Κεφάλαιο 6:
Σημασιολογική ανάλυση

Σύνταξη και σημασιολογία

- Σύνταξη: μορφή και δομή των καλώς σχηματισμένων προγραμμάτων
- Σημασιολογία: ερμηνεία των καλώς σχηματισμένων προγραμμάτων
  - Στατική σημασιολογία: εντοπισμός σημασιολογικών σφαλμάτων κατά τη διάρκεια της μεταγλώττισης
  - Δυναμική σημασιολογία: απόδοση ερμηνείας στα προγράμματα κατά την εκτέλεσή τους

Στατική σημασιολογία (i)

- Περιβάλλοντα τύπων

$$\Gamma_1 \;=\; \{\, \mathtt{i} \mapsto \mathit{integer}, \mathtt{x} \mapsto \mathit{real} \,\}$$

- Σχέση αντιστοίχισης τύπων

$$\Gamma \vdash E : \tau$$

- Κανόνες τύπων

$$\frac{\Gamma \vdash E_1 : \mathit{integer} \qquad \Gamma \vdash E_2 : \mathit{integer}}{\Gamma \vdash E_1 \texttt{+} E_2 : \mathit{integer}}$$

Στατική σημασιολογία (ii)

- Παραγωγές τύπων

$$\dfrac{\dfrac{\Gamma_1 \vdash \texttt{i} : integer \quad \Gamma_1 \vdash \texttt{1} : integer}{\Gamma_1 \vdash \texttt{i+1} : integer} \qquad \Gamma_1 \vdash \texttt{x} : real}{\Gamma_1 \vdash \texttt{(i+1)*x} : real}$$

- Η αντιστοίχιση τύπων με κανόνες τύπων επεκτείνεται σε όλα τα τμήματα προγράμματος

$$\dfrac{\Gamma \vdash E : boolean \quad \Gamma \vdash S : stmt}{\Gamma \vdash \texttt{while}\ E\ \texttt{do}\ S : stmt}$$

Δυναμική σημασιολογία (i)

- Λειτουργική σημασιολογία (operational semantics)
  - $\Rightarrow$ ακολουθία υπολογιστικών βημάτων
- Δηλωτική σημασιολογία (denotational semantics)
  - $\Rightarrow$ μαθηματική συνάρτηση από το πεδίο των δεδομένων εισόδου στο πεδίο των αποτελεσμάτων
- Αξιωματική σημασιολογία (axiomatic semantics)
  - $\Rightarrow$ η ερμηνεία καθορίζεται έμμεσα μέσω λογικών προτάσεων που περιγράφουν ιδιότητες του προγράμματος

Δυναμική σημασιολογία (ii)

- Η εντολή ανάθεσης $I{=}E$
  - Λειτουργική σημασιολογία

$$\frac{\langle\, E, \sigma \,\rangle \;\rightarrow\; v}{\langle\, I{=}E, \sigma \,\rangle \;\rightarrow\; \sigma[I \mapsto v]}$$

  - Δηλωτική σημασιολογία

$$\mathcal{C}[\![\, I{=}E \,]\!](s) \quad = \quad s[I \mapsto \mathcal{E}[\![\, E \,]\!](s)]$$

  - Αξιωματική σημασιολογία

$$\{\, P[I \mapsto E] \,\}\;\; I{=}E \;\; \{\, P \,\}$$

Σημασιολογικός έλεγχος (i)

- Έλεγχος τύπων
  - π.χ. οι τελεστές εφαρμόζονται σε κατάλληλα τελούμενα
- Έλεγχος ροής
  - π.χ. όχι `continue` έξω από βρόχο (C)
  - π.χ. οι μεταβλητές αρχικοποιούνται πριν τη χρήση (Java)
- Έλεγχος ύπαρξης ονομάτων
  - π.χ. οι μεταβλητές ορίζονται πριν τη χρήση τους
- Έλεγχος μοναδικότητας
  - π.χ. οι σταθερές σε μία δομή case είναι μοναδικές (Pascal)
- Έλεγχος συνέπειας
  - π.χ. σωστό όνομα υποπρογράμματος στο `end` (Ada)

Σημασιολογικός έλεγχος (ii)

- Απουσία σημασιολογικών σφαλμάτων $6\Rightarrow$ απουσία σφαλμάτων εκτέλεσης
- Μερικές φορές όμως είναι δυνατό να προβλεφθούν σφάλματα εκτέλεσης

```
program p;
var x, y : integer;
begin
   read(x);
   y := 5/(x-x)
end.
```

Σύστημα τύπων

- Βασικοί τύποι (integer, boolean, real, char, ...)
- Σύνθετοι τύποι
  - Πίνακες (arrays)
  - Ζεύγη (products) και πλειάδες (tuples)
  - Εγγραφές (records)
  - Δείκτες (pointers)
  - Συναρτήσεις (functions)
- Τύποι και τιμές πρώτης τάξης (first class)

Απλές εκφράσεις, βασικοί τύποι (i)

