

Μεταγλωττιστές

Νίκος Παπασπύρου Κωστής Σαγώνας
{nickie}@softlab.ntua.gr {kostis}@cs.ntua.gr

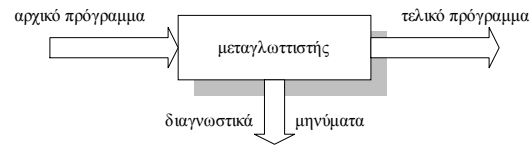


Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχ. και Μηχ. Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού
Πολυτεχνειούπολη, 15780 Ζωγράφου.

Μάρτιος 2009

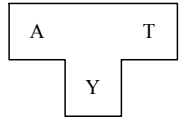
Εισαγωγή (i)

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



Εισαγωγή (ii)

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



$$\llbracket P \rrbracket_L : Inp(P) \rightarrow Out(P)$$

- ▶ Σημασία ενός μεταγλωττιστή C

$$\llbracket C \rrbracket_{L_Y} : L_A \rightarrow L_T$$

1 / 123

2 / 123

3 / 123

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)

4 / 123

5 / 123

6 / 123

Κατασκευή μεταγλωττιστή (i)

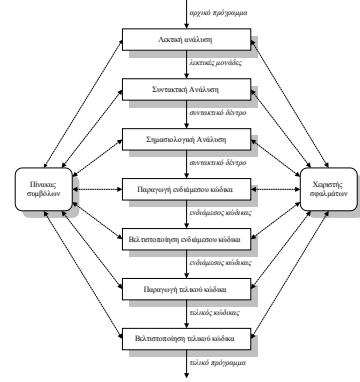
- ▶ Βασικές απαιτήσεις:
 - ▶ Να λειτουργεί **σωστά**
 - ▶ Να **συμμορφώνεται** με τις προδιαγραφές της αρχικής και της τελικής γλώσσας
 - ▶ Να μεταγλωττίζει προγράμματα **κάθε μεγέθους**

Κατασκευή μεταγλωττιστή (ii)

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

Φάσεις και προϊόντα της μεταγλώττισης

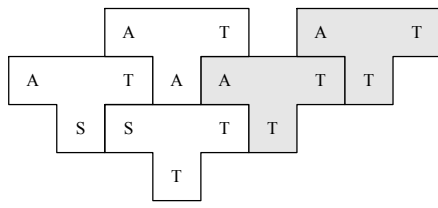
- ▶ Λεκτική ανάλυση
- ▶ Συντακτική ανάλυση
- ▶ Σημασιολογική ανάλυση
- ▶ Παραγωγή ενδιάμεσου κώδικα
- ▶ Βελτιστοποίηση
- ▶ Παραγωγή τελικού κώδικα



Θέματα υλοποίησης

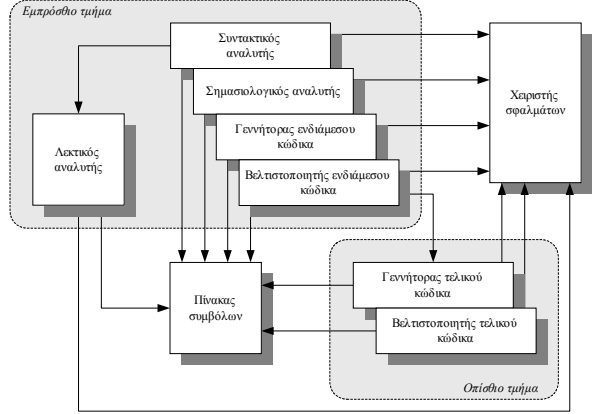
- ▶ Οργάνωση σε **περάσματα**
- ▶ Οργάνωση σε **εμπρόσθιο** και **οπίσθιο** τμήμα (front-end / back-end)
- ▶ Έλεγχος **ορθότητας**
- ▶ Είδη **διαγνωστικών μηνυμάτων** και ανάνηψη
 - ▶ Εσωτερικά (internal)
 - ▶ Σφάλματα (errors)
 - ▶ Προειδοποιητικά μηνύματα (warnings)
 - ▶ Απλά μηνύματα (messages)

