Γλώσσες Προγραμματισμού II

http://courses.softlab.ntua.gr/pl2/

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 τ
- Σχέση αντιστοίχισης τύπων $\quad \Gamma \vdash e : \tau$
- Περιβάλλον τύπων Γ:
  απεικόνιση μεταβλητών x σε τύπους τ, π.χ.

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

- Κανόνες τύπων

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

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) $\quad \text{erase} : L_T \rightarrow L_U$
- Το πρόβλημα της εξαγωγής τύπων: δεδομένης μίας έκφρασης $e_U \in L_U$, βρες μία έκφραση $e_T \in L_T$ τέτοια ώστε
  - $\text{erase}(e_T) = e_U \quad$ και
  - $\Gamma \vdash e_T : \tau \qquad$ (για κάποια τ και Γ)

Εξαγωγή τύπων (iv)

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

$$\begin{aligned} E &= \{\, \alpha = \beta, \\ &\qquad \gamma \to \alpha = (\beta \to \alpha) \to \gamma \,\} \end{aligned}$$

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

Μερικά παραδείγματα (i)

Με τον interpreter της OCaml 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 $\texttt{inc} : \alpha \rightarrow \beta$, δηλαδή $\texttt{x} : \alpha$ και $\texttt{x} + 1 : \beta$
  2. πρέπει $\texttt{x} : int$ άρα $\alpha = int$
  3. τότε $\texttt{x} + 1 : int$ άρα και $\beta = int$
  4. επομένως $\texttt{inc} : int \rightarrow int$

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 $\{ @1, @2, \ldots \}$ 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

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

- Εφαρμογή αντικατάστασης τύπων: ταυτόχρονα και μία φορά

$$\sigma(\alpha) \;=\; int \qquad\qquad \sigma(\beta) \;=\; bool \to \alpha$$
$$\sigma(\beta \to \gamma) \;=\; (bool \to \alpha) \to \gamma$$

- 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 σ και ενός τύπου τ ώστε $\sigma(\Gamma) \vdash \sigma(e) : \tau$ για το αρχικό περιβάλλον Γ
- 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`: $\Gamma_b = \{\, \texttt{f} : \texttt{@1} \rightarrow \texttt{@2} \rightarrow \texttt{@3},\ \texttt{m} : \texttt{@4} \rightarrow \texttt{@5} \rightarrow \texttt{@6} \,\}$

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)

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

- 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:

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

Περιορισμοί και επίλυση (ii)

- Παραγωγή τύπων με περιορισμούς

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

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

Περιορισμοί και επίλυση (iii)

- Ο ενοποιητής σ είναι πιο γενικός από τον σ' αν υπάρχει αντικατάσταση σ_δ τέτοια ώστε $\sigma' = \sigma_\delta \circ \sigma$
- Πιο γενικός ενοποιητής (most general unifier): ενοποιητής σ τέτοιος ώστε να είναι πιο γενικός από κάθε άλλον ενοποιητή σ'
- 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 λ-λογισμό (i)

- Τύποι

$$\sigma, \tau \;::=\; \alpha \mid (\sigma \to \tau)$$

Το $\to$ είναι δεξιά προσεταιριστικό, π.χ.

$$(\alpha \to (\beta \to \gamma)) \qquad \alpha \to \beta \to \gamma$$

- Κανόνες τύπων à-la Curry

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

$$\frac{\Gamma \vdash M : (\sigma \to \tau) \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (M\, N) : \tau}$$

Εφαρμογή στο λ-λογισμό (ii)

- Ενοποίηση: επίλυση συνόλου περιορισμών

unify($\emptyset$) $= \sigma_0$ {η κενή αντικατάσταση}

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

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

unify(C)

else if $\tau_1 \equiv \alpha$ και δεν εμφανίζεται στο τ_2 **then**

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

else if $\tau_2 \equiv \alpha$ και δεν εμφανίζεται στο τ_1 **then**

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**

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

else

η ενοποίηση αποτυγχάνει