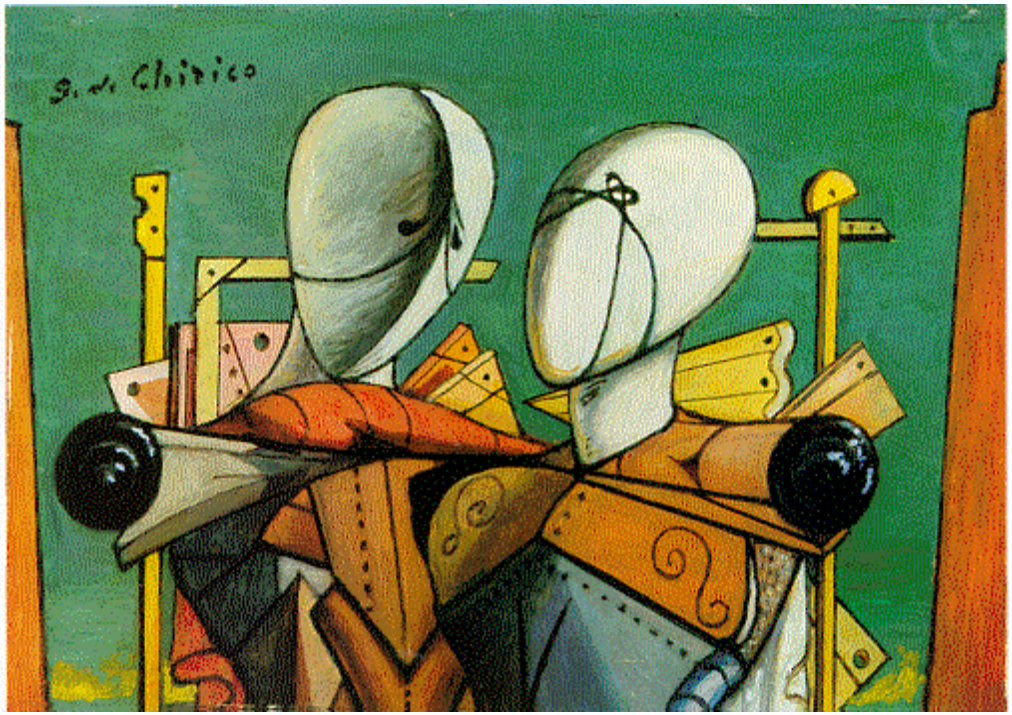
Συμπερασμός Τύπων και Πολυμορφισμός



Giorgio de Chirico, *Etorre e Andromaca, 1915-1925*

Κωστής Σαγώνας <kostis@cs.ntua.gr>
Ζωή Παρασκευοπούλου <zoepar@softlab.ntua.gr>

Εισαγωγή: Σύγκριση μεταξύ γλωσσών

C:
```
int f(char a, char b) {
  return a == b;
}
```

ML:
```
- fun f(a, b) = (a = b);
val f = fn : ''a * ''a -> bool
```

- Η συνάρτηση σε ML γράφεται πιο εύκολα: ο προγραμματιστής δε χρειάζεται να ορίσει τύπους
- Η συνάρτηση σε ML είναι πιο ευέλικτη: μπορεί να χρησιμοποιηθεί για κάθε τύπο (που υποστηρίζει ισότητα)
- Συναρτήσεις σαν και την παραπάνω, οι οποίες δουλεύουν για πολλούς τύπους, ονομάζονται **πολυμορφικές**

Περιεχόμενα

- Συμπερασμός τύπων (type inference)
  - Καλό παράδειγμα αλγόριθμου και εφαρμογής στατικής ανάλυσης προγραμμάτων
  - Θα δούμε τον αλγόριθμο μέσω παραδειγμάτων
- Είδη Πολυμορφισμού
  - Υπερφόρτωση (overloading)
  - Μέσω αυτόματης μετατροπής τύπων (type coercion)
  - Παραμετρικός πολυμορφισμός (parametric polymorphism)
  - Πολυμορφισμός υποτύπων (subtype polymorphism)

Συμπερασμός τύπων

Έλεγχος τύπων έναντι συμπερασμού τύπων

- **Έλεγχος τύπων**

```
int f(int x) { return x+1; };
int g(int y) { return f(y+1)*2; };
```

  - Ελέγχουμε αν στο σώμα της συνάρτησης αν η χρήση των τελεστών/συναρτήσεων είναι συνεπής ως προς τους τύπους που υποστηρίζουν
  - Χρησιμοποιούμε τον τύπο που έχει δηλωθεί για κάθε παράμετρο

