Μεταγλωττιστές

Νίκος Παπασπύρου
nickie@softlab.ntua.gr

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

Εισαγωγή (i)

- Υλοποίηση γλωσσών προγραμματισμού
- Είδη μεταγλωττιστών:
  - Απλοί
  - Αντίστροφοι (decompilers)
  - Μετα-μεταγλωττιστές (meta-compilers)
- Ειδικές περιπτώσεις μεταγλωττιστών:
  - Προεπεξεργαστές (preprocessors)
  - Συμβολομεταφραστές (assemblers)
  - Γεννήτορες προγραμμάτων (program generators)

Εισαγωγή (ii)

- Συναφή εργαλεία
  - Διερμηνείς (interpreters)
  - Διαχειριστές βιβλιοθηκών (library managers)
  - Συνδέτες (linkers)
  - Φορτωτές (loaders)
  - Εκδότες προγραμμάτων (program editors)
  - Εντοπιστές σφαλμάτων (debuggers)
  - Στατιστικοί αναλυτές (profilers)

Κατασκευή μεταγλωττιστή (i)

- Βασικές απαιτήσεις:
  - Να λειτουργεί σωστά
  - Να συμμορφώνεται με τις προδιαγραφές της αρχικής και της τελικής γλώσσας
  - Να μεταγλωττίζει προγράμματα κάθε μεγέθους

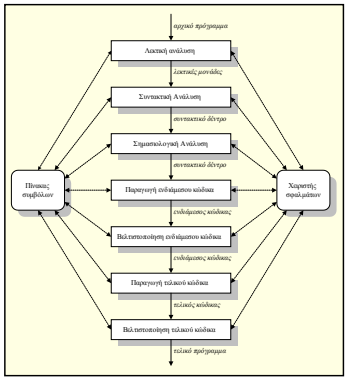
Κατασκευή μεταγλωττιστή (ii)

- Επιπρόσθετες απαιτήσεις:
  - Να παράγει αποδοτικό κώδικα
  - Να έχει μικρό χρόνο μεταγλώττισης
  - Να έχει μικρές απαιτήσεις μνήμης κατά τη μεταγλώττιση
  - Να δίνει καλά διαγνωστικά μηνύματα
  - Να συνεχίζει ύστερα από λάθη
  - Να είναι μεταφέρσιμος

Φάσεις της μεταγλώττισης

- Λεκτική ανάλυση
- Συντακτική ανάλυση
- Σημασιολογική ανάλυση
- Παραγωγή ενδιάμεσου κώδικα
- Βελτιστοποίηση
- Παραγωγή τελικού κώδικα

Φάσεις και προϊόντα

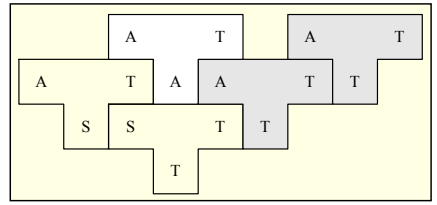


Θέματα υλοποίησης

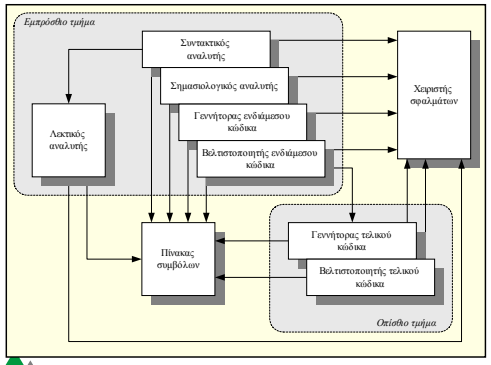
- Οργάνωση σε περάσματα
- Οργάνωση σε εμπρόσθιο και οπίσθιο τμήμα (front-end / back-end)
- Έλεγχος ορθότητας
- Είδη διαγνωστικών μηνυμάτων και ανάνηψη
  - Εσωτερικά (internal)
  - Σφάλματα (errors)
  - Προειδοποιητικά μηνύματα (warnings)
  - Απλά μηνύματα (messages)

Εκκίνηση – bootstrapping

- Βήμα 1: Μεταγλωττιστής για $S \subset A$ στην T.
- Βήμα 2: Μεταγλωττιστής για την A στην S.
- Βήμα 3: Μεταγλωττιστής για την A στην A.



Οργάνωση σε ένα πέρασμα



Τυπικές γλώσσες

- Γραμματική $G = (T, N, P, S)$
  - T: τερματικά σύμβολα a
  - N: μη τερματικά σύμβολα A
  - P: κανόνες παραγωγής $\alpha \to \beta$
  - S: αρχικό σύμβολο
- Παραγωγές: αν $\alpha, \beta, \gamma, \delta \in (T \cup N)^*$
  και $(\alpha \to \beta) \in P$
  τότε $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$
- Γλώσσα: $L(G) = \{\, \alpha \in T^* \mid S \Rightarrow^+ \alpha \,\}$

Ιεραρχία Chomsky

- Τύπου 0: όλες οι γραμματικές, $\alpha \to \beta$
- Τύπου 1: γραμματικές με συμφραζόμενα (context-sensitive), $\alpha \to \beta$ με $|\alpha| \le |\beta|$
- Τύπου 2: γραμματικές χωρίς συμφραζόμενα (context-free) $A \to \beta$
- Τύπου 3: κανονικές γραμματικές (regular) $A \to \mathsf{a}B$ ή $A \to \mathsf{a}$
- Ειδική περίπτωση: γλώσσες που παράγουν την κενή συμβολοσειρά

Αναγνωριστές

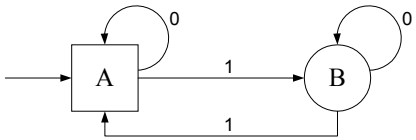
- Τύπου 0: μηχανή Turing
- Τύπου 1: γραμμικά περιορισμένη μηχανή Turing
- Τύπου 2: αυτόματα στοίβας (push-down automata)
  - Χρήσιμα στη συντακτική ανάλυση
- Τύπου 3: πεπερασμένα αυτόματα (finite automata)
  - Χρήσιμα στη λεκτική ανάλυση

Κανονικές γλώσσες

- Κανονικές εκφράσεις (regular expressions)
  - Κενή συμβολοσειρά: ϵ
  - Κάθε σύμβολο: $a \in \Sigma$
  - Παράθεση δύο κανονικών εκφράσεων: (rs)
  - Διάζευξη δύο κανονικών εκφράσεων: $(r|s)$
  - Κλείσιμο (ή άστρο) Kleene: (r^*)
- Συντομογραφίες:
  - απαλοιφή περιττών παρενθέσεων
  - r^+ $[a_1, a_2, \ldots, a_n]$ $r?$.

Πεπερασμένα αυτόματα

- Καταστάσεις και μεταβάσεις
- Ντετερμινιστικά (ΝΠΑ), μη ντετερμινιστικά (ΜΠΑ) και ΜΠΑ με κενές μεταβάσεις (ΜΠΑ-ϵ)



- Η γλώσσα των συμβολοσειρών που αποτελούνται από 0 και 1 και περιέχουν άρτιο αριθμό 1

Αναγωγές και ισοδυναμίες

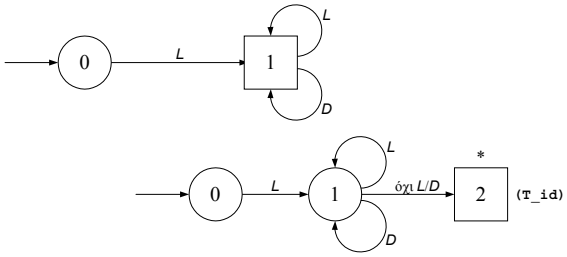
- κανονική γραμματική $\Rightarrow$ ΜΠΑ-ϵ
- ΜΠΑ-ϵ $\Rightarrow$ κανονική γραμματική
- κανονική έκφραση $\Rightarrow$ ΜΠΑ-ϵ
- ΜΠΑ-ϵ $\Rightarrow$ κανονική έκφραση
- ΜΠΑ-ϵ $\Rightarrow$ ΝΠΑ
- Ελαχιστοποίηση ΝΠΑ

Λεκτική ανάλυση

- Λεκτικές μονάδες (tokens)
- Αναγνωρίζονται με πεπερασμένα αυτόματα που:
  - διαβάζουν ενδεχομένως περισσότερους χαρακτήρες
  - οπισθοδρομούν αν χρειαστεί
  - διαθέτουν έξοδο που χρησιμοποιείται στη συντακτική ανάλυση
- Ειδικός συμβολισμός: διαγράμματα μετάβασης

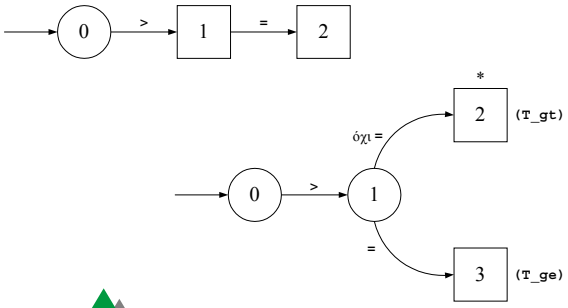
Διαγράμματα μετάβασης (i)

- Αναγνωριστικά της Pascal



Διαγράμματα μετάβασης (ii)

- Τελεστές $>$ και $>=$



Κατασκευή του ΛΑ (i)

- Καταγραφή και ταξινόμηση χαρακτήρων

$$mapping \ : \ (\text{ASCII} \cup \{\,\texttt{EOF}\,\}) \rightarrow \Sigma$$

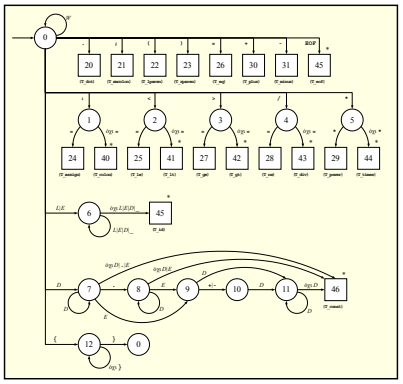
- Καταγραφή και ταξινόμηση λεκτικών μονάδων
  - Κωδικοποίηση λεκτικών μονάδων
  - Ακολουθία χαρακτήρων (lexeme)
- Σχεδίαση του διαγράμματος μετάβασης
- Υλοποίηση του λεκτικού αναλυτή

Κατασκευή του ΛΑ (ii)

- Επιμέρους θέματα
  - Τρόπος διαχωρισμού λεκτικών μονάδων
  - Σχόλια
  - Διάκριση πεζών / κεφαλαίων γραμμάτων
  - Ενδιάμεση μνήμη (buffer)
  - Ανάνηψη από σφάλματα

Κατασκευή του ΛΑ (iii)

Σχεδίαση συνολικού διαγράμματος μετάβασης



Κατασκευή του ΛΑ (iv)

- Εναλλακτικοί τρόποι υλοποίησης:
  - Χειρωνακτικά
  - Με πίνακα μεταβάσεων
  - Με το μεταεργαλείο flex

Υλοποίηση ΛΑ με το flex (i)

- Μεταεργαλείο flex: γεννήτορας ΛΑ
- Είσοδος: μεταπρόγραμμα που περιγράφει τις λεκτικές μονάδες
- Έξοδος: πρόγραμμα σε C
  - Η συνάρτηση yylex υλοποιεί το ΛΑ
  - Επιστρέφει τον κωδικό της λεκτικής μονάδας που αναγνωρίστηκε, ή 0 στο τέλος της συμβολοσειράς εισόδου
  - Τοποθετεί στη μεταβλητή yytext την αντίστοιχη ακολουθία χαρακτήρων (lexeme)

Υλοποίηση ΛΑ με το flex (ii)

- Δομή του μεταπρογράμματος

Μέρος Α
%%
Μέρος Β
%%
Μέρος Γ

Και τα τρία μέρη μπορούν να είναι κενά

Υλοποίηση ΛΑ με το flex (iii)

- Μέρος Α, περιέχει
  - Σχόλια, όπως στη C
  - Κώδικα C, μέσα σε %{ και %}
  - Μνημονικά ονόματα ως συντομογραφίες κανονικών εκφράσεων
  - Δηλώσεις αρχικών καταστάσεων

Υλοποίηση ΛΑ με το flex (iv)

- Μέρος Α, παράδειγμα

```
%{
#define T_eof             0
#define T_id              1
   ...
#define T_while          52

void ERROR (const char msg []);
%}

L [A-Za-z]          /* letters     */
D [0-9]             /* digits      */
W [ \t\n]           /* white space */
```


Υλοποίηση ΛΑ με το flex (v)

- Μέρος Β, περιέχει κανόνες της μορφής

κανονική έκφραση ενέργεια

- Κάθε ενέργεια είναι μια εντολή της C
- Λειτουργία:
  - Διαβάζεται το μακρύτερο πρόθεμα της συμβολοσειράς εισόδου που μπορεί να αναγνωριστεί από κάποια κανονική έκφραση
  - Εκτελείται η αντίστοιχη ενέργεια

Υλοποίηση ΛΑ με το flex (vi)

- Κανονικές εκφράσεις

`a`	Ο χαρακτήρας a.
`.`	Οποιοσδήποτε χαρακτήρας εκτός της αλλαγής γραμμής.
$\backslash x$	Αν x ένα από τα a, b, f, n, r, t, v ή 0, τότε όπως στη C, αλλιώς ο ίδιος ο χαρακτήρας x.
`\123`	Ο χαρακτήρας ASCII με οκταδική τιμή 123.
`\x3f`	Ο χαρακτήρας ASCII με δεκαεξαδική τιμή 3F.
`"abc"`	Η συμβολοσειρά abc.
`[abc]`	Ένας από τους χαρακτήρες a, b ή c.
`[a-z]`	Ένας από τους χαρακτήρες a έως z.
`[ac-fs]`	Ένας από τους χαρακτήρες a, c έως f, ή s.

