
- Έστω  $L_U$  μια παραλλαγή της ίδιας γλώσσας στην οποία οι δηλώσεις τύπων παραλείπονται
- Έστω μια συνάρτηση σβησίματος τύπων  
(type erasure function)  $\text{erase} : L_T \rightarrow L_U$
- Το πρόβλημα της εξαγωγής τύπων:  
δεδομένης μίας έκφρασης  $e_U \in L_U$ ,  
βρες μία έκφραση  $e_T \in L_T$  τέτοια ώστε
  - $\text{erase}(e_T) = e_U$  και
  - $\Gamma \vdash e_T : \tau$  (για χάποια  $\tau$  και  $\Gamma$ )



# Εξαγωγή τύπων

(iv)

- Ιδέα: μετασχηματισμός του προβλήματος  
 $? \vdash e : ?$  σε ένα σύνολο  $E$  που περιέχει  
εξισώσεις τύπων, π.χ.

$$E = \{ \alpha = \beta, \\ \gamma \rightarrow \alpha = (\beta \rightarrow \alpha) \rightarrow \gamma \}$$

- Αν το  $E$  έχει λύση, βρίσκουμε  $\Gamma$  και τ τέτοια  
ώστε  $\Gamma \vdash e : \tau$
- Διαφορετικά δεν υπάρχουν  $\Gamma$  και τ τέτοια ώστε  
 $\Gamma \vdash e : \tau$

# Μερικά παραδείγματα (i)

Με τον interpreter της OCaml [www.ocaml.org](http://www.ocaml.org)

```
# let inc (x : int) : int = x + 1;;
val f : int -> int = <fun>
# let inc x = x + 1;;
val f : int -> int = <fun>
```

- Πώς βρήκε τον τύπο;
  1. έστω  $\text{inc} : \alpha \rightarrow \beta$ , δηλαδή  $x : \alpha$  και  $x + 1 : \beta$
  2. πρέπει  $x : \text{int}$  **άρα**  $\alpha = \text{int}$
  3. τότε  $x + 1 : \text{int}$  **άρα**  $\beta = \text{int}$
  4. επομένως  $\text{inc} : \text{int} \rightarrow \text{int}$

# Μερικά παραδείγματα (ii)

- Πολυμορφισμός

```
# let id x = x;;
val id : 'a -> 'a = <fun>
# let fst (x, y) = x;;
val fst : 'a * 'b -> 'a = <fun>
# let rec map f l =
    match l with
    | []      -> []
    | h :: t -> f h :: map f t;;
val map : ('a -> 'b) -> 'a list -> 'b list = <fun>
```

# Μερικά παραδείγματα

(iii)

- Περιορισμός: let polymorphism

```
# let strange f = (f 5, f "hello");;
```

Characters 24-31:

```
let strange f = (f 5, f "hello");;  
^ ^ ^ ^ ^ ^
```

This expression has type string  
but is here used with type int

- ο τύπος του f είναι μονομορφικός!

# Εξαγωγή τύπων στην πράξη (i)

- Προσθέτουμε μεταβλητές στη γλώσσα των τύπων
- Έστω  $\{@1, @2, \dots\}$  ένα αριθμήσιμο υποσύνολο των μεταβλητών τύπων (ακόμα άγνωστοι τύποι)
- Συμπληρώνουμε τους τύπους που λείπουν στην αρχική έκφραση βάζοντας φρέσκες μεταβλητές

```
let f g x = g x (x + 1)  
and m a b = a * b
```

γίνεται:

```
let f (g : @1) (x : @2) : @3 = g x (x + 1)  
and m (a : @4) (b : @5) : @6 = a * b
```

# Εξαγωγή τύπων στην πράξη (ii)

- Αντικατάσταση τύπων (type substitution): απεικόνιση μεταβλητών τύπων σε τύπους

$$\sigma = [\alpha \mapsto int, \beta \mapsto bool \rightarrow \alpha]$$

- Εφαρμογή αντικατάστασης τύπων: ταυτόχρονα και μία φορά

$$\sigma(\alpha) = int \qquad \qquad \sigma(\beta) = bool \rightarrow \alpha$$

$$\sigma(\beta \rightarrow \gamma) = (bool \rightarrow \alpha) \rightarrow \gamma$$

- Σύνθεση αντικαταστάσεων  $\sigma_1 \circ \sigma_2$  έτσι ώστε  $(\sigma_1 \circ \sigma_2)(\tau) = \sigma_1(\sigma_2(\tau))$

# Εξαγωγή τύπων στην πράξη (iii)

- Το πρόβλημα  $? \vdash e : ?$  ανάγεται στην εύρεση μίας αντικατάστασης  $\sigma$  και ενός τύπου  $\tau$  ώστε  $\sigma(\Gamma) \vdash \sigma(e) : \tau$  για το αρχικό περιβάλλον  $\Gamma$
- Παράδειγμα

```
let f (g : @1) (x : @2) : @3 = g x (x + 1)
and m (a : @4) (b : @5) : @6 = a * b
in  f m 6
```

- Αρχικό περιβάλλον:  $\Gamma = \emptyset$
- Αρχικό περιβάλλον για το σώμα  $f\ m\ 6$ :  
$$\Gamma_b = \{ f : @1 \rightarrow @2 \rightarrow @3, m : @4 \rightarrow @5 \rightarrow @6 \}$$