- **Συμπερασμός τύπων**

```
int f(int x) { return x+1; };
int g(int y) { return f(y+1)*2; };
```

  - Οι δηλώσεις συναρτήσεων και μεταβλητών μπορεί να μην έχουν επισημειώσεις τύπων
  - Κοιτάμε στον κώδικα και «μαντεύουμε» ποιοι τύποι θα έπρεπε να είχαν δηλωθεί ώστε το πρόγραμμα να είναι συνεπές ως προς τη χρήση των τύπων

- Η ML έχει σχεδιαστεί ώστε ο συμπερασμός τύπων να είναι βατός (tractable)

Χρησιμότητα

- Τύποι και έλεγχος τύπων
  - Τα συστήματα τύπων εξελίσσονται συνεχώς από την Algol 60 και μετά, βελτιώνοντας την αξιοπιστία των γλωσσών προγραμματισμού.
  - **Ασφάλεια των προγραμμάτων:** εντοπισμός σφαλμάτων πριν από την εκτέλεση
  - **Βελτίωση της απόδοσης:** ο μεταγλωττιστής μπορεί να κάνει βελτιστοποιήσεις βάσει τύπων
- Συμπερασμός τύπων
  - Από τις σημαντικότερες εξελίξεις τόσο στη θεωρία όσο και στην πρακτική των γλωσσών προγραμματισμού.
  - Επιτρέπει στους προγραμματιστές να **παραλείπουν** τη **ρητή δήλωση** τύπων, ενώ ο μεταγλωττιστής τους υπολογίζει αυτόματα
  - Επιρροή σε πάρα πολλές σύγχρονες γλώσσες προγραμματισμού (Haskell, OCaml, Rust, Scala, Kotlin, TypeScript, …)

Συμπερασμός τύπων στην ML

- Παράδειγμα

```
- fun add2 x = 2 + x;
val add2 = fn : int -> int
```

- Πώς συμπεραίνουμε τον παραπάνω τύπο;
  - Ο **+** έχει δύο τύπους: **`int * int -> int`**
    ή **`real * real -> real`**
  - Η σταθερά **2** έχει τύπο **`int`**
  - Εδώ χρησιμοποιούμε τον τύπο : **`int * int -> int`**
  - Αυτό με τη σειρά του σημαίνει ότι **`x:int`**
  - Επομένως η συνάρτηση **`add2`** έχει τύπο **`int -> int`**

Οι υπερφορτωμένοι τελεστές και συναρτήσεις, όπως ο + είναι σπάνιοι.

Τα περισσότερα σύμβολα στην ML έχουν μοναδικό τύπο.

Σε πολλές περιπτώσεις, ο μοναδικός αυτός τύπος είναι πολυμορφικός.

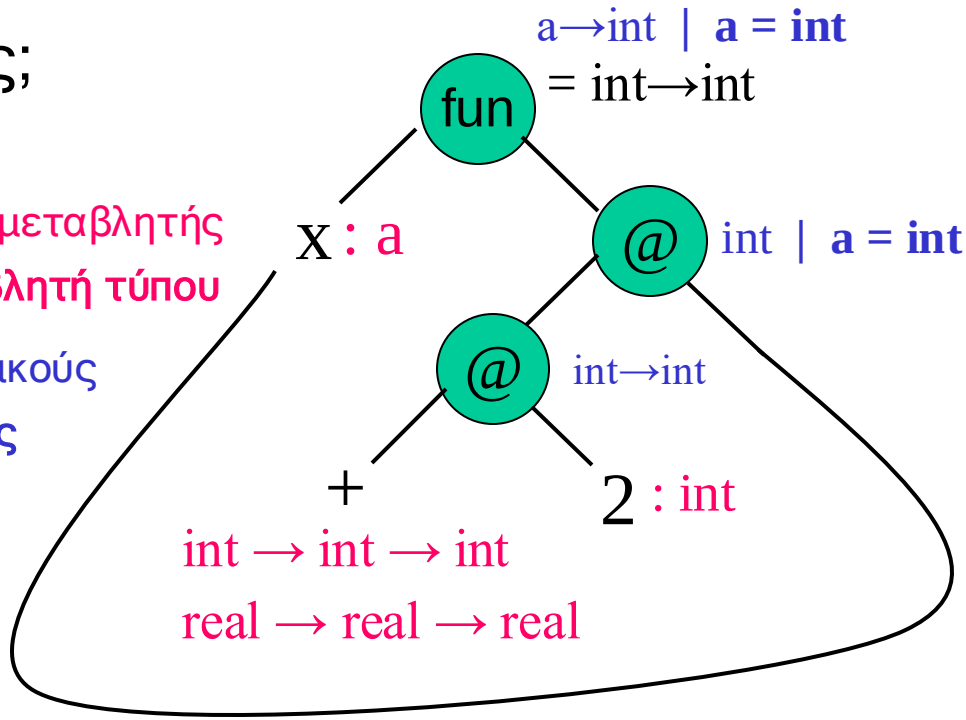
Μια διαφορετική παρουσίαση του συμπερασμού