Υλοποίηση ΛΑ με το flex (vii)

- Κανονικές εκφράσεις (συνέχεια)

`[^a-z]`	Ένας από τους χαρακτήρες εκτός όσων ανήκουν στην περιοχή a έως z.
`{name}`	Η κανονική έκφραση με μνημονικό όνομα name.
rs	Η παράθεση των r και s.
$r\|s$	Η διάζευξη των r και s.
r/s	Η κανονική έκφραση r αλλά μόνο αν ακολουθεί η κανονική έκφραση s.
(r)	Η κανονική έκφραση r. Οι παρενθέσεις χρησιμοποιούνται για ομαδοποίηση.

Υλοποίηση ΛΑ με το flex (viii)

- Κανονικές εκφράσεις (συνέχεια)

r*	Η r επαναλαμβάνεται μηδέν ή περισσότερες φορές.
r+	Η r επαναλαμβάνεται μια ή περισσότερες φορές.
r?	Η r είναι προαιρετική (επαναλαμβάνεται μηδέν ή μια φορά).
r{7}	Η r επαναλαμβάνεται ακριβώς 7 φορές.
r{3,5}	Η r επαναλαμβάνεται από 3 έως 5 φορές.
r{4,}	Η r επαναλαμβάνεται 4 ή περισσότερες φορές.
^r	Η r αλλά μόνο στην αρχή μιας γραμμής.
$r$$	Η r αλλά μόνο στο τέλος μιας γραμμής.

Υλοποίηση ΛΑ με το flex (ix)

- Κανονικές εκφράσεις (συνέχεια)

<S>r	Η r αλλά μόνο όταν η τρέχουσα αρχική κατάσταση είναι η S.
<S_1,S_2,S_3>r	Η r, αλλά μόνο όταν η τρέχουσα αρχική κατάσταση είναι μια από τις S_1, S_2 ή S_3.
<*>r	Η r σε οποιαδήποτε αρχική κατάσταση.
<<EOF>>	Το τέλος του αρχείου εισόδου.

Υλοποίηση ΛΑ με το flex (x)

- Μέρος Β, παράδειγμα

```
"and"                          { return T_and;    }
   ...
"while"                        { return T_while;  }

":="                           { return T_assign; }
":"                            { return T_colon;  }

{L}({L}|{D}|_)*                 { return T_id;     }
{D}+(\.{D}*(e\-?{D}+)?)?       { return T_const;  }
{W}+                           { /* nothing */   }
"(*"([^*]+|\*+[^*)])*\*+")"    { /* nothing */   }

.                          { ERROR("illegal token"); }
```


Υλοποίηση ΛΑ με το flex (xi)

- Μέρος Γ, περιέχει κώδικα C
- Παράδειγμα

```
void ERROR (const char msg [])
{
   fprintf(stderr, "ERROR: %s\n", msg);
   exit(1);
}
```


Υλοποίηση ΛΑ με το flex (xii)

- Παράδειγμα (συνέχεια)

```
int main ()
{
   int token;

   do {
      token = yylex();
      printf("token=%d, lexeme=\"%s\"\n",
             token, yytext);
   } while (token != T_eof);

   return 0;
}
```


Υλοποίηση ΛΑ με το flex (xiii)

- Παράδειγμα: Αρίθμηση γραμμών

```
int lineno = 1;

[ \t]+     { /* nothing */ }
\n         { lineno++;     }

void ERROR (const char msg [])
{
    fprintf(stderr, "ERROR, line %d: %s\n",
            lineno, msg);
    exit(1);
}
```

- Πρόβλημα: Λάθος αρίθμηση σε σχόλια

Υλοποίηση ΛΑ με το flex (xiv)

- Αρχικές καταστάσεις
  - Κοινές: %s
  - Αποκλειστικές: %x
- Ενεργοί κανόνες σε κάποια κατάσταση
- Μετάβαση μεταξύ καταστάσεων: BEGIN(s)
- Αρχική κατάσταση κατά την έναρξη λειτουργίας του ΛΑ: INITIAL

Υλοποίηση ΛΑ με το flex (xv)

- Παράδειγμα: Αρίθμηση γραμμών (διόρθωση)

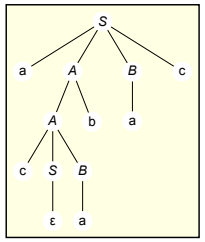
```
%x COMMENT

"(*"                { BEGIN(COMMENT); }
<COMMENT>"*)"       { BEGIN(INITIAL); }
<COMMENT>\n         { lineno++;       }
<COMMENT>"*"        { /* nothing */   }
<COMMENT>[^*\n]+    { /* nothing */   }
```


Γλώσσες χωρίς συμφραζόμενα

- Γραμματικές χωρίς συμφραζόμενα: $A \to \alpha$
- Αριστερότερη / δεξιότερη παραγωγή (leftmost / rightmost derivation)
- Συντακτικά δέντρα (parse trees)

$S \Rightarrow_L aABC \Rightarrow_L aAbBc$
$\Rightarrow_L acSBbBc \Rightarrow_L acBbBc$
$\Rightarrow_L acabBc \Rightarrow_L acabac$

$S \Rightarrow_R aABc \Rightarrow_R aAac$
$\Rightarrow_R aAbac \Rightarrow_R acSBbac$
$\Rightarrow_R acSabac \Rightarrow_R acabac$



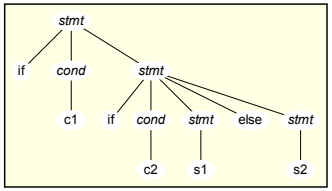
Διφορούμενες γραμματικές (i)

- Μια γραμματική είναι διφορούμενη (ambiguous) όταν υπάρχουν δύο ή περισσότερα συντακτικά δέντρα για την ίδια παραγόμενη συμβολοσειρά
- Γραμματικές και γλώσσες εγγενώς διφορούμενες (inherently ambiguous)
- Παράδειγμα διφορούμενης γραμματικής: ξεκρέμαστο if (dangling if)

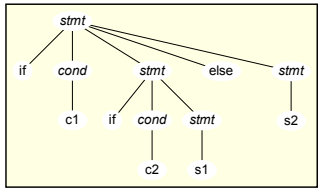
stmt → if *cond stmt* else *stmt* | if *cond stmt* | s1 | s2
cond → c1 | c2

και η συμβολοσειρά: if c1 if c2 s1 else s2

Διφορούμενες γραμματικές (ii)



if c1 (if c2 s1 else s2)



if c1 (if c2 s1) else s2

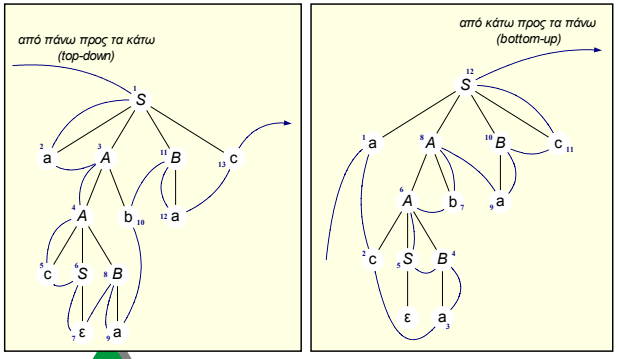
Συμβολισμός BNF/EBNF

- Backus-Naur Form
  - Σύμβολο ::= στους κανόνες
  - Μη τερματικά σύμβολα σε γωνιακές παρενθέσεις, π.χ. ⟨expr⟩
  - Σύμβολο | για διάζευξη
- Extended Backus-Naur Form
  - Τερματικά σύμβολα σε εισαγωγικά
  - Παρενθέσεις για ομαδοποίηση
  - Αγκύλες για προαιρετικά τμήματα
  - Σύμβολα * και + για επανάληψη

Συντακτική ανάλυση

- Συντακτικό δέντρο (parse tree)
- Κατασκευάζεται με δύο τρόπους:
  - Από πάνω προς τα κάτω (top-down) δηλαδή ξεκινώντας από τη ρίζα και προχωρώντας προς τα φύλλα
  - Από κάτω προς τα πάνω (bottom-up) δηλαδή ξεκινώντας από τα φύλλα και προχωρώντας προς τη ρίζα

Top-down και bottom-up



Βοηθητικές έννοιες (i)

- Σύνολα FIRST
  - Έστω συμβολοσειρά $\alpha \in (T \cup N)^*$
  - Το σύνολο $\text{FIRST}(\alpha) \subseteq (T \cup \{\epsilon\})$ περιέχει τα τερματικά σύμβολα από τα οποία αρχίζουν οι συμβολοσειρές που παράγονται από την α
  - Αν $\alpha \Rightarrow \epsilon$, τότε $\epsilon \in \text{FIRST}(\alpha)$

Βοηθητικές έννοιες (ii)

- Σύνολα FOLLOW
  - Έστω μη τερματικό σύμβολο A
  - Το σύνολο $\text{FOLLOW}(A) \subseteq (T \cup \{\texttt{EOF}\})$ περιέχει τα τερματικά σύμβολα που μπορούν να ακολουθούν το A στη διάρκεια μιας παραγωγής
  - Αν το A μπορεί να είναι το τελευταίο σύμβολο σε μια παραγωγή, τότε $\texttt{EOF} \in \text{FOLLOW}(A)$

Υπολογισμός FIRST (i)

- $\text{FIRST}(\epsilon) = \{\epsilon\}$
- $\text{FIRST}(\mathsf{a}\beta) = \{\mathsf{a}\}$
- αν $\epsilon \notin \text{FIRST}(A)$
  τότε $\text{FIRST}(A\beta) = \text{FIRST}(A)$
- αν $\epsilon \in \text{FIRST}(A)$
  τότε $\text{FIRST}(A\beta) = (\text{FIRST}(A) - \{\epsilon\}) \cup \text{FIRST}(\beta)$
- για κάθε κανόνα $A \to \alpha$, πρέπει $\text{FIRST}(\alpha) \subseteq \text{FIRST}(A)$

Υπολογισμός FIRST (ii)

- Παράδειγμα

$$
\begin{aligned}
\text{FIRST}(E) &= \{\mathsf{id}, (\,\} \\
\text{FIRST}(T) &= \{\mathsf{id}, (\,\} \\
\text{FIRST}(F) &= \{\mathsf{id}, (\,\} \\
\text{FIRST}(E') &= \{+, \epsilon\} \\
\text{FIRST}(T') &= \{*, \epsilon\}
\end{aligned}
$$

$$
\begin{aligned}
E &\to T\ E' \\
E' &\to \epsilon \\
E' &\to +\ T\ E' \\
T &\to F\ T' \\
T' &\to \epsilon \\
T' &\to *\ F\ T' \\
F &\to (\ E\ ) \\
F &\to \mathsf{id}
\end{aligned}
$$

Υπολογισμός FOLLOW (i)

- EOF $\in$ FOLLOW(S)
- για κάθε κανόνα $A \to \alpha B \beta$
  - (FIRST(β) $- \{\epsilon\}$) $\subseteq$ FOLLOW(B)
  - αν $\epsilon \in$ FIRST(β)
    τότε FOLLOW(A) $\subseteq$ FOLLOW(B)