Εκκίνηση – bootstrapping



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

Οργάνωση σε ένα πέρασμα



Κεφάλαιο 2: Τυπικές γλώσσες

Τυπικές γλώσσες (i)

Βασικές έννοιες

- ▶ Αλφάβητο
- ▶ Σύμβολο
- ▶ Συμβολοσειρά
- ▶ Μήκος συμβολοσειράς
- ▶ Σύνολο συμβολοσειρών μήκους n
- ▶ Σύνολο όλων των συμβολοσειρών

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

Σ

a

α

$|\alpha|$

Σ^n

Τυπικές γλώσσες (ii)

Βασικές έννοιες (συνέχεια)

- ▶ Κενή συμβολοσειρά
- ▶ Παράθεση συμβολοσειρών
- ▶ Παράθεση συμβολοσειράς με τον εαυτό της
- ▶ Πρόθεμα, επίθεμα, υποσυμβολοσειρά

$$\alpha^0 = \epsilon$$

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

ϵ

$\alpha\beta$

13 / 123

14 / 123

15 / 123

Τυπικές γλώσσες (iii)

Βασικές έννοιες (συνέχεια)

- ▶ Γλώσσα
- ▶ Ένωση γλωσσών
- ▶ Παράθεση γλωσσών
- ▶ Παράθεση γλώσσας με τον εαυτό της

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

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

$$L \subseteq \Sigma^*$$

- ▶ 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
 - ▶ N : **μη τερματικά** σύμβολα
 - ▶ P : **κανόνες** παραγωγής
 - ▶ 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: $L(G) = \{\alpha \in T^* \mid S \Rightarrow^+ \alpha\}$

a

A

$\alpha \rightarrow \beta$

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

16 / 123

17 / 123

18 / 123

Τυπικές γλώσσες (vi)

Αναγνωριστές

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

Κανονικές γλώσσες (i)

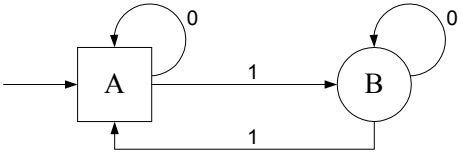
- ▶ **Κανονικές γραμματικές**
 - ▶ Μόνο κανόνες $A \rightarrow aB$ ή $A \rightarrow a$
 - ▶ ισοδύναμα $A \rightarrow Ba$ ή $A \rightarrow a$
- ▶ **Κανονικές εκφράσεις** (regular expressions)
 - ▶ Κενή συμβολοσειρά: ϵ
 - ▶ Κάθε σύμβολο του Σ : a
 - ▶ Παράθεση δύο κανονικών εκφράσεων: (rs)
 - ▶ Διάζευξη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

Κανονικές γλώσσες (ii)

Παραδείγματα κανονικών εκφράσεων

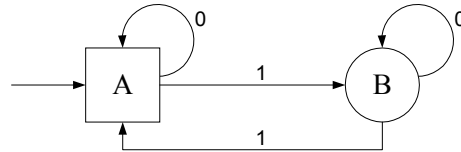
- ▶ Ακέραιες σταθερές χωρίς πρόσημο στην Pascal
 - ▶ ένα ή περισσότερα δεκαδικά ψηφία $[0-9]^+$
 - ▶ Αριθμητικές σταθερές χωρίς πρόσημο στη C
 - ▶ ακέραιο μέρος που **δεν** αρχίζει με μηδέν, εκτός αν είναι μηδενικό (γιατί;)
 - ▶ προαιρετικά: υποδιαστολή και κλασματικό μέρος
 - ▶ προαιρετικά: εκθέτης 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;
- ▶ Τη γλώσσα των συμβολοσειρών που αποτελούνται από 0 και 1 και περιέχουν άρτιο αριθμό 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