- Παράδειγμα

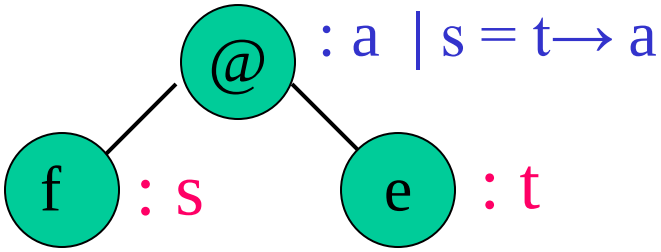
```
- fun add2 x = 2+x;
val add2 = fn : int -> int
```

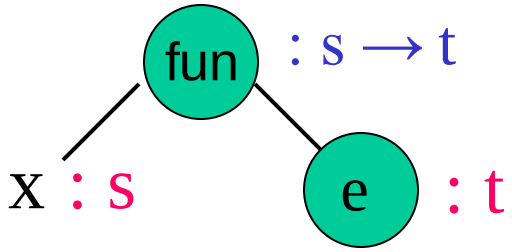
Γράφος για
`fn x => ((+ 2) x)`

- Πώς συμπεραίνεται ο τύπος;
  1. Αναθέτουμε τύπους στα φύλλα.
     Αν δεν γνωρίζουμε τον τύπο μιας μεταβλητής χρησιμοποιούμε μια **φρέσκια μεταβλητή τύπου**
  2. Προωθούμε τύπους στους εσωτερικούς κόμβους και γεννάμε **περιορισμούς**
  3. Επιλύουμε μέσω αντικαταστάσεων (ενοποίησης)



Εφαρμογή και ορισμός συναρτήσεων





- **Εφαρμογή συνάρτησης**
  - Η f έχει τύπο συνάρτησης πεδίο ορισμού → πεδίο τιμών
  - Το πεδίο ορισμού της f είναι ίδιο με τον τύπο του ορίσματος e
  - Ο τύπος του αποτελέσματος είναι ο τύπος του πεδίου τιμών της f
  - Εάν χρειάζεται χρησιμοποιούμε μια φρέσκια μεταβλητή τύπου για το αποτέλεσμα (π.χ. a)
- **Ορισμός συνάρτησης**
  - Ο τύπος της συνάρτησης είναι πεδίο ορισμού → πεδίο τιμών
  - Πεδίο ορισμού είναι ο τύπος της μεταβλητής x
  - Πεδίο τιμών είναι ο τύπος του αποτελέσματος του σώματος e της συνάρτησης

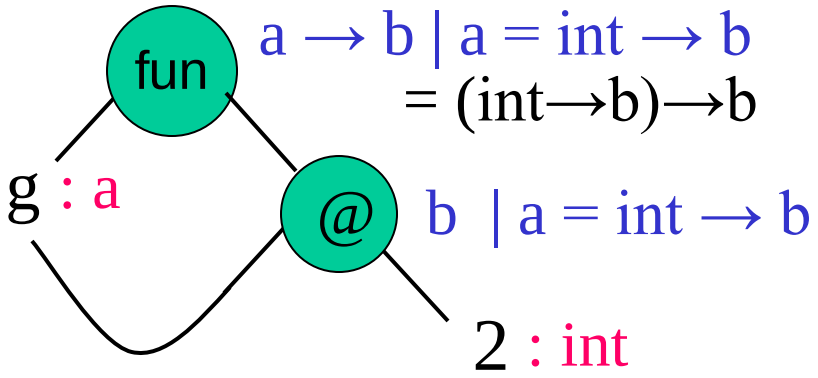
Τύποι με μεταβλητές τύπων

- Παράδειγμα

```
- fun f g = g 2;
val f = fn : (int -> 'a) -> 'a
```

Γράφος για
`fun g => g 2`

- Πώς συμπεραίνεται ο τύπος;



Αναθέτουμε τύπους στα φύλλα

Προωθούμε τύπους στους εσωτερικούς κόμβους και γεννάμε περιορισμούς

Επιλύουμε τους περιορισμούς (εννοποίηση) και βρίσκοντας αντικαταστάσεις για τις μεταβλητές τύπων