Υπολογισμός FOLLOW (ii)

- Παράδειγμα

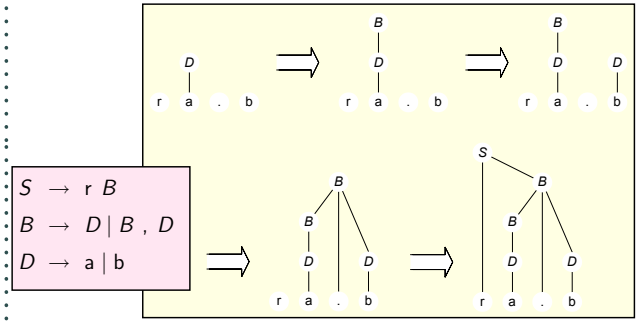
$$
\begin{aligned}
\text{FOLLOW}(E) &= \{), \texttt{EOF} \} \\
\text{FOLLOW}(T) &= \{ +,), \texttt{EOF} \} \\
\text{FOLLOW}(F) &= \{ *, +,), \texttt{EOF} \} \\
\text{FOLLOW}(E') &= \{), \texttt{EOF} \} \\
\text{FOLLOW}(T') &= \{ +,), \texttt{EOF} \} \\
\\
\text{FIRST}(E') &= \{ +, \epsilon \} \\
\text{FIRST}(T') &= \{ *, \epsilon \}
\end{aligned}
$$

$$
\begin{aligned}
E &\to T\ E' \\
E' &\to \epsilon \\
E' &\to +\ T\ E' \\
T &\to F\ T' \\
T' &\to \epsilon \\
T' &\to *\ F\ T' \\
F &\to (\ E\ ) \\
F &\to \text{id}
\end{aligned}
$$

ΣΑ bottom-up (i)

- Η συντακτική ανάλυση ξεκινά από τα φύλλα
- Κάθε φορά, αναζητά:
  - τον αριστερότερο κόμβο του δέντρου
  - που δεν έχει ακόμα κατασκευαστεί
  - ενώ όλα τα παιδιά του έχουν κατασκευαστεί
- Επαναλαμβάνει μέχρι να κατασκευαστεί η ρίζα
- Ελάττωση (reducing): η επιλογή των κόμβων που θα αποτελέσουν τα παιδιά ενός νέου κόμβου

ΣΑ bottom-up (ii)



ΣΑ bottom-up (iii)

- ΣΑ ολίσθησης-ελάττωσης (shift-reduce)
  - Χρησιμοποιούν μια (αρχικά κενή) στοίβα όπου τοποθετούν σύμβολα της γραμματικής
  - Ολίσθηση (shift): μεταφορά ενός συμβόλου από την είσοδο στην κορυφή της στοίβας
  - Ελάττωση (reduce): αφαίρεση από την κορυφή της στοίβας του δεξιού μέλους ενός κανόνα και πρόσθεση του αριστερού μέλους
  - Επιτυχία: η στοίβα περιέχει μόνο το S και τα σύμβολα της εισόδου έχουν εξαντληθεί

ΣΑ bottom-up (iv)

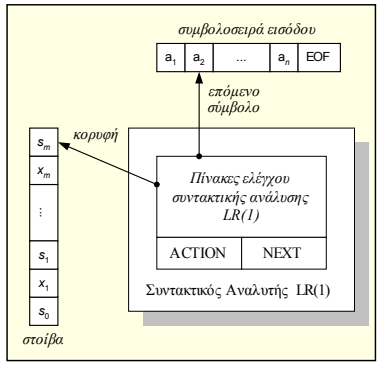
βήμα	στοίβα	είσοδος	περιγραφή κίνησης
0	ϵ	r a , b	ολίσθηση
1	r	a , b	ολίσθηση
2	r a	, b	ελάττωση με $D \to$ a
3	r D	, b	ελάττωση με $B \to D$
4	r B	, b	ολίσθηση
			(όχι ελάττωση με $S \to$ r B)
5	r B ,	b	ολίσθηση
6	r B , b	ϵ	ελάττωση με $D \to$ b
7	r B , D	ϵ	ελάττωση με $B \to B$, D
			(όχι ελάττωση με $B \to D$)
8	r B	ϵ	ελάττωση με $S \to$ r B
9	S	ϵ	αναγνώριση

shift/reduce conflict

reduce/reduce conflict

ΣΑ bottom-up (v)

- LR(k)
- LR(0)
- SLR(1)
- LALR(1)
- LR(1)



Υλοποίηση ΣΑ με το bison (i)

- Μεταεργαλείο bison: γεννήτορας ΣΑ LALR(1)
- Είσοδος: μεταπρόγραμμα που περιγράφει τη σύνταξη και τις σημασιολογικές ρουτίνες
- Έξοδος: πρόγραμμα σε C
  - Η συνάρτηση yyparse υλοποιεί το ΣΑ
  - Επιστρέφει 0 αν αναγνωριστεί η συμβολοσειρά εισόδου ή 1 σε περίπτωση συντακτικού σφάλματος
  - Συνεργάζεται με το λεκτικό αναλυτή (συνάρτηση yylex)

Υλοποίηση ΣΑ με το bison (ii)

- Δομή του μεταπρογράμματος

Μέρος Α
%%
Μέρος Β
%%
Μέρος Γ

Και τα τρία μέρη μπορούν να είναι κενά

Υλοποίηση ΣΑ με το bison (iii)

- Μέρος Α, περιέχει
  - Σχόλια, όπως στη C
  - Κώδικα C, μέσα σε %{ και %}
  - Δηλώσεις λεκτικών μονάδων
  - Δηλώσεις τελεστών της αρχικής γλώσσας (προτεραιότητα, προσεταιριστικότητα)
  - Δήλωση του συνόλου σημασιολογικών τιμών (τύπος YYSTYPE ή με χρήση του %union)
  - Δήλωση του τύπου της σημασιολογικής τιμής κάθε συμβόλου

Υλοποίηση ΣΑ με το bison (iv)

- Μέρος Α, παράδειγμα

```
%{
void ERROR (const char msg []);
%}

%token T_program "program"
%token T_div T_mod
%token T_if T_then T_else

%nonassoc '=' '<' '>'
%left '+' '-'
%left '*' '/' T_div T_mod
```


Υλοποίηση ΣΑ με το bison (v)

- Μέρος Α, παράδειγμα (συνέχεια)

```
%union{
  int i;
  double f;
  char str[80];
}

%token<str> T_id
%token<i>   T_int_const
%token<f>   T_float_const

%type<f>    expression
```

Υλοποίηση ΣΑ με το bison (vi)

- Μέρος Β, περιέχει:
  - τους κανόνες παραγωγής σε μορφή BNF
  - σημασιολογικές ρουτίνες που εκτελούνται κατά τη συντακτική ανάλυση
- Οι κανόνες έχουν τη μορφή:

$$\begin{array}{rcl} A & : & x_1^1\ x_2^1\ \ldots\ x_{m_1}^1 \\ & | & x_1^2\ x_2^2\ \ldots\ x_{m_2}^2 \\ & & \ldots \\ & | & x_1^n\ x_2^n\ \ldots\ x_{m_n}^n \\ & ; & \end{array}$$

Υλοποίηση ΣΑ με το bison (vii)

- Μέρος Γ, περιέχει κώδικα C
- Το μεταπρόγραμμα του bison αναλαμβάνει τον κεντρικό έλεγχο του μεταγλωττιστή που επιτυγχάνεται με τη συνεργασία των παρακάτω:
  - του λεκτικού αναλυτή
  - του συντακτικού αναλυτή
  - του πίνακα συμβόλων
  - του σημασιολογικού αναλυτή
  - του γεννήτορα ενδιάμεσου κώδικα

Υλοποίηση ΣΑ με το bison (viii)

- Μέρος Β, παράδειγμα

```
program     : { count=0; } block_list
              { printf("Counted %d block(s)\n",
                       count); }
            ;

block_list  : /* nothing */
            | block_list block { count++; }
            ;

block       : "begin" block_list "end"
            ;
```


Υλοποίηση ΣΑ με το bison (ix)

- Μέρος Γ, παράδειγμα

```
void yyerror (const char * msg)
{
   fprintf(stderr,
           "syntax error in line %d: %s\n",
           linecount, msg);
   exit(1);
}

int main ()
{
   return yyparse();
}
```


Υλοποίηση ΣΑ με το bison (x)

- Παράδειγμα με σημασιολογικές τιμές

$$\begin{array}{rcl} E & \rightarrow & T \\ E & \rightarrow & E + T \\ T & \rightarrow & F \\ T & \rightarrow & T * F \\ F & \rightarrow & (E) \\ F & \rightarrow & \text{num} \end{array}$$

- Ζητούμενο: να κατασκευαστεί ΣΑ που να υπολογίζει την τιμή μιας αριθμητικής έκφρασης

Υλοποίηση ΣΑ με το bison (xi)

- Παράδειγμα (συνέχεια)

```
%{
typedef int YYSTYPE;
%}

%token T_num

%%

program :
   expression { printf("Value: %d\n", $1); }
 ;
```

Υλοποίηση ΣΑ με το bison (xii)

- Παράδειγμα (συνέχεια)

```
expression :
   term { $$ = $1; }
 | expression '+' term { $$ = $1 + $3; }
 ;

term :
   factor { $$ = $1; }
 | term '*' factor { $$ = $1 * $3; }
 ;
```


Υλοποίηση ΣΑ με το bison (xiii)

- Παράδειγμα (συνέχεια)

```
factor :
   '(' expression ')' { $$ = $2; }
 | T_num { $$ = $1; }
 ;

%%
```

- *Παραλείπονται στο Μέρος Γ:*
  - η συνάρτηση yylex (πιθανώς σε ξεχωριστό αρχείο, αν χρησιμοποιηθεί το flex)
  - οι συναρτήσεις yyerror και main

Υλοποίηση ΣΑ με το bison (xiv)

- Παράδειγμα — Υλοποίηση ΛΑ χειρωνακτικά

```
int yylex ()
{
   int c;

   while (isspace(c = fgetc(stdin)));
   if (isdigit(c)) {
      yylval = c - '0';
      while (isdigit(c = fgetc(stdin)))
         yylval = yylval * 10 + c - '0';
      ungetc(c, stdin);
      return T_num;
   }
```

Στο μέρος Γ

Υλοποίηση ΣΑ με το bison (xv)

- Παράδειγμα (συνέχεια)

```
   if (strchr("+*()", c)) return c;
   if (c != EOF)
      fprintf(stderr,
              "Illegal character: %c\n", c);
   return 0;
}
```

- Αυτοματοποίηση της μεταγλώττισης του ΣΑ

```
mytest1: mytest1.y
         bison mytest1.y
         gcc -o mytest1 mytest1.tab.c
```

Makefile

Υλοποίηση ΣΑ με το bison (xvi)

- Παράδειγμα — Υλοποίηση ΛΑ με το flex

```
%{
#include "mytest2.tab.h"
%}

%%

[0-9]+ { yylval = atoi(yytext); return T_num; }

\(|\)|\+|\* { return yytext[0]; }
[ \t\n]+    { /* nothing */ }
.           { yyerror("illegal character"); }

%%
```

mytest2.l

Υλοποίηση ΣΑ με το bison (xvii)

- Αυτοματοποίηση της μεταγλώττισης ΛΑ και ΣΑ

```
mytest2: mytest2.l mytest2.y
         bison -d mytest2.y
         flex -s mytest2.l
         gcc -o mytest2 mytest2.tab.c lex.yy.c \
                -lfl
```

Makefile

- Επίλυση συγκρούσεων στο bison
  - shift-reduce: πάντα reduce
  - reduce-reduce: ο πρώτος κανόνας

ΣΑ top-down (i)

- Η συντακτική ανάλυση ξεκινά από τη ρίζα
- Κάθε φορά, αναζητά:
  - το μη τερματικό σύμβολο που θα αντικατασταθεί
    ⇒ συνήθως επιλέγεται το αριστερότερο
  - τον κανόνα παραγωγής που θα εφαρμοστεί
    ⇒ βάσει των επόμενων k λεκτικών μονάδων στη συμβολοσειρά εισόδου: LL(k)
- Επαναλαμβάνεται μέχρι να εξαντληθούν τα μη τερματικά

Γραμματικές LL(1)

