

Κεφάλαιο 3 :

Σύνταξη Γλωσσών Προγραμματισμού

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Γιάννης Γαροφαλάκης, Σπύρος Σιούτας,
Παναγιώτης Χατζηδούκας

Εισαγωγή

- Οι διαφορές των ΓΠ στις *συντακτικές δομές* τους, είναι μεγαλύτερες από τις διαφορές τους στις *εννοιολογικές δομές*. Π.χ. το στοιχείο του πίνακα A στη θέση 1, γράφεται:

A(1) στις **FORTRAN, COBOL, PL/1, Ada**

A[1] στις **Pascal, C, C++, Java, Python, ...**

A<1> στη **SNOBOL**

- Στόχος συντακτικού:

Κανόνες επικοινωνίας της πληροφορίας μεταξύ προγραμματιστή και μεταφραστή/διερμηνέα.

Γενικά κριτήρια Συντακτικών Κανόνων

- Αναγνωσιμότητα (readability)

Self-documenting: «φυσική» μορφή εντολών, δομημένες εντολές, keywords και noise words, ενσωματωμένα σχόλια, μεγάλο μήκος ονομάτων, μνημονικά σύμβολα

- Ευκολία Γραφής (writeability)

Πολλές φορές αντίθετο με αναγνωσιμότητα

- Ευκολία Μετάφρασης

Πολλές συντακτικές δομές → δυσκολία στη μετάφραση

- Έλλειψη Ασαφειών

Π.χ. στη C:

```
if (n>0) {  
    if (a>b) z=a;  
    else z=b; }
```

ΣΥΝΤΑΚΤΙΚΑ ΣΤΟΙΧΕΙΑ ΜΙΑΣ ΓΠ (1)

1. Σύνολο Χαρακτήρων (Αλφάβητο)

- Γράμματα κεφαλαία και μικρά (A,B,...,Z,a,b,...z)
- Αριθμητικά ψηφία (0,1,...,9)
- Ειδικοί χαρακτήρες (, . ; & \$ * # () [] ...)

Δύο αρχικές **κωδικοποιήσεις** χαρακτήρων για Η/Υ:

- a. EBCDIC (Extended Binary Coded Decimal Interchange Code) από την IBM (8 bit $\rightarrow 2^8 = 256$ χαρακτήρες)
- b. ASCII (American Standard Code for Information Interchange) από ANSI (7 bit $\rightarrow 2^7 = 128$ χαρακτήρες)

Τώρα:

UNICODE (Universal Character Set)

16 bit $\rightarrow 2^{16} = 65.536$ χαρακτήρες

Συμβατή κωδικοποίηση με ASCII

ΣΥΝΤΑΚΤΙΚΑ ΣΤΟΙΧΕΙΑ ΜΙΑΣ ΓΠ (2)

2. Αναγνωριστικά (identifiers)

Ονόματα μεταβλητών, συναρτήσεων, κ.λπ., που ορίζει ο χρήστης. Π.χ. **SUM** (αλλά όχι **SU#M**, **SU*M**)

3. Σύμβολα πράξεων

+ * - / ** && || AND OR NOT := (εντολή ανάθεσης) ...

4. Λέξεις Κλειδιά (keywords) και Δεσμευμένες Λέξεις (reserved words)

Λέξεις Κλειδιά: Χρησιμοποιούνται από τη ΓΠ

Δεσμευμένες Λέξεις: Λέξεις Κλειδιά που δεν μπορεί ο προγραμματιστής να αλλάξει τη χρήση τους

- Η **C** έχει 28 δεσμευμένες λέξεις (int, else, for, ...)
- Η αρχική **FORTRAN** δεν είχε τα DO, IF ως δεσμευμένες λέξεις

Συντακτικά Στοιχεία μιας ΓΠ (3)

ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΔΕΣΜΕΥΜΕΝΩΝ ΛΕΞΕΩΝ

- Ευανάγνωστα Προγράμματα
- Εύκολη εύρεση στον Πίνακα Συμβόλων από Μεταφραστή
- Διευκολύνουν τον εντοπισμό και τη διόρθωση λαθών

ΜΕΙΟΝΕΚΤΗΜΑΤΑ ΔΕΣΜΕΥΜΕΝΩΝ ΛΕΞΕΩΝ

- Όσο αυξάνονται, πιο δύσκολο να τις θυμάται ο χρήστης
- Δύσκολη η επέκταση της ΓΠ με νέες δεσμευμένες λέξεις, διότι τα παλιά προγράμματα μπορεί να τις χρησιμοποιούν ως ονόματα μεταβλητών

ΣΥΝΤΑΚΤΙΚΑ ΣΤΟΙΧΕΙΑ ΜΙΑΣ ΓΠ (4)

5. Σχόλια και Θόρυβος

Σχόλια: π.χ. `/* ... */` στη **C**, εντολή **REM** στην **BASIC**

Θόρυβος: π.χ. στην **COBOL**: `GO TO <label>`

6. Κενά

Διάφορες χρήσεις. Π.χ. στη **SNOBOL** είναι το σύμβολο της συγχώνευσης strings

7. Διαχωριστικά (delimiters)

- Στη **C**: `{ ... }`. Στις **ALGOL, Pascal**: `begin ... end`
- Ομαδοποίηση εντολών
- Καλό για άρση ασαφειών

Συντακτικά Στοιχεία μιας ΓΠ (4)

8. Εκφράσεις (expressions)