Χρήση πολυμορφικών συναρτήσεων

- Συνάρτηση

```
- fun f g = g 2;
val f = fn : (int -> 'a) -> 'a
```

- Πιθανές χρήσεις

```
- fun add2 x = x + 2;
val add2 = fn : int -> int
- f add2;
val it = 4 : int
```

```
- fun isEven x = (x mod 2) = 0;
val isEven = fn : int -> bool
- f isEven;
val it = true : bool
```

Αναγνώριση σφάλματος τύπων

- Έστω η συνάρτηση:

```
- fun f g = g 2;
val f = fn : (int -> 'a) -> 'a
```

- Λάθος χρήση:

```
- fun not x = if x then false else true;
val not = fn : bool -> bool
- f not;
```

- **Σφάλμα τύπου:** δεν είναι δυνατόν ο τύπος `bool -> bool` να είναι στιγμιότυπο του τύπου `int -> 'a`

Ακόμα ένα παράδειγμα συμπερασμού τύπων

- Έστω η συνάρτηση

```
- fun f (g,x) = g (g x);
val f = fn : ('a -> 'a) * 'a -> 'a
```

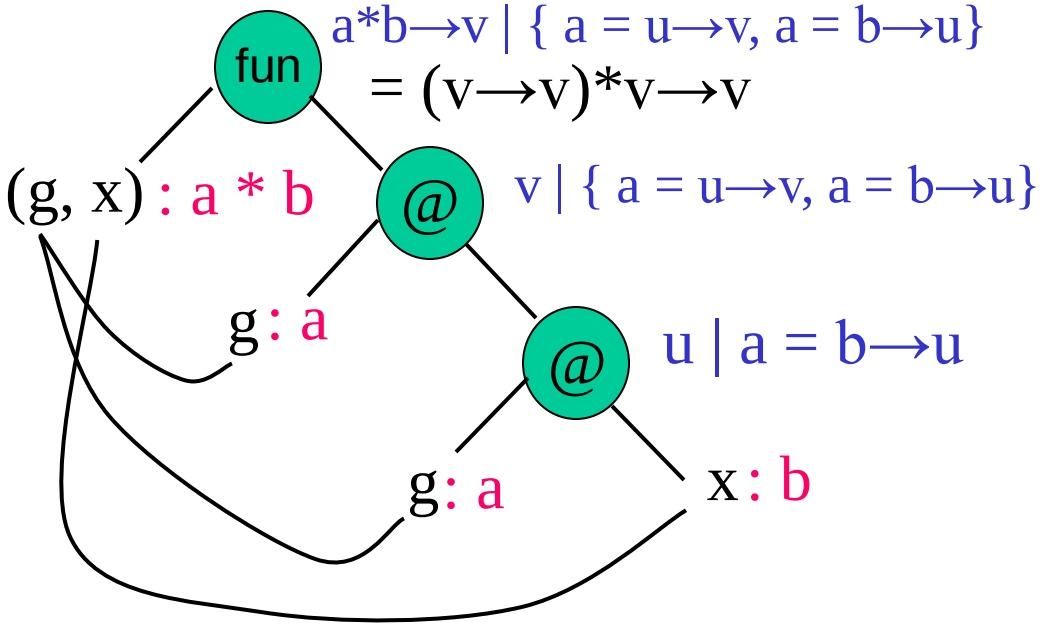
- Συμπερασμός τύπου

Αναθέτουμε τύπους στα φύλλα

Προωθούμε τύπους στους εσωτερικούς κόμβους και γεννάμε περιορισμούς

Επιλύουμε μέσω ενοποίησης

Γράφος για **`fun(g,x)=> g(g x)`**



Πολυμορφικοί τύποι δεδομένων

- Τύποι δεδομένων με μεταβλητές τύπου

```
- datatype 'a lst = Nil | Cons of 'a * ('a lst);
Nil : 'a lst
Cons : 'a * ('a lst) -> 'a lst
```

- Πολυμορφική συνάρτηση

```
- fun my_len Nil = 0
=   | my_len Cons(h, t) = 1 + my_len t;
val my_len = fn : 'a lst -> int
```

- Συμπερασμός τύπων
  - Συμπεραίνουμε κάποιο τύπο για κάθε περίπτωση ξεχωριστά
  - Συνδυάζουμε τους τύπους με τον περιορισμό ότι πρέπει να είναι συμβατοί μεταξύ τους (οι τύποι ενοποιούνται αν αυτό είναι αναγκαίο)