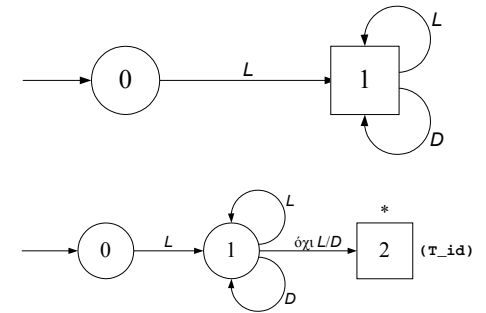
Κεφάλαιο 3: Λεκτική ανάλυση

Λεκτική ανάλυση

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

Διαγράμματα μετάβασης (i)

- ▶ Αναγνωριστικά της Pascal



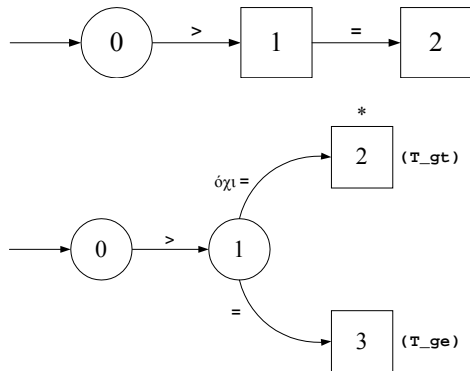
25 / 123

26 / 123

27 / 123

Διαγράμματα μετάβασης (ii)

- ▶ Τελεστές > και >=



28 / 123

Κατασκευή του ΛΑ (i)

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

Κατασκευή του ΛΑ (ii)

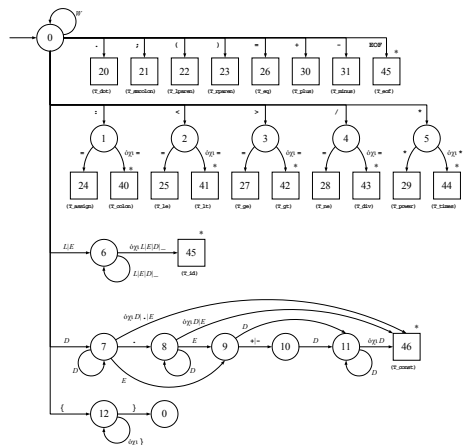
- ▶ Επιμέρους θέματα
 - ▶ Τρόπος **διαχωρισμού** λεκτικών μονάδων
 - ▶ **Σχόλια**
 - ▶ Διάκριση **πεζών / κεφαλαίων** γραμμάτων
 - ▶ **Ενδιάμεση μνήμη** (buffer)
 - ▶ **Ανάληψη** από σφάλματα

29 / 123

30 / 123

Κατασκευή του ΛΑ (iii)

Σχεδίαση
συνολικού
διαγράμματος
μετάβασης



31 / 123

Κατασκευή του ΛΑ (iv)

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

32 / 123

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)

33 / 123

Υλοποίηση ΛΑ με το flex (ii)

- ▶ **Δομή του μεταπρογράμματος**

Μέρος Α
%%
Μέρος Β
%%
Μέρος Γ

Και τα τρία μέρη μπορούν να είναι κενά

34 / 123

Υλοποίηση ΛΑ με το flex (iii)

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

35 / 123

Υλοποίηση ΛΑ με το flex (iv)

- ▶ **Μέρος Α**, παράδειγμα

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

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

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

36 / 123

Υλοποίηση ΛΑ με το flex (v)

- ▶ **Μέρος Β**, περιέχει κανόνες της μορφής
κανονική έκφραση ενέργεια
- ▶ Κάθε ενέργεια είναι μια εντολή της C
- ▶ **Λειτουργία:**
 - ▶ Διαβάζεται το **μακρύτερο πρόθεμα** της συμβολοσειράς εισόδου που μπορεί να αναγνωριστεί από κάποια κανονική έκφραση
 - ▶ Εκτελείται η αντίστοιχη ενέργεια

37 / 123

Υλοποίηση ΛΑ με το flex (vi)

- ▶ **Κανονικές εκφράσεις**
 - a** Ο χαρακτήρας a.
 - .** Οποιοσδήποτε 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.

