

Μεταγλωτιστές

Νίκος Παπασπύρου Κωστής Σαγώνας

{nickie}@softlab.ntua.gr {kostis}@cs.ntua.gr



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

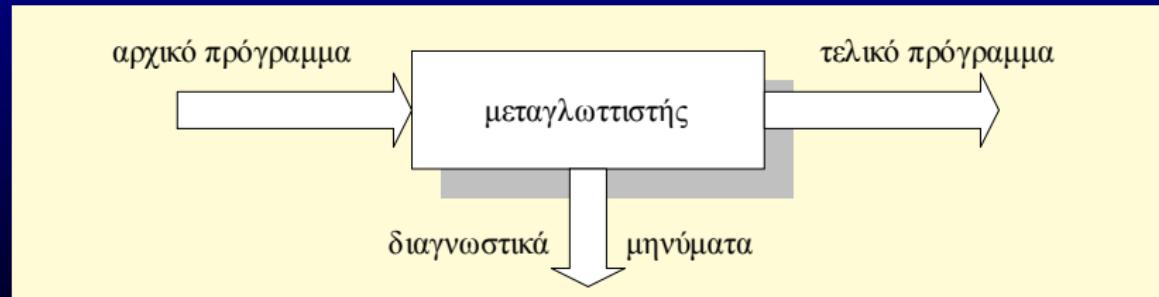
Φεβρουάριος 2010

Εισαγωγή (i)

- ▶ Γλώσσες προγραμματισμού
- ▶ Μεταγλωττιστές
- ▶ Αναγκαιότητα και ιστορική αναδρομή

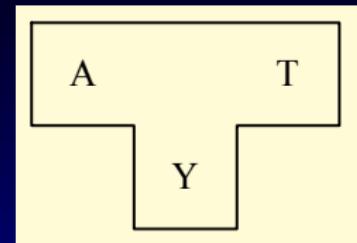
Εισαγωγή (i)

- ▶ Γλώσσες προγραμματισμού
- ▶ Μεταγλωττιστές
- ▶ Αναγκαιότητα και ιστορική αναδρομή



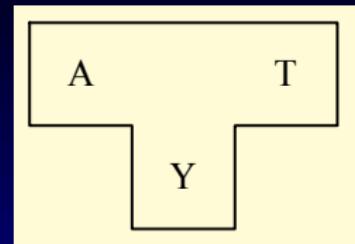
Εισαγωγή (ii)

- Αρχική γλώσσα L_A
- Τελική γλώσσα L_T
- Γλώσσα υλοποίησης L_Y



Εισαγωγή (ii)

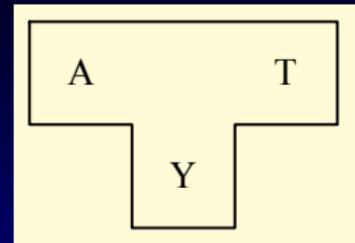
- ▶ Αρχική γλώσσα L_A
- ▶ Τελική γλώσσα L_T
- ▶ Γλώσσα υλοποίησης L_Y
- ▶ Σημασιολογία του προγράμματος P στη γλώσσα L



$$[\![P]\!]_L : Inp(P) \rightarrow Out(P)$$

Εισαγωγή (ii)

- ▶ Αρχική γλώσσα L_A
- ▶ Τελική γλώσσα L_T
- ▶ Γλώσσα υλοποίησης L_Y
- ▶ Σημασιολογία του προγράμματος P στη γλώσσα L



$$[\![P]\!]_L : Inp(P) \rightarrow Out(P)$$

- ▶ Σημασία ενός μεταγλωττιστή C

$$[\![C]\!]_{L_Y} : L_A \rightarrow L_T$$

Εισαγωγή (iii)

- ▶ Ορθότητα του μεταγλωττιστή: “το μεταγλωττισμένο πρόγραμμα πρέπει να είναι ισοδύναμο με το αρχικό”

Εισαγωγή (iii)

- Ορθότητα του μεταγλωττιστή: “το μεταγλωττισμένο πρόγραμμα πρέπει να είναι ισοδύναμο με το αρχικό”

$$\llbracket P \rrbracket_{L_A} = \llbracket \llbracket C \rrbracket_{L_Y}(P) \rrbracket_{L_T}$$

Εισαγωγή (iv)

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

Εισαγωγή (v)

► 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)

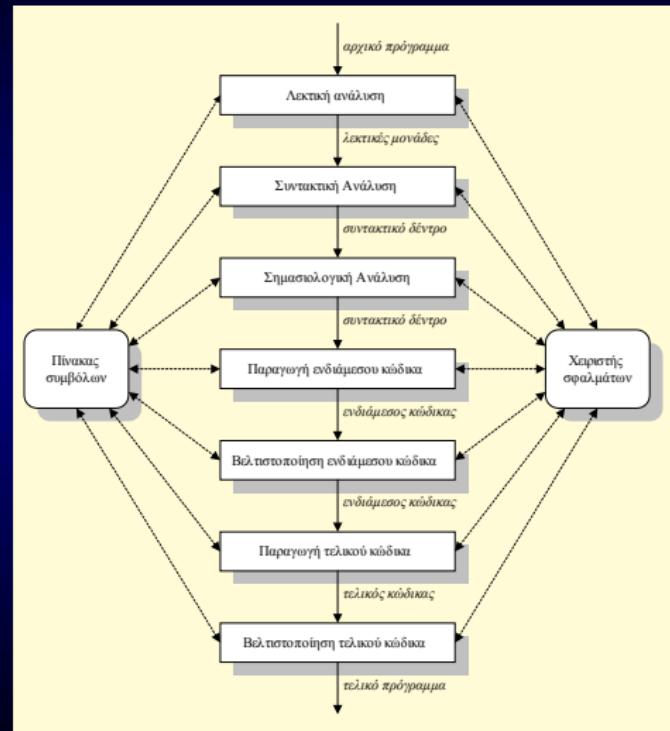
- ▶ Βασικές απαιτήσεις:
 - ▶ Να λειτουργεί σωστά
 - ▶ Να συμμορφώνεται με τις προδιαγραφές της αρχικής και της τελικής γλώσσας
 - ▶ Να μεταγλωτίζει προγράμματα κάθε μεγέθους

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 και προϊόντα της μεταγλώττισης

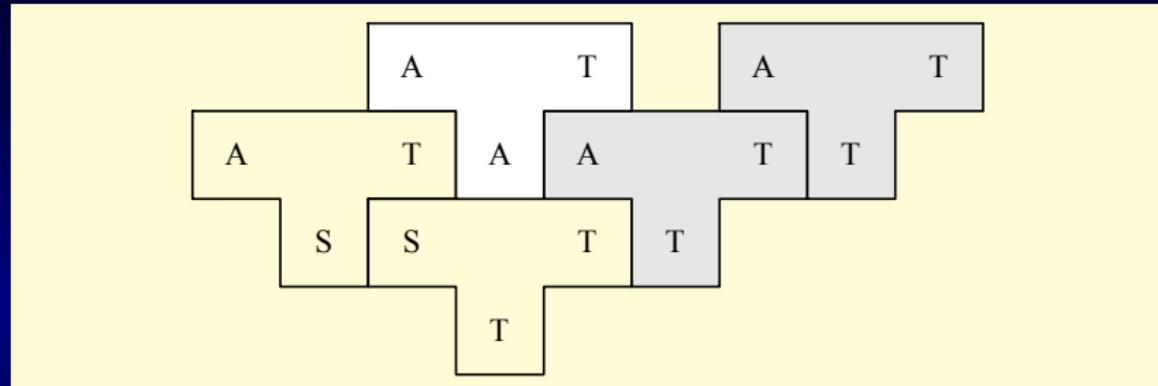
- ▶ Λεκτική ανάλυση
- ▶ Συντακτική ανάλυση
- ▶ Σημασιολογική ανάλυση
- ▶ Παραγωγή ενδιάμεσου κώδικα
- ▶ Βελτιστοποίηση
- ▶ Παραγωγή τελικού κώδικα



Θέματα υλοποίησης

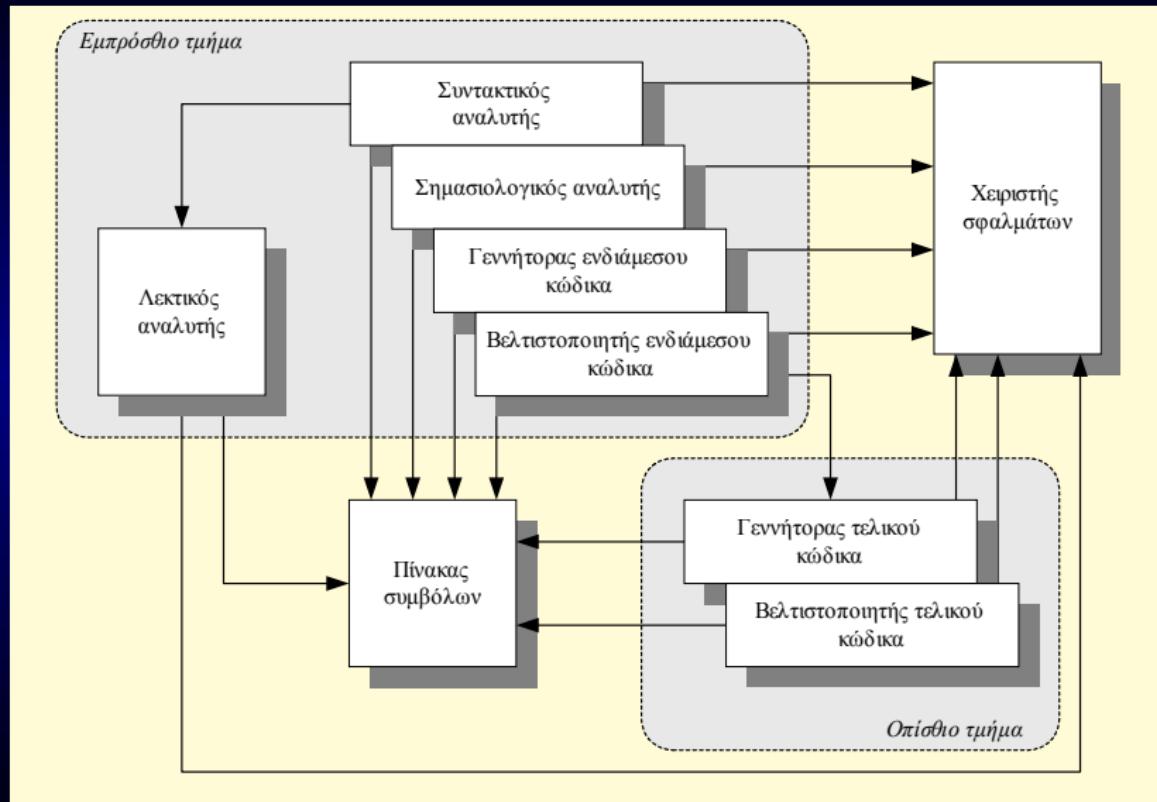
- ▶ Οργάνωση σε περάσματα
- ▶ Οργάνωση σε εμπρόσθιο και οπίσθιο τμήμα
(front-end / back-end)
- ▶ Έλεγχος ορθότητας
- ▶ Είδη διαγνωστικών μηνυμάτων και ανάνηψη
 - ▶ Εσωτερικά (internal)
 - ▶ Σφάλματα (errors)
 - ▶ Προειδοποιητικά μηνύματα (warnings)
 - ▶ Απλά μηνύματα (messages)

Εκκίνηση – bootstrapping



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

Οργάνωση σε ένα πέρασμα



Κεφάλαιο 2: Τυπικές γλώσσες

Τυπικές γλώσσες (i)

Βασικές έννοιες

- ▶ Αλφάβητο Σ
- ▶ Σύμβολο a
- ▶ Συμβολοσειρά α
- ▶ Μήκος συμβολοσειράς $|\alpha|$
- ▶ Σύνολο συμβολοσειρών μήκους n Σ^n
- ▶ Σύνολο όλων των συμβολοσειρών

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

Τυπικές γλώσσες (ii)

Βασικές έννοιες (συνέχεια)

- ▶ Κενή συμβολοσειρά ϵ
- ▶ Παράθεση συμβολοσειρών $\alpha\beta$

- ▶ Παράθεση συμβολοσειράς με τον εαυτό της

$$\alpha^0 = \epsilon$$

$$\alpha^{n+1} = \alpha\alpha^n$$

- ▶ Πρόθεμα, επίθεμα, υποσυμβολοσειρά

Τυπικές γλώσσες (iii)

Βασικές έννοιες (συνέχεια)

► Γλώσσα

$$L \subseteq \Sigma^*$$

► Ένωση γλωσσών

$$L_1 \cup L_2 = \{ \alpha \mid \alpha \in L_1 \vee \alpha \in L_2 \}$$

► Παράθεση γλωσσών

$$L_1 L_2 = \{ \alpha \beta \mid \alpha \in L_1 \wedge \beta \in L_2 \}$$

► Παράθεση γλώσσας με τον εαυτό της

$$L^0 = \{ \epsilon \}$$

$$L^{n+1} = LL^n$$

► Κλείσιμο ή άστρο του Kleene

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

$$L^+ = LL^*$$

Τυπικές γλώσσες (iv)

Γεννητικά μοντέλα

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

Τυπικές γλώσσες (v)

Ιεραρχία Chomsky

- ▶ Τύπου 0: όλες οι γραμματικές, $\alpha \rightarrow \beta$
- ▶ Τύπου 1: γραμματικές με συμφραζόμενα (context-sensitive), $\alpha \rightarrow \beta$ με $|\alpha| \leq |\beta|$
- ▶ Τύπου 2: γραμματικές χωρίς συμφραζόμενα (context-free) $A \rightarrow \beta$
- ▶ Τύπου 3: κανονικές γραμματικές (regular)
 $A \rightarrow aB$ ή $A \rightarrow a$
- ▶ Ειδική περίπτωση: γλώσσες που παράγουν την κενή συμβολοσειρά

Τυπικές γλώσσες (vi)

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 (i)

- ▶ Κανονικές γραμματικές
 - ▶ Μόνο κανόνες $A \rightarrow aB$ ή $A \rightarrow a$
 - ▶ ισοδύναμα $A \rightarrow Ba$ ή $A \rightarrow a$
- ▶ Κανονικές εκφράσεις (regular expressions)
 - ▶ Κενή συμβολοσειρά: ϵ
 - ▶ Κάθε σύμβολο του Σ : a
 - ▶ Παράθεση δύο κανονικών εκφράσεων: (rs)
 - ▶ Διάζευξη δύο κανονικών εκφράσεων: $(r|s)$
 - ▶ Κλείσιμο (ή άστρο) Kleene: (r^*)
- ▶ Συντομογραφίες:
 - ▶ απαλοιφή περιττών παρενθέσεων
 - ▶ $r^+ [a_1, a_2, \dots, a_n] [a_1 - a_2] r? .$

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία

[0-9]⁺

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία $[0\text{-}9]^+$
- ▶ Αριθμητικές σταθερές χωρίς πρόσημο στη C

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία $[0-9]^+$
- ▶ Αριθμητικές σταθερές χωρίς πρόσημο στη C
 - ▶ ακέραιο μέρος που δεν αρχίζει με μηδέν, εκτός αν είναι μηδενικό (γιατί;)

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία $[0-9]^+$
- ▶ Αριθμητικές σταθερές χωρίς πρόσημο στη C
 - ▶ ακέραιο μέρος που δεν αρχίζει με μηδέν,
εκτός αν είναι μηδενικό (γιατί;)

([1-9][0-9]*|0)

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία $[0-9]^+$
- ▶ Αριθμητικές σταθερές χωρίς πρόσημο στη C
 - ▶ ακέραιο μέρος που δεν αρχίζει με μηδέν, εκτός αν είναι μηδενικό (γιατί;)
 - ▶ προαιρετικά: υποδιαστολή και κλασματικό μέρος

([1-9][0-9]*|0)

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία $[0-9]^+$
- ▶ Αριθμητικές σταθερές χωρίς πρόσημο στη C
 - ▶ ακέραιο μέρος που δεν αρχίζει με μηδέν, εκτός αν είναι μηδενικό (γιατί;)
 - ▶ προαιρετικά: υποδιαστολή και κλασματικό μέρος

([1-9][0-9]*|0)(\.[0-9]+)?

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία
- ▶ [0-9]⁺
- ▶ Αριθμητικές σταθερές χωρίς πρόσημο στη C
 - ▶ ακέραιο μέρος που δεν αρχίζει με μηδέν, εκτός αν είναι μηδενικό (γιατί;)
 - ▶ προαιρετικά: υποδιαστολή και κλασματικό μέρος
 - ▶ προαιρετικά: εκθέτης με ή χωρίς πρόσημο

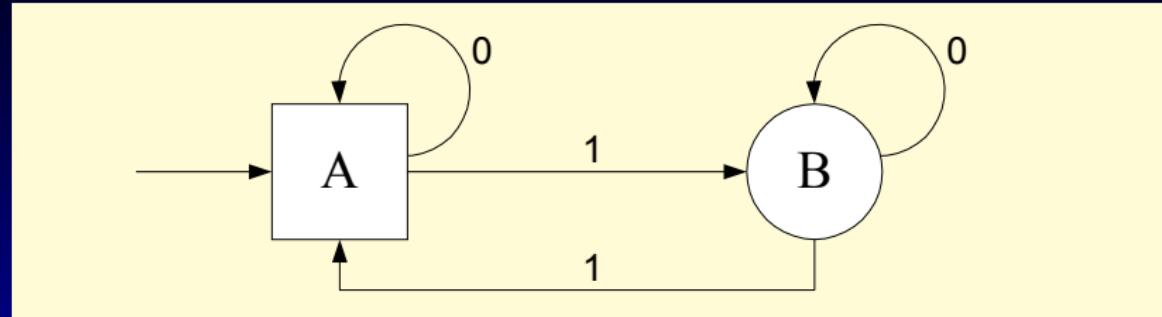
([1-9][0-9]*|0)(\.[0-9]+)?