Κύρια σημεία για το συμπερασμό τύπων

- Υπολογίζουμε τον τύπο της έκφρασης
  - Δε χρειαζόμαστε δηλώσεις για τον τύπο των μεταβλητών
  - Βρίσκουμε τον *πιο γενικό τύπο* μέσω επίλυσης περιορισμών
  - Το παραπάνω αυτόματα οδηγεί σε πολυμορφισμό συναρτήσεων
- Στατικός έλεγχος τύπων χωρίς προδιαγραφές τύπων
- Πολλές φορές οδηγεί σε καλύτερη αναγνώριση σφαλμάτων από ό,τι ο κοινός έλεγχος τύπων
  - Ο συμπερασμός τύπων μπορεί να αναδείξει κάποιο προγραμματιστικό λάθος **ακόμα και αν δεν υπάρχει σφάλμα τύπων** (βλέπε παράδειγμα στην επόμενη διαφάνεια)

Χρήσιμη πληροφορία από συμπερασμό τύπων

- Μια συνάρτηση για λίστες:

```
fun my_rev Nil = Nil
  | my_rev Cons(_,t) = my_rev t
```

- Ο τύπος που συμπεραίνεται από την ML είναι:

```
my_rev : 'a lst -> 'b lst
```

- Τι δείχνει αυτός ο τύπος;

  Αφού η αναστροφή μιας λίστας δεν (πρέπει να) αλλάζει τον τύπο των στοιχείων της λίστας, πρέπει να υπάρχει κάποιο λάθος στον παραπάνω ορισμό της "**`my_rev`**"

Πολυμορφισμός

Διαφορετικά Είδη Πολυμορφισμού

- Πολυμορφισμός: η δυνατότητα μιας συνάρτησης (ή τελεστή) να εφαρμοστεί σε δεδομένα διαφορετικού τύπου
- Υπάρχουν διάφορες προσεγγίσεις:
  - Πολυμορφισμός μέσω υπερφόρτωσης
  - Πολυμορφισμός μέσω εξαναγκασμού τύπου
  - Παραμετρικός πολυμορφισμός
  - Πολυμορφισμός υποτύπων

Πολυμορφισμός μέσω υπερφόρτωσης

Υπερφόρτωση (overloading)

- Μια συνάρτηση (ή ένας τελεστής) είναι **υπερφορτωμένη** όταν έχει τουλάχιστον δύο ορισμούς για διαφορετικούς τύπους ορισμάτων
- Πολλές γλώσσες έχουν υπερφορτωμένους τελεστές

ML:

```
val x = 1 + 2
val y = 1.0 + 2.0
```

Python:

```
a = 1 + 2
b = 1.0 + 2.0
c = "hello " + "there»
d = [1,2,3] + [4,5,6]
```

- Επίσης, κάποιες γλώσσες επιτρέπουν τον ορισμό νέων υπερφορτωμένων συναρτήσεων ή τελεστών

Προσθήκη σε ήδη υπερφορτωμένους τελεστές

- Κάποιες γλώσσες, όπως η C++, επιτρέπουν πρόσθετη υπερφόρτωση των ήδη υπερφορτωμένων τελεστών

```
class complex {
  double rp, ip; // real part, imaginary part
public:
  complex(double r, double i) {rp = r; ip = i;}
  friend complex operator+(complex, complex);
  friend complex operator*(complex, complex);
};

void f(complex a, complex b, complex c) {
  complex d = a + b * c;
  …
}
```

Υπερφόρτωση τελεστών στη C++

- Η C++ επιτρέπει σχεδόν σε όλους τους τελεστές την πρόσθετη υπερφόρτωση, συμπεριλαμβανομένων των:
  - Πιο συχνά χρησιμοποιούμενων τελεστών (**+**,**-**,*****,**/**,**%**,**^**,**&**,**|**,**~**,**!**, **=**,**<**,**>**, **+=**,**-=**,**=**,***=**,**/=**,**%=**,**^=**,**&=**,**|=**,**<<**,**>>**,**>>=**,**<<=**,**==**, **!=**,**<=**,**>=**,**&&**,**||**,**++**,**--**,**->***,**,**)
  - Αποδεικτοδότησης (dereferencing) (***p** και **p->x**)
  - Χρήσης δεικτών (**a[i]**)
  - Κλήσης συνάρτησης (**f(a,b,c)**)
  - Δέσμευσης και αποδέσμευσης μνήμης (**new** και **delete**)