38 / 123

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.

39 / 123

Υλοποίηση ΛΑ με το flex (viii)

- ▶ **Κανονικές εκφράσεις (συνέχεια)**
 - ^r** Η *r* αλλά μόνο στην αρχή μιας γραμμής.
 - r\$** Η *r* αλλά μόνο στο τέλος μιας γραμμής.
 - <<EOF>>** Το τέλος του αρχείου εισόδου.
 - r/s** Η κανονική έκφραση *r* αλλά μόνο αν ακολουθεί η κανονική έκφραση *s*.
 - <S>r** Η *r* αλλά μόνο όταν η τρέχουσα αρχική κατάσταση είναι η *S*.
 - <S₁, S₂, S₃>r** Η *r*, αλλά μόνο όταν η τρέχουσα αρχική κατάσταση είναι μια από τις *S₁*, *S₂* ή *S₃*.
 - <*>r** Η *r* σε οποιαδήποτε αρχική κατάσταση.

40 / 123

Υλοποίηση ΛΑ με το flex (ix)

- ▶ **Μέρος Β**, παράδειγμα

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

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

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

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

41 / 123

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);
}
```

42 / 123

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;
}
```

43 / 123

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

44 / 123

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

45 / 123

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 */ }
```

46 / 123

Κεφάλαιο 2: Τυπικές γλώσσες

(μέρος 2ο)

47 / 123

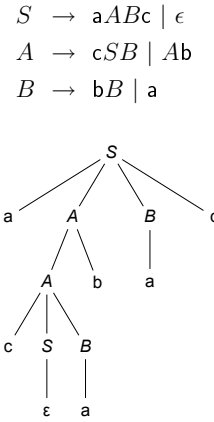
Γλώσσες χωρίς συμφραζόμενα (i)

- ▶ 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)

48 / 123

Γλώσσες χωρίς συμφραζόμενα (ii)

- ▶ Μία παραγωγή
 $[S] \Rightarrow aA[B]c \Rightarrow a[A]bBc$
 $\Rightarrow acS[B]bBc \Rightarrow ac[S]abBc$
 $\Rightarrow acab[B]c \Rightarrow acabac$
- ▶ 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
 $[S] \Rightarrow_R aA[B]c \Rightarrow_R a[A]ac$
 $\Rightarrow_R a[A]bac \Rightarrow_R acS[B]bac$
 $\Rightarrow_R ac[S]abac \Rightarrow_R acabac$

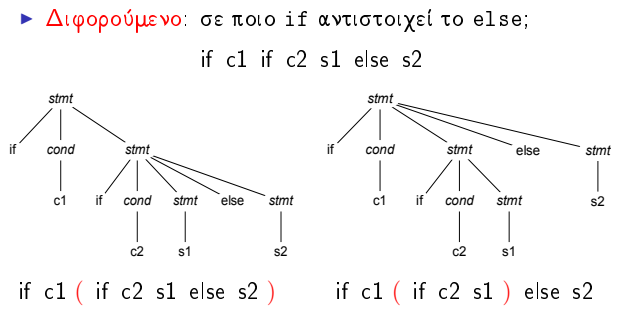


Διφορούμενες γραμματικές (i)

- ▶ 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: **ξεκρέμαστο if** (dangling if)
 $stmt \rightarrow if\ cond\ stmt\ else\ stmt \mid if\ cond\ stmt \mid s1 \mid s2$
 $cond \rightarrow c1 \mid c2$



Τρόποι παράστασης γραμματικών (i)

- ▶ **Backus-Naur Form** (BNF)
- ▶ Σύμβολο ::= στους κανόνες
- ▶ Μη τερματικά σύμβολα σε γωνιακές παρενθέσεις, π.χ. <expr>
- ▶ Σύμβολο | για διάζευξη

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

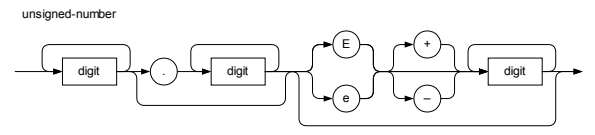
Τρόποι παράστασης γραμματικών (ii)