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

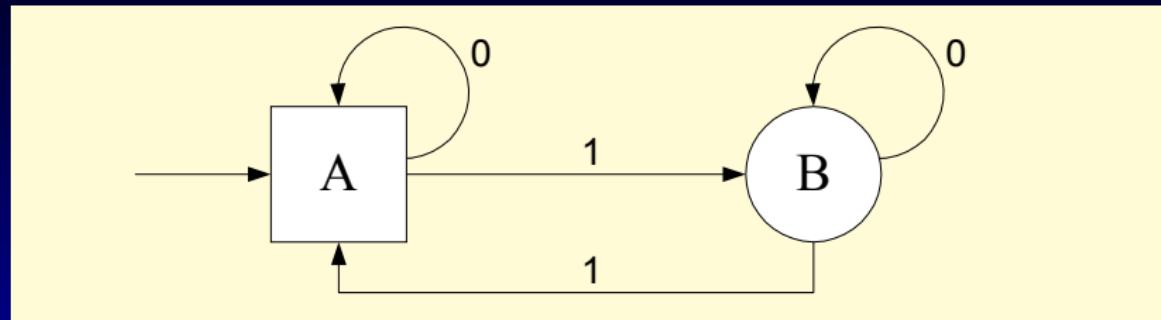
- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία
 $[0-9]^+$
- ▶ Αριθμητικές σταθερές χωρίς πρόσημο στη C
 - ▶ ακέραιο μέρος που δεν αρχίζει με μηδέν,
εκτός αν είναι μηδενικό (γιατί;)
 - ▶ προαιρετικά: υποδιαστολή και κλασματικό μέρος
 - ▶ προαιρετικά: εκθέτης με ή χωρίς πρόσημο
 $([1-9][0-9]^*|0)(\.[0-9]^+)?((E|e)(+|-)?[0-9]^+)?$

Πεπερασμένα αυτόματα (i)



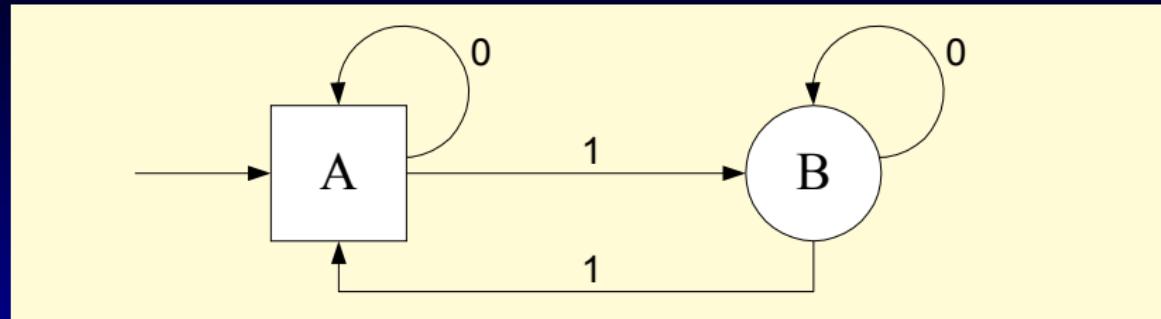
- ▶ Καταστάσεις και μεταβάσεις
- ▶ Ντετερμινιστικά (ΝΠΑ), μη ντετερμινιστικά (ΜΠΑ) και ΜΠΑ με κενές μεταβάσεις (ΜΠΑ- ϵ)
- ▶ Αρχική κατάσταση, τελικές καταστάσεις

Πεπερασμένα αυτόματα (ii)



- ▶ Ποια γλώσσα αναγνωρίζει;

Πεπερασμένα αυτόματα (ii)



- ▶ Ποια γλώσσα αναγνωρίζει;
- ▶ Τη γλώσσα των συμβολοσειρών που αποτελούνται από 0 και 1 και περιέχουν άριθμό 1

Κανονικές γλώσσες, ανασκόπηση

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

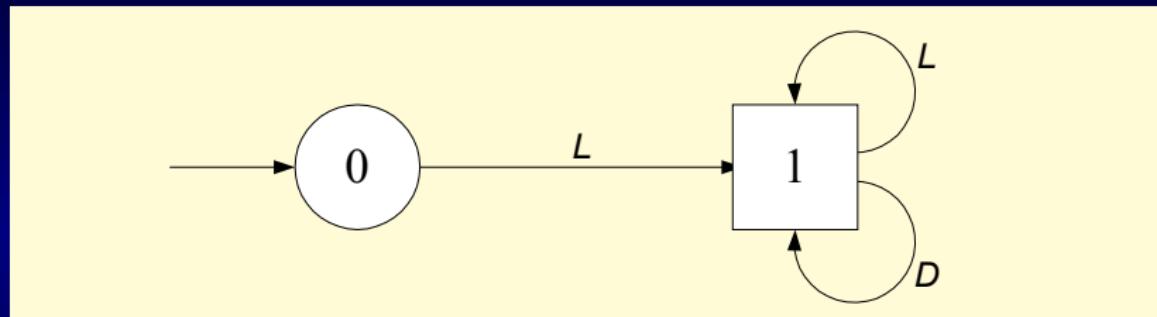
Κεφάλαιο 3: Λεκτική ανάλυση

Λεξική ανάλυση

- ▶ Λεξικές 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

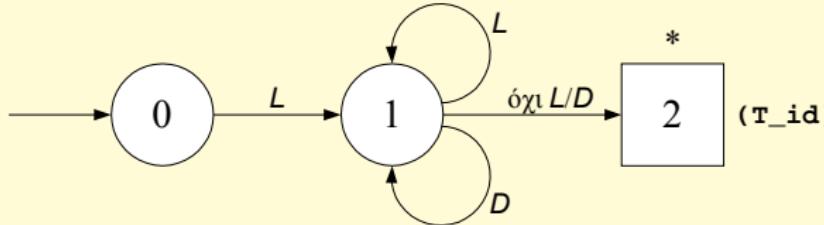
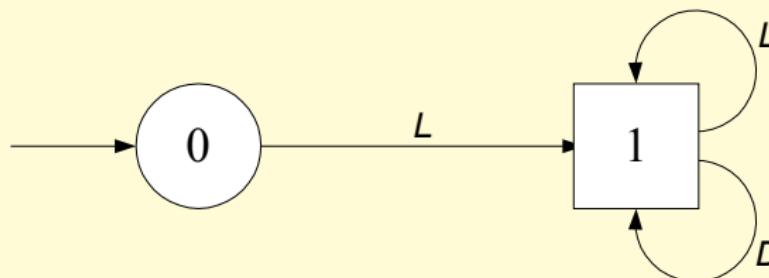
Διαγράμματα μετάβασης (i)

- Αναγνωριστικά της Pascal



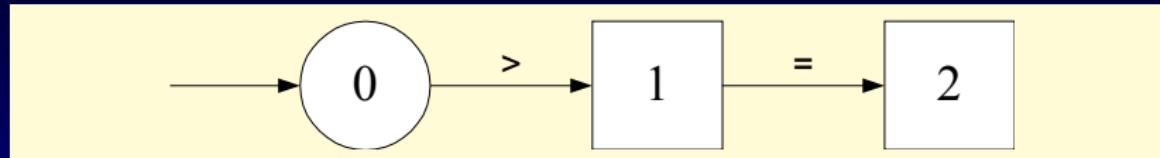
Διαγράμματα μετάβασης (i)

- Αναγνωριστικά της Pascal



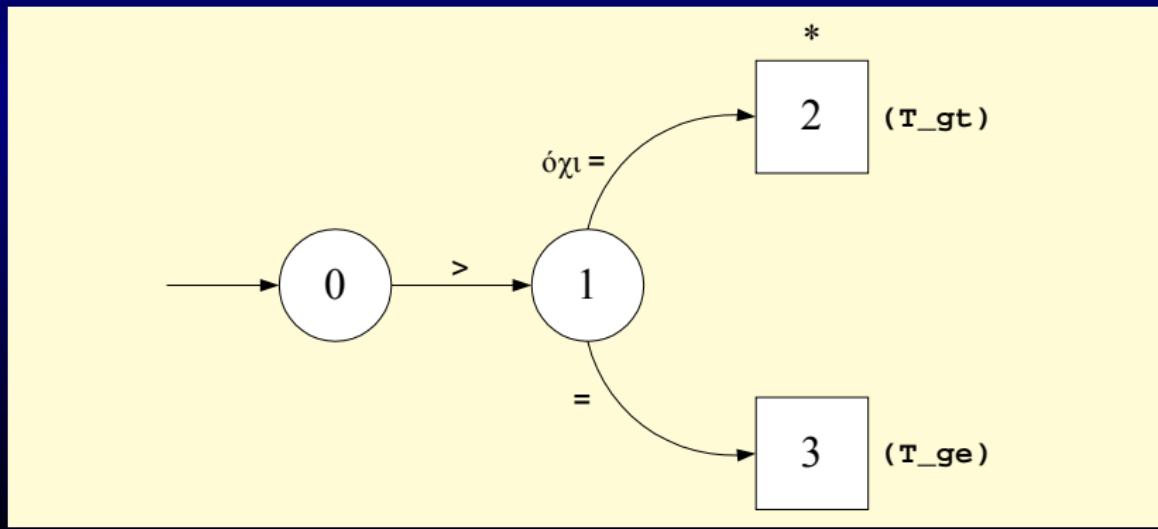
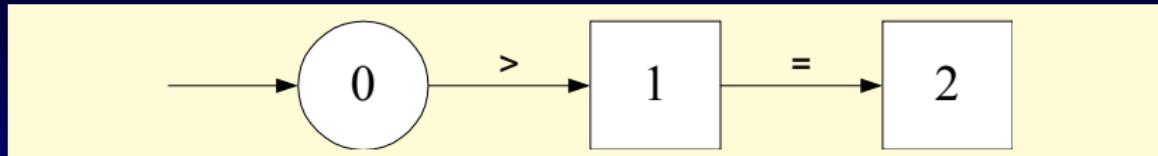
Διαγράμματα μετάβασης (ii)

► Τελεστές > και >=



Διαγράμματα μετάβασης (ii)

► Τελεστές > και >=



Κατασκευή του ΛΑ (i)

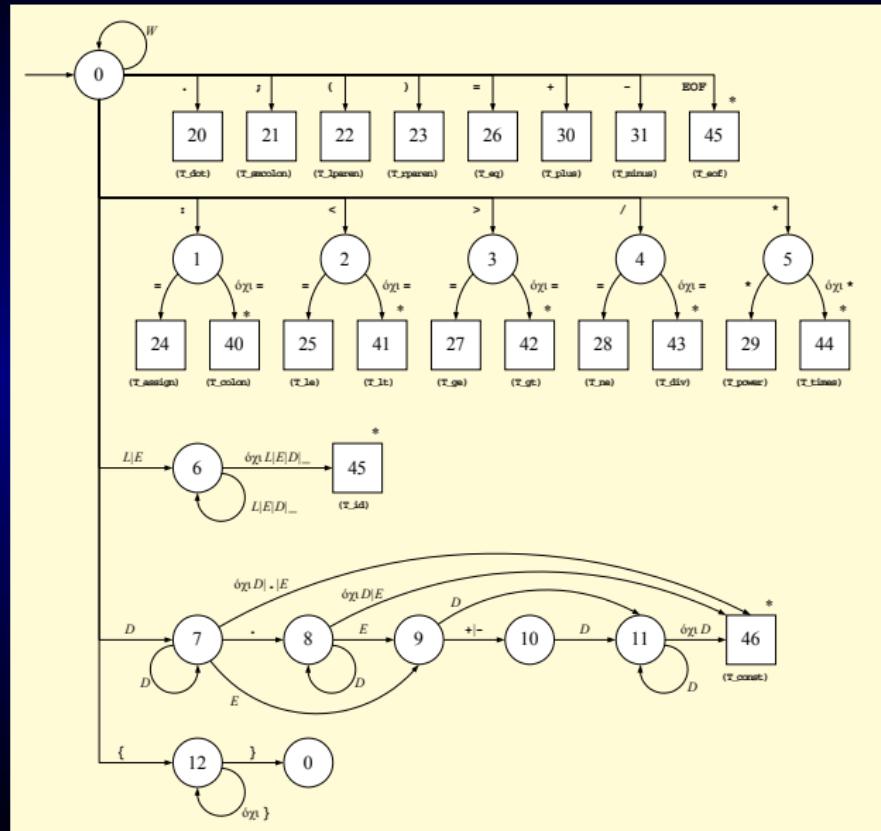
- ▶ Καταγραφή και ταξινόμηση χαρακτήρων
 $mapping : (\text{ASCII} \cup \{\text{EOF}\}) \rightarrow \Sigma$
- ▶ Καταγραφή και ταξινόμηση λεκτικών μονάδων
 - ▶ Κωδικοποίηση λεκτικών μονάδων
 - ▶ Ακολουθία χαρακτήρων (lexeme)
- ▶ Σχεδίαση του διαγράμματος μετάβασης
- ▶ Υλοποίηση του λεκτικού αναλυτή

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 `yyltext` 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 A, 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 A, 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 B, 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.
- .
- Οποιοισδήποτε χαρακτήρας εκτός της αλλαγής γραμμής.
- \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.
- [^a-z] 'Ένας από τους χαρακτήρες εκτός όσων ανήκουν στην περιοχή a έως z.

Υλοποίηση ΛΑ με το flex (vii)

► Κανονικές εκφράσεις (συνέχεια)

{name} Η κανονική έκφραση με μνημονικό όνομα name.

rs Η παράθεση των r και s.

r | s Η διάζευξη των r και s.

(r) Η κανονική έκφραση r. Οι παρενθέσεις χρησιμοποιούνται για ομαδοποίηση.

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.

Υλοποίηση ΛΑ με το flex (viii)

► Κανονικές εκφράσεις (συνέχεια)

- \hat{r} Η r αλλά μόνο στην αρχή μιας γραμμής.
- $r\$$ Η r αλλά μόνο στο τέλος μιας γραμμής.
- $<<\text{EOF}>>$ Το τέλος του αρχείου εισόδου.
- r/s Η κανονική έκφραση r αλλά μόνο αν ακολουθεί η κανονική έκφραση s .
- $~~r~~$ Η r αλλά μόνο όταν η τρέχουσα αρχική κατάσταση είναι η S .
- 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.

Υλοποίηση ΛΑ με το flex (ix)

► Μέρος B, 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 */  }
"(>(*([^\n]+|\n*+[^\n]))*\n*)" { /* nothing */  }

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

Υλοποίηση ΛΑ με το flex (x)

- Μέρος Γ, περιέχει κώδικα C

- Παράδειγμα

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

Υλοποίηση ΛΑ με το flex (xi)

- ▶ 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 (xii)

- ▶ 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 (xiii)

- ▶ 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 (xiv)

- ▶ Παράδειγμα: Αρίθμηση γραμμών (διόρθωση)
%x COMMENT

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

Κεφάλαιο 2: Τυπικές γλώσσες

(μέρος 2ο)

Γλώσσες χωρίς συμφραζόμενα (i)

- ▶ Γραμματικές χωρίς συμφραζόμενα: $A \rightarrow \alpha$
 - ▶ Σε κάθε παραγωγή ένα μη τερματικό σύμβολο αντικαθίσταται, βάσει ενός κανόνα
 - ▶ Πολλές διαφορετικές παραγωγές διαφέρουν μόνο στη σειρά των αντικαταστάσεων

Γλώσσες χωρίς συμφραζόμενα (i)

- ▶ Γραμματικές χωρίς συμφραζόμενα: $A \rightarrow \alpha$
 - ▶ Σε κάθε παραγωγή ένα μη τερματικό σύμβολο αντικαθίσταται, βάσει ενός κανόνα
 - ▶ Πολλές διαφορετικές παραγωγές διαφέρουν μόνο στη σειρά των αντικαταστάσεων
- ▶ Αριστερότερη / δεξιότερη παραγωγή (leftmost / rightmost derivation)

Γλώσσες χωρίς συμφραζόμενα (i)

- ▶ Γραμματικές χωρίς συμφραζόμενα: $A \rightarrow \alpha$
 - ▶ Σε κάθε παραγωγή ένα μη τερματικό σύμβολο αντικαθίσταται, βάσει ενός κανόνα
 - ▶ Πολλές διαφορετικές παραγωγές διαφέρουν μόνο στη σειρά των αντικαταστάσεων
- ▶ Αριστερότερη / δεξιότερη παραγωγή (leftmost / rightmost derivation)
- ▶ Συντακτικά δέντρα (parse trees)

Γλώσσες χωρίς συμφραζόμενα (ii)

$S \rightarrow aABc \mid \epsilon$
$A \rightarrow cSB \mid Ab$
$B \rightarrow bB \mid a$

Γλώσσες χωρίς συμφραζόμενα (ii)

► Μία παραγωγή

$$\begin{aligned} \boxed{S} &\Rightarrow aA\boxed{B}c \Rightarrow a\boxed{A}bBc \\ &\Rightarrow acS\boxed{B}bBc \Rightarrow ac\boxed{S}abBc \\ &\Rightarrow acab\boxed{B}c \Rightarrow acabac \end{aligned}$$

$$\begin{aligned} S &\rightarrow aABc \mid \epsilon \\ A &\rightarrow cSB \mid Ab \\ B &\rightarrow bB \mid a \end{aligned}$$

Γλώσσες χωρίς συμφραζόμενα (ii)

► Μία παραγωγή

$\boxed{S} \Rightarrow aA\boxed{B}c \Rightarrow a\boxed{A}bBc$
 $\Rightarrow acS\boxed{B}bBc \Rightarrow ac\boxed{S}abBc$
 $\Rightarrow acab\boxed{B}c \Rightarrow acabac$

$S \rightarrow aABc \mid \epsilon$
$A \rightarrow cSB \mid Ab$
$B \rightarrow bB \mid a$

► Αριστερότερη παραγωγή

$\boxed{S} \Rightarrow_L a\boxed{A}BC \Rightarrow_L a\boxed{A}bBc$
 $\Rightarrow_L ac\boxed{S}BbBc \Rightarrow_L ac\boxed{B}bBc$
 $\Rightarrow_L acab\boxed{B}c \Rightarrow_L acabac$

Γλώσσες χωρίς συμφραζόμενα (ii)

► Μία παραγωγή

$[S] \Rightarrow aA[B]c \Rightarrow a[A]bBc$
 $\Rightarrow acS[B]bBc \Rightarrow ac[S]abBc$
 $\Rightarrow acab[B]c \Rightarrow acabac$

$S \rightarrow aABc \mid \epsilon$
 $A \rightarrow cSB \mid Ab$
 $B \rightarrow bB \mid a$

► Αριστερότερη παραγωγή

$[S] \Rightarrow_L a[A]BC \Rightarrow_L a[A]bBc$
 $\Rightarrow_L ac[S]BbBc \Rightarrow_L ac[B]bBc$
 $\Rightarrow_L acab[B]c \Rightarrow_L acabac$

► Δεξιότερη παραγωγή

$[S] \Rightarrow_R aA[B]c \Rightarrow_R a[A]ac$
 $\Rightarrow_R a[A]bac \Rightarrow_R acS[B]bac$
 $\Rightarrow_R ac[S]abac \Rightarrow_R acabac$

Γλώσσες χωρίς συμφραζόμενα (ii)

► Μία παραγωγή

$[S] \Rightarrow aA[B]c \Rightarrow a[A]bBc$
 $\Rightarrow acS[B]bBc \Rightarrow ac[S]abBc$
 $\Rightarrow acab[B]c \Rightarrow acabac$

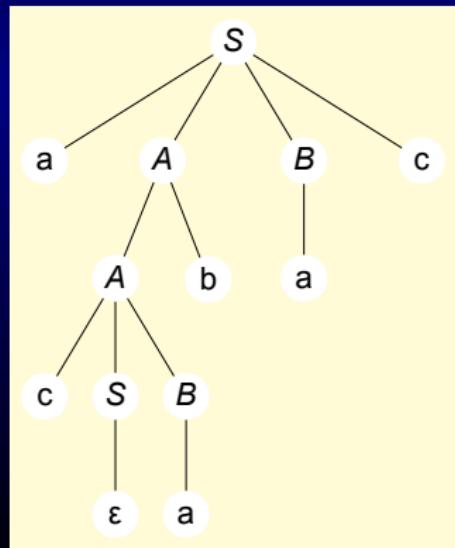
$S \rightarrow aABc \mid \epsilon$
 $A \rightarrow cSB \mid Ab$
 $B \rightarrow bB \mid a$

► Αριστερότερη παραγωγή

$[S] \Rightarrow_L a[A]BC \Rightarrow_L a[A]bBc$
 $\Rightarrow_L ac[S]BbBc \Rightarrow_L ac[B]bBc$
 $\Rightarrow_L acab[B]c \Rightarrow_L acabac$

► Δεξιότερη παραγωγή

$[S] \Rightarrow_R aA[B]c \Rightarrow_R a[A]ac$
 $\Rightarrow_R a[A]bac \Rightarrow_R acS[B]bac$
 $\Rightarrow_R acab[S]abac \Rightarrow_R acabac$



Διφορούμενες γραμματικές (i)

- Δύο γραμματικές είναι ισοδύναμες όταν παράγουν την ίδια γλώσσα.