- Απαραίτητες προϋποθέσεις:
  - Απουσία αριστερής αναδρομής (άμεσης ή έμμεσης)
  - Απουσία κοινού προθέματος σε εναλλακτικούς κανόνες
- Μερικές φορές είναι δυνατός ο μετασχηματισμός μιας γραμματικής σε ισοδύναμη LL(1)
  ⇒ απαλοιφή αριστερής αναδρομής
  ⇒ αριστερή παραγοντοποίηση

Μετασχηματισμός σε LL(1)

- Αντικατάσταση

$$\begin{array}{l} A \to \alpha_1 \mid \ldots \mid \alpha_n \\ B \to \beta_1 A \beta_2 \end{array} \Rightarrow \begin{array}{l} A \to \alpha_1 \mid \ldots \mid \alpha_n \\ B \to \beta_1\alpha_1\beta_2 \mid \ldots \mid \beta_1\alpha_n\beta_2 \end{array}$$

- Αριστερή παραγοντοποίηση

$$A \to \alpha\beta_1 \mid \ldots \mid \alpha\beta_n \Rightarrow \begin{array}{l} A \to \alpha B \\ B \to \beta_1 \mid \ldots \mid \beta_n \end{array}$$

- Απαλοιφή άμεσης αριστερής αναδρομής

$$\begin{array}{l} A \to A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m \end{array} \Rightarrow \begin{array}{l} A \to \beta_1 B \mid \ldots \mid \beta_m B \\ B \to \alpha_1 B \mid \ldots \mid \alpha_n B \mid \epsilon \end{array}$$

ΣΑ αναδρομικής κατάβασης

$A \to \alpha_1 \mid \ldots \mid \alpha_n$

μετατρέπεται σε κώδικα της μορφής:

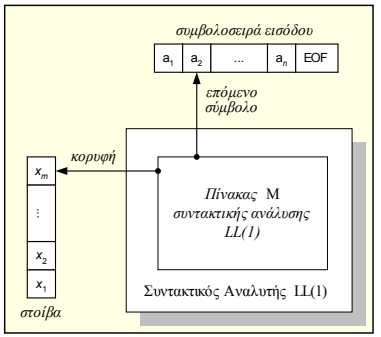
αν $token \in \text{FIRST}(\alpha_1)$ **τότε**
  κώδικας για την αναγνώριση της α_1
  ...
αλλιώς αν $token \in \text{FIRST}(\alpha_n)$ **τότε**
  κώδικας για την αναγνώριση της α_n
αλλιώς αν $\epsilon \notin \text{FIRST}(\alpha_1) \cup \ldots \cup \text{FIRST}(\alpha_n)$ **τότε**
  συντακτικό σφάλμα
αλλιώς αν $token \notin \text{FOLLOW}(A)$ **τότε**
  συντακτικό σφάλμα
τέλος αν

ΣΑ LL(1) (i)

- Χρησιμοποιούν μια στοίβα όπου τοποθετούν σύμβολα της γραμματικής — αρχικά μόνο το S
- Κάθε φορά εξετάζεται η κορυφή της στοίβας:
  - Αν είναι τερματικό σύμβολο και είναι το ίδιο με το επόμενο της συμβολοσειράς εισόδου, τότε αφαιρούνται και τα δύο
  - Αν είναι μη τερματικό σύμβολο, τότε ανάλογα με το επόμενο της συμβολοσειράς εισόδου εφαρμόζεται κάποιος κανόνας
- Επιτυχία: η στοίβα και η συμβολοσειρά εισόδου είναι άδειες

ΣΑ LL(1) (ii)

Ο αλγόριθμος κατασκευής του πίνακα M ορίζει την οικογένεια των γλωσσών LL(1)



Κατασκευή ΣΑ LL(1)

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \epsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \epsilon$
$F \rightarrow (\ E\ ) \mid \mathsf{id}$

$\mathrm{FIRST}(E) = \mathrm{FIRST}(T) = \mathrm{FIRST}(F) = \{\mathsf{id}, (\}$
$\mathrm{FIRST}(E') = \{+, \epsilon\}$
$\mathrm{FIRST}(T') = \{*, \epsilon\}$
$\mathrm{FOLLOW}(E) = \mathrm{FOLLOW}(E') = \{), \mathtt{EOF}\}$
$\mathrm{FOLLOW}(T) = \mathrm{FOLLOW}(T') = \{+,), \mathtt{EOF}\}$
$\mathrm{FOLLOW}(F) = \{*, +,), \mathtt{EOF}\}$

	id	+	*	(	)	EOF
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow$ id			$F \rightarrow (E)$		

Λειτουργία ΣΑ LL(1)

0	E	id + id * id EOF	$E \rightarrow T\ E'$
1	$E'\ T$	id + id * id EOF	$T \rightarrow F\ T'$
2	$E'\ T'\ F$	id + id * id EOF	$F \rightarrow$ id
3	$E'\ T'$ id	id + id * id EOF	
4	$E'\ T'$	+ id * id EOF	$T' \rightarrow \epsilon$
5	E'	+ id * id EOF	$E' \rightarrow +\ T\ E'$
6	$E'\ T +$	+ id * id EOF	
7	$E'\ T$	id * id EOF	$T \rightarrow F\ T'$
8	$E'\ T'\ F$	id * id EOF	$F \rightarrow$ id
9	$E'\ T'$ id	id * id EOF	
10	$E'\ T'$	* id EOF	$T' \rightarrow *\ F\ T'$
11	$E'\ T'\ F\ *$	* id EOF	
12	$E'\ T'\ F$	id EOF	$F \rightarrow$ id
13	$E'\ T'$ id	id EOF	
14	$E'\ T'$	EOF	$T' \rightarrow \epsilon$
15	E'	EOF	$E' \rightarrow \epsilon$
16	ϵ	EOF	αναγνώριση

Κατηγορικές γραμματικές (i)

- Κατηγορική γραμματική (attribute grammar): γραμματική χωρίς συμφραζόμενα όπου κάθε σύμβολο φέρει ένα σύνολο κατηγορημάτων
- Οι τιμές των κατηγορημάτων υπολογίζονται βάσει του συντακτικού δέντρου:
  - Συνθετικά κατηγορήματα: οι τιμές τους εξαρτώνται μόνο από κατηγορήματα των παιδιών κάθε κόμβου
  - Κληρονομούμενα κατηγορήματα: οι τιμές τους εξαρτώνται μόνο από κατηγορήματα του 'πατέρα' και των 'αδελφών' κάθε κόμβου

Κατηγορικές γραμματικές (ii)

$E \rightarrow E + T$	$\{ E^1.val := E^2.val + T.val \}$
$E \rightarrow T$	$\{ E.val := T.val \}$
$T \rightarrow T * F$	$\{ T^1.val := T^2.val * F.val \}$
$T \rightarrow F$	$\{ T.val := F.val \}$
$F \rightarrow (\ E\ )$	$\{ F.val := E.val \}$
$F \rightarrow$ num	$\{ F.val :=$ num$.val \}$

Σημασιολογικοί κανόνες

Πίνακας συμβόλων

- Συγκεντρώνει πληροφορίες για τα ονόματα που εμφανίζονται στο αρχικό πρόγραμμα
- Ονόματα είναι:
  - το πρόγραμμα
  - οι μεταβλητές
  - τα υποπρογράμματα (διαδικασίες, συναρτήσεις)
  - οι παράμετροι των υποπρογραμμάτων
  - οι ετικέττες εντολών
  - οι σταθερές
  - οι τύποι δεδομένων

Χαρακτηριστικά ονομάτων

- Κατηγορία αποθήκευσης (storage class)
  - Καθολικές μεταβλητές (global variables)
  - Μεταβλητές στοίβας (stack variables)
  - Στατικές μεταβλητές (static variables)
- Εμβέλεια (scope)
- Ορατότητα (visibility)
- Διάρκεια ζωής (lifetime)

Περιεχόμενα πίνακα συμβόλων

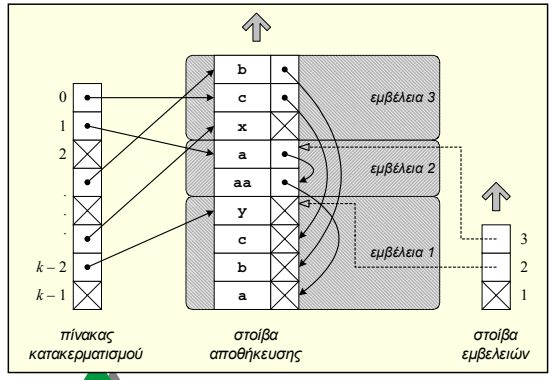
- Εμβέλεια (έμμεσα)
- Ορατότητα (έμμεσα)
- Διάρκεια ζωής
- Τύπος
- Θέση (διεύθυνση μνήμης, καταχωρητής, ...)
- Αριθμός παραμέτρων υποπρογράμματος
- Τύπος παραμέτρων υποπρογράμματος
- Τρόπος περάσματος παραμέτρων υποπρογράμματος
- Τύπος αποτελέσματος συνάρτησης

Οργάνωση πίνακα συμβόλων

- Βασικές λειτουργίες
  - Προσθήκη ονόματος
  - Αναζήτηση ονόματος
  - Διαγραφή ονόματος ή ομάδας ονομάτων
- Κόστος προσθήκης ή αναζήτησης ανάλογα με την υλοποίηση:

γραμμική λίστα	$O(n)$
δυαδικό δέντρο αναζήτησης	$O(\log n)$
πίνακας κατακερματισμού	$O(n/k)$

Υλοποίηση με ΠΚ



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

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

- 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

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

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

$$\frac{\Gamma \vdash E : \mathit{boolean} \qquad \Gamma \vdash S : \mathit{stmt}}{\Gamma \vdash \texttt{while}\ E\ \texttt{do}\ S : \mathit{stmt}}$$

Δυναμική σημασιολογία (i)

- Λειτουργική σημασιολογία (operational semantics)
  ⇒ ακολουθία υπολογιστικών βημάτων
- Δηλωτική σημασιολογία (denotational semantics)
  ⇒ μαθηματική συνάρτηση από το πεδίο των δεδομένων εισόδου στο πεδίο των αποτελεσμάτων
- Αξιωματική σημασιολογία (axiomatic semantics)
  ⇒ 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

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

  - Αξιωματική σημασιολογία

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

Σημασιολογικός έλεγχος

- Έλεγχος τύπων
- Έλεγχος ροής
- Έλεγχος ύπαρξης ονομάτων
- Έλεγχος μοναδικότητας
- Έλεγχος συνέπειας

Σύστημα τύπων (i)

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

Σύστημα τύπων (ii)

- Μετατροπές τύπων (type casting)
- Υπερφόρτωση τελεστών (operator overloading)
- Πολυμορφικοί τελεστές (polymorphic operators)
- Υποσύνολα τύπων και υπο-τύποι (subtypes)
- Πολυμορφικά συστήματα τύπων (polymorphic type systems)
- Στατική και δυναμική αντιστοίχιση τύπων (type binding)
- Εξαγωγή τύπων (type inference)

Δυναμικός έλεγχος τύπων

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

```
a[i] := 42
```

  αν `i` ≥ 0 και `i` ≤ 100 τότε
    κώδικας για την ανάθεση του 42 στο `a[i]`
  αλλιώς
    σφάλμα εκτέλεσης
  τέλος αν

Ενδιάμεσος κώδικας (i)

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

Ενδιάμεσος κώδικας (ii)

- Μετάφραση οδηγούμενη από τη σύνταξη (syntax-directed translation)
  - Για κάθε δομή της γλώσσας προσδιορίζεται ο αντίστοιχος ενδιάμεσος κώδικας
  - Διευρύνεται ο συντακτικός αναλυτής με σημασιολογικές ρουτίνες που παράγουν ενδιάμεσο κώδικα
- Σχέδιο παραγωγής ενδιάμεσου κώδικα
- Μεταβλητές ιδιοτήτων (attributes) για κάθε σύμβολο της γραμματικής

Ενδιάμεση γλώσσα (i)

- Τετράδες (quadruples)

  n: op, x, y, z
- Παράδειγμα: `b*b-4*a*c`

  1: $*, \mathrm{b}, \mathrm{b}, \$1$
  2: $*, 4, \mathrm{a}, \$2$
  3: $*, \$2, \mathrm{c}, \$3$
  4: $-, \$1, \$3, \$4$

Ενδιάμεση γλώσσα (ii)