- ▶ **Extended Backus-Naur Form** (EBNF)
- ▶ Τερματικά σύμβολα σε εισαγωγικά
- ▶ Παρενθέσεις για ομαδοποίηση
- ▶ Αγκύλες για προαιρετικά τμήματα
- ▶ Σύμβολα * και + για επανάληψη

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

Τρόποι παράστασης γραμματικών (iii)

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

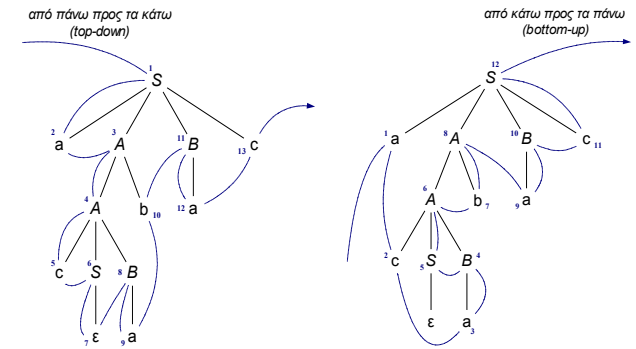


Κεφάλαιο 4: Συντακτική ανάλυση

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



55 / 123

56 / 123

57 / 123

Βοηθητικές έννοιες (i)

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

Βοηθητικές έννοιες (ii)

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

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

58 / 123

59 / 123

60 / 123

Υπολογισμός FIRST (ii)

▶ Παράδειγμα

- FIRST(E) = {id, (}
- FIRST(T) = {id, (}
- FIRST(F) = {id, (}
- FIRST(E') = {+, ε}
- FIRST(T') = {*, ε}

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

Υπολογισμός FOLLOW (i)

- ▶ EOF ∈ FOLLOW(S)
- ▶ για κάθε κανόνα $A \rightarrow \alpha B \beta$
 - ▶ (FIRST(β) - {ε}) ⊆ FOLLOW(B)
 - ▶ αν ε ∈ FIRST(β) τότε FOLLOW(A) ⊆ FOLLOW(B)

Υπολογισμός FOLLOW (ii)

▶ Παράδειγμα

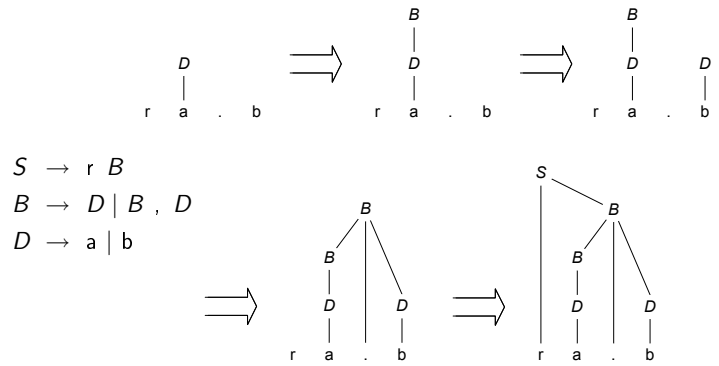
- FOLLOW(E) = {), EOF}
- FOLLOW(T) = {+,), EOF}
- FOLLOW(F) = {*, +,), EOF}
- FOLLOW(E') = {), EOF}
- FOLLOW(T') = {+,), EOF}
- FIRST(E') = {+, ε}
- FIRST(T') = {*, ε}

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

ΣΑ bottom-up (i)

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

ΣΑ bottom-up (ii)



ΣΑ bottom-up (iii)

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

ΣΑ bottom-up (iv)

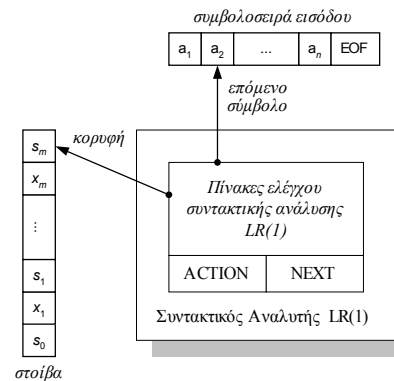
$S \rightarrow r B$
 $B \rightarrow D | B , D$

βήμα	στοίβα	είσοδος	περιγραφή
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

67 / 123

ΣΑ bottom-up (v)

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



68 / 123

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`)

69 / 123

Υλοποίηση ΣΑ με το bison (ii)

- ▶ **Δομή του μεταπρογράμματος**

Μέρος Α
`%%`
 Μέρος Β
`%%`
 Μέρος Γ

Και τα τρία μέρη μπορούν να είναι κενά

70 / 123

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, προσηταιριστικότητα)
 - ▶ Δήλωση του **συνόλου σημασιολογικών τιμών** (τύπος `YYSTYPE` ή με χρήση του `%union`)
 - ▶ Δήλωση του **τύπου της σημασιολογικής τιμής** κάθε συμβόλου

71 / 123

Υλοποίηση ΣΑ 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
            
```