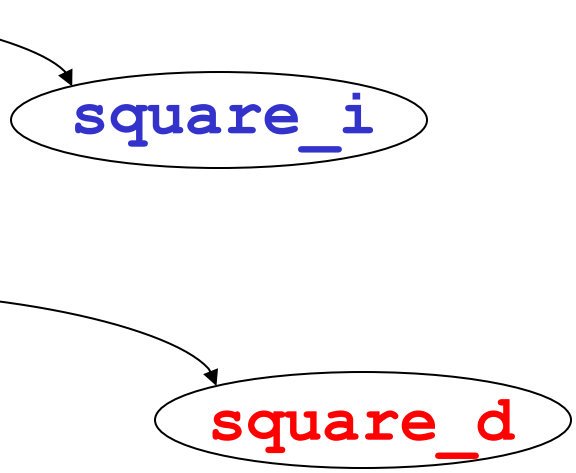
Ορισμός υπερφορτωμένων συναρτήσεων

- Κάποιες γλώσσες, όπως η C++, επιτρέπουν την υπερφόρτωση των ονομάτων των συναρτήσεων

```
int square(int x) {
  return x*x;
}

double square(double x) {
  return x*x;
}
```

Όμως η υπερφόρτωση εξαφανίζεται στη C++

```
int square(int x) {
  return x*x;
}

double square(double x) {
  return x*x;
}

void f() {
  int a = square(3);
  double b = square(3.0);
}
```



Δίνουμε καινούργια (μοναδικά) ονόματα σε υπερφορτωμένους ορισμούς συναρτήσεων…

Εξαφάνιση υπερφόρτωσης στη C++

```
int square_i(int x) {
  return x*x;
}

double square_d(double x) {
  return x*x;
}

void f() {
  int a = square_i(3);
  double b = square_d(3.0);
}
```

Και στη συνέχεια μετονομάζουμε τις κλήσεις (ανάλογα με τους τύπους των ορισμάτων τους)

Υλοποίηση υπερφόρτωσης στη C++

- Οι μεταγλωττιστές συνήθως υλοποιούν την υπερφόρτωση:
  - Δημιουργούν μονομορφικές συναρτήσεις, μια για κάθε ορισμό
  - Εφευρίσκουν ένα νέο όνομα για κάθε ορισμό το οποίο κωδικοποιεί την πληροφορία για τους τύπους
  - Κάθε κλήση χρησιμοποιεί το κατάλληλο όνομα ανάλογα με τους τύπους των παραμέτρων

C++:

```
int shazam(int a, int b) {return a+b;}
double shazam(double a, double b) {return a+b;}
```

Assembler:

```
shazam__Fii:
        lda $30,-32($30)
        .frame $15,32,$26,0
        …
shazam__Fdd:
        lda $30,-32($30)
        .frame $15,32,$26,0
        …
```

Πολυμορφισμός μέσω εξαναγκασμού μετατροπής τύπων

Αυτόματος εξαναγκασμός τύπου (Coercion)

- Σε πολλές γλώσσες ο μεταγλωττιστής εξαναγκάζει την αυτόματη μετατροπή τύπου (type coercion), ακόμα και σε περιπτώσεις που οι μετατροπές δεν είναι άμεσα δηλωμένες από τον προγραμματιστή

Δήλωση μετατροπής τύπου στη Java:

```
double x;
x = (double) 42;
```

Coercion στη Java:

```
double x;
x = 42;
```

Αυτόματη μετατροπή παραμέτρων

- Διαφορετικές γλώσσες υποστηρίζουν διαφορετικές μετατροπές σε διαφορετικές περιπτώσεις: σε αναθέσεις, σε δυαδικούς τελεστές, σε μοναδιαίους τελεστές, σε παραμέτρους, κ.λπ.
- Όταν μια γλώσσα υποστηρίζει αυτόματους εξαναγκασμούς μετατροπής τύπου σε παραμέτρους μιας κλήσης συνάρτησης (ή σε μια χρήση τελεστή), τότε η συνάρτηση (ή ο τελεστής) είναι **πολυμορφική (πολυμορφικός)**

Παράδειγμα: Java

```
void f(double x) {
   …
}

f((byte) 1);
f((short) 2);
f('a');
f(3);
f(4L);   // long
f(5.6F); // float
```

Η συνάρτηση **f** μπορεί να κληθεί με κάθε τύπο παραμέτρου που μπορεί να μετατραπεί αυτόματα σε **double** στη Java

Ορισμός αυτόματων μετατροπών τύπων

- Οι γλώσσες ξοδεύουν μεγάλο μέρος του τυπικού ορισμού τους στο να ορίσουν επακριβώς τους επιτρεπόμενους αυτόματους εξαναγκασμούς μετατροπής τύπου και το πώς αυτοί λαμβάνουν χώρα
- Κάποιες γλώσσες, ειδικά κάποιες παλιές γλώσσες όπως η Algol 68 και η PL/I, επιτρέπουν πολλές αυτόματες μετατροπές τύπων
- Κάποιες άλλες, όπως η ML, δεν επιτρέπουν καμία
- Οι περισσότερες, όπως η Java, είναι κάπου ενδιάμεσα

Παράδειγμα: Java

5.6.1 **Unary Numeric Promotion**

Some operators apply *unary numeric promotion* to a single operand, which must produce a value of a numeric type: If the operand is of compile-time type **`byte`**, **`short`**, or **`char`**, unary numeric promotion promotes it to a value of type **`int`** by a widening conversion (§5.1.2). Otherwise, a unary numeric operand remains as is and is not converted.