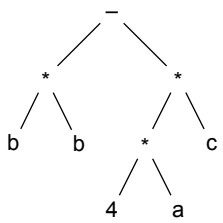
- Τριάδες (triples)

  n: op, x, y
- Παράδειγμα: `b*b-4*a*c`

  1: $*, \mathrm{b}, \mathrm{b}$
  2: $*, 4, \mathrm{a}$
  3: $*, (2), \mathrm{c}$
  4: $-, (1), (3)$

Ενδιάμεση γλώσσα (iii)

- Αφηρημένα συντακτικά δέντρα (abstract syntax trees)
- Παράδειγμα: `b*b-4*a*c`



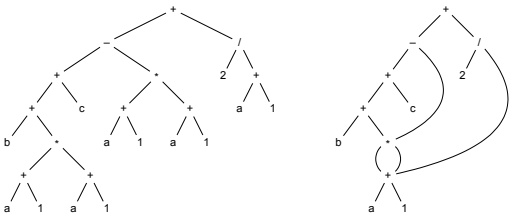
Ενδιάμεση γλώσσα (iv)

- Προθεματικός και επιθεματικός κώδικας (prefix/postfix code)
- Παράδειγμα: `b*b-4*a*c`

  $- * \mathrm{b}\ \mathrm{b} * * 4\ \mathrm{a}\ \mathrm{c}$ προθεματικός
  $\mathrm{b}\ \mathrm{b} * 4\ \mathrm{a} * \mathrm{c} * -$ επιθεματικός

Ενδιάμεση γλώσσα (v)

- Κατευθυνόμενοι ακυκλικοί γράφοι (directed acyclic graphs)
- Παράδειγμα: `b+(a+1)*(a+1)+c-(a+1)*(a+1)+2/(a+1)`



Γλώσσα τετράδων

- Μορφή τετράδας:

  n: op, x, y, z

  όπου:
  - n: ετικέτα τετράδας (φυσικός αριθμός)
  - op: τελεστής
  - x, y, z: τελούμενα
- Ανάλογα με το είδος του τελεστή, κάποια τελούμενα ενδεχομένως παραλείπονται

Τελούμενα (i)

- Σταθερά
  - ακέραια, πραγματική, λογική
  - χαρακτήρας, συμβολοσειρά, `nil`
- Όνομα
  - μεταβλητή
  - παραμέτρος υποπρογράμματος
  - υποπρόγραμμα
- Προσωρινή μεταβλητή: $\$n$

Τελούμενα (ii)

- Ετικέτα
  - εντολής στο αρχικό πρόγραμμα
  - τετράδας
- Τρόπος περάσματος
  - V : κατ' αξία
  - R : κατ' αναφορά
  - RET : θέση αποτελέσματος συνάρτησης
- Κενό : $-$
- Προσωρινά κενό : $*$ (για backpatching)

Τελεστές (i)

- unit, $I, -, -$
- endu, $I, -, -$
  αρχή και τέλος δομικής μονάδας
- op, x, y, z $op \in \{+, -, *, /, \%\}$
  $z := x\ op\ y$
- $:=, x, -, z$
  $z := x$
- op, x, y, z $op \in \{=, <>, >, <, >=, <=\}$
  αν $x\ op\ y$ τότε πήγαινε στην τετράδα z

Τελεστές (ii)

- ifb, $x, -, z$
  αν η λογική τιμή x είναι αληθής τότε πήγαινε στην τετράδα z
- jump, $-, -, z$
  πήγαινε στην τετράδα z
- call, $-, -, I$
  κάλεσε τη δομική μονάδα I

Τελεστές (iii)

- par, $x, m, -$
  πέρασε την πραγματική παράμετρο x με τρόπο περάσματος m
- retv, x, $-, -$
  κάνε το αποτέλεσμα της τρέχουσας συνάρτησης ίσο με x
- ret, $-, -, -$
  επιστροφή από την τρέχουσα δομική μονάδα

Μεταβλητές ιδιοτήτων

- *PLACE*: θέση όπου βρίσκεται αποθηκευμένη η τιμή μιας l-value ή μιας r-value
- *TYPE*: τύπος μιας l-value ή μιας r-value
- *NEXT*: λίστα από ετικέτες τετράδων που περιέχουν άλματα στην επόμενη εντολή
- *TRUE*, *FALSE*: λίστες από ετικέτες τετράδων που περιέχουν άλματα στον κώδικα που πρέπει να εκτελεστεί αν μια συνθήκη είναι αληθής ή ψευδής

Βοηθητικές υπορουτίνες (i)

- NEXTQUAD()
  Επιστρέφει τον αριθμό της επόμενης τετράδας
- GENQUAD(op, x, y, z)
  Γεννά την επόμενη τετράδα op, x, y, z
- NEWTEMP(t)
  Δημιουργεί μια νέα προσωρινή μεταβλητή τύπου t
- EMPTYLIST()
  Δημιουργεί μια κενή λίστα ετικετών τετράδων

Βοηθητικές υπορουτίνες (ii)

- MAKELIST(x)
  Δημιουργεί μια λίστα ετικετών τετράδων που περιέχει μόνο το στοιχείο x
- MERGE($l_1, \ldots, l_n$)
  Συνένωση των λιστών ετικετών τετράδων $l_1 \ldots l_n$
- BACKPATCH(l, z)
  Αντικαθιστά σε όλες τις τετράδες που περιέχονται στην l την άγνωστη ετικέτα τετράδας με τη z (backpatching)

Αριθμητικές εκφράσεις

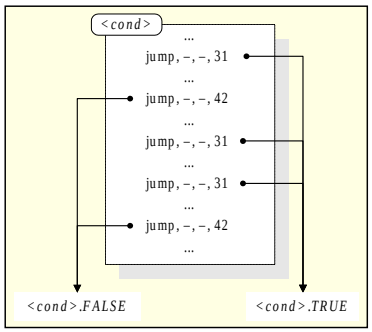
- Ακέραιες σταθερές

⟨r-value⟩ ::= ⟨integer-const⟩ { P_1 }
P_1 : { ⟨r-value⟩.*PLACE* = ⟨integer-const⟩; }

- Τελεστές με δύο τελούμενα

⟨r-value⟩ ::= ⟨expr⟩ ⟨binop⟩ ⟨expr⟩ { P_{14} }
P_{14} : { W = NEWTEMP(⟨r-value⟩.*TYPE*);
GENQUAD(⟨binop⟩.*NAME*,
⟨expr⟩^{1}.*PLACE*,
⟨expr⟩^{2}.*PLACE*, W);
⟨r-value⟩.*PLACE* = W; }

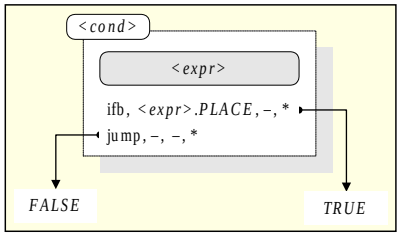
Λογικές εκφράσεις (i)



Λογικές εκφράσεις (ii)

- Λογικές εκφράσεις σε συμβολισμό 0/1

⟨cond⟩ ::= ⟨expr⟩



Λογικές εκφράσεις (iii)

- Λογικές εκφράσεις σε συμβολισμό 0/1

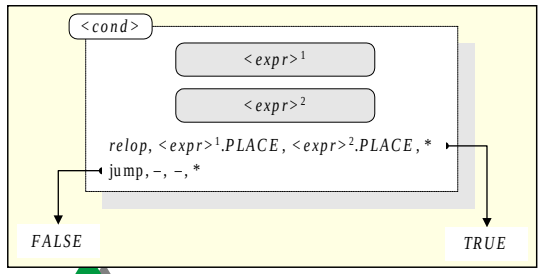
⟨cond⟩ ::= ⟨expr⟩ { P_{21} }

P_{21} : { ⟨cond⟩.*TRUE* = MAKELIST(NEXTQUAD());
GENQUAD(ifb, ⟨expr⟩.*PLACE*, –, ∗);
⟨cond⟩.*FALSE* = MAKELIST(NEXTQUAD());
GENQUAD(jump, –, –, ∗); }

Λογικές εκφράσεις (iv)

- Τελεστές σύγκρισης

⟨cond⟩ ::= ⟨expr⟩^{1} ⟨relop⟩ ⟨expr⟩^{2}



Λογικές εκφράσεις (v)

- Τελεστές σύγκρισης

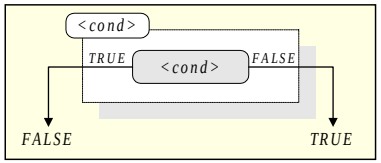
⟨cond⟩ ::= ⟨expr⟩^{1} ⟨relop⟩ ⟨expr⟩^{2} { P_{23} }

P_{23} : { ⟨cond⟩.*TRUE* = MAKELIST(NEXTQUAD());
GENQUAD(⟨relop⟩.*NAME*,
⟨expr⟩^{1}.*PLACE*,
⟨expr⟩^{2}.*PLACE*, ∗);
⟨cond⟩.*FALSE* = MAKELIST(NEXTQUAD());
GENQUAD(jump, –, –, ∗); }

Λογικές εκφράσεις (vi)

- Άρνηση

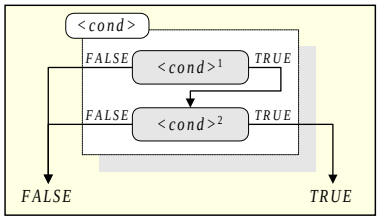
⟨cond⟩ ::= "**not**" ⟨cond⟩



Λογικές εκφράσεις (vii)

- Σύζευξη

⟨cond⟩ ::= ⟨cond⟩^{1} "**and**" ⟨cond⟩^{2}



Λογικές εκφράσεις (viii)

- Σύζευξη

⟨cond⟩ ::= ⟨cond⟩$_1$ “**and**” { P_{25} } ⟨cond⟩$_2$ { P_{26} }

P_{25} : { BACKPATCH(⟨cond⟩^{1}.*TRUE*, NEXTQUAD()); }

P_{26} : { ⟨cond⟩.*FALSE* = MERGE(⟨cond⟩^{1}.*FALSE*, ⟨cond⟩^{2}.*FALSE*);
⟨cond⟩.*TRUE* = ⟨cond⟩^{2}.*TRUE*; }

Απλές εντολές

- Κενή εντολή

⟨stmt⟩ ::= ϵ { P_{29} }

P_{29} : { ⟨stmt⟩.*NEXT* = EMPTYLIST(); }

- Εντολή ανάθεσης

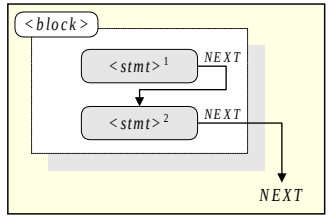
⟨stmt⟩ ::= ⟨l-value⟩ “:=” ⟨expr⟩ { P_{30} }

P_{30} : { GENQUAD(“:=”, ⟨expr⟩.*PLACE*, −, ⟨l-value⟩.*PLACE*);
⟨stmt⟩.*NEXT* = EMPTYLIST(); }

Σύνθετη εντολή (i)

⟨stmt⟩ ::= ⟨block⟩

⟨block⟩ ::= “**begin**” ⟨stmt⟩ (“;” ⟨stmt⟩)* “**end**”



Σύνθετη εντολή (ii)

⟨stmt⟩ ::= ⟨block⟩ { P_{34} }

P_{34} : { ⟨stmt⟩.*NEXT* = ⟨block⟩.*NEXT*; }

⟨block⟩ ::= “**begin**” ⟨stmt⟩^{1} { P_{35} }
(“;” { P_{36} } ⟨stmt⟩^{2} { P_{37} })* “**end**” { P_{38} }

P_{35} : { L = ⟨stmt⟩^{1}.*NEXT*; }

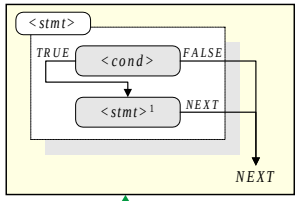
P_{36} : { BACKPATCH(L, NEXTQUAD()); }

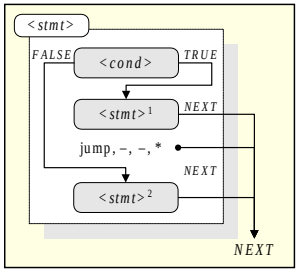
P_{37} : { L = ⟨stmt⟩^{2}.*NEXT*; }

P_{38} : { ⟨block⟩.*NEXT* = L; }

Εντολή if (i)