72 / 123

Υλοποίηση ΣΑ με το bison (v)

- ▶ **Μέρος Α**, παράδειγμα (συνέχεια)

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

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

%type<f> expression
```

73 / 123

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:

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

74 / 123

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"
      ;
```

75 / 123

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 γεννήτορα ενδιαμέσου κώδικα

76 / 123

Υλοποίηση ΣΑ με το bison (ix)

- ▶ **Μέρος Γ**, παράδειγμα

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

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

77 / 123

Υλοποίηση ΣΑ με το bison (x)

- ▶ Παράδειγμα με **σημασιολογικές τιμές**

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

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

78 / 123

Υλοποίηση ΣΑ με το bison (xi)

- ▶ Παράδειγμα (συνέχεια)

```
%{
typedef int YYSTYPE;
%}

%token T_num

%%

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

79 / 123

Υλοποίηση ΣΑ με το bison (xii)

- ▶ Παράδειγμα (συνέχεια)

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

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

80 / 123

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`

81 / 123

Υλοποίηση ΣΑ με το bison (xiv)

- ▶ Παράδειγμα — Υλοποίηση ΛΑ χειρωνακτικά

```
int yylex ()
{
    int c;

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

Στο μέρος Γ

82 / 123

Υλοποίηση ΣΑ με το bison (xv)

- ▶ Παράδειγμα (συνέχεια)

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

- ▶ Αυτοματοποίηση της μεταγλώττισης του ΣΑ

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

Makefile

83 / 123

Υλοποίηση ΣΑ με το bison (xvi)

- ▶ Παράδειγμα — Υλοποίηση ΛΑ με το flex

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

%%

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

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

%%
mytest2.l
```

84 / 123

Υλοποίηση ΣΑ με το bison (xvii)

- ▶ **Αυτοματοποίηση** της μεταγλώττισης ΛΑ και ΣΑ

```
mytest2: mytest2.1 mytest2.y          Makefile
        bison -d mytest2.y
        flex -s mytest2.1
        gcc -o mytest2 mytest2.tab.c lex.yy.c \
        -lf1
```

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

85 / 123

ΣΑ top-down (i)

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

86 / 123

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

87 / 123

Μετασχηματισμός σε LL(1)

- ▶ **Αντικατάσταση**

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

- ▶ **Αριστερή παραγοντοποίηση**

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

- ▶ **Απαλοιφή άμεσης αριστερής αναδρομής**

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

88 / 123

ΣΑ αναδρομικής κατάβασης

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

μετατρέπεται σε κώδικα της μορφής:

```
if token ∈ FIRST(α1) then
  κώδικας για την αναγνώριση της α1
  ...
else if token ∈ FIRST(αn) then
  κώδικας για την αναγνώριση της αn
else if ε ∉ FIRST(α1) ∪ ... ∪ FIRST(αn) then
  συντακτικό σφάλμα
else if token ∉ FOLLOW(A) then
  συντακτικό σφάλμα
end if
```

89 / 123

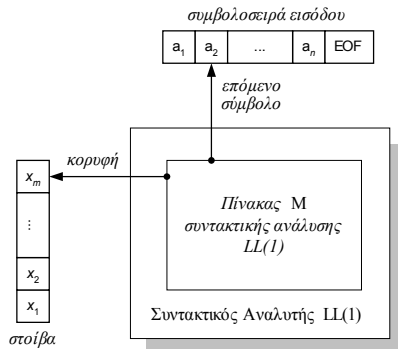
ΣΑ LL(1) (i)

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

90 / 123

ΣΑ LL(1) (ii)

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



Κατασκευή ΣΑ LL(1)

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

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

Λειτουργία ΣΑ LL(1)

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

91 / 123

92 / 123

93 / 123

Κεφάλαιο 2: Τυπικές γλώσσες (μέρος 3ο)

Κατηγορικές γραμματικές (i)

- ▶ **Κατηγορική γραμματική** (attribute grammar): γραμματική χωρίς συμπραζόμενα 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

94 / 123

Κατηγορικές γραμματικές (ii)

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

Σημασιολογικοί κανόνες

95 / 123

96 / 123

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 ετικέτες εντολών
 - ▶ οι σταθερές
 - ▶ οι τύποι δεδομένων

Χαρακτηριστικά ονομάτων

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

97 / 123

98 / 123

99 / 123

Περιεχόμενα πίνακα συμβόλων

Οργάνωση 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

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

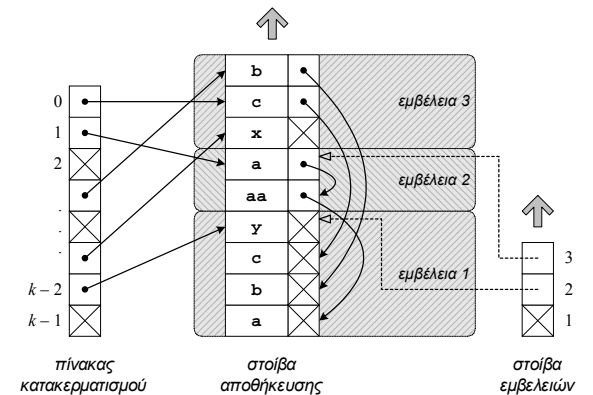
γραμμική λίστα	$O(n)$
δυναμικό δέντρο αναζήτησης	$O(\log n)$
πίνακας κατακερματισμού	$O(n/k)$

100 / 123

101 / 123

102 / 123

Υλοποίηση με ΠΚ



Κεφάλαιο 6: Σημασιολογική ανάλυση

Σύνταξη και σημασιολογία

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

103 / 123

Στατική σημασιολογία (i)

- ▶ Περιβάλλοντα τύπων

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

- ▶ Σχέση αντιστοίχισης τύπων

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

- ▶ Κανόνες τύπων

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

104 / 123

105 / 123

Στατική σημασιολογία (ii)

- ▶ Παραγωγές τύπων

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

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

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

106 / 123

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

107 / 123

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**

$$\{P[I \mapsto E] \mid I=E \{P\}$$

108 / 123

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)

109 / 123

Σημασιολογικός έλεγχος (ii)

- ▶ Απουσία σημασιολογικών σφαλμάτων \neq 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.
```

110 / 123

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)

111 / 123

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
```

112 / 123

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");  
} ;
```

113 / 123

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")  
} ;
```

114 / 123

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)

115 / 123

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");
} ;
```

116 / 123

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");
} ;
```

117 / 123

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)

118 / 123

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

119 / 123

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

120 / 123

Αντιστοίχιση τύπων

▶ Στατική

- ▶ ρητή
- ▶ έμμεση

▶ Δυναμική

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

121 / 123

Εξαγωγή τύπων

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

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

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

122 / 123

Δυναμικός έλεγχος τύπων

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

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

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

123 / 123