Συναρτήσεις που προσπελαύνουν δεδομένα σε ένα πρόγραμμα και επιστρέφουν μια τιμή. Π.χ. :

$$A+B*C$$

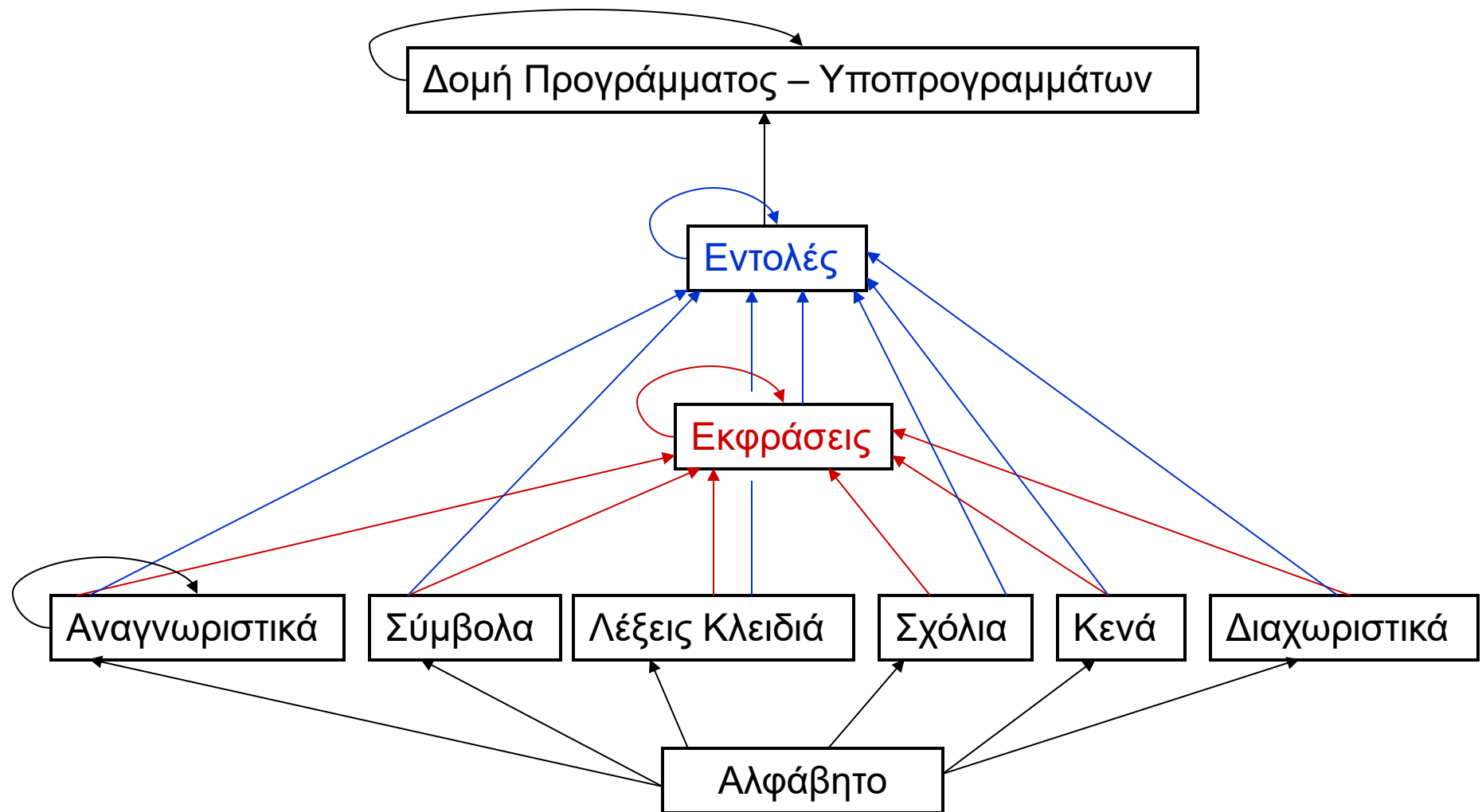
9. Εντολές (statements)

Δημιουργούνται από συνδυασμούς εκφράσεων και άλλων συντακτικών στοιχείων. Π.χ. :

$$D = (A+B*C) - E/2$$

10. Δομή Προγράμματος – Υποπρογραμμάτων

Ιεραρχία Συντακτικών Στοιχείων



Δομή Προγράμματος – Υποπ/μάτων (1)

a. Ξεχωριστός Ορισμός Υποπρογράμματος

- C, FORTRAN, Java, Python
- Όλα τα τμήματα κώδικα είναι “υποπρογράμματα”
- Κάθε υποπρόγραμμα θεωρείται διακριτή συντακτική μονάδα, μπορεί να μεταφράζεται χωριστά και να συνδέονται όλα όταν γίνεται φόρτωση

b. Ξεχωριστός Ορισμός Δεδομένων

- Java, Python, C++, Smalltalk: Μηχανισμός κλάσεων
- Ομαδοποίηση των λειτουργιών που χειρίζονται ένα δεδομένο data object

c. Φωλιασμένος Ορισμός Υποπρογράμματος

- Pascal, ALGOL, JavaScript, Python
- Ιεραρχικός ορισμός υπ/τος μέσα σε υπ/μα. Δυνατότητα ορισμού εμβέλειας. Στατικός έλεγχος τύπων.

Δομή Προγράμματος – Υποπ/μάτων (2)

d. *Ξεχωριστός Ορισμός Interface*

- C, C++, Ada
- Δυνατότητα χρήσης file operations του ΛΣ (.h και .c file specs του make στη C)

e. *Περιγραφές δεδομένων χωριστές από τις εκτελέσιμες εντολές*

- COBOL
- Όλα τα data είναι global. Data και procedure divisions