⟨stmt⟩ ::= “**if**” ⟨cond⟩ “**then**” ⟨stmt⟩ [“**else**” ⟨stmt⟩]





Εντολή if (ii)

⟨stmt⟩ ::= “**if**” ⟨cond⟩ { P_{39} } “**then**” ⟨stmt⟩^{1}
[“**else**” { P_{40} } ⟨stmt⟩^{2} { P_{41} }] { P_{42} }

P_{39} : { BACKPATCH(⟨cond⟩.*TRUE*, NEXTQUAD());
L_1 = ⟨cond⟩.*FALSE*;
L_2 = EMPTYLIST(); }

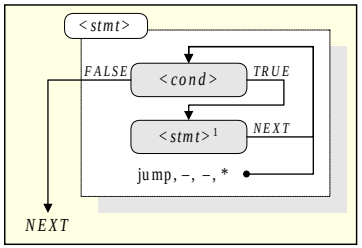
P_{40} : { L_1 = MAKELIST(NEXTQUAD());
GENQUAD(jump, −, −, ∗);
BACKPATCH(⟨cond⟩.*FALSE*, NEXTQUAD()); }

P_{41} : { L_2 = ⟨stmt⟩^{2}.*NEXT*; }

P_{42} : { ⟨stmt⟩.*NEXT* = MERGE(L_1, ⟨stmt⟩^{1}.*NEXT*, L_2); }

Εντολή while (i)

⟨stmt⟩ ::= “**while**” ⟨cond⟩ “**do**” ⟨stmt⟩



Εντολή while (ii)

⟨stmt⟩ ::= “**while**” { P_{43} } ⟨cond⟩ “**do**” { P_{44} } ⟨stmt⟩^{1} { P_{45} }

P_{43} : { Q = NEXTQUAD(); }

P_{44} : { BACKPATCH(⟨cond⟩.$TRUE$, NEXTQUAD()); }

P_{45} : { BACKPATCH(⟨stmt⟩^{1}.$NEXT$, Q);
GENQUAD(jump, –, –, Q);
⟨stmt⟩.$NEXT$ = ⟨cond⟩.$FALSE$; }

Κλήση υποπρογραμμάτων (i)

⟨call⟩ ::= ⟨id⟩ “(” [⟨expr⟩ (“,” ⟨expr⟩)*] “)”
⟨r-value⟩ ::= ⟨call⟩
⟨stmt⟩ ::= ⟨call⟩

- Πέρασμα παραμέτρων με τετράδες `par`
- Πέρασμα θέσης αποτελέσματος με τετράδα `par` αν πρόκειται για συνάρτηση
- Κλήση με τετράδα `call`

Κλήση υποπρογραμμάτων (ii)

⟨call⟩ ::= ⟨id⟩ “(” { P_{46} } [⟨expr⟩^{1} { P_{47} }
(“,” ⟨expr⟩^{2} { P_{48} })*] “)” { P_{49} }

P_{46} : { $N = 1$; }

P_{47} : { GENQUAD(“**par**”, ⟨expr⟩^{1}.$PLACE$,
PARAMMODE(⟨id⟩, N), –);
$N = N + 1$; }

P_{48} : { GENQUAD(“**par**”, ⟨expr⟩^{2}.$PLACE$,
PARAMMODE(⟨id⟩, N), –);
$N = N + 1$; }

Κλήση υποπρογραμμάτων (iii)

⟨call⟩ ::= ⟨id⟩ “(” { P_{46} } [⟨expr⟩^{1} { P_{47} }
(“,” ⟨expr⟩^{2} { P_{48} })*] “)” { P_{49} }

(συνέχεια)

P_{49} : { **if** (ISFUNCTION(⟨id⟩)) {
W = NEWTEMP(FUNCRESULT(⟨id⟩));
GENQUAD(par, RET, W, –);
⟨call⟩.$PLACE$ = W;
}
GENQUAD(call, –, –, ⟨id⟩); }

Κλήση υποπρογραμμάτων (iv)

- Κλήση συνάρτησης

  ⟨r-value⟩ ::= ⟨call⟩ { P_{50} }
  P_{50} : { ⟨r-value⟩.$PLACE$ = ⟨call⟩.$PLACE$; }

- Κλήση διαδικασίας

  ⟨stmt⟩ ::= ⟨call⟩ { P_{51} }
  P_{51} : { ⟨stmt⟩.$NEXT$ = EMPTYLIST(); }

Κλήση υποπρογραμμάτων (v)

- Επιστροφή από υποπρόγραμμα

$\langle \text{stmt} \rangle$ ::= "**return**" [$\langle \text{expr} \rangle$ { P_{52} }] { P_{53} }

P_{52} : { GENQUAD(retv, $\langle \text{expr} \rangle.PLACE, -, -$); }

P_{53} : { GENQUAD(ret, $-, -, -$); }

- Δήλωση υποπρογράμματος

$\langle \text{body} \rangle$ ::= ($\langle \text{local} \rangle$)* { P_{56} } $\langle \text{block} \rangle$ ";" { P_{57} }

P_{56} : { GENQUAD(unit, $I, -, -$); }

P_{57} : { BACKPATCH($\langle \text{block} \rangle.NEXT$, NEXTQUAD()); GENQUAD(endu, $I, -, -$); }

Τελικός κώδικας (i)

- Από θεωρητικής άποψης, το πρόβλημα της κατασκευής βέλτιστου τελικού κώδικα δεν έχει λύση (undecidable)
- Μορφές τελικού κώδικα:
  - Γλώσσα μηχανής σε απόλυτη μορφή (absolute)
  - Γλώσσα μηχανής σε επανατοποθετήσιμη και διασυνδέσιμη μορφή (relocatable, linkable)
  - Συμβολική γλώσσα (assembly)
  - Άλλη γλώσσα χαμηλού επιπέδου

Τελικός κώδικας (ii)

- Επιμέρους προβλήματα:
  - Επιλογή εντολών
    ⇒ Πώς μεταφράζεται κάθε εντολή του ενδιάμεσου κώδικα
    ⇒ Πώς μεταφράζονται ακολουθίες τέτοιων εντολών
  - Διαχείριση της μνήμης στο χρόνο εκτέλεσης
    ⇒ Πού αποθηκεύονται τα δεδομένα
    ⇒ Πώς γίνεται η επικοινωνία ανάμεσα στις δομικές μονάδες

Τελικός υπολογιστής (i)

- Χαρακτηριστικά:
  - Επεξεργαστής: Intel 8086
  - Λειτουργικό σύστημα: MS-DOS
  - Μοντέλο μνήμης: COM / tiny
    ⇒ Συνολική μνήμη $\leq 64\,\text{K}$
    ⇒ Οργάνωση σε ένα segment
    ⇒ Αρχική διεύθυνση του προγράμματος η $100h$
  - Συμβολική γλώσσα: συμβατή με το συμβολομεταφραστή MASM (Microsoft macro assembler)

Τελικός υπολογιστής (ii)

- Καταχωρητές, μεγέθους 16 bit
  - Γενικής φύσης: `ax`, `bx`, `cx`, `dx`
    ⇒ σε ζεύγη των 8 bit: `ah`, `al`, κ.λπ.
  - Καταχωρητές δείκτες: `sp` (δείκτης στοίβας) και `bp` (δείκτης βάσης)
  - Καταχωρητές αναφοράς: `si` και `di`
  - Καταχωρητές τμημάτων: `cs` (code), `ds` (data), `ss` (stack) και `es` (extra)
  - Ειδικοί καταχωρητές: `ip` (instruction pointer) και καταχωρητής σημαιών (flags)

Τελικός υπολογιστής (iii)

- Διευθύνσεις:

$$address = segment * 16 + offset$$

- Μορφή εντολής:

[*label*] *opname* [$operand_1$ [, $operand_2$]]

Τελικός υπολογιστής (iv)

- Εντολές:
  - Μεταφοράς: `mov`, `lea`
  - Αριθμητικών πράξεων: `add`, `sub`, `neg`, `imul`, `idiv`, `cmp`
  - Λογικών πράξεων: `and`, `or`, `xor`, `not`, `test`
  - Άλματος: `jmp`, `jz`, `jnz`, `jl`, `jle`, `jg`, `jge`
  - Διαχείρισης στοίβας: `push`, `pop`
  - Υποπρογραμμάτων: `call`, `ret`
  - Πράξεων κινητής υποδιαστολής (x87 FPU)

Εντολές μεταφοράς

- `mov` *destination, source* (move)

```
mov ax,42
mov ax,bx
mov ax,[1000h]
mov ax,[si]
mov ax,[si+6]
mov ax,[bp+6]
mov ax,[si+bp+6]
```

- `lea` *destination, source* (load effective address)
- Καθορισμός μεγέθους δεδομένων

```
mov ax,word ptr [bp+6]
mov al,byte ptr [bp+6]
```


Αριθμητικές πράξεις

- `add` op_1, op_2 — $op_1 := op_1 + op_2$
- `sub` op_1, op_2 — $op_1 := op_1 - op_2$
- `neg` op — $op := -op$
- `imul` op — $(\mathtt{dx}, \mathtt{ax}) := \mathtt{ax} * op$
- `idiv` op — $\mathtt{ax} := (\mathtt{dx}, \mathtt{ax})$ **div** op; $\mathtt{dx} := (\mathtt{dx}, \mathtt{ax})$ **mod** op
- `cmp` op_1, op_2 — σύγκρινε τα op_1 και op_2, ενημέρωσε τις σημαίες

Λογικές πράξεις

- `and` op_1, op_2 — $op_1 := op_1$ **and** op_2
- `or` op_1, op_2 — $op_1 := op_1$ **or** op_2
- `not` op — $op :=$ **not** op
- `xor` op_1, op_2 — $op_1 := op_1$ **xor** op_2

Εντολές άλματος

- `jmp` *address* — χωρίς συνθήκη
- `jz` *address* ή `je` *address* — μηδέν / ίσο
- `jnz` *address* ή `jne` *address* — όχι μηδέν / διάφορο
- `jl` *address* — μικρότερο
- `jle` *address* — μικρότερο ή ίσο
- `jg` *address* — μεγαλύτερο
- `jge` *address* — μεγαλύτερο ή ίσο

Εντολές στοίβας

- `push` op — πρόσθεση στη στοίβα: $\mathtt{sp} := \mathtt{sp} - 2$, $[\mathtt{sp}] := op$
- `pop` op — αφαίρεση από τη στοίβα: $op := [\mathtt{sp}]$, $\mathtt{sp} := \mathtt{sp} + 2$

⇒ Η στοίβα αυξάνει προς τα κάτω, δηλαδή προς μικρότερες διευθύνσεις

Εντολές υποπρογραμμάτων

- `call` *address* — κλήση
  $\texttt{sp} := \texttt{sp} - 2, \quad [\texttt{sp}] := \texttt{ip}, \quad \texttt{ip} := \textit{address}$
- `ret` — επιστροφή
  $\texttt{ip} := [\texttt{sp}], \quad \texttt{sp} := \texttt{sp} + 2$

⇒ Η τιμή του `ip` που τοποθετείται στη στοίβα από την `call` είναι η διεύθυνση της εντολής που ακολουθεί την `call`

Εντολές x87 FPU (i)

⇒ Ειδική στοίβα 8 καταχωρητών: ST(0), ... ST(7)

- `fld` *source* — (load real & push)
  ```
  fld tbyte ptr @real1
  ```
- `fild` *source* — (load integer & push)
  ```
  fild word ptr [bp − 2]
  ```
- `fstp` *destination* — (pop & store real)
  ```
  fld tbyte ptr [bp − 10]
  ```
- `fistp` *destination* — (pop & store integer)
  ```
  fild word ptr [bp − 2]
  ```

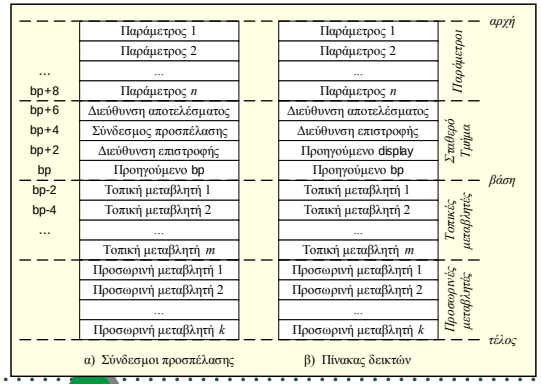

Εντολές x87 FPU (ii)

- `faddp ST(1),ST(0)` — $\texttt{ST(1)} := \texttt{ST(1)} + \texttt{ST(0)}$ & pop
- `fsubp ST(1),ST(0)` — $\texttt{ST(1)} := \texttt{ST(1)} - \texttt{ST(0)}$ & pop
- `fmulp ST(1),ST(0)` — $\texttt{ST(1)} := \texttt{ST(1)} * \texttt{ST(0)}$ & pop
- `fdivp ST(1),ST(0)` — $\texttt{ST(1)} := \texttt{ST(1)}/\texttt{ST(0)}$ & pop
- `fchs` — $\texttt{ST(0)} := -\texttt{ST(0)}$
- `fcompp` — $\texttt{ST(1)} \gtreqless \texttt{ST(0)}$ & pop both
- `fstsw` *destination* — (store x87 FPU flags)
  ```
  fstsw ax
  fstsw word ptr [bp − 2]
  ```

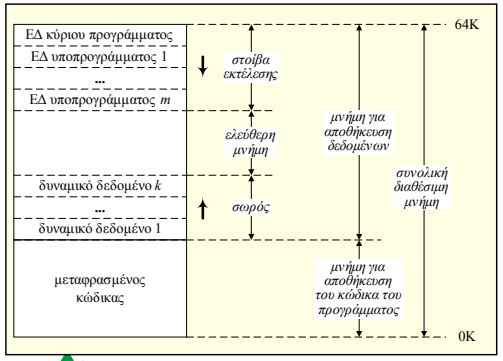

Διαχείριση μνήμης (i)

- Δομή ενοτήτων (block structure)
  - Μη τοπικά δεδομένα
- Εγγράφημα δραστηριοποίησης (activation record)
  - Παράμετροι
  - Αποτέλεσμα
  - Πληροφορίες κατάστασης μηχανής
  - Τοπικές μεταβλητές
  - Προσωρινές μεταβλητές

Διαχείριση μνήμης (ii)



Διαχείριση μνήμης (iii)



Προσπέλαση ονομάτων

- Τοπικά: $[\texttt{bp} + \mathit{offset}]$
- Μη τοπικά: $[\texttt{si} + \mathit{offset}]$

  $\Rightarrow$ ο `si` πρέπει να δείχνει στη βάση του εγγραφήματος δραστηριοποίησης όπου τα δεδομένα είναι τοπικά
- Το πρόβλημα ανάγεται στον εντοπισμό του αντίστοιχου εγγραφήματος δραστηριοποίησης
- Λύσεις που βασίζονται στο βάθος φωλιάσματος:
  - Σύνδεσμοι προσπέλασης (access links)
  - Πίνακες δεικτών (link tables / displays)

Σύνδεσμοι προσπέλασης (i)

- Αρχή λειτουργίας
  - Έστω ότι η δομική μονάδα p βρίσκεται φωλιασμένη μέσα στη δομική μονάδα q

  $\Rightarrow$ Στο ΕΔ της p τοποθετείται ένα σύνδεσμος προς τη βάση του ΕΔ της πιο πρόσφατης κλήσης της q
- Κατά την κλήση υποπρογραμμάτων, απαιτείται τελικός κώδικας για την ενημέρωση των συνδέσμων προσπέλασης

Σύνδεσμοι προσπέλασης (ii)

- Τρόπος χρήσης
  - Έστω ότι ζητείται το δεδομένο a που είναι τοπικό σε μια δομική μονάδα με βάθος φωλιάσματος n_a
  - Έστω ότι βρισκόμαστε σε μια δομική μονάδα p με βάθος φωλιάσματος $n_p \geq n_a$

  $\Rightarrow$ Ακολουθούμε $n_p - n_a$ συνδέσμους προσπέλασης
- Κατά την προσπέλαση ονομάτων, απαιτείται τελικός κώδικας για την υλοποίηση των παραπάνω

Πέρασμα παραμέτρων

- Κλήση κατ' αξία (call by value)
- Κλήση κατ' αναφορά (call by reference)
- Κλήση κατ' όνομα (call by name)
- Κλήση κατ' ανάγκη (call by need)
- Κλήση κατ' αξία και αποτέλεσμα (call by value-result)

$\Rightarrow$ Τρόπος υλοποίησης καθενός

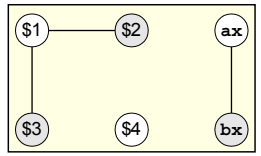
Δέσμευση καταχωρητών (i)

- Πρόβλημα 1: επιλογή τελουμένων που θα αποθηκευτούν σε καταχωρητές
- Πρόβλημα 2: επιλογή καταχωρητών όπου θα αποθηκευτούν τα τελούμενα
- Το πρόβλημα της βέλτιστης χρήσης καταχωρητών είναι NP-complete, ακόμα και χωρίς ειδικούς περιορισμούς
- Η λύση του ανάγεται στην κατασκευή του γράφου αλληλεπιδράσεων μεταξύ των μεταβλητών και στο χρωματισμό αυτού με τόσα χρώματα όσοι οι καταχωρητές

Δέσμευση καταχωρητών (ii)

- Παράδειγμα: `d := b*b-4*a*c`

1: $*, \mathrm{b}, \mathrm{b}, \$1$
2: $*, 4, \mathrm{a}, \$2$
3: $*, \$2, \mathrm{c}, \$3$
4: $-, \$1, \$3, \$4$
5: $:=, \$4, -, \mathrm{d}$

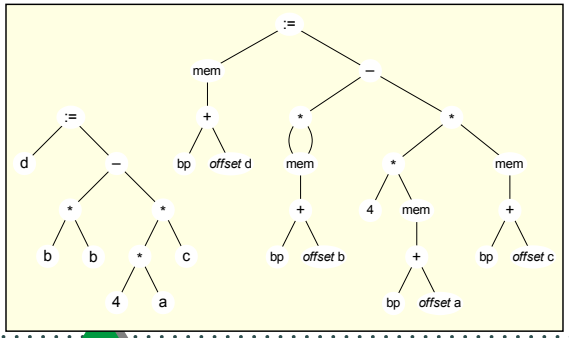


Επιλογή εντολών (i)

- Απλή αλλά κακή προσέγγιση: ενιαίο σχήμα παραγωγής τελικού κώδικα για κάθε δομή του ενδιάμεσου κώδικα
- Καλύτερη προσέγγιση: πλακόστρωση (tiling)
  - τεμαχισμός του ενδιάμεσου κώδικα σε τμήματα
  - κάθε τμήμα αντιστοιχεί σε μια εντολή
- Βέλτιστη αλλά χρονοβόρα προσέγγιση: δυναμικός προγραμματισμός (dynamic programming)

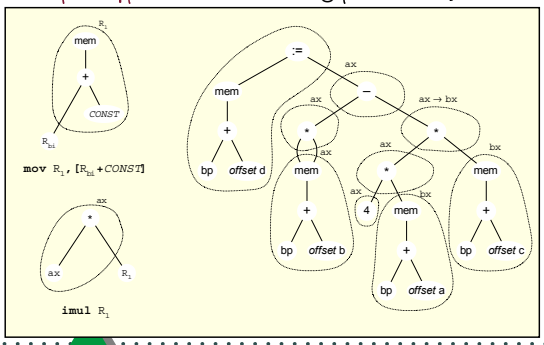
Επιλογή εντολών (ii)

- Παράδειγμα: `d := b*b-4*a*c`



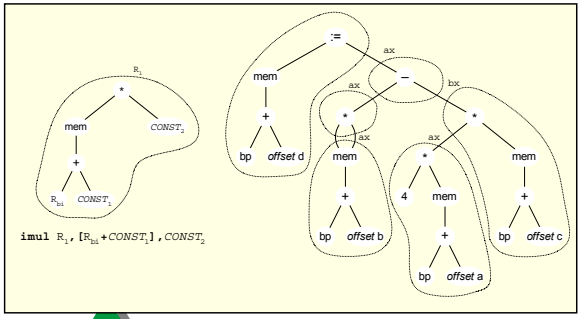
Επιλογή εντολών (iii)

- Παράδειγμα: tiling με εντολές του 8086



Επιλογή εντολών (iv)

- Παράδειγμα: tiling με εντολές του 80386



Το τελικό πρόγραμμα (i)

- Σκελετός:

```
xseg  segment  public 'code'
      assume   cs : xseg, ds : xseg, ss : xseg
      org      100h
main  proc     near
      call     near ptr program
      mov      ax, 4C00h
      int      21h
main  endp

... τελικός κώδικας που παράγεται ...

xseg  ends
      end      main
```


Το τελικό πρόγραμμα (ii)

- Βιβλιοθήκη χρόνου εκτέλεσης (run-time library)

```
extrn function : proc
```

- Σταθερές συμβολοσειρές και κινητής υποδιαστολής

```
@str1   db 'this is'
        db 10
        db 'an example'
        db 0

@real1  dt 1e-10
@real2  dt 2.0
```

Βοηθητικές ρουτίνες (i)

- getAR(a) (φόρτωση διεύθυνσης ΕΔ)

```
mov  si, word ptr [bp + 4]
mov  si, word ptr [si + 4]
...                                  } (n_cur − n_a − 1 φορές)
mov  si, word ptr [si + 4]
```

- updateAL() (ενημέρωση συνδέσμων προσπέλασης)

```
(α)   push  bp                                   αν n_p < n_x
(β)   push  word ptr [bp + 4]                    αν n_p = n_x
(γ)   mov  si, word ptr [bp + 4]                 αν n_p > n_x
      mov  si, word ptr [si + 4]
      ...                                        } (n_p − n_x − 1 φορές)
      mov  si, word ptr [si + 4]
      push  word ptr [si + 4]
```


Βοηθητικές ρουτίνες (ii)

- load(R, a) (φόρτωση τελουμένου)

	Είδος του a	Κώδικας που παράγεται
(α1)	αριθμητική σταθερά	`mov R, a`
(α2)	λογική σταθερά `true`	`mov R, 1`
(α3)	λογική σταθερά `false`	`mov R, 0`
(α4)	σταθερά χαρακτήρα	`mov R, ASCII(a)`
(α6)	σταθερά `nil`	`mov R, 0`
(β1)	τοπική οντότητα: μεταβλητή, παράμετρος κατ΄ αξία, ή προσωρινή μεταβλητή	`mov R, size ptr [bp + offset]`
(β2)	τοπική οντότητα: παράμετρος κατ΄ αναφορά	`mov si, word ptr [bp + offset]` `mov R, size ptr [si]`

Βοηθητικές ρουτίνες (iii)

- load(R, a) (φόρτωση τελουμένου)

	Είδος του a	Κώδικας που παράγεται
(γ1)	μη τοπική οντότητα: μεταβλητή, παράμετρος κατ΄ αξία, ή προσωρινή μεταβλητή	getAR(a) `mov R, size ptr [si + offset]`

(γ2)	μη τοπική οντότητα: παράμετρος κατ΄ αναφορά	getAR(a)
		`mov si, word ptr [si + offset]`
		`mov R, size ptr [si]`

(δ)	$[x]$	load(di, x) `mov R, size ptr [di]`
(ε)	$\{x\}$	loadAddr(R, x)

Βοηθητικές ρουτίνες (iv)

- loadAddr(R, a) (φόρτωση διεύθυνσης τελουμένου)

	Είδος του a	Κώδικας που παράγεται
(α5)	σταθερή συμβολοσειρά	`lea R, byte ptr a`
(β1)	τοπική οντότητα: παράμετρος κατ΄ αξία, ή προσωρινή μεταβλητή	`lea R, size ptr [bp + offset]`
(β2)	τοπική οντότητα: παράμετρος κατ΄ αναφορά	`mov R, word ptr [bp + offset]`
(γ1)	μη τοπική οντότητα: παράμετρος κατ΄ αξία, ή προσωρινή μεταβλητή	getAR(a) `lea R, size ptr [si + offset]`
(γ2)	μη τοπική οντότητα: παράμετρος κατ΄ αναφορά	getAR(a) `mov R, word ptr [si + offset]`
(δ)	$[x]$	load(R, x)

Βοηθητικές ρουτίνες (v)

- Παρόμοια υλοποίηση για τις:
  - loadReal(a)
  - store(R, a)
  - storeReal(a)
- Ρουτίνες για ετικέτες τελικού κώδικα:
  - name(p) _p_*num*
  - endof(p) @p_*num*
  - label(n) @n
  - label(ℓ) @p_*num*_ℓ

Παραγωγή κώδικα (i)

- Τετράδα :=, x, −, z

load(R, x) loadReal(x)
store(R, z) storeReal(z)

- Τετράδα array, x, y, z

```
load(ax, y)
mov   cx, size
imul  cx
loadAddr(cx, x)
add   ax, cx
store(ax, z)
```

Παραγωγή κώδικα (ii)

- Τετράδες $+, x, y, z$ $-, x, y, z$

```
load(ax, x)                 loadReal(x)
load(dx, y)                 loadReal(y)
instr ax, dx                finstr ST(1), ST(0)
store(ax, z)                store(ax, z)
```

instr = `add` ή `sub` *finstr* = `faddp` κ.λπ.

- Τετράδες $*, x, y, z$ $/, x, y, z$ $\%, x, y, z$

```
load(ax, x)     load(ax, x)     load(ax, x)
load(cx, y)     xor  dx, dx     xor  dx, dx
imul cx         load(cx, y)     load(cx, y)
store(ax, z)    idiv cx         idiv cx
                store(ax, z)    store(dx, z)
```


Παραγωγή κώδικα (iii)

- Τετράδες $=, x, y, z$ $<>, x, y, z$ $<, x, y, z$ $>, x, y, z$ $<=, x, y, z$ $>=, x, y, z$

```
load(ax, x)                 loadReal(x)
load(dx, y)                 loadReal(y)
cmp  ax, dx                 fcompp
instr label(z)              fstsw  ax
                            test   ax, value
instr = je, jne, κ.λπ.      instr  label(z)
```

value και *instr* από Πίνακα 9.1 σελ. 249

Παραγωγή κώδικα (iv)

- Τετράδα ifb, x, −, z

```
load(al, x)
or   al, al
jnz  label(z)
```

- Τετράδα jump, −, −, z

```
jmp  label(z)
```

- Τετράδα jumpl, −, −, z

```
jmp  label(z)
```

- Τετράδα label, −, −, z

```
label(z) :
```


Παραγωγή κώδικα (v)

- Τετράδα unit, x, −, −

```
name(x)  proc  near
         push  bp
         mov   bp, sp
         sub   sp, size
```

- Τετράδα endu, x, −, −

```
endof(x) :  mov   sp, bp
            pop   bp
            ret
name(x)     endp
```


Παραγωγή κώδικα (vi)

- Τετράδα call, −, −, z

```
sub   sp, 2                 αν z είναι διαδικασία
updateAL()
call  near ptr name(z)
add   sp, size + 4
```

- Τετράδα ret, −, −, −

```
jmp  endof(current)
```


Παραγωγή κώδικα (vii)

- Τετράδα par, x, y, −
  - αν y = V και x είναι 16 bit

```
load(ax, x)
push  ax
```

  - αν y = V και x είναι 8 bit

```
load(al, x)
sub   sp, 1
mov   si, sp
mov   byte ptr [si], al
```

Παραγωγή κώδικα (vii)

- Τετράδα $\text{par}, x, y, -$ (συνέχεια)
  - αν $y = V$ και x είναι 80 bit

```
loadReal(x)
sub   sp,10
mov   si,sp
fstp  tbyte ptr [si]
```

  - αν $y = R$ ή RET

```
loadAddr(si,x)
push  si
```


Προχωρημένα θέματα

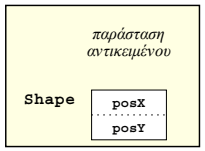
- Μεταγλώττιση αντικειμενοστρεφών γλωσσών
- Χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού
  - λογισμικό οργανωμένο ως σύνολο από αλληλεπιδρώντα αντικείμενα
  - αντικείμενο / κλάση
  - κελυφοποίηση (encapsulation)
  - κληρονομικότητα (inheritance)
  - πολυμορφισμός υποτύπων (subtype polymorphism)

Αντικείμενα και μέθοδοι (i)

- Αντικείμενο = εγγραφή (record) + μέθοδοι
- Αναπαράσταση αντικειμένων: ίδια με την αναπαράσταση εγγραφών
  - Δεσμεύεται χώρος για τα πεδία του αντικειμένου
  - Οι μέθοδοι δε συμπεριλαμβάνονται στην αναπαράσταση
  - Οι κλήσεις μεθόδων μεταφράζονται σε κλήσεις συναρτήσεων με μια επιπλέον παράμετρο: ένα δείκτη στο αντικείμενο όπου αναφέρονται

Αντικείμενα και μέθοδοι (ii)

```
class Shape
  var posX, posY : real;

  procedure move (dx, dy : real);
  begin
    posX := posX + dx;
    posY := posY + dy
  end;
end;
```



$\Downarrow$

```
procedure Shape@move (var self : Shape; dx, dy : real);
begin
  self.posX := self.posX + dx;
  self.posY := self.posY + dy
end;
```

`s.move(1, 2)` $\Rightarrow$ `Shape@move(s, 1, 2)`

Κατασκευαστές & καταστροφείς

- Ίδια αντιμετώπιση με τις μεθόδους

```
class Shape
  ...
  procedure constructor (x, y : real);
  ...
  procedure destructor ();
  ...
end;
```

$\Downarrow$

```
procedure Shape@constructor (var self : Shape;
                             x, y : real);
procedure Shape@destructor  (var self : Shape);
```

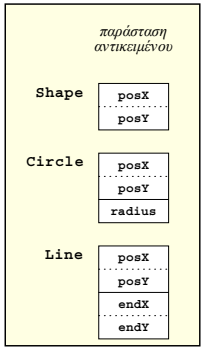
```
var s : Shape(10, 20);  ⇒  Shape@constructor(s, 10, 20);
                           ...
                           Shape@destructor(s);
```


Απλή κληρονομικότητα (i)

- Υποκλάση / υπερκλάση
- Επισκίαση μεθόδων (method override)
- Υποτύποι (subtypes)
- Αναπαράσταση αντικειμένων
  - Τα πεδία της υπερκλάσης τοποθετούνται πρώτα (στην αρχή του αντικειμένου)
  - Αυτό εφαρμόζεται αναδρομικά
  - Τα πεδία της υποκλάσης ακολουθούν

Απλή κληρονομικότητα (ii)

```
class Circle extends Shape
  var radius : real;
  procedure scale (s : real);
  ...
end;

class Line extends Shape
  var endX, endY : real;
  procedure move (dx, dy : real);
  ...
end;
```



Απλή κληρονομικότητα (iii)

- Στατικό δέσιμο μεθόδων (static binding)

```
var s : Shape(10, 20);
    c : Circle(30, 30, 10);
    l : Line(10, 20, 30, 30);
    p : ^Shape;

...
p := @l;

s.move(5, 5);
c.move(5, 5);
l.move(5, 5);
c.scale(2);
p^.move(5, 5);
```

$\Rightarrow$

```
Shape@move(s, 5, 5);
Shape@move(c, 5, 5);
Line@move(l, 5, 5);
Circle@scale(c, 2);
Shape@move(p^, 5, 5)
```


Πολυμορφισμός υποτύπων (i)

- Δυναμικό δέσιμο μεθόδων (dynamic binding)
  - Όταν γίνεται κλήση μεθόδων μέσω δεικτών ή αναφορών σε αντικείμενα της υπερκλάσης, καλούνται οι αντίστοιχες μέθοδοι των υποκλάσεων
  - Πίνακας ανταπόκρισης μεθόδων (method dispatch table) ή περιγραφέας κλάσης (class descriptor)
  - Κοινός για όλα τα αντικείμενα μιας κλάσης

Πολυμορφισμός υποτύπων (ii)

```
class Shape
  var posX, posY : real;
  dynamic procedure move (dx, dy : real);
end;

class Line extends Shape
  var endX, endY : real;
  dynamic procedure move (dx, dy : real);
end;

var p : ^Shape;

...
p := @l;

p^.move(5, 5)
```

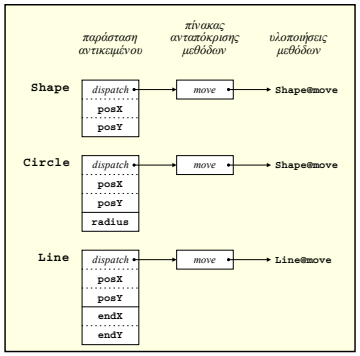

Πολυμορφισμός υποτύπων (iii)

```
var p : ^Shape;

...
p := @l;

p^.move(5, 5)
```

$\Downarrow$

```
f := p^.dispatch^.move;
f^(p^, 5, 5)
```

- Ομοίως για αφηρημένες (abstract) μεθόδους



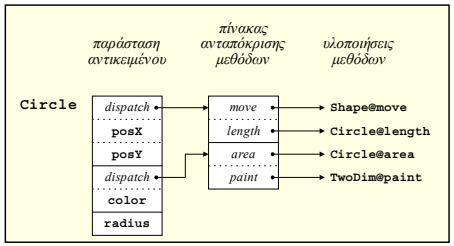
Πολλαπλή κληρονομικότητα (i)

- Πρόβλημα με την αναπαράσταση των αντικειμένων σε συνδυασμό με τους υποτύπους

```
class TwoDim
   var color : ColorType;
   abstract function area () : real;
   dynamic procedure paint (c : ColorType);
end;

class Circle extends Shape, TwoDim
   ...
   dynamic function area () : real;
end;
```

Πολλαπλή κληρονομικότητα (ii)

```
var c  : Circle(...);
    p1 : ^Shape;
    p2 : ^TwoDim;

...
p1 := @c;
p2 := @c
```



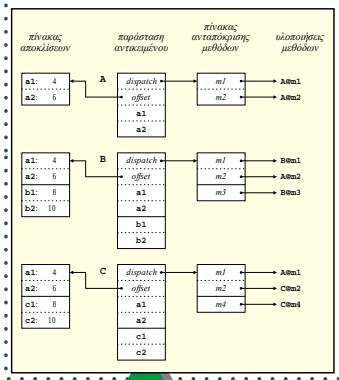
Πολλαπλή κληρονομικότητα (iii)

- Εξαρτημένη πολλαπλή κληρονομικότητα

```
class A
   var a1, a2 : integer;
   dynamic procedure m1 ();
   dynamic procedure m2 ();
end;

class B extends A
   var b1, b2 : integer;
   dynamic procedure m1 ();
   dynamic procedure m3 ();
end;
```

```
class C extends A
   var c1, c2 : integer;
   dynamic procedure m2 ();
   dynamic procedure m4 ();
end;

class D extends B, C
   var d1, d2 : integer;
   dynamic procedure m2 ();
   dynamic procedure m3 ();
   dynamic procedure m5 ();
end;
```


Πολλαπλή κληρονομικότητα (iv)



- Πίνακας αποκλίσεων (offset table)

  p^.a1 ⇒

  (p + p^.offset^.a1)^

Έλεγχος υποτύπων (i)

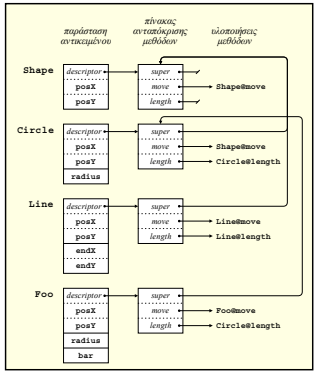
Ερώτηση 1: σε ποια κλάση ανήκει το αντικείμενο X;
Απάντηση: απλώς σύγκρινε τους περιγραφείς κλάσης

Ερώτηση 2: ανήκει το αντικείμενο X στην κλάση $\mathcal{C}$;
Απάντηση: ...λίγο πιο δύσκολη...

```
var p : ^Shape;
...
if p^ instanceOf Circle then
   ...1...
else
   ...2...
```

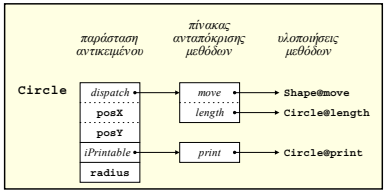

Έλεγχος υποτύπων (ii)

```
  d := p^.descriptor;
loop:
  if d = Circle@descriptor
  then begin
    ...1...
    goto next
  end
  else if d = nil then
  begin
    ...2...
    goto next
  end
  else
    d = d^.super;
  goto loop;
next:
```



Διαπροσωπείες (i)

```
interface Printable
   dynamic procedure print ();
end;

class Circle extends Shape implements Printable
   ...
   dynamic procedure print ();
end;
```



Διαπροσωπείες (ii)

```
var c : Circle(...);
    p : ^Printable;

...
p := @c;

p^.print()
```

⇓

```
p.self := @c;
p.iPrintable := c.iPrintable
```