Διφορούμενες γραμματικές (i)

- Δύο γραμματικές είναι ισοδύναμες όταν παράγουν την ίδια γλώσσα.
- Μια γραμματική είναι διφορούμενη (ambiguous) αν υπάρχουν δύο ή περισσότερα συντακτικά δέντρα για την ίδια παραγόμενη συμβολοσειρά

Διφορούμενες γραμματικές (i)

- ▶ Δύο γραμματικές είναι ισοδύναμες όταν παράγουν την ίδια γλώσσα.
- ▶ Μια γραμματική είναι διφορούμενη (*ambiguous*) αν υπάρχουν δύο ή περισσότερα συντακτικά δέντρα για την ίδια παραγόμενη συμβολοσειρά
- ▶ Γραμματικές και γλώσσες εγγενώς διφορούμενες (*inherently ambiguous*)

Διφορούμενες γραμματικές (i)

- ▶ Δύο γραμματικές είναι ισοδύναμες όταν παράγουν την ίδια γλώσσα.
- ▶ Μια γραμματική είναι διφορούμενη (*ambiguous*) αν υπάρχουν δύο ή περισσότερα συντακτικά δέντρα για την ίδια παραγόμενη συμβολοσειρά
- ▶ Γραμματικές και γλώσσες εγγενώς διφορούμενες (*inherently ambiguous*)
- ▶ Χρήση διφορούμενων γραμματικών στην περιγραφή της σύνταξης γλωσσών προγραμματισμού

Διφορούμενες γραμματικές (ii)

- ▶ Παράδειγμα: ζεχρέμαστο if (dangling if)

$stmt \rightarrow \text{if } cond\ stmt\ \text{else } stmt\ |\ \text{if } cond\ stmt\ |\ s1\ |\ s2$

$cond \rightarrow c1\ |\ c2$

Διφορούμενες γραμματικές (ii)

- ▶ Παράδειγμα: ζεχρέμαστο if (dangling if)

$stmt \rightarrow \text{if } cond\ stmt\ \text{else}\ stmt\ |\ \text{if}\ cond\ stmt\ |\ s1\ |\ s2$

$cond \rightarrow c1\ |\ c2$

- ▶ Διφορούμενο: σε ποιο if αντιστοιχεί το else;

if c1 if c2 s1 else s2

Διφορούμενες γραμματικές (ii)

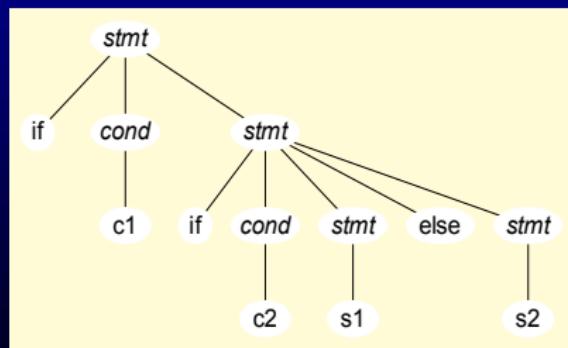
- Παράδειγμα: ζεχρέμαστο if (dangling if)

$stmt \rightarrow \text{if } cond \ stmt \ \text{else } stmt \mid \text{if } cond \ stmt \mid s1 \mid s2$

$cond \rightarrow c1 \mid c2$

- Διφορούμενο: σε ποιο if αντιστοιχεί το else;

if c1 if c2 s1 else s2



if c1 (if c2 s1 else s2)

Διφορούμενες γραμματικές (ii)

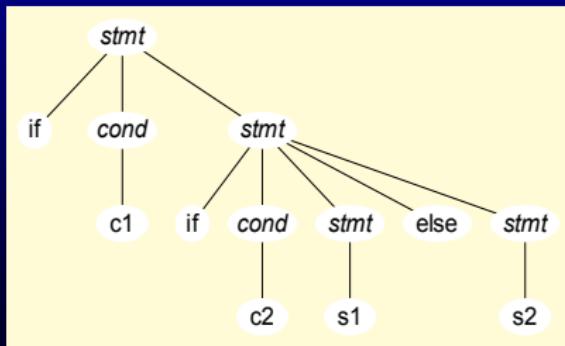
- Παράδειγμα: ζεχρέμαστο if (dangling if)

$stmt \rightarrow \text{if } cond \ stmt \ \text{else } stmt \mid \text{if } cond \ stmt \mid s1 \mid s2$

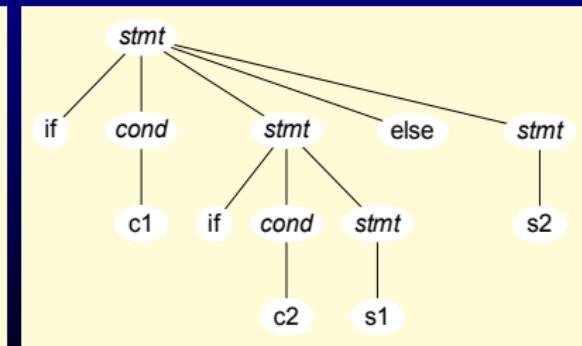
$cond \rightarrow c1 \mid c2$

- Διφορούμενο: σε ποιο if αντιστοιχεί το else;

if c1 if c2 s1 else s2



if c1 (if c2 s1 else s2)



if c1 (if c2 s1) else s2

Τρόποι παράστασης γραμματικών (i)

► Backus-Naur Form (BNF)

- Σύμβολο ::= στους κανόνες
- Μη τερματικά σύμβολα σε γωνιακές παρενθέσεις,
π.χ. $\langle \text{expr} \rangle$
- Σύμβολο | για διάζευξη

Τρόποι παράστασης γραμματικών (i)

► Backus-Naur Form (BNF)

- Σύμβολο ::= στους κανόνες
 - Μη τερματικά σύμβολα σε γωνιακές παρενθέσεις,
 $\pi.\chi.$ $\langle \text{expr} \rangle$
 - Σύμβολο | για διάζευξη
-

$\langle \text{unsigned-number} \rangle$::=	$\langle \text{integer-part} \rangle$	$\langle \text{dec-fraction} \rangle$	$\langle \text{exp-part} \rangle$																
$\langle \text{integer-part} \rangle$::=	$\langle \text{digit} \rangle$	$\langle \text{integer-part} \rangle$	$\langle \text{digit} \rangle$																
$\langle \text{dec-fraction} \rangle$::=	.	$\langle \text{integer-part} \rangle$	ϵ																
$\langle \text{exp-part} \rangle$::=	E	$\langle \text{sign} \rangle$	$\langle \text{integer-part} \rangle$																
			e	$\langle \text{sign} \rangle$																
			$\langle \text{integer-part} \rangle$	ϵ																
$\langle \text{sign} \rangle$::=	+		-																
				ϵ																
$\langle \text{digit} \rangle$::=	0		1		2		3		4		5		6		7		8		9

Τρόποι παράστασης γραμματικών (ii)

► Extended Backus-Naur Form (EBNF)

- ▶ Τερματικά σύμβολα σε εισαγωγικά
- ▶ Παρενθέσεις για ομαδοποίηση
- ▶ Αγκύλες για προαιρετικά τμήματα
- ▶ Σύμβολα * και + για επανάληψη

Τρόποι παράστασης γραμματικών (ii)

► Extended Backus-Naur Form (EBNF)

- ▶ Τερματικά σύμβολα σε εισαγωγικά
- ▶ Παρενθέσεις για ομαδοποίηση
- ▶ Αγκύλες για προαιρετικά τμήματα
- ▶ Σύμβολα * και + για επανάληψη

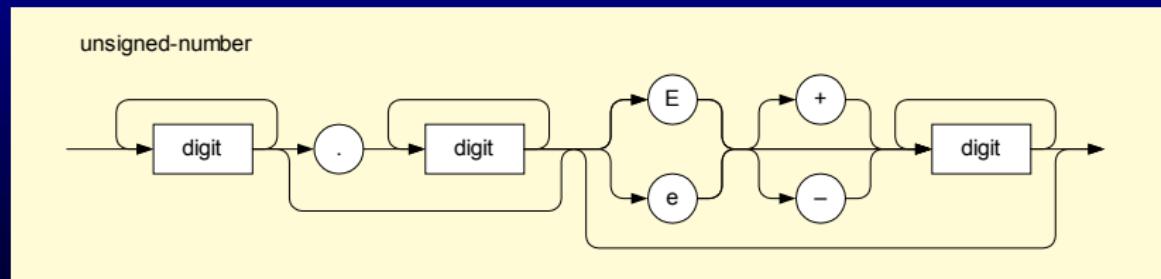
$$\langle \text{unsigned-number} \rangle ::= \langle \text{digit} \rangle^+ [\text{“.”} \langle \text{digit} \rangle^+] [(\text{“E”} \mid \text{“e”}) [\text{“+”} \mid \text{“-”}] \langle \text{digit} \rangle^+]$$
$$\langle \text{digit} \rangle ::= \text{“0”} \mid \text{“1”} \mid \text{“2”} \mid \text{“3”} \mid \text{“4”} \mid \text{“5”} \mid \text{“6”} \mid \text{“7”} \mid \text{“8”} \mid \text{“9”}$$

Τρόποι παράστασης γραμματικών (iii)

- ▶ Συντακτικά διαγράμματα
 - ▶ Τερματικά σύμβολα σε οβάλ
 - ▶ Μη τερματικά σύμβολα σε ορθογώνια
 - ▶ Διαδοχή συμβόλων (παράθεση) με βέλη

Τρόποι παράστασης γραμματικών (iii)

- ▶ Συντακτικά διαγράμματα
 - ▶ Τερματικά σύμβολα σε οβάλ
 - ▶ Μη τερματικά σύμβολα σε ορθογώνια
 - ▶ Διαδοχή συμβόλων (παράθεση) με βέλη



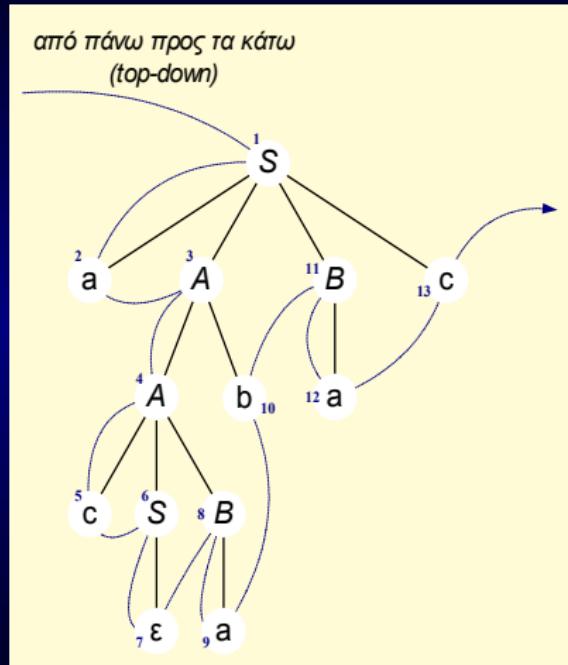
Κεφάλαιο 4:

Συντακτική ανάλυση

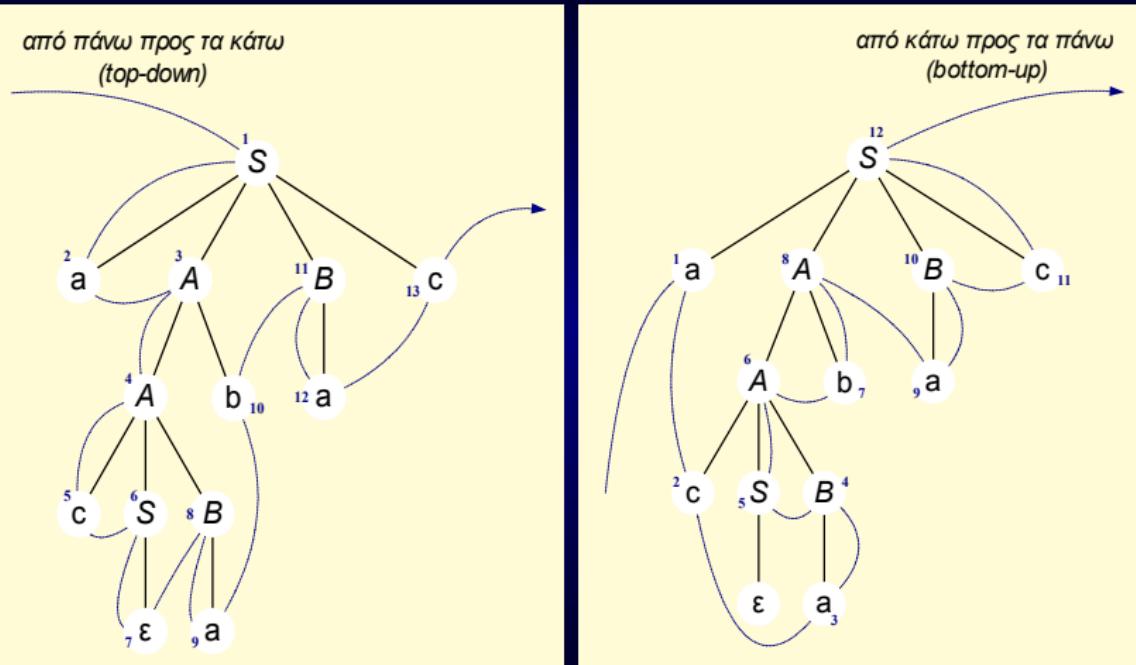
Συντακτική ανάλυση

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

Top-down και bottom-up



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

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
- ▶ $\text{Av } \alpha \Rightarrow^* a\beta \quad \text{τότε } a \in \text{FIRST}(\alpha)$
- ▶ $\text{Av } \alpha \Rightarrow^* \epsilon \quad \text{τότε } \epsilon \in \text{FIRST}(\alpha)$

Βοηθητικές έννοιες (ii)

► Σύνολα FOLLOW

- ▶ Έστω μη τερματικό σύμβολο A
- ▶ Το σύνολο $\text{FOLLOW}(A) \subseteq T \cup \{\text{EOF}\}$ 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 $\text{EOF} \in \text{FOLLOW}(A)$

Βοηθητικές έννοιες (ii)

► Σύνολα FOLLOW

- ▶ Έστω μη τερματικό σύμβολο A
- ▶ Το σύνολο $\text{FOLLOW}(A) \subseteq T \cup \{\text{EOF}\}$ 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 $\text{EOF} \in \text{FOLLOW}(A)$
- ▶ Αν $S \Rightarrow^* \alpha A a \beta$ τότε $a \in \text{FOLLOW}(A)$
- ▶ Αν $S \Rightarrow^* \alpha A$ τότε $\text{EOF} \in \text{FOLLOW}(A)$

Υπολογισμός FIRST (i)

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

Υπολογισμός FIRST (ii)

► Παράδειγμα

$$\text{FIRST}(E) = \{\text{id}, ()\}$$

$$\text{FIRST}(T) = \{\text{id}, ()\}$$

$$\text{FIRST}(F) = \{\text{id}, ()\}$$

$$\text{FIRST}(E') = \{+, \epsilon\}$$

$$\text{FIRST}(T') = \{*, \epsilon\}$$

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

Υπολογισμός FOLLOW (i)

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

Υπολογισμός FOLLOW (ii)

► Παράδειγμα

$$\text{FOLLOW}(E) = \{), \text{EOF} \}$$

$$\text{FOLLOW}(T) = \{ +,), \text{EOF} \}$$

$$\text{FOLLOW}(F) = \{ *, +,), \text{EOF} \}$$

$$\text{FOLLOW}(E') = \{), \text{EOF} \}$$

$$\text{FOLLOW}(T') = \{ +,), \text{EOF} \}$$

$$\text{FIRST}(E') = \{ +, \epsilon \}$$

$$\text{FIRST}(T') = \{ *, \epsilon \}$$

$$E \rightarrow T E'$$

$$E' \rightarrow \epsilon$$

$$E' \rightarrow + T E'$$

$$T \rightarrow F T'$$

$$T' \rightarrow \epsilon$$

$$T' \rightarrow * F T'$$

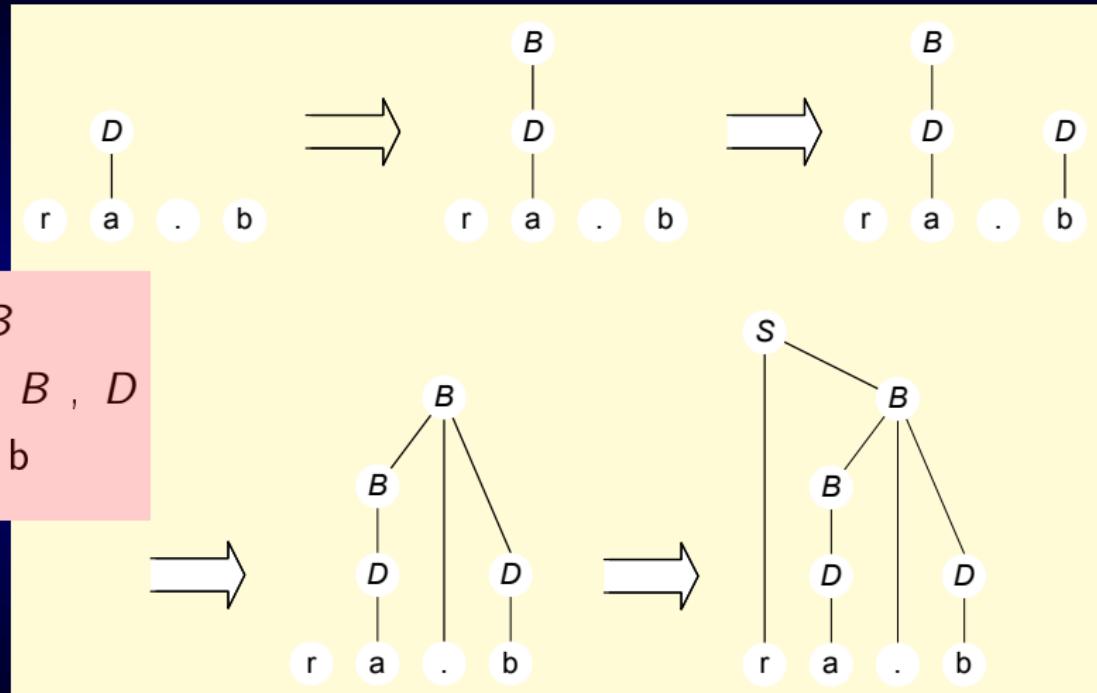
$$F \rightarrow (E)$$

$$F \rightarrow \text{id}$$

ΣA 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

ΣA bottom-up (ii)



ΣA 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

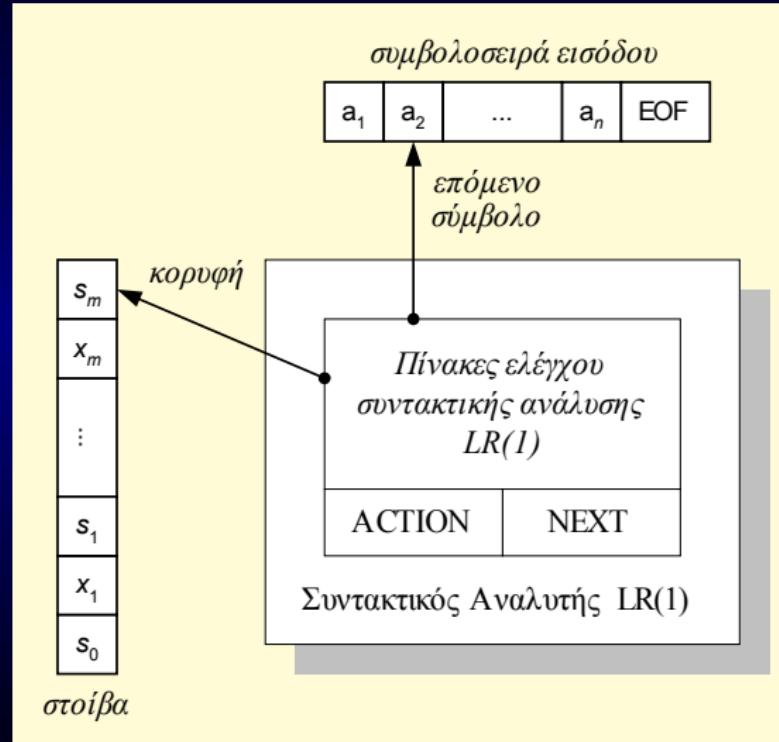
ΣA bottom-up (iv)