Unary numeric promotion is performed on expressions in the following situations: the dimension expression in array creations (§15.9); the index expression in array access expressions (§15.12); operands of the unary operators plus **`+`** (§15.14.3) and minus **`-`** (§15.14.4) ...

The Java Language Specification
James Gosling, Bill Joy, Guy Steele

Αυτόματες μετατροπές τύπων και υπερφόρτωση

- Η αυτόματη μετατροπή τύπων συνήθως έχει περίεργες αλληλεπιδράσεις με την υπερφόρτωση συναρτήσεων
- Αυτό συμβαίνει διότι
  - Η υπερφόρτωση χρησιμοποιεί τους τύπους για την επιλογή του ορισμού που θα χρησιμοποιηθεί
  - Η αυτόματη μετατροπή τύπων χρησιμοποιεί τον ορισμό για να αποφασίσει τι είδους μετατροπή θα πρέπει να γίνει

Παραδείγματα… σύγχυσης

- Έστω ότι, όπως στη C++, η γλώσσα επιτρέπει την αυτόματη μετατροπή **`char`** σε **`int`** ή σε **`double`**
- Ποια **`square`** καλείται σε μια κλήση **`square('a')`**;

```
int square(int x) {
  return x*x;
}
```

```
double square(double x) {
  return x*x;
}
```

- Έστω ότι, όπως στη C++, η γλώσσα επιτρέπει την αυτόματη μετατροπή **`char`** σε **`int`**
- Ποια **`f`** καλείται σε μια κλήση **`f('a','b')`**;

```
void f(int x, char y) {
  …
}
```

```
void f(char x, int y) {
  …
}
```

Παραμετρικός Πολυμορφισμός

Παραμετρικός πολυμορφισμός

- Μια συνάρτηση είναι **παραμετρικά πολυμορφική** εάν έχει τύπο που περιέχει μία ή περισσότερες μεταβλητές τύπου
- Ένας τύπος με μεταβλητές τύπων είναι ένας **πολυτύπος**
- Παραμετρικός πολυμορφισμός συναντιέται σε πολλές γ' γλώσσες όπως οι ML, C++ (templates), Java και Rust (generics)

Παράδειγμα: C++ Function Templates

```
template<typename X>
X max(X a, X b) {
  return a>b ? a : b;
}

void g(int a, int b, char c, char d) {
  int  m1 = max(a,b);
  char m2 = max(c,d);
}
```

Ο τελεστής σύγκρισης **>** *μπορεί να είναι πρόσθετα υπερφορτωμένος, οπότε η μεταβλητή τύπου* **X** *δεν περιορίζεται μόνο σε τύπους για τους οποίους ο τελεστής* **>** *είναι προκαθορισμένος.*

Παράδειγμα: Συναρτήσεις σε ML

```
- fun identity x = x;
val identity = fn : 'a -> 'a
- identity 42;
val it = 42 : int
- identity "hello";
val it = "hello" : string
- fun reverse x =
=   if null x then nil
=   else (reverse (tl x)) @ [(hd x)];
val reverse = fn : 'a list -> 'a list
```

Υλοποίηση παραμετρικού πολυμορφισμού

- Το ένα άκρο: πολλά αντίγραφα του κώδικα
  - Δημιουργείται ένα σύνολο από μονομορφικές συναρτήσεις, μία για κάθε πιθανό στιγμιότυπο των μεταβλητών τύπου
    - Κάθε αντίγραφο είναι μια μονομορφική υλοποίηση
    - Η οποία όμως μπορεί να βελτιστοποιηθεί/προσαρμοστεί στο συγκεκριμένο τύπο
- Το άλλο άκρο: ο ίδιος κώδικας
  - Δημιουργείται μία μόνο υλοποίηση και χρησιμοποιείται για όλες τις κλήσεις (αληθινός καθολικός πολυμορφισμός)
  - Δε μπορεί να βελτιστοποιηθεί για χρήση συγκεκριμένων τύπων
- Βεβαίως υπάρχουν και πολλές ενδιάμεσες υλοποιήσεις