- Σημασιολογική ανάλυση στο bison
- Σκοπός: υπολογισμός του πεδίου `type`

```
%{
typedef enum { TY_int, TY_real, TY_bool } Type;
%}

%union{
  char * n;
  Type t;
  struct { Type type; /* other fields */ } v;
  ...
}

%type<n> T_id
%type<t> typename
%type<v> expression
```

Απλές εκφράσεις, βασικοί τύποι (ii)

```
expression :
    T_intconst            { $$.type = TY_int;  }
  | T_realconst           { $$.type = TY_real; }
  | '(' expression ')'    { $$.type = $2.type; } ;


expression : T_id
  {
    Entry * id = lookup($1);

    if (id != NULL && id->kind == K_variable)
      $$.type = id->type;
    else
      yyerror("identifier not found");
  } ;
```

Απλές εκφράσεις, βασικοί τύποι (iii)

```
expression : expression ”mod” expression
  {
    if ($1.type == TY_int && $3.type == TY_int)
      $$.type = TY_int;
    else
      yyerror(”type mismatch”);
  } ;


statement : ”while” expression ”do” statement
  {
    if ($2.type != TY_bool)
      yyerror(”condition type mismatch”)
  } ;
```

Μετατροπές τύπων

- Άμεση μετατροπή (type casting)

```
expression : '(' typename ')' expression
  {
    if (isCastAllowed($2, $4.type))
      $$.type = $2;
    else
      yyerror("illegal type cast");
  } ;
```

- Έμμεση μετατροπή – εξαναγκασμός (coercion)

Υπερφόρτωση τελεστών

```
expression : expression '+' expression
  {
    if ($1.type == TY_int)
      if ($3.type == TY_int)       $$.type = TY_int;
      else if ($3.type == TY_real) $$.type = TY_real;
      else yyerror("type mismatch");
    else if ($1.type == TY_real)
      if ($3.type == TY_int)       $$.type = TY_real;
      else if ($3.type == TY_real) $$.type = TY_real;
      else yyerror("type mismatch");
    else
      yyerror("type mismatch");
  } ;
```

Πολυμορφικοί τελεστές

```
typedef struct type_str {
   enum { TY_integer, TY_real, TY_boolean, TY_ptr } code;
   struct type_str * ptr_type;
} Type;

expression : expression '^'
  {
    if ($1.type.code == TY_ptr)
      $$.type = *($$.type.ptr_type);
    else
      yyerror("type mismatch");
  } ;
```

Συνώνυμα και ισοδυναμία τύπων

- Συνώνυμα τύπων (type aliases)

```
type complex = record
                   re, im : real
               end;
```

- Ισοδυναμία τύπων (type equivalence)
  - Δομική ισοδυναμία
  - Ονομαστική ισοδυναμία
  - (Δηλωτική ισοδυναμία)

Υποσύνολα τύπων και υποτύποι

$$\frac{\Gamma \vdash E : \tau \qquad \tau <: \tau'}{\Gamma \vdash E : \tau'}$$

```
var indexSmall   : 1..10;
    indexLarge   : 1..1000;
    indexGeneral : integer;

indexGeneral := indexSmall;  (* OK: 1..10 <: integer *)

indexLarge := indexGeneral;  (* δυναμικός έλεγχος! *)
```

- Υποτύποι
  - $\Rightarrow$ κυρίως σε αντικειμενοστρεφείς γλώσσες

Πολυμορφικοί τύποι

- Παραμετρικός πολυμορφισμός (templates, generics)

```
template <class T>
bool exists (int length, T array [], T element)
{
   for (int i=0; i < length; i++)
      if (array[i] == element)
         return true;
   return false;
}
```

- Συναρτησιακές γλώσσες

Αντιστοίχιση τύπων

- Στατική
  - ρητή
  - έμμεση
- Δυναμική

```
x := 5;          (*              x : integer *)
x := ”hello”;    (* τώρα όμως    x : string *)
```

Εξαγωγή τύπων

```
val pi      = 3.14159;   (* pi  : real                  *)
fun inc x   = x + 1;     (* inc : int -> int            *)
fun add x y = x + y;     (* add : int -> int -> int *)


fun add1 (x : real) y = x + y;
fun add2 x y : real = x + y;
                (* add1, add2 : real -> real -> real *)


fun id x = x;                          (* id : 'a -> 'a *)
```

Δυναμικός έλεγχος τύπων

- Επιβάλλεται όταν υπάρχει δυναμική αντιστοίχιση τύπων
- Πολλές φορές όμως απαιτείται και σε στατική αντιστοίχιση τύπων, π.χ. έλεγχος ορίων σε arrays:

```
var a : array [0..100] of integer;
...
a[i] := 42
```

if `i` ≥ 0 and `i` ≤ 100 **then**
  κώδικας για την ανάθεση του `42` στο `a[i]`
else
  σφάλμα εκτέλεσης
end if

Κεφάλαιο 7:
Ενδιάμεσος κώδικας

Ενδιάμεσος κώδικας (i)

- Λόγοι ύπαρξης
  - Διευκολύνει το έργο της μετάφρασης
  - Διευκολύνει τη βελτιστοποίηση
  - Διευκολύνει την κατάτμηση σε εμπρόσθιο και οπίσθιο τμήμα