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

$$\begin{aligned} S &\rightarrow r B \\ B &\rightarrow D \mid B , D \\ D &\rightarrow a \mid b \end{aligned}$$

ΣA 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 A, 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 ΣA με το bison (iv)

► Μέρος A, 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 A, 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:

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

Υλοποίηση ΣΑ με το bison (vii)

► 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 (viii)

- ▶ 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 γεννητορα ενδιάμεσου κώδικα

Υλοποίηση ΣA 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

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow F$

$T \rightarrow T * F$

$F \rightarrow (E)$

$F \rightarrow \text{num}$

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

Στο μέρος Γ

Υλοποίηση ΣA με το bison (xv)

- ▶ Παράδειγμα (σ υνέχεια)

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

- ▶ Αυτοματοποίηση της μεταγλώττισης του ΣA

```
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]+ { yyval = 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 shift
- ▶ reduce-reduce: ο πρώτος κανόνας

ΣA 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 λεξικών 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

$$\begin{array}{lcl} A \rightarrow \alpha_1 | \dots | \alpha_n & \Rightarrow & A \rightarrow \alpha_1 | \dots | \alpha_n \\ B \rightarrow \beta_1 A \beta_2 & & B \rightarrow \beta_1 \alpha_1 \beta_2 | \dots | \beta_1 \alpha_n \beta_2 \end{array}$$

► Αριστερή παραγοντοποίηση

$$A \rightarrow \alpha \beta_1 | \dots | \alpha \beta_n \Rightarrow \begin{array}{l} A \rightarrow \alpha B \\ B \rightarrow \beta_1 | \dots | \beta_n \end{array}$$

► Απαλοιφή άμεσης αριστερής αναδρομής

$$\begin{array}{lcl} A \rightarrow A\alpha_1 | \dots | A\alpha_n | \beta_1 | \dots | \beta_m \\ \Rightarrow \qquad \qquad \qquad A \rightarrow \beta_1 B | \dots | \beta_m B \\ \qquad \qquad \qquad B \rightarrow \alpha_1 B | \dots | \alpha_n B | \epsilon \end{array}$$

ΣA αναδρομικής κατάβασης

$A \rightarrow \alpha_1 \mid \dots \mid \alpha_n$

μετατρέπεται σε κώδικα της μορφής:

if $token \in \text{FIRST}(\alpha_1)$ **then**

κώδικας για την αναγνώριση της α_1

...

else if $token \in \text{FIRST}(\alpha_n)$ **then**

κώδικας για την αναγνώριση της α_n

else if $\epsilon \notin \text{FIRST}(\alpha_1) \cup \dots \cup \text{FIRST}(\alpha_n)$ **then**

συντακτικό σφάλμα

else if $token \notin \text{FOLLOW}(A)$ **then**

συντακτικό σφάλμα

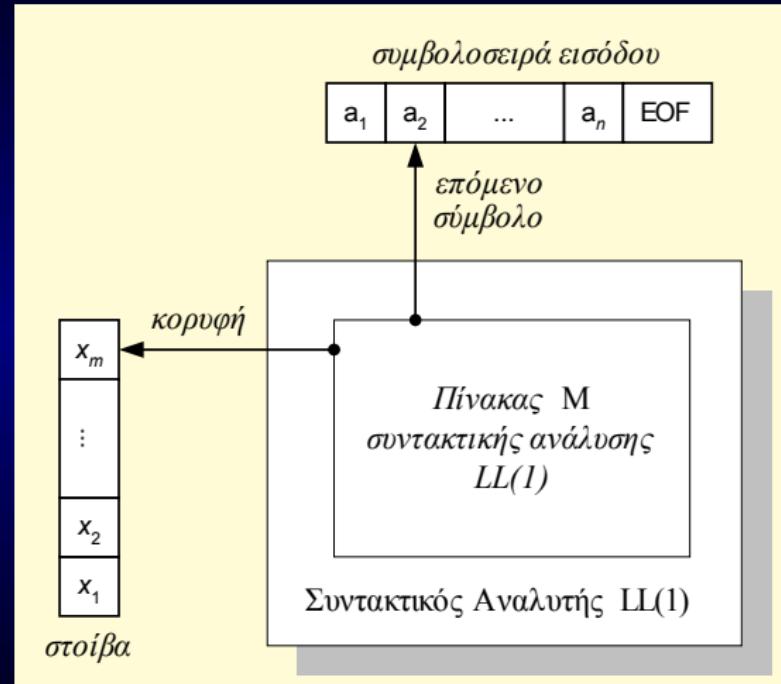
end if

ΣA $LL(1)$ (i)

- ▶ Χρησιμοποιούν μια **στοίβα** όπου το ποθετούν 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

ΣA $LL(1)$ (ii)

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



Kατασκευή ΣΑ LL(1)

$E \rightarrow T E'$	$\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{\text{id}, ()\}$
$E' \rightarrow + T E' \epsilon$	$\text{FIRST}(E') = \{+, \epsilon\}$
$T \rightarrow F T'$	$\text{FIRST}(T') = \{*, \epsilon\}$
$T' \rightarrow * F T' \epsilon$	$\text{FOLLOW}(E) = \text{FOLLOW}(E') = \{(), \text{EOF}\}$
$F \rightarrow (E) \text{id}$	$\text{FOLLOW}(T) = \text{FOLLOW}(T') = \{+, (), \text{EOF}\}$ $\text{FOLLOW}(F) = \{*, +, (), \text{EOF}\}$

	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	$F \rightarrow \text{id}$			$F \rightarrow (E)$		

Λειτουργία ΣA $LL(1)$

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	$F \rightarrow \text{id}$
3	$E' T' \text{id}$	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	$F \rightarrow \text{id}$
9	$E' T' \text{id}$	id * id EOF	
10	$E' T'$	* id EOF	$T' \rightarrow * F T'$
11	$E' T' F *$	* id EOF	
12	$E' T' F$	id EOF	$F \rightarrow \text{id}$
13	$E' T' \text{id}$	id EOF	
14	$E' T'$	EOF	$T' \rightarrow \epsilon$
15	E'	EOF	$E' \rightarrow \epsilon$
16	ϵ	EOF	αναγνώριση

Κεφάλαιο 2: Τυπικές γλώσσες

(μέρος 3ο)

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)

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

Σημασιολογικοί χανόνες

Κεφάλαιο 5: Πίνακας συμβόλων

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 στοιβας (stack variables)
 - ▶ Στατικές μεταβλητές (static variables)
- ▶ Εμβέλεια (scope)
- ▶ Ορατότητα (visibility)
- ▶ Διάρκεια ζωής (lifetime)

Περιεχόμενα πίνακα συμβόλων

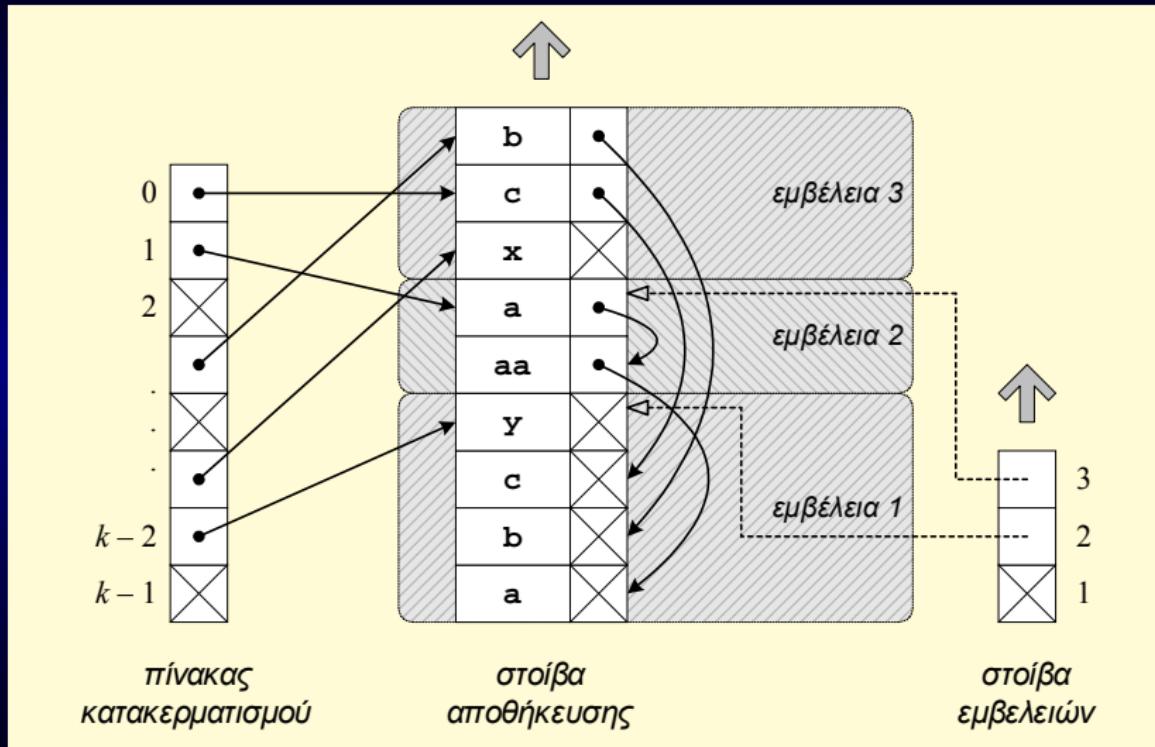
- ▶ Εμβέλεια (έμμεσα)
- ▶ Ορατότητα (έμμεσα)
- ▶ Διάρκεια ζωής
- ▶ Τύπος
- ▶ Θέση (διεύθυνση μνήμης, καταχωρητής, ...)
- ▶ Αριθμός παραμέτρων υποπρογράμματος
- ▶ Τύπος παραμέτρων υποπρογράμματος
- ▶ Τρόπος περάσματος παραμέτρων υποπρογράμματος
- ▶ Τύπος αποτελέσματος συνάρτησης

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



Κεφάλαιο 6:

Σημασιολογική ανάλυση

Σύνταξη και σημασιολογία

- ▶ Σύνταξη: μορφή και δομή των καλώς σχηματισμένων προγραμμάτων

Σύνταξη και σημασιολογία

- ▶ **Σύνταξη:** μορφή και δομή των χαλώς σχηματισμένων προγραμμάτων
- ▶ **Σημασιολογία:** ερμηνεία των χαλώς σχηματισμένων προγραμμάτων

Σύνταξη και σημασιολογία

- ▶ **Σύνταξη:** μορφή και δομή των χαλώς σχηματισμένων προγραμμάτων
- ▶ **Σημασιολογία:** ερμηνεία των χαλώς σχηματισμένων προγραμμάτων
 - ▶ **Στατική σημασιολογία:** εντοπισμός σημασιολογικών σφαλμάτων κατά τη διάρκεια της μεταγλώττισης

Σύνταξη και σημασιολογία

- ▶ **Σύνταξη:** μορφή και δομή των χαλώς σχηματισμένων προγραμμάτων
- ▶ **Σημασιολογία:** ερμηνεία των χαλώς 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

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

Στατική σημασιολογία (i)

- ▶ Περιβάλλοντα τύπων

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

- ▶ Σχέση αντιστοίχισης τύπων

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

Στατική σημασιολογία (i)

- ▶ Περιβάλλοντα τύπων

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

- ▶ Σχέση αντιστοίχισης τύπων

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

- ▶ Κανόνες τύπων

$$\frac{\Gamma \vdash E_1 : \text{integer} \quad \Gamma \vdash E_2 : \text{integer}}{\Gamma \vdash E_1 + E_2 : \text{integer}}$$

Στατική σημασιολογία (ii)

► Παραγωγές τύπων

$$\frac{\Gamma_1 \vdash i : \text{integer} \quad \Gamma_1 \vdash 1 : \text{integer}}{\Gamma_1 \vdash i+1 : \text{integer}} \qquad \Gamma_1 \vdash x : \text{real}$$

$$\Gamma_1 \vdash (i+1)*x : \text{real}$$

Στατική σημασιολογία (ii)

- ▶ Παραγωγές τύπων

$$\frac{\Gamma_1 \vdash i : \text{integer} \quad \Gamma_1 \vdash 1 : \text{integer}}{\Gamma_1 \vdash i+1 : \text{integer}} \qquad \frac{}{\Gamma_1 \vdash x : \text{real}}$$

$$\Gamma_1 \vdash (i+1)*x : \text{real}$$

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

$$\frac{\Gamma \vdash E : \text{boolean} \quad \Gamma \vdash S : \text{stmt}}{\Gamma \vdash \text{while } E \text{ do } S : \text{stmt}}$$

Δυναμική σημασιολογία (i)

- ▶ Λειτουργική σημασιολογία (operational semantics)

Δυναμική σημασιολογία (i)

- ▶ Λειτουργική σημασιολογία (operational semantics)
⇒ ακολουθία υπολογιστικών βημάτων

Δυναμική σημασιολογία (i)

- ▶ Λειτουργική σημασιολογία (operational semantics)
⇒ ακολουθία υπολογιστικών βημάτων
- ▶ Δηλωτική σημασιολογία (denotational semantics)

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

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)

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 $I=E$

Δυναμική σημασιολογία (ii)

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

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

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

$$\mathcal{C}\llbracket I=E \rrbracket(s) = s[I \mapsto \mathcal{E}\llbracket E \rrbracket(s)]$$

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 (i)

- Έλεγχος τύπων

Σημασιολογικός έλεγχος (i)

► Έλεγχος τύπων

π.χ. οι τελεστές εφαρμόζονται σε κατάλληλα τελούμενα

Σημασιολογικός έλεγχος (i)

- ▶ Έλεγχος τύπων
 - π.χ. οι τελεστές εφαρμόζονται σε κατάλληλα τελούμενα
- ▶ Έλεγχος ροής

Σημασιολογικός έλεγχος (i)

► Έλεγχος τύπων

π.χ. οι τελεστές εφαρμόζονται σε κατάλληλα τελούμενα

► Έλεγχος ροής

π.χ. όχι continue έξω από βρόχο

(C)

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)

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

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

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

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)

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

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 $\not\Rightarrow$ απουσία σφαλμάτων εκτέλεσης

Σημασιολογικός έλεγχος (ii)

- ▶ Απουσία σημασιολογικών σφαλμάτων $\not\Rightarrow$ απουσία σφαλμάτων εκτέλεσης
- ▶ Μερικές φορές όμως είναι δυνατό να προβλεφθούν σφάλματα εκτέλεσης

Σημασιολογικός έλεγχος (ii)

- ▶ Απουσία σημασιολογικών σφαλμάτων $\not\Rightarrow$ 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; } ;
```

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");
} ;
```

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");
} ;
```

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;
```

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;
```

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

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

Υποσύνολα τύπων και υποτύποι

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

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

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

Υποσύνολα τύπων και υποτύποι

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

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

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

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

Υποσύνολα τύπων και υποτύποι

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

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

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

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

► Υποτύποι

⇒ 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;
}
```

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

Αντιστοίχιση τύπων

- ▶ Στατική
 - ▶ ρητή
 - ▶ έμμεση
- ▶ Δυναμική
 - x := 5; (* x : integer *)
 - x := "hello"; (* τώρα ομως x : string *)

Eξαγωγή 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 *)
```

Eξαγωγή 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 *)
```

Eξαγωγή 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
```

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 \geq 0 **and** i \leq 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

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

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

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

Ενδιάμεση γλώσσα (i)

► Τετράδες (quadruples)

n : op, x, y, z

► Παραδειγμα:

$$b*b - 4*a*c$$

1: *, b, b, \$1

2: *, 4, a, \$2

3: *, \$2, c, \$3

4: -, \$1, \$3, \$4

Ενδιάμεση γλώσσα (ii)

- Τριάδες (triples)

n : op, x, y

Ενδιάμεση γλώσσα (ii)

- ▶ Τριάδες (triples)

n : op, x, y

- ▶ Παράδειγμα:

$$b*b - 4*a*c$$

- 1: *, b, b
- 2: *, 4, a
- 3: *, (2), c
- 4: -, (1), (3)

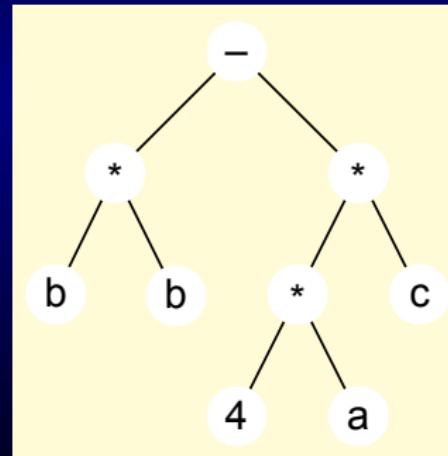
Ενδιάμεση γλώσσα (iii)

- ▶ Αφηρημένα συντακτικά δέντρα
(abstract syntax trees)

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)

Ενδιάμεση γλώσσα (iv)

- ▶ Προθεματικός και επιθεματικός χώδικας (prefix/postfix code)
- ▶ Παράδειγμα:
 $b*b-4*a*c$
 — * b b * * 4 a c προθεματικός
 b b * 4 a * c * — επιθεματικός

Ενδιάμεση γλώσσα (v)

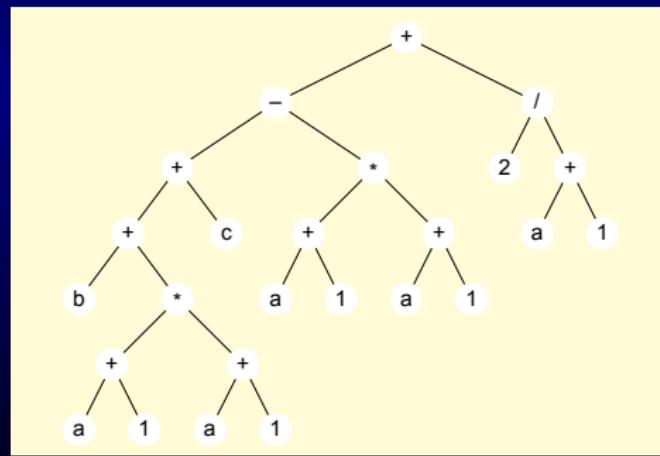
- ▶ Κατευθυνόμενοι ακυκλικοί γράφοι
(directed acyclic graphs)

Ενδιάμεση γλώσσα (v)

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

Ενδιάμεση γλώσσα (v)

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

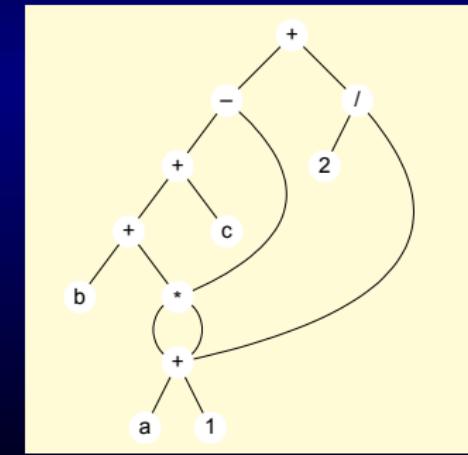
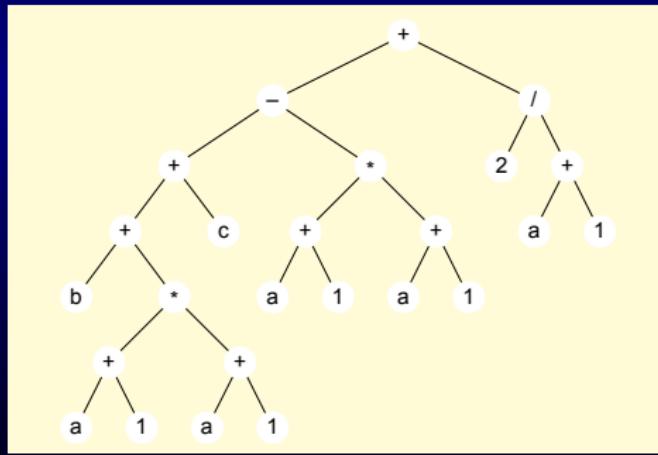


Ενδιάμεση γλώσσα (v)

- 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, χάποια τελούμενα ενδεχομένως παραλείπονται

Τελούμενα (i)

- ▶ Σταθερά
 - ▶ ακέραια, πραγματική, λογική
 - ▶ χαρακτήρας, συμβολοσειρά, nil
- ▶ Όνομα
 - ▶ μεταβλητή, παραμέτρος, υποπρόγραμμα
- ▶ Προσωρινή μεταβλητή: \$n
- ▶ Αποτέλεσμα συνάρτησης: \$\$
- ▶ Αποδεικτοδότηση: [x] x απλό τελούμενο
- ▶ Διεύθυνση: {x} x απλό τελούμενο

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)

$T\varepsilon\lambda\varepsilon\sigma\tau\acute{\varepsilon}\varsigma$ (i)

- ▶ **unit**, $I, -, -$
- ▶ **endu**, $I, -, -$
αρχή και τέλος δομικής μονάδας
- ▶ op, x, y, z $op \in \{+, -, *, /, \%\}$
 $z := x \ op \ y$
- ▶ $:=, x, -, z$
 $z := x$
- ▶ **array**, x, y, z
 $z := \eta$ διεύθυνση του στοιχείου $x[y]$

Τελεστές (ii)

- ▶ op, x, y, z $op \in \{=, <\!, >, <, >=, <=\}$
αν x op y τότε πήγαινε στην τετράδα z
- ▶ ifb, $x, -, z$
αν η λογική τιμή x είναι αληθής τότε πήγαινε στην τετράδα z
- ▶ jump, $-, -, z$
πήγαινε στην τετράδα z
- ▶ label, $I, -, -$
jumpl, $-, -, I$
ορισμός ετικέτας και άλμα προς αυτήν

Τελεστές (iii)

- ▶ `call, −, −, I`
κάλεσε τη δομική μονάδα I
- ▶ `par, x, m, −`
πέρασε την πραγματική παράμετρο x με τρόπο περάσματος m
- ▶ `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, \dots, l_n)

Συνένωση των λιστών ετικετών τετράδων $l_1 \dots l_n$

- ▶ BACKPATCH(l, z)

Αντικαθιστά σε όλες τις τετράδες που περιέχονται στην l την άγνωστη ετικέτα τετράδας με τη z (**backpatching**)

Αριθμητικές εκφράσεις

► Ακέραιες σταθερές

$\langle \text{r-value} \rangle ::= \langle \text{integer-const} \rangle \{ P_1 \}$

$P_1 ::= \{ \langle \text{r-value} \rangle . PLACE = \langle \text{integer-const} \rangle; \}$

Αριθμητικές εκφράσεις

► Ακέραιες σταθερές

$\langle \text{r-value} \rangle ::= \langle \text{integer-const} \rangle \{ P_1 \}$

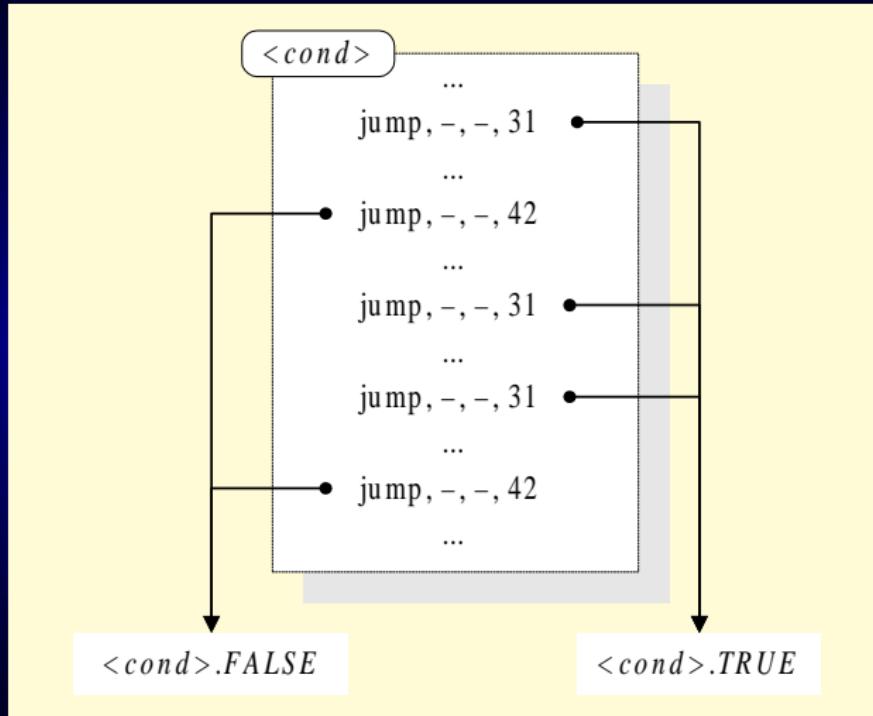
$P_1 ::= \{ \langle \text{r-value} \rangle.PLACE = \langle \text{integer-const} \rangle; \}$

► Τελεστές με δύο τελούμενα

$\langle \text{r-value} \rangle ::= \langle \text{expr} \rangle \langle \text{binop} \rangle \langle \text{expr} \rangle \{ P_{14} \}$

$P_{14} ::= \{ W = \text{NEWTEMP}(\langle \text{r-value} \rangle.TYPE);$
 $\quad \text{GENQUAD}(\langle \text{binop} \rangle.NAME,$
 $\quad \quad \langle \text{expr} \rangle^1.PLACE,$
 $\quad \quad \langle \text{expr} \rangle^2.PLACE, W);$
 $\quad \langle \text{r-value} \rangle.PLACE = W; \}$

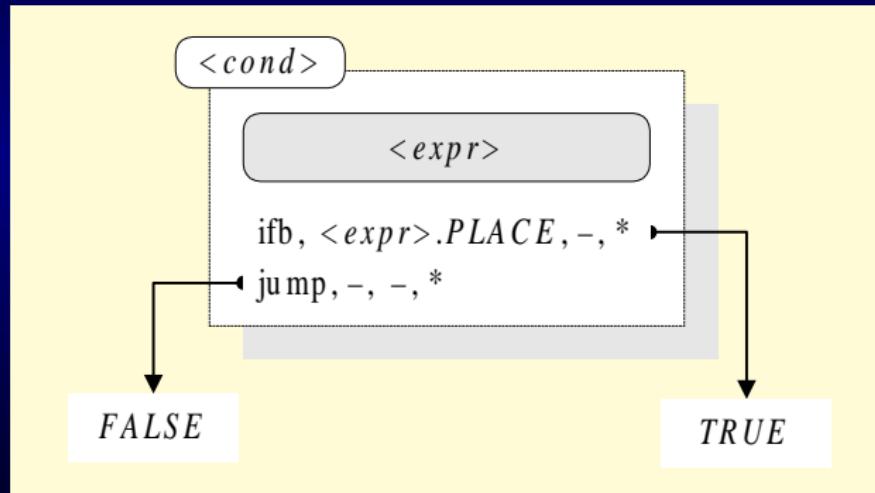
Λογικές εκφράσεις (i)



Λογικές εκφράσεις (ii)

- Λογικές εκφράσεις σε συμβολισμό 0/1

$\langle \text{cond} \rangle ::= \langle \text{expr} \rangle$



Λογικές εκφράσεις (iii)

► Λογικές εκφράσεις σε συμβολισμό 0/1

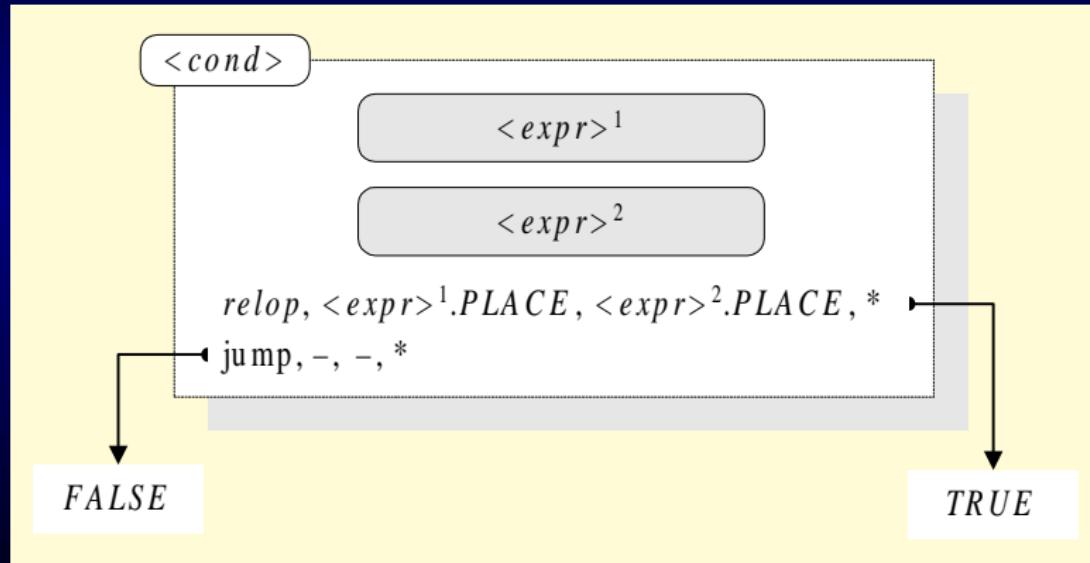
$\langle \text{cond} \rangle ::= \langle \text{expr} \rangle \{ P_{21} \}$

$P_{21} : \{ \langle \text{cond} \rangle . \text{TRUE} = \text{MAKELIST}(\text{NEXTQUAD}());$
 $\qquad \text{GENQUAD}(\text{ifb}, \langle \text{expr} \rangle . \text{PLACE}, -, *);$
 $\qquad \langle \text{cond} \rangle . \text{FALSE} = \text{MAKELIST}(\text{NEXTQUAD}());$
 $\qquad \text{GENQUAD}(\text{jump}, -, -, *); \}$

Λογικές εκφράσεις (iv)

► Τελεστές σύγχρισης

$\langle \text{cond} \rangle ::= \langle \text{expr} \rangle^1 \ \langle \text{relop} \rangle \ \langle \text{expr} \rangle^2$



Λογικές εκφράσεις (v)

► Τελεστές σύγχρισης

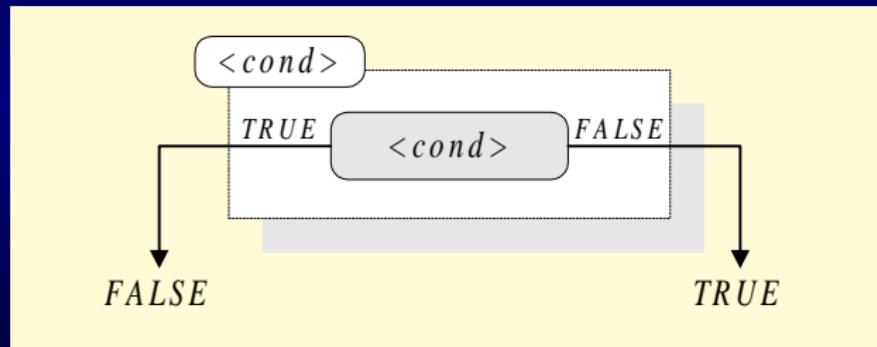
$\langle \text{cond} \rangle ::= \langle \text{expr} \rangle^1 \langle \text{relop} \rangle \langle \text{expr} \rangle^2 \{ P_{23} \}$

$P_{23} ::= \{ \langle \text{cond} \rangle.\text{TRUE} = \text{MAKELIST}(\text{NEXTQUAD}());$
 $\qquad \qquad \qquad \text{GENQUAD}(\langle \text{relop} \rangle.\text{NAME},$
 $\qquad \qquad \qquad \langle \text{expr} \rangle^1.\text{PLACE},$
 $\qquad \qquad \qquad \langle \text{expr} \rangle^2.\text{PLACE}, *);$
 $\langle \text{cond} \rangle.\text{FALSE} = \text{MAKELIST}(\text{NEXTQUAD}());$
 $\qquad \qquad \qquad \text{GENQUAD}(\text{jump}, -, -, *); \}$

Λογικές εκφράσεις (vi)

► Άρνηση

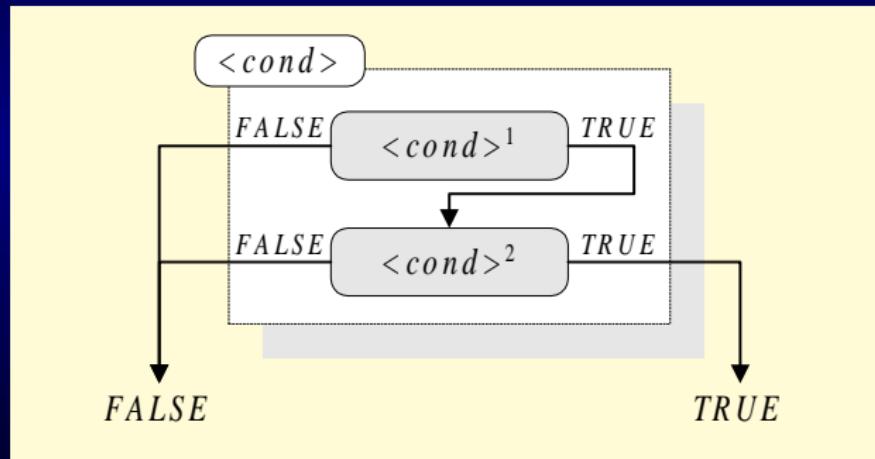
$\langle \text{cond} \rangle ::= \text{"not"} \ \langle \text{cond} \rangle$



Λογικές εκφράσεις (vii)

► Σύζευξη

$\langle \text{cond} \rangle ::= \langle \text{cond} \rangle^1 \text{ ``and'' } \langle \text{cond} \rangle^2$



Λογικές εκφράσεις (viii)

► Σύζευξη

$\langle \text{cond} \rangle ::= \langle \text{cond} \rangle_1 \text{ ``and'' } \{ P_{25} \} \; \langle \text{cond} \rangle_2 \; \{ P_{26} \}$

$P_{25} ::= \{ \text{BACKPATCH}(\langle \text{cond} \rangle^1.\text{TRUE}, \text{NEXTQUAD}()); \}$

$P_{26} ::= \{ \langle \text{cond} \rangle.\text{FALSE} = \text{MERGE}(\langle \text{cond} \rangle^1.\text{FALSE},$
 $\qquad\qquad\qquad \langle \text{cond} \rangle^2.\text{FALSE});$
 $\qquad\qquad\qquad \langle \text{cond} \rangle.\text{TRUE} = \langle \text{cond} \rangle^2.\text{TRUE}; \}$

Απλές εντολές

► Κενή εντολή

$\langle \text{stmt} \rangle ::= \epsilon \{ P_{29} \}$

$P_{29} : \{ \langle \text{stmt} \rangle.NEXT = \text{EMPTYLIST}(); \}$

Απλές εντολές

► Κενή εντολή

$\langle \text{stmt} \rangle ::= \epsilon \{ P_{29} \}$

$P_{29} : \{ \langle \text{stmt} \rangle.NEXT = \text{EMPTYLIST}(); \}$

► Εντολή ανάθεσης

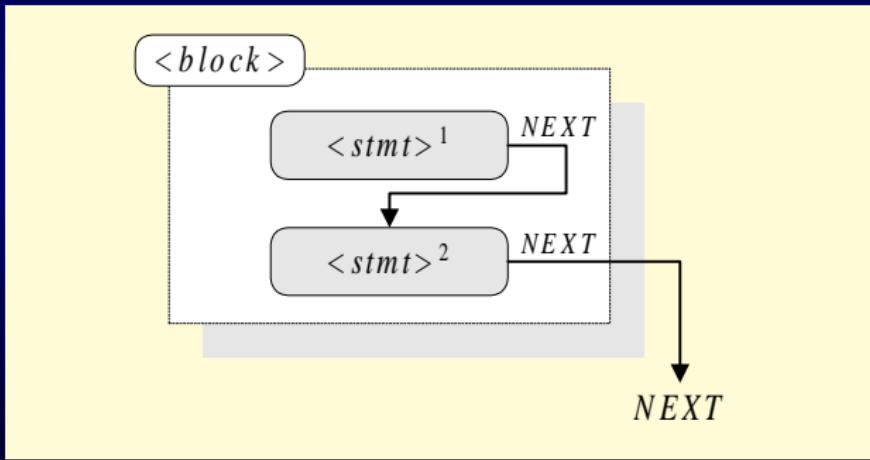
$\langle \text{stmt} \rangle ::= \langle \text{l-value} \rangle “:=” \langle \text{expr} \rangle \{ P_{30} \}$

$P_{30} : \{ \text{GENQUAD}(“:=”, \langle \text{expr} \rangle.PLACE, -, \langle \text{l-value} \rangle.PLACE); \langle \text{stmt} \rangle.NEXT = \text{EMPTYLIST}(); \}$

Σύνθετη εντολή (i)

$\langle \text{stmt} \rangle ::= \langle \text{block} \rangle$

$\langle \text{block} \rangle ::= \text{"begin"} \; \langle \text{stmt} \rangle \; (\; ; \; \langle \text{stmt} \rangle \;)^* \; \text{"end"}$



Σύνθετη εντολή (ii)

$\langle \text{stmt} \rangle ::= \langle \text{block} \rangle \{ P_{34} \}$

$P_{34} ::= \{ \langle \text{stmt} \rangle.NEXT = \langle \text{block} \rangle.NEXT; \}$

$\langle \text{block} \rangle ::= \text{"begin"} \langle \text{stmt} \rangle^1 \{ P_{35} \}$
 $\quad (\text{";"} \{ P_{36} \} \langle \text{stmt} \rangle^2 \{ P_{37} \})^* \text{"end"} \{ P_{38} \}$

$P_{35} ::= \{ L = \langle \text{stmt} \rangle^1.NEXT; \}$

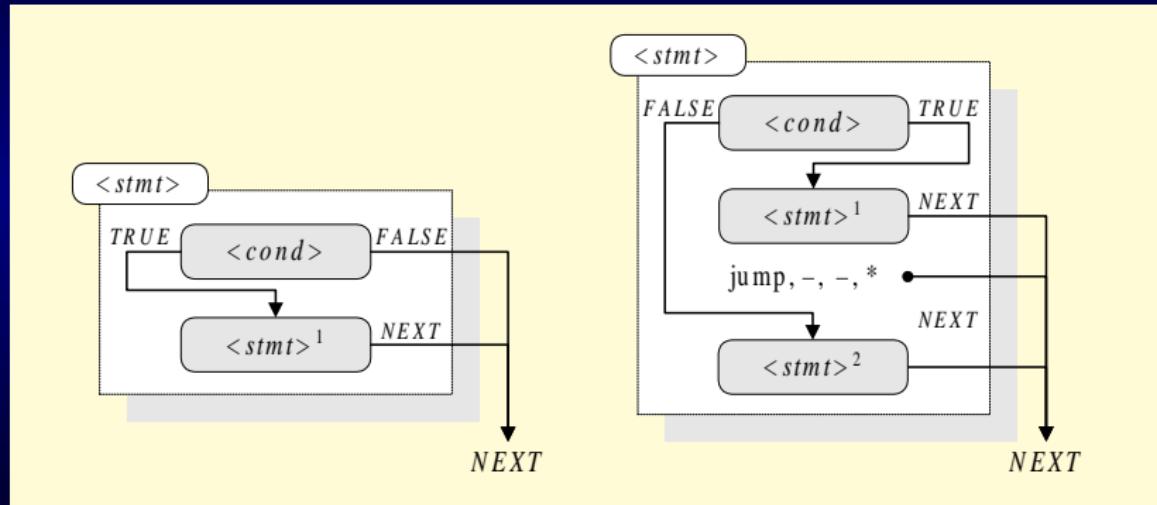
$P_{36} ::= \{ \text{BACKPATCH}(L, \text{NEXTQUAD}()); \}$

$P_{37} ::= \{ L = \langle \text{stmt} \rangle^2.NEXT; \}$

$P_{38} ::= \{ \langle \text{block} \rangle.NEXT = L; \}$

$E \nu \tau o \lambda \eta$ if (i)

$\langle \text{stmt} \rangle ::= \text{"if"} \; \langle \text{cond} \rangle \; \text{"then"} \; \langle \text{stmt} \rangle \; [\; \text{"else"} \; \langle \text{stmt} \rangle \;]$



$E \nu \tau o \lambda \eta$ if (ii)

$\langle \text{stmt} \rangle ::= \text{"if"} \; \langle \text{cond} \rangle \; \{ P_{39} \} \; \text{"then"} \; \langle \text{stmt} \rangle^1$
[“else” { P_{40} } $\langle \text{stmt} \rangle^2 \{ P_{41} \}$] { P_{42} }

$P_{39} : \{ \text{BACKPATCH}(\langle \text{cond} \rangle.\text{TRUE}, \text{NEXTQUAD}());$
 $L_1 = \langle \text{cond} \rangle.\text{FALSE};$
 $L_2 = \text{EMPTYLIST}(); \}$

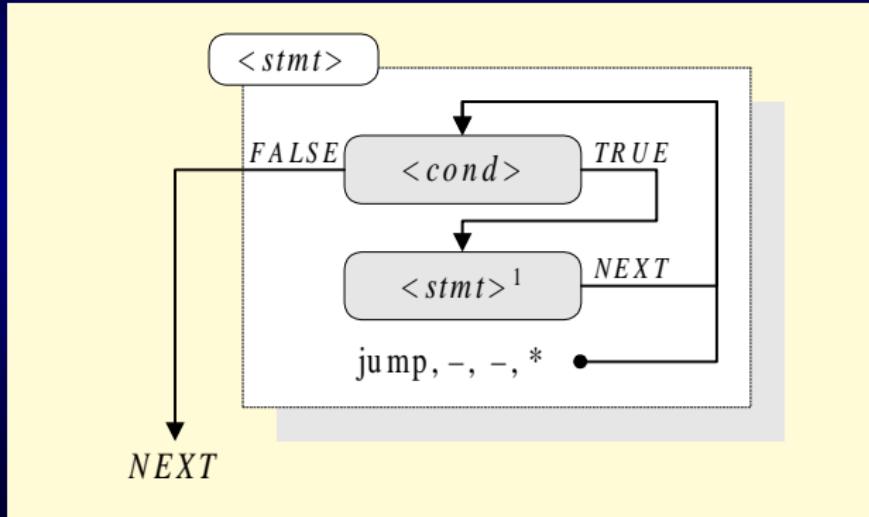
$P_{40} : \{ L_1 = \text{MAKELIST}(\text{NEXTQUAD}());$
 $\text{GENQUAD}(\text{jump}, -, -, *);$
 $\text{BACKPATCH}(\langle \text{cond} \rangle.\text{FALSE}, \text{NEXTQUAD}()); \}$

$P_{41} : \{ L_2 = \langle \text{stmt} \rangle^2.\text{NEXT}; \}$

$P_{42} : \{ \langle \text{stmt} \rangle.\text{NEXT} = \text{MERGE}(L_1, \langle \text{stmt} \rangle^1.\text{NEXT}, L_2); \}$

Εντολή while (i)

$\langle \text{stmt} \rangle ::= \text{"while"} \; \langle \text{cond} \rangle \; \text{"do"} \; \langle \text{stmt} \rangle$



$E \nu \tau o \lambda \dot{\eta}$ while (ii)

$\langle \text{stmt} \rangle ::= \text{"while"} \{ P_{43} \} \langle \text{cond} \rangle \text{"do"} \{ P_{44} \} \langle \text{stmt} \rangle^1 \{ P_{45} \}$

$P_{43} ::= \{ Q = \text{NEXTQUAD}(); \}$

$P_{44} ::= \{ \text{BACKPATCH}(\langle \text{cond} \rangle.\text{TRUE}, \text{NEXTQUAD}()); \}$

$P_{45} ::= \{ \text{BACKPATCH}(\langle \text{stmt} \rangle^1.\text{NEXT}, Q);$
 $\quad \text{GENQUAD}(\text{jump}, -, -, Q);$
 $\quad \langle \text{stmt} \rangle.\text{NEXT} = \langle \text{cond} \rangle.\text{FALSE}; \}$

Κλήση υποπρογραμμάτων (i)

```
 $\langle \text{call} \rangle ::= \langle \text{id} \rangle \text{ ``(``} [ \langle \text{expr} \rangle \text{ ( ``,``} \langle \text{expr} \rangle \text{ )}* ] \text{ ``)''}$ 
 $\langle \text{r-value} \rangle ::= \langle \text{call} \rangle$ 
 $\langle \text{stmt} \rangle ::= \langle \text{call} \rangle$ 
```

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

Kλήση υποπρογραμμάτων (ii)

$\langle \text{call} \rangle ::= \langle \text{id} \rangle \text{ ``(``} \{ P_{46} \} [\langle \text{expr} \rangle^1 \{ P_{47} \} \\ (\text{ ``,”} \langle \text{expr} \rangle^2 \{ P_{48} \})^*] \text{ ``)’’} \{ P_{49} \}$

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

$P_{47} : \{ \text{ GENQUAD(`par’,} \langle \text{expr} \rangle^1.PLACE, \\ \text{ PARAMMODE}(\langle \text{id} \rangle, N), -); \\ N = N + 1; \}$

$P_{48} : \{ \text{ GENQUAD(`par’,} \langle \text{expr} \rangle^2.PLACE, \\ \text{ PARAMMODE}(\langle \text{id} \rangle, N), -); \\ N = N + 1; \}$

Κλήση υποπρογραμμάτων (iii)

$\langle \text{call} \rangle ::= \langle \text{id} \rangle \text{ ``(``} \{ P_{46} \} [\langle \text{expr} \rangle^1 \{ P_{47} \} \\ (\text{ ``,”} \langle \text{expr} \rangle^2 \{ P_{48} \})^*] \text{ ``)’’} \{ P_{49} \}$

(συνέχεια)

$P_{49} : \{ \text{ if } (\text{ISFUNCTION}(\langle \text{id} \rangle)) \{$

$W = \text{NEWTEMP}(\text{FUNCRESULT}(\langle \text{id} \rangle));$

$\text{GENQUAD}(\text{par}, \text{RET}, W, -);$

$\langle \text{call} \rangle.\text{PLACE} = W;$

$\}$

$\text{GENQUAD}(\text{call}, -, -, \langle \text{id} \rangle); \}$

Κλήση υποπρογραμμάτων (iv)

► Κλήση συνάρτησης

$\langle \text{r-value} \rangle ::= \langle \text{call} \rangle \{ P_{50} \}$

$P_{50} ::= \{ \langle \text{r-value} \rangle . PLACE = \langle \text{call} \rangle . PLACE; \}$

Κλήση υποπρογραμμάτων (iv)

► Κλήση συνάρτησης

$\langle \text{r-value} \rangle ::= \langle \text{call} \rangle \{ P_{50} \}$

$P_{50} ::= \{ \langle \text{r-value} \rangle . PLACE = \langle \text{call} \rangle . PLACE; \}$

► Κλήση διαδικασίας

$\langle \text{stmt} \rangle ::= \langle \text{call} \rangle \{ P_{51} \}$

$P_{51} ::= \{ \langle \text{stmt} \rangle . NEXT = \text{EMPTYLIST}(); \}$

Κλήση υποπρογραμμάτων (v)

► Επιστροφή από υποπρόγραμμα

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

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

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

Κλήση υποπρογραμμάτων (v)

► Επιστροφή από υποπρόγραμμα

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

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

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

► Δήλωση υποπρογράμματος

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

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

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

Παράδειγμα (i)

```
procedure quicksort (var a : array of integer;
                     m, n : integer);
  var i, j, temp : integer;
begin
  if n <= m then return;
  i := m; j := n;
  while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
    if i <= j then begin
      temp := a[i]; a[i] := a[j]; a[j] := temp;
      i := i+1; j := j-1
    end
  end;
  quicksort(a, m, j); quicksort(a, i, n)
end;
```

Παράδειγμα (ii)

```
procedure quicksort (var a : array of integer;  
                     m, n : integer);  
  var i, j, temp : integer;  
begin  
  if n <= m then return;  
  i := m; j := n;
```

Παράδειγμα (ii)

```
procedure quicksort (var a : array of integer;  
                     m, n : integer);  
  var i, j, temp : integer;  
begin  
  if n <= m then return;  
  i := m; j := n;
```

1: unit, quicksort, -, -

Παράδειγμα (ii)

```
procedure quicksort (var a : array of integer;  
                     m, n : integer);  
  var i, j, temp : integer;  
begin  
  if n <= m then return;  
  i := m; j := n;
```

1: unit, quicksort, -, -

2: <=, n, m, *

Παράδειγμα (ii)

```
procedure quicksort (var a : array of integer;  
                     m, n : integer);  
  var i, j, temp : integer;  
begin  
  if n <= m then return;  
  i := m; j := n;
```

1: unit, quicksort, -, -

2: <=, n, m, *

3: jump, -, -, *

Παράδειγμα (ii)

```
procedure quicksort (var a : array of integer;  
                     m, n : integer);  
  var i, j, temp : integer;  
begin  
  if n <= m then return;  
  i := m; j := n;
```

1: unit, quicksort, -, -

2: <=, n, m, *

3: jump, -, -, *

4: ret, -, -, -

Παράδειγμα (ii)

```
procedure quicksort (var a : array of integer;  
                     m, n : integer);  
  var i, j, temp : integer;  
begin  
  if n <= m then return;  
  i := m; j := n;
```

1: unit, quicksort, -, -

2: <=, n, m, 4

3: jump, -, -, *

4: ret, -, -, -

Παράδειγμα (ii)

```
procedure quicksort (var a : array of integer;  
                     m, n : integer);  
  var i, j, temp : integer;  
begin  
  if n <= m then return;  
  i := m; j := n;
```

- 1: unit, quicksort, -, -
- 2: <=, n, m, 4
- 3: jump, -, -, 5
- 4: ret, -, -, -

Παράδειγμα (ii)

```
procedure quicksort (var a : array of integer;  
                     m, n : integer);  
  var i, j, temp : integer;  
begin  
  if n <= m then return;  
  i := m; j := n;
```

- 1: unit, quicksort, -, -
- 2: <=, n, m, 4
- 3: jump, -, -, 5
- 4: ret, -, -, -
- 5: :=, m, -, i

Παράδειγμα (ii)

```
procedure quicksort (var a : array of integer;  
                     m, n : integer);  
  var i, j, temp : integer;  
begin  
  if n <= m then return;  
  i := m; j := n;
```

- 1: unit, quicksort, -, -
- 2: <=, n, m, 4
- 3: jump, -, -, 5
- 4: ret, -, -, -
- 5: :=, m, -, i
- 6: :=, n, -, j

Παράδειγμα (iii)

```
while i <= j do begin  
    while a[i] < a[(m+n) div 2] do i := i+1;  
    while a[j] > a[(m+n) div 2] do j := j-1;
```

Παράδειγμα (iii)

```
while i <= j do begin  
    while a[i] < a[(m+n) div 2] do i := i+1;  
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, *

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, *

8: jump, -, -, *

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9

8: jump, -, -, *

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9

8: jump, -, -, *

9: array, a, i, \$1

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- 7: <=, i, j, 9
- 8: jump, -, -, *
- 9: array, a, i, \$1
- 10: +, m, n, \$2

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
```

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
12: array, a, $3, $4
```

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- 7: <=, i, j, 9
- 8: jump, -, -, *
- 9: array, a, i, \$1
- 10: +, m, n, \$2
- 11: /, \$2, 2, \$3
- 12: array, a, \$3, \$4
- 13: <, [\$1], [\$4], *

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
12: array, a, $3, $4
13: <, [$1], [$4], *
14: jump, -, -, *
```

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
12: array, a, $3, $4
13: <, [$1], [$4], 15
14: jump, -, -, *
```

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- 7: <=, i, j, 9
- 8: jump, -, -, *
- 9: array, a, i, \$1
- 10: +, m, n, \$2
- 11: /, \$2, 2, \$3
- 12: array, a, \$3, \$4
- 13: <, [\$1], [\$4], 15
- 14: jump, -, -, *
- 15: +, i, 1, \$5

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
12: array, a, $3, $4
13: <, [$1], [$4], 15
14: jump, -, -, *
15: +, i, 1, $5
16: :=, $5, -, i
```

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- | | |
|-------------------------|-------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | |
| 9: array, a, i, \$1 | |
| 10: +, m, n, \$2 | |
| 11: /, \$2, 2, \$3 | |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, * | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- | | |
|-------------------------|-------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | |
| 9: array, a, i, \$1 | |
| 10: +, m, n, \$2 | |
| 11: /, \$2, 2, \$3 | |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- | | |
|-------------------------|----------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | |
| 10: +, m, n, \$2 | |
| 11: /, \$2, 2, \$3 | |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- | | |
|-------------------------|----------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | |
| 11: /, \$2, 2, \$3 | |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- | | |
|-------------------------|----------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- | | |
|-------------------------|------------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9	17: jump, -, -, 9
8: jump, -, -, *	18: array, a, j, \$6
9: array, a, i, \$1	19: +, m, n, \$7
10: +, m, n, \$2	20: /, \$7, 2, \$8
11: /, \$2, 2, \$3	21: array, a, \$8, \$9
12: array, a, \$3, \$4	22: >, [\$6], [\$9], *
13: <, [\$1], [\$4], 15	
14: jump, -, -, 18	
15: +, i, 1, \$5	
16: :=, \$5, -, i	

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9	17: jump, -, -, 9
8: jump, -, -, *	18: array, a, j, \$6
9: array, a, i, \$1	19: +, m, n, \$7
10: +, m, n, \$2	20: /, \$7, 2, \$8
11: /, \$2, 2, \$3	21: array, a, \$8, \$9
12: array, a, \$3, \$4	22: >, [\$6], [\$9], *
13: <, [\$1], [\$4], 15	23: jump, -, -, *
14: jump, -, -, 18	
15: +, i, 1, \$5	
16: :=, \$5, -, i	

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

- | | |
|-------------------------|-------------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | 22: >, [\$6], [\$9], 24 |
| 13: <, [\$1], [\$4], 15 | 23: jump, -, -, * |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9	17: jump, -, -, 9
8: jump, -, -, *	18: array, a, j, \$6
9: array, a, i, \$1	19: +, m, n, \$7
10: +, m, n, \$2	20: /, \$7, 2, \$8
11: /, \$2, 2, \$3	21: array, a, \$8, \$9
12: array, a, \$3, \$4	22: >, [\$6], [\$9], 24
13: <, [\$1], [\$4], 15	23: jump, -, -, *
14: jump, -, -, 18	24: -, j, 1, \$10
15: +, i, 1, \$5	
16: :=, \$5, -, i	

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9	17: jump, -, -, 9
8: jump, -, -, *	18: array, a, j, \$6
9: array, a, i, \$1	19: +, m, n, \$7
10: +, m, n, \$2	20: /, \$7, 2, \$8
11: /, \$2, 2, \$3	21: array, a, \$8, \$9
12: array, a, \$3, \$4	22: >, [\$6], [\$9], 24
13: <, [\$1], [\$4], 15	23: jump, -, -, *
14: jump, -, -, 18	24: -, j, 1, \$10
15: +, i, 1, \$5	25: :=, \$10, -, j
16: :=, \$5, -, i	

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9	17: jump, -, -, 9
8: jump, -, -, *	18: array, a, j, \$6
9: array, a, i, \$1	19: +, m, n, \$7
10: +, m, n, \$2	20: /, \$7, 2, \$8
11: /, \$2, 2, \$3	21: array, a, \$8, \$9
12: array, a, \$3, \$4	22: >, [\$6], [\$9], 24
13: <, [\$1], [\$4], 15	23: jump, -, -, *
14: jump, -, -, 18	24: -, j, 1, \$10
15: +, i, 1, \$5	25: :=, \$10, -, j
16: :=, \$5, -, i	26: jump, -, -, 18

Παράδειγμα (iii)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9	17: jump, -, -, 9
8: jump, -, -, *	18: array, a, j, \$6
9: array, a, i, \$1	19: +, m, n, \$7
10: +, m, n, \$2	20: /, \$7, 2, \$8
11: /, \$2, 2, \$3	21: array, a, \$8, \$9
12: array, a, \$3, \$4	22: >, [\$6], [\$9], 24
13: <, [\$1], [\$4], 15	23: jump, -, -, 27
14: jump, -, -, 18	24: -, j, 1, \$10
15: +, i, 1, \$5	25: :=, \$10, -, j
16: :=, \$5, -, i	26: jump, -, -, 18

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, *

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, *

28: jump, -, -, *

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

- 27: <=, i, j, 29
- 28: jump, -, -, *
- 29: array, a, i, \$11
- 30: :=, [\$11], -, temp
- 31: array, a, i, \$12

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

- 27: <=, i, j, 29
- 28: jump, -, -, *
- 29: array, a, i, \$11
- 30: :=, [\$11], -, temp
- 31: array, a, i, \$12
- 32: array, a, j, \$13

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

- 27: <=, i, j, 29
- 28: jump, -, -, *
- 29: array, a, i, \$11
- 30: :=, [\$11], -, temp
- 31: array, a, i, \$12
- 32: array, a, j, \$13
- 33: :=, [\$13], -, [\$12]

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

34: array, a, j, \$14

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

36: +, i, 1, \$15

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

36: +, i, 1, \$15

37: :=, \$15, -, i

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

36: +, i, 1, \$15

37: :=, \$15, -, i

38: -, j, 1, \$16

Παράδειγμα (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

36: +, i, 1, \$15

37: :=, \$15, -, i

38: -, j, 1, \$16

39: :=, \$16, -, j

Παράδειγμα (v)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
    if i <= j then begin
        temp := a[i]; a[i] := a[j]; a[j] := temp;
        i := i+1; j := j-1
    end
end;
```

Παράδειγμα (v)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
    if i <= j then begin
        temp := a[i]; a[i] := a[j]; a[j] := temp;
        i := i+1; j := j-1
    end
end;
```

7: <=, i, j, 9

8: jump, -, -, *

Παράδειγμα (v)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
    if i <= j then begin
        temp := a[i]; a[i] := a[j]; a[j] := temp;
        i := i+1; j := j-1
    end
end;
```

7: <=, i, j, 9

8: jump, -, -, *

...

27: <=, i, j, 29

28: jump, -, -, *

Παράδειγμα (v)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
    if i <= j then begin
        temp := a[i]; a[i] := a[j]; a[j] := temp;
        i := i+1; j := j-1
    end
end;
```

7: <=, i, j, 9

8: jump, -, -, *

...

27: <=, i, j, 29

28: jump, -, -, *

...

40: jump, -, -, 7

Παράδειγμα (v)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
    if i <= j then begin
        temp := a[i]; a[i] := a[j]; a[j] := temp;
        i := i+1; j := j-1
    end
end;
```

7: <=, i, j, 9

8: jump, -, -, *

...

27: <=, i, j, 29

28: jump, -, -, 7

...

40: jump, -, -, 7

Παράδειγμα (v)

```
while i <= j do begin
    while a[i] < a[(m+n) div 2] do i := i+1;
    while a[j] > a[(m+n) div 2] do j := j-1;
    if i <= j then begin
        temp := a[i]; a[i] := a[j]; a[j] := temp;
        i := i+1; j := j-1
    end
end;
```

7: <=, i, j, 9

8: jump, -, -, 41

...

27: <=, i, j, 29

28: jump, -, -, 7

...

40: jump, -, -, 7

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, -

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, -

42: par, m, V, -

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, -

42: par, m, V, -

43: par, j, V, -

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

- 41: par, a, R, -
- 42: par, m, V, -
- 43: par, j, V, -
- 44: call, -, -, quicksort

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

- 41: par, a, R, -
- 42: par, m, V, -
- 43: par, j, V, -
- 44: call, -, -, quicksort
- 45: par, a, R, -

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

- 41: par, a, R, -
- 42: par, m, V, -
- 43: par, j, V, -
- 44: call, -, -, quicksort
- 45: par, a, R, -
- 46: par, i, V, -

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

- 41: par, a, R, -
- 42: par, m, V, -
- 43: par, j, V, -
- 44: call, -, -, quicksort
- 45: par, a, R, -
- 46: par, i, V, -
- 47: par, n, V, -

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

- 41: par, a, R, -
- 42: par, m, V, -
- 43: par, j, V, -
- 44: call, -, -, quicksort
- 45: par, a, R, -
- 46: par, i, V, -
- 47: par, n, V, -
- 48: call, -, -, quicksort

Παράδειγμα (vi)

```
quicksort(a, m, j); quicksort(a, i, n)
end;
```

- 41: par, a, R, -
- 42: par, m, V, -
- 43: par, j, V, -
- 44: call, -, -, quicksort
- 45: par, a, R, -
- 46: par, i, V, -
- 47: par, n, V, -
- 48: call, -, -, quicksort
- 49: endu, quicksort, -, -

Κεφάλαιο 9: Τελικός κώδικας

Τελικός κώδικας (i)

- ▶ Από θεωρητικής άποψης, το πρόβλημα της κατασκευής βέλτιστου τελικού κώδικα δεν έχει λύση (undecidable)

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, με γέθους 16 bit
 - ▶ Γενικής φύσης: ax, bx, cx, dx
⇒ σε ζεύγη των 8 bit: ah, al, x.lπ.
 - ▶ 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, cwd
- ▶ 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 $(dx, ax) := ax * op$
- ▶ **idiv** op $ax := (dx, ax) \text{ div } op$
 $dx := (dx, ax) \text{ mod } op$
- ▶ **cwd** επέκταση προσήμου του ax στον dx
- ▶ **cmp** op_1, op_2 σύγκρινε τα op_1 και op_2
ενημέρωσε τις σημαίες

Λογικές πράξεις

- ▶ **and** op_1, op_2 $op_1 := op_1 \text{ and } op_2$
- ▶ **or** op_1, op_2 $op_1 := op_1 \text{ or } op_2$
- ▶ **not** op $op := \text{not } op$
- ▶ **xor** op_1, op_2 $op_1 := op_1 \text{ xor } op_2$
- ▶ **test** op_1, op_2 $op_1 \text{ and } op_2$
ενημέρωσε τις σημαίες

Εντολές áλματος

- ▶ *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 στοιβας

- ▶ push op πρόσθεση στη στοίβα
 $sp := sp - 2, [sp] := op$
 - ▶ pop op αφαίρεση από τη στοίβα
 $op := [sp], sp := sp + 2$
- ⇒ Η στοίβα αυξάνει προς τα κάτω, δηλαδή προς μικρότερες διευθύνσεις

Εντολές υποπρογραμμάτων

► call *address*

κλήση

$sp := sp - 2, [sp] := ip, ip := address$

► ret

επιστροφή

$ip := [sp], sp := sp + 2$

⇒ 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
- ▶ **fld source** (load integer & push)
fld word ptr [bp - 2]
- ▶ **fstp destination** (pop & store real)
fld tbyte ptr [bp - 10]
- ▶ **fistp destination** (pop & store integer)
fld word ptr [bp - 2]

Evτολέσ x87 FPU (ii)

- ▶ **faddp** ST(1), ST(0) $ST(1) := ST(1) + ST(0) \& \text{pop}$
- ▶ **fsubp** ST(1), ST(0) $ST(1) := ST(1) - ST(0) \& \text{pop}$
- ▶ **fmulp** ST(1), ST(0) $ST(1) := ST(1) * ST(0) \& \text{pop}$
- ▶ **fdivp** ST(1), ST(0) $ST(1) := ST(1)/ST(0) \& \text{pop}$
 $ST(0) := -ST(0)$
- ▶ **fchs**
- ▶ **fcompp** $ST(1) \stackrel{\geq}{\ll} ST(0) \& \text{pop both}$
- ▶ **fstsw** *destination* (store x87 FPU flags)
 - fstsw ax**
 - fstsw word ptr [bp - 2]**

Διαχείριση μνήμης (i)

- ▶ Δομή ενοτήτων (block structure)
 - ▶ Μη τοπικά δεδομένα

Διαχείριση μνήμης (i)

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

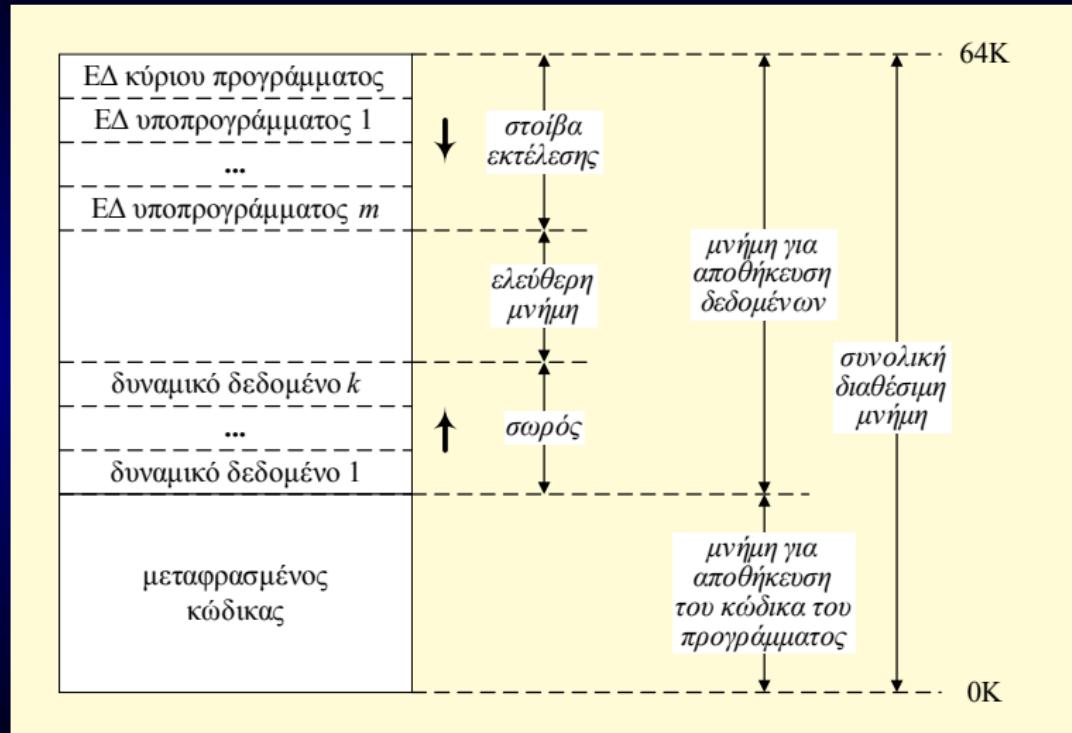
Διαχείριση μνήμης (ii)

	Παράμετρος 1	Παράμετρος 1	αρχή
	Παράμετρος 2	Παράμετρος 2	
...	
bp+8	Παράμετρος n	Παράμετρος n	
bp+6	Διεύθυνση αποτελέσματος	Διεύθυνση αποτελέσματος	
bp+4	Σύνδεσμος προσπέλασης	Διεύθυνση επιστροφής	
bp+2	Διεύθυνση επιστροφής	Προηγούμενο display	
bp	Προηγούμενο bp	Προηγούμενο bp	
bp-2	Τοπική μεταβλητή 1	Τοπική μεταβλητή 1	
bp-4	Τοπική μεταβλητή 2	Τοπική μεταβλητή 2	
...	
	Τοπική μεταβλητή m	Τοπική μεταβλητή m	
	Προσωρινή μεταβλητή 1	Προσωρινή μεταβλητή 1	
	Προσωρινή μεταβλητή 2	Προσωρινή μεταβλητή 2	
	
	Προσωρινή μεταβλητή k	Προσωρινή μεταβλητή k	

a) Σύνδεσμοι προσπέλασης

b) Πίνακας δεικτών

Διαχείριση μνήμης (iii)



Προσπέλαση ονομάτων

- ▶ Τοπικά: $[bp + offset]$
- ▶ Μη τοπικά: $[si + offset]$
 - ⇒ 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
⇒ 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 $n_p \geq n_a$
⇒ 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)

⇒ Τρόπος υλοποίησης καθενός

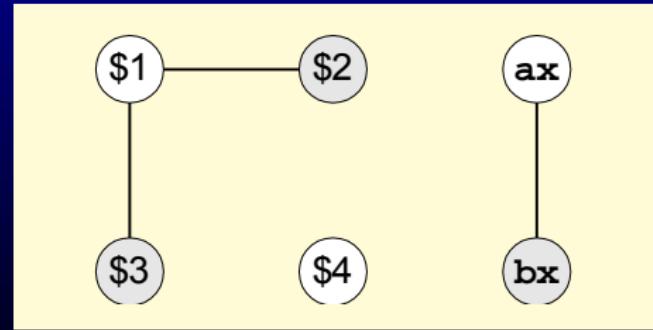
Δέσμευση καταχωρητών (i)

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



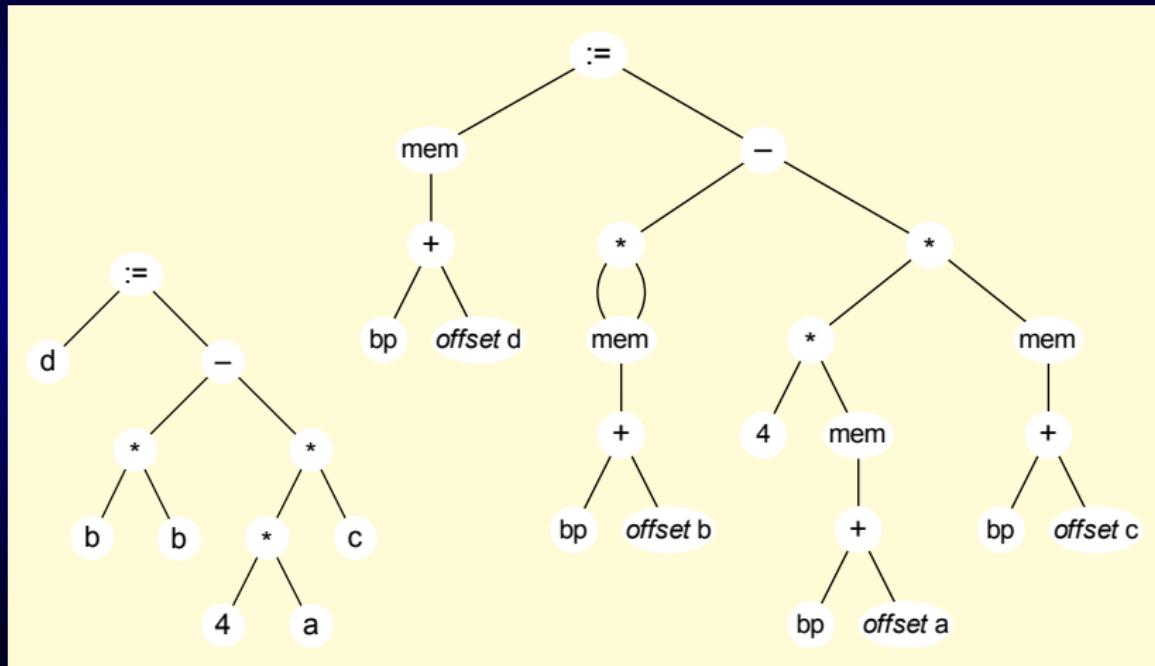
Επιλογή εντολών (i)

- ▶ 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:

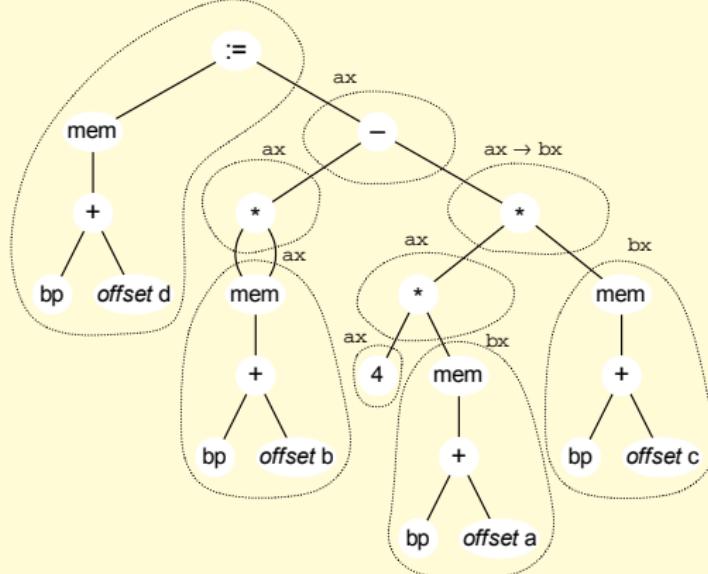
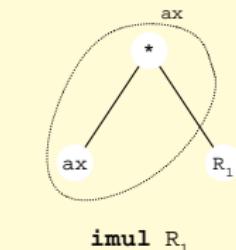
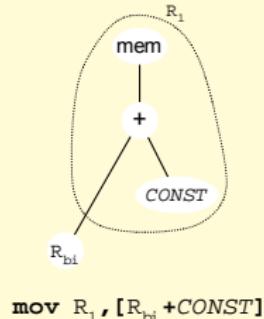
$d := b*b - 4*a*c$



Επιλογή εντολών (iii)

► Παράδειγμα:

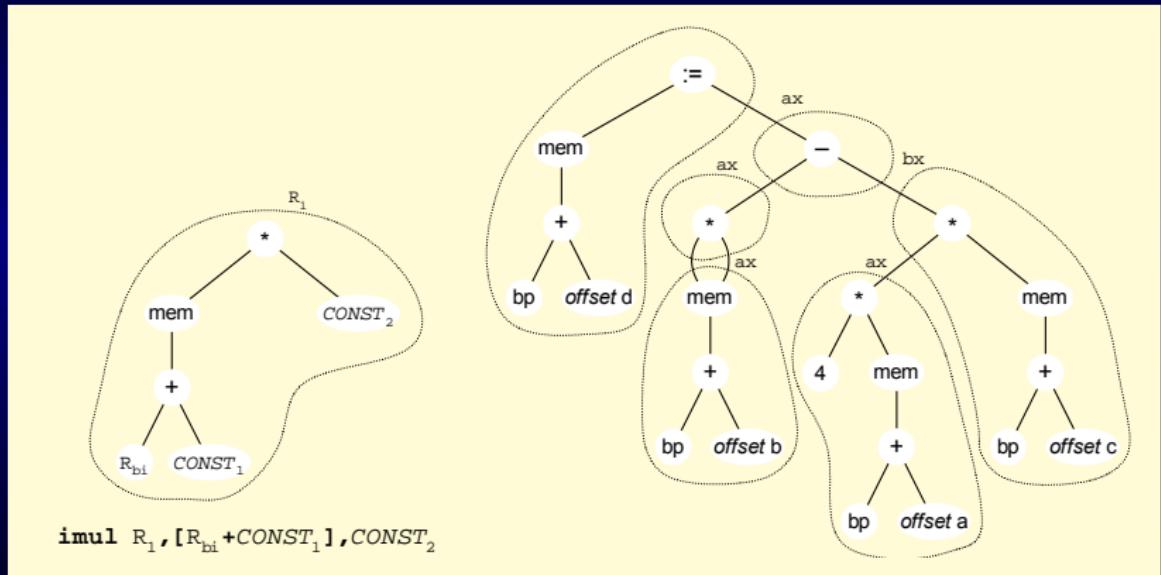
tiling με εντολές του 8086



Επιλογή εντολών (iv)

► Παράδειγμα:

tiling με εντολές του 80386



To 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
```

To 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]
    ...
    mov  si,word ptr [si + 4]
```

} ($n_{\text{cur}} - n_a - 1$ φορές)

Βοηθητικές ρουτίνες (i)

- getAR(a) (φόρτωση διεύθυνσης ΕΔ)

```
    mov si,word ptr [bp + 4]
    mov si,word ptr [si + 4]
    ...
    mov si,word ptr [si + 4]
```

} ($n_{\text{cur}} - n_a - 1$ φορές)

- 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)

► $\text{load}(R, a)$ (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	<code>lea R, byte ptr a</code>
(β1)	τοπική οντότητα: παράμετρος κατ' αξία, ή προσωρινή μεταβλητή	<code>lea R, size ptr [bp + offset]</code>
(β2)	τοπική οντότητα: παράμετρος κατ' αναφορά	<code>mov R, word ptr [bp + offset]</code>
(γ1)	μη τοπική οντότητα: παράμετρος κατ' αξία, ή προσωρινή μεταβλητή	<code>getAR(a)</code> <code>lea R, size ptr [si + offset]</code>
(γ2)	μη τοπική οντότητα: παράμετρος κατ' αναφορά	<code>getAR(a)</code> <code>mov R, word ptr [si + offset]</code>
(δ)	$[x]$	<code>load(R, x)</code>

Βοηθητικές ρουτίνες (v)

- ▶ Παρόμοια υλοποίηση για τις:
 - ▶ `loadReal(a)`
 - ▶ `store(R, a)`
 - ▶ `storeReal(a)`

Βοηθητικές ρουτίνες (v)

- ▶ Παρόμοια υλοποίηση για τις:
 - ▶ $\text{loadReal}(a)$
 - ▶ $\text{store}(R, a)$
 - ▶ $\text{storeReal}(a)$
- ▶ Ρουτίνες για ετικέτες τελικού κώδικα:
 - ▶ $\text{name}(p)$ $_p_num$
 - ▶ $\text{endof}(p)$ $@p_num$
 - ▶ $\text{label}(n)$ $@n$
 - ▶ $\text{label}(\ell)$ $@p_num_l$

Παραγωγή κώδικα (i)

► Τετράδα :=, x, −, z

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

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)
<i>instr ax, dx</i>	<i>finstr ST(1), ST(0)</i>
store(ax, z)	storeReal(z)

instr = add ή sub *finstr = faddp x.λπ.*

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)		storeReal(z)
$instr = add \text{ } \& \text{ } sub$		$finstr = faddp \text{ } x.\lambda\pi.$

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

load(ax, x)	load(ax, x)	load(ax, x)
load(cx, y)	cwd	cwd
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

instr = je, jne, x.lπ.

test ax, value

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)
```

Παραγωγή κώδικα (iv)

► Τετράδα ifb, x, -, z

load(al,x)

or al,al

jnz label(z)

► Τετράδα jump, -, -, z

jmp label(z)

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)

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
```

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
```

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

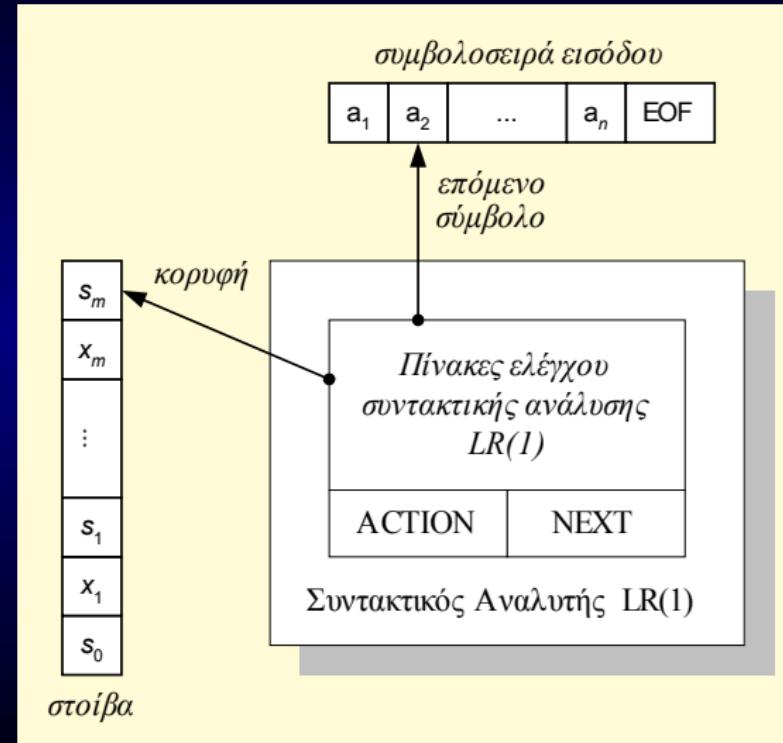
Παραγωγή κώδικα (vii)

- ▶ Τετράδα 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

Κεφάλαιο 4: Συντακτική ανάλυση

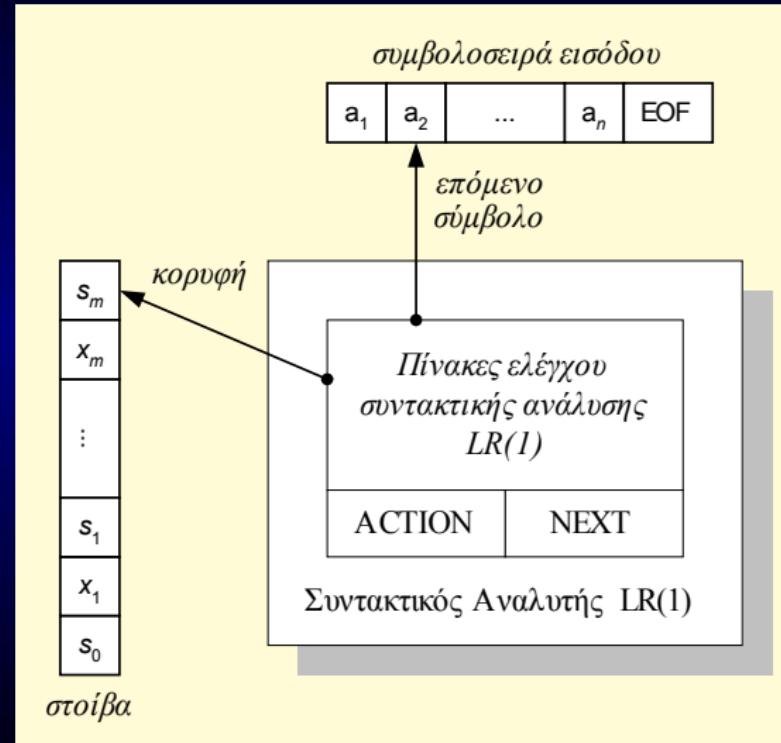
(μέρος 3ο)

Συντακτικοί αναλυτές $SLR(1)$



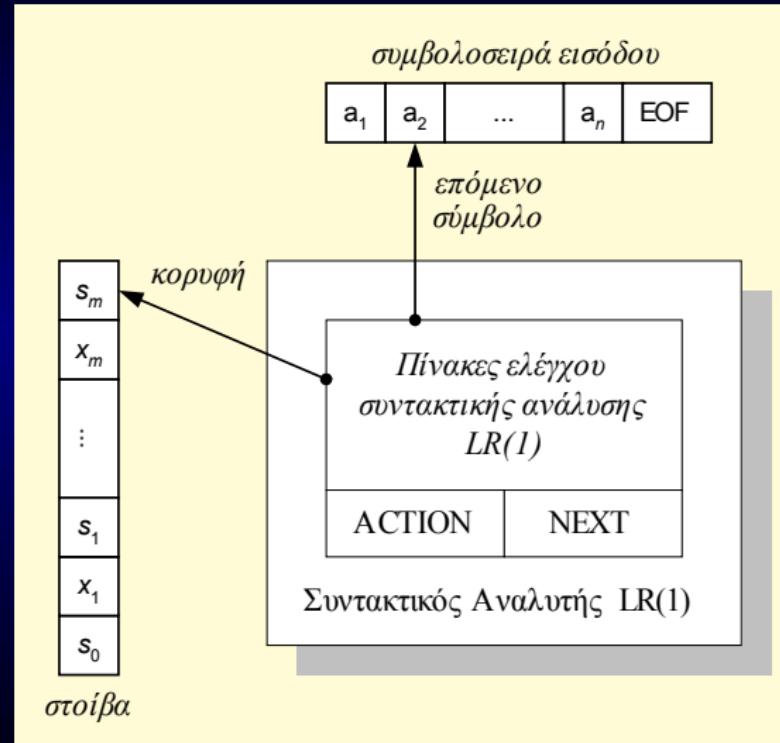
Συντακτικοί αναλυτές $SLR(1)$

► Αρχικά: s_0



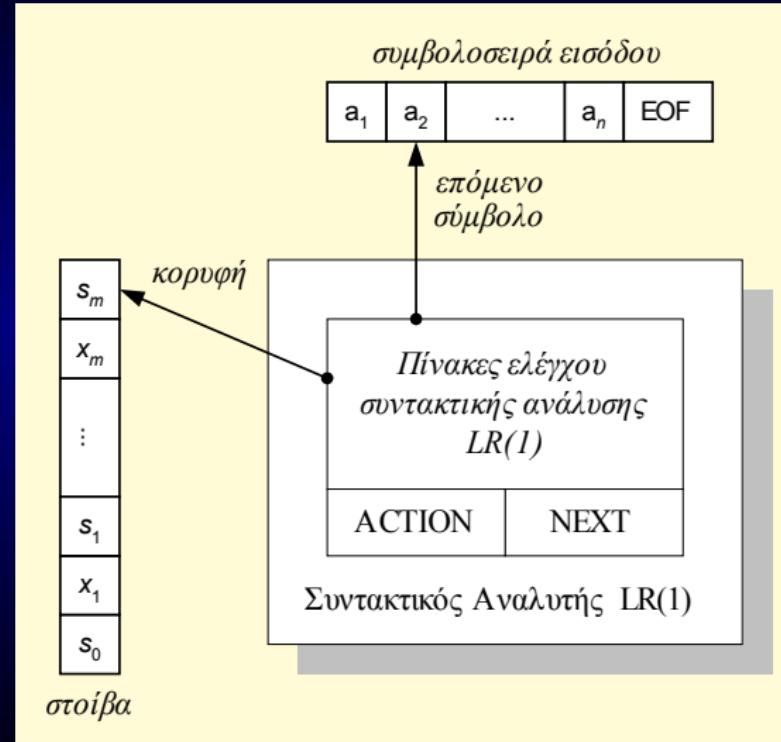
Συντακτικοί αναλυτές $SLR(1)$

- Αρχικά: s_0
- Είσοδος: a_k
- Κατάσταση: s_m
- $ACTION(s_m, a_k)$



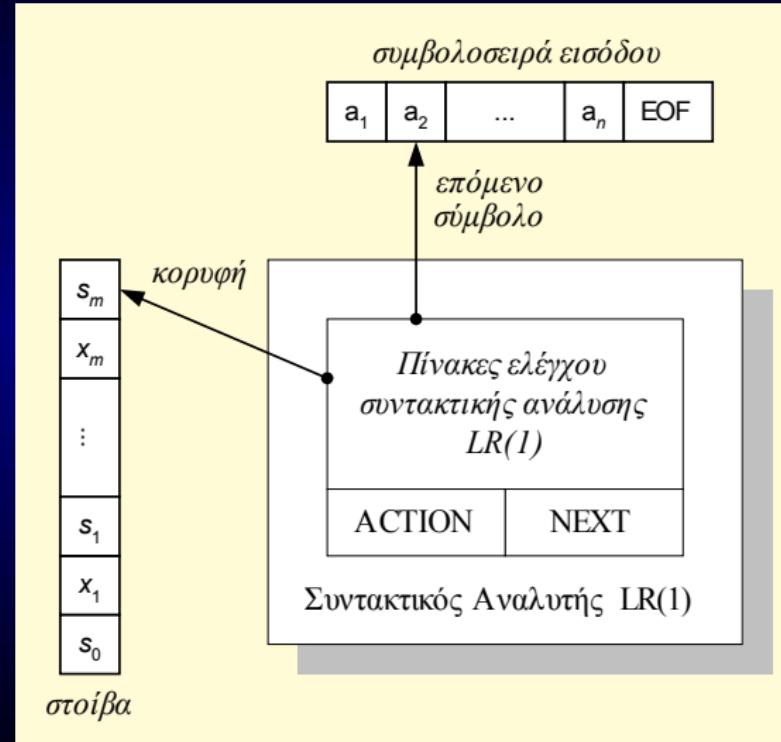
Συντακτικοί αναλυτές $SLR(1)$

- Αρχικά: s_0
- Είσοδος: a_k
- Κατάσταση: s_m
- $ACTION(s_m, a_k)$
- Ολίσθηση



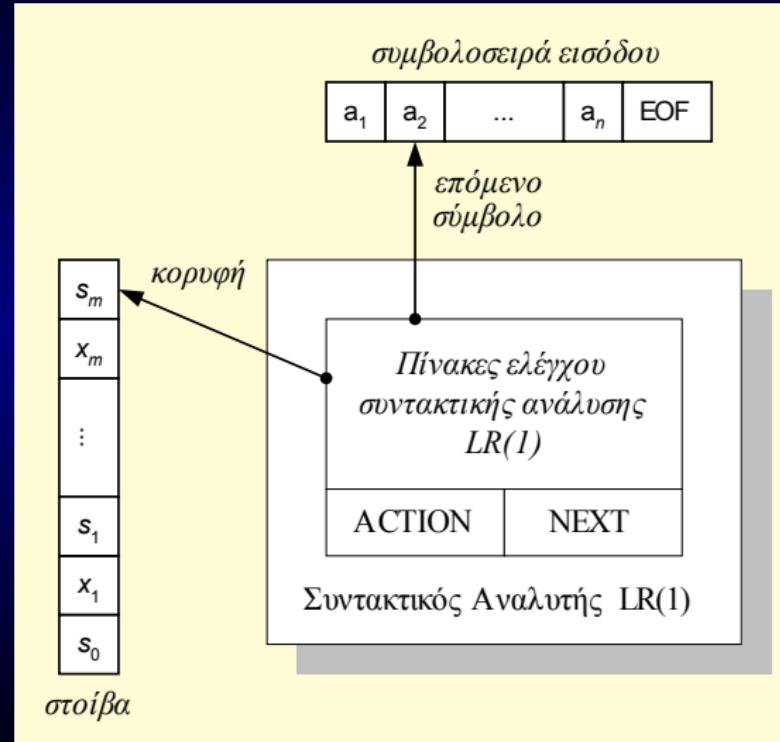
Συντακτικοί αναλυτές $SLR(1)$

- Αρχικά: s_0
- Είσοδος: a_k
- Κατάσταση: s_m
- $ACTION(s_m, a_k)$
- Ολίσθηση
- Ελάττωση



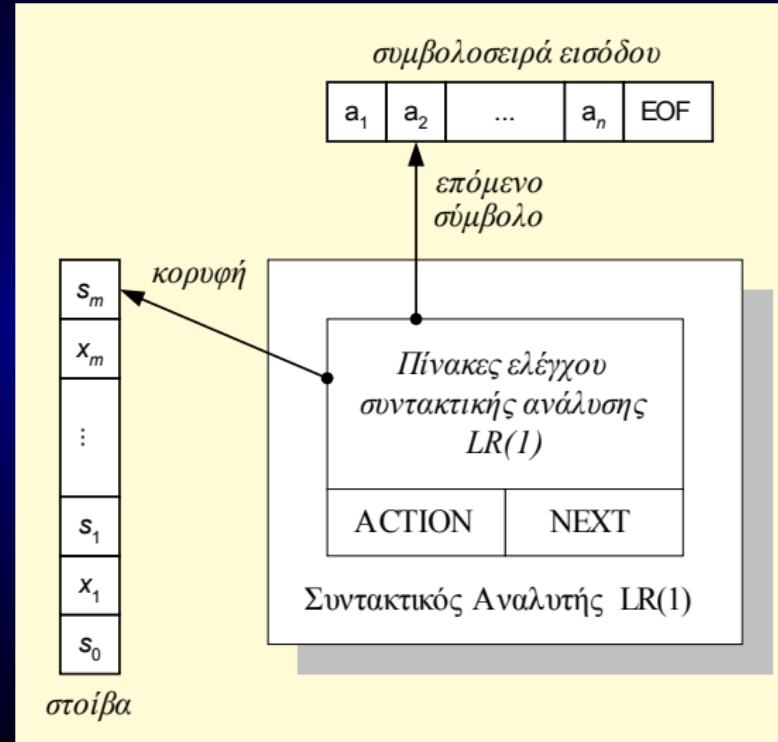
Συντακτικοί αναλυτές $SLR(1)$

- Αρχικά: s_0
- Είσοδος: a_k
- Κατάσταση: s_m
- $\text{ACTION}(s_m, a_k)$
- Ολίσθηση
- Ελάττωση
- Αποδοχή



Συντακτικοί αναλυτές $SLR(1)$

- Αρχικά: s_0
- Είσοδος: a_k
- Κατάσταση: s_m
- $\text{ACTION}(s_m, a_k)$
- Ολίσθηση
- Ελάττωση
- Αποδοχή
- Επόμενη κατάσταση:
 $\text{NEXT}(s_m, x_{m+1})$



Βοηθητικές έννοιες (i)

- Στοιχείο (item)

Βοηθητικές έννοιες (i)

- ▶ Στοιχείο (item)
- ▶ Έστω π.χ. ο συντακτικός κανόνας:
 $A \rightarrow xy$

Βοηθητικές έννοιες (i)

- ▶ **Στοιχείο** (item)
 - ▶ Έστω π.χ. ο συντακτικός κανόνας:

$$A \rightarrow xy$$

- ▶ Αντιστοιχούν τρία στοιχεία:

$$A \rightarrow \bullet xy$$

$$A \rightarrow x \bullet y$$

$$A \rightarrow xy \bullet$$

Βοηθητικές έννοιες (i)

- ▶ **Στοιχείο** (item)
 - ▶ Έστω π.χ. ο συντακτικός κανόνας:
$$A \rightarrow xy$$
 - ▶ Αντιστοιχούν τρία στοιχεία:
$$A \rightarrow \bullet xy$$
$$A \rightarrow x \bullet y$$
$$A \rightarrow xy \bullet$$
 - ▶ Το στοιχείο π.χ. $A \rightarrow x \bullet y$ δείχνει ότι
 - ▶ από τον κανόνα παραγωγής $A \rightarrow xy$
 - ▶ έχει μέχρι τώρα αναγνωριστεί το σύμβολο x
 - ▶ και απομένει να αναγνωριστεί το σύμβολο y

Βοηθητικές έννοιες (ii)

- ▶ Συνάρτηση CLOSURE
- ▶ Έστω σύνολο στοιχείων I
- ▶ Το σύνολο στοιχείων $\text{CLOSURE}(I)$ είναι το ελάχιστο σύνολο για το οποίο
 - ▶ $I \subseteq \text{CLOSURE}(I)$
 - ▶ Αν $A \rightarrow \alpha \bullet B \beta$ ανήκει στο $\text{CLOSURE}(I)$ και υπάρχει κανόνας $B \rightarrow \gamma$ τότε $B \rightarrow \bullet \gamma$ ανήκει στο $\text{CLOSURE}(I)$

Βοηθητικές έννοιες (iii)

- ▶ Συνάρτηση GOTO
- ▶ 'Εστω σύνολο στοιχείων I
- ▶ 'Εστω $x \in T \cup N$
- ▶ $\text{GOTO}(I, x) = \text{CLOSURE}(J)$
 $J = \{ A \rightarrow \alpha x \bullet \beta \mid A \rightarrow \alpha \bullet x \beta \in I \}$

Βοηθητικές έννοιες (iii)

- ▶ Συνάρτηση GOTO
- ▶ 'Εστω σύνολο στοιχείων I
- ▶ 'Εστω $x \in T \cup N$
- ▶ $\text{GOTO}(I, x) = \text{CLOSURE}(J)$
 $J = \{ A \rightarrow \alpha x \bullet \beta \mid A \rightarrow \alpha \bullet x \beta \in I \}$
- ▶ Δηλαδή, το $\text{GOTO}(I, x)$ περιέχει όλα τα στοιχεία της γραμματικής που μπορούν να προκύψουν από τα στοιχεία του I αν αναγνωριστεί το σύμβολο x

Κατασκευή ΣΑ SLR(1)

- ▶ Υπολογισμός συνόλου καταστάσεων
 - ▶ Προσθήκη κανόνα $S' \rightarrow S$
 - ▶ Οι καταστάσεις είναι σύνολα στοιχείων
 - ▶ Έναρξη με $I_0 = \text{CLOSURE}(\{ S' \rightarrow \bullet S \})$
 - ▶ Υπολογισμός $\text{GOTO}(I_n, x)$ για κάθε $x \in T \cup N$
 - ▶ Προσθήκη νέων καταστάσεων

Κατασκευή ΣΑ SLR(1)

- ▶ Υπολογισμός συνόλου καταστάσεων
 - ▶ Προσθήκη κανόνα $S' \rightarrow S$
 - ▶ Οι καταστάσεις είναι σύνολα στοιχείων
 - ▶ Έναρξη με $I_0 = \text{CLOSURE}(\{ S' \rightarrow \bullet S \})$
 - ▶ Υπολογισμός $\text{GOTO}(I_n, x)$ για κάθε $x \in T \cup N$
 - ▶ Προσθήκη νέων καταστάσεων
- ▶ Υπολογισμός πινάκων ACTION και GOTO
 - ▶ Συγχώνευση σε έναν πίνακα