# Εξαγωγή τύπων στην πράξη (iv)

- Παράδειγμα (συνέχεια)

```
let f (g : @1) (x : @2) : @3 = g x (x + 1)
and m (a : @4) (b : @5) : @6 = a * b
in  f m 6
```

- Μια λύση (η μοναδική)

$$\begin{aligned}\sigma &= [ @1 \mapsto int \rightarrow int \rightarrow int, @2 \mapsto int, @3 \mapsto int, \\ &\quad @4 \mapsto int, @5 \mapsto int, @6 \mapsto int ] \\ \tau &= int\end{aligned}$$

- Γενικά οι λύσεις δεν είναι μοναδικές!

```
let id (x : @1) : @2 = x
```

# Περιορισμοί και επίλυση (i)

- Πώς βρίσκεται η λύση;
- Περιορισμός (constraint): εξίσωση τύπων  $\tau_1 = \tau_2$
- Πρόβλημα 1: εύρεση συνόλου περιορισμών  $C$

```
let f (g : @1) (x : @2) : @3 = g x (x + 1)
and m (a : @4) (b : @5) : @6 = a * b
in f m 6
```

οδηγεί στο σύνολο περιορισμών:

$$\begin{aligned} C = & \{ @1 = @2 \rightarrow \text{int} \rightarrow @3, @2 = \text{int}, \\ & @4 = \text{int}, @5 = \text{int}, @6 = \text{int}, \\ & @1 = @4 \rightarrow @5 \rightarrow @6, @2 = \text{int} \} \end{aligned}$$

# Περιορισμοί και επίλυση (ii)

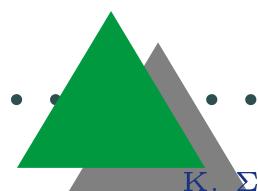
- Παραγωγή τύπων με περιορισμούς

$$\Gamma \vdash E : \tau' \mid C$$

- Πρόβλημα 2: επίλυση συνόλου περιορισμών
- Μια αντικατάσταση  $\sigma$  λέγεται **ενοποιητής** (unifier) για τον περιορισμό  $\tau_1 = \tau_2$  αν οι τύποι  $\sigma(\tau_1)$  και  $\sigma(\tau_2)$  ταυτίζονται  $\sigma(\tau_1) \equiv \sigma(\tau_2)$
- **Λύση** για το πρόβλημα της εξαγωγής τύπων:
  - ένας ενοποιητής  $\sigma$  για κάθε περιορισμό του  $C$
  - ο τύπος  $\tau = \sigma(\tau')$

# Περιορισμοί και επίλυση (iii)

- Ο ενοποιητής  $\sigma$  είναι πιο γενικός από τον  $\sigma'$  αν υπάρχει αντικατάσταση  $\sigma_\delta$  τέτοια ώστε  $\sigma' = \sigma_\delta \circ \sigma$
- Πιο γενικός ενοποιητής (most general unifier): ενοποιητής  $\sigma$  τέτοιος ώστε να είναι πιο γενικός από κάθε άλλον ενοποιητή  $\sigma'$
- Αν υπάρχει πιο γενικός ενοποιητής, αυτός δίνει τον πρωτεύοντα τύπο (principal type)
- Ο αλγόριθμος **W** για το λ-λογισμό (διαφ. 42) υπολογίζει τον πιο γενικό ενοποιητή



# Εφαρμογή στο $\lambda$ -λογισμό (i)

- Τύποι

$$\sigma, \tau ::= \alpha \mid (\sigma \rightarrow \tau)$$

To → είναι δεξιά προσεταιριστικό, π.χ.

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

- Κανόνες τύπων à-la Curry

$$\frac{(x : \sigma) \in \Gamma}{\Gamma \vdash x : \sigma} \quad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x. M) : (\sigma \rightarrow \tau)}$$

$$\frac{\Gamma \vdash M : (\sigma \rightarrow \tau) \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (M N) : \tau}$$

# Εφαρμογή στο λ-λογισμό (ii)

- **Ενοποίηση**: επίλυση συνόλου περιορισμών

$$\text{unify}(\emptyset) = \sigma_0 \quad \{\eta \text{ κενή αντικατάσταση}\}$$

$$\text{unify}(\{\tau_1 = \tau_2\} \cup C) =$$

**if**  $\tau_1 \equiv \tau_2$  **then**

$\text{unify}(C)$

**else if**  $\tau_1 \equiv \alpha$  **και** δεν εμφανίζεται στο  $\tau_2$  **then**

$\text{unify}([\alpha \mapsto \tau_2]C) \circ [\alpha \mapsto \tau_2]$

**else if**  $\tau_2 \equiv \alpha$  **και** δεν εμφανίζεται στο  $\tau_1$  **then**

$\text{unify}([\alpha \mapsto \tau_1]C) \circ [\alpha \mapsto \tau_1]$

**else if**  $\tau_1 \equiv \tau_{11} \rightarrow \tau_{12}$  **και**  $\tau_2 \equiv \tau_{21} \rightarrow \tau_{22}$  **then**

$\text{unify}(C \cup \{\tau_{11} = \tau_{21}, \tau_{12} = \tau_{22}\})$

**else**

    η ενοποίηση αποτυγχάνει