Πολυμορφισμός Υποτύπων

Πολυμορφισμός υποτύπων

- Μια συνάρτηση (ή ένας τελεστής) είναι **πολυμορφική ως προς υποτύπους** εάν κάποια από τις παραμέτρους τύπων της έχει υποτύπους
- Είναι σημαντική πηγή πολυμορφισμού σε γλώσσες με πλούσια δομή υποτύπων
- Τέτοιες είναι οι περισσότερες αντικειμενοστρεφείς γλώσσες προγραμματισμού (π.χ. η Java)

Παράδειγμα: Java

```
class Car {
  void brake() { … }
}

class ManualCar extends Car
{
  void clutch() { … }
}

void g(Car z) {
  z.brake();
}

void f(Car x, ManualCar y) {
  g(x);
  g(y);
}
```

Υποτύπος της κλάσης **Car** *είναι η* **ManualCar**

Η συνάρτηση **g** *έχει έναν απεριόριστο αριθμό τύπων—ένα για κάθε κλάση που είναι μια υποκλάση της κλάσης* **Car**

Λέμε ότι αυτό είναι πολυμορφισμός υποτύπων

Ορισμοί (Ανακεφαλαίωση)

Πολυμορφισμός

- Είδαμε 4 κατηγορίες πολυμορφισμού
- Υπάρχουν και άλλες χρήσεις του πολυμορφισμού
  - Πολυμορφισμός μεταβλητών, κλάσεων, πακέτων, συναρτήσεων
  - Είναι άλλο ένα όνομα για κλήση μεθόδων κατά το χρόνο εκτέλεσης: όταν μια κλήση `x.f()` μπορεί να καλέσει διαφορετικές μεθόδους ανάλογα με την κλάση του αντικειμένου `x` κατά το χρόνο εκτέλεσης
- Ορισμός που καλύπτει όλες τις χρήσεις:

- Μια συνάρτηση (ή ένας τελεστής) είναι ***πολυμορφική*** εάν έχει τουλάχιστον δύο πιθανούς τύπους
  - Λέμε ότι έχει *περιστασιακό πολυμορφισμό (ad hoc polymorphism)* εάν έχει τουλάχιστον δύο αλλά πεπερασμένο πλήθος πιθανών τύπων
  - Λέμε ότι έχει *καθολικό πολυμορφισμό (universal polymorphism)* εάν έχει άπειρο πλήθος πιθανών τύπων

Υπερφόρτωση

- Περιστασιακός πολυμορφισμός (ad hoc polymorphism)
- Ένας ορισμός για κάθε διαφορετικό τύπο
- Αλλά οι ορισμοί αυτοί είναι πεπερασμένοι σε ένα πεπερασμένο πρόγραμμα

Αυτόματη μετατροπή τύπων παραμέτρων

- Περιστασιακός πολυμορφισμός (ad hoc polymorphism)
- Όσο υπάρχουν πεπερασμένοι διαφορετικοί τύποι, υπάρχουν πεπερασμένοι το πλήθος διαφορετικοί τρόποι που μπορεί να γίνει η αυτόματη μετατροπή τύπων των παραμέτρων

Παραμετρικός πολυμορφισμός

- ***Καθολικός*** πολυμορφισμός
- Δεν υπάρχει κάποιος περιορισμός για τα στιγμιότυπα μιας μεταβλητής τύπου
- Ο αριθμός των πιθανών τύπων που μπορούν να χρησιμοποιηθούν είναι απεριόριστος

Πολυμορφισμός υποτύπων

- ***Καθολικός*** πολυμορφισμός
- Η δυνατότητα ένα αντικείμενο μιας υποκλάσης να χρησιμοποιείται εκεί όπου αναμένεται ένα αντικείμενο της υπερκλάσης του
- Δεν υπάρχει κάποιος περιορισμός στο
- Συνηθισμένος σε αντικειμενοστραφείς γλώσσες προγραμματισμού, όπως η Java,

Συμπερασματικά

- Συμπερασμός τύπων
  - Προσπαθεί να εξάγει τον καλύτερο τύπο για κάθε έκφραση, με βάση πληροφορία για (κάποια από) τα σύμβολα της έκφρασης
- Πολυμορφισμός
  - Όταν κάποια συνάρτηση ή αλγόριθμος μπορεί να δουλέψει σε πολλούς τύπους δεδομένων
- Υπερφόρτωση (overloading)
  - Όταν σύμβολα έχουν πολλαπλές χρήσεις οι οποίες επιλύονται στο χρόνο μεταγλώττισης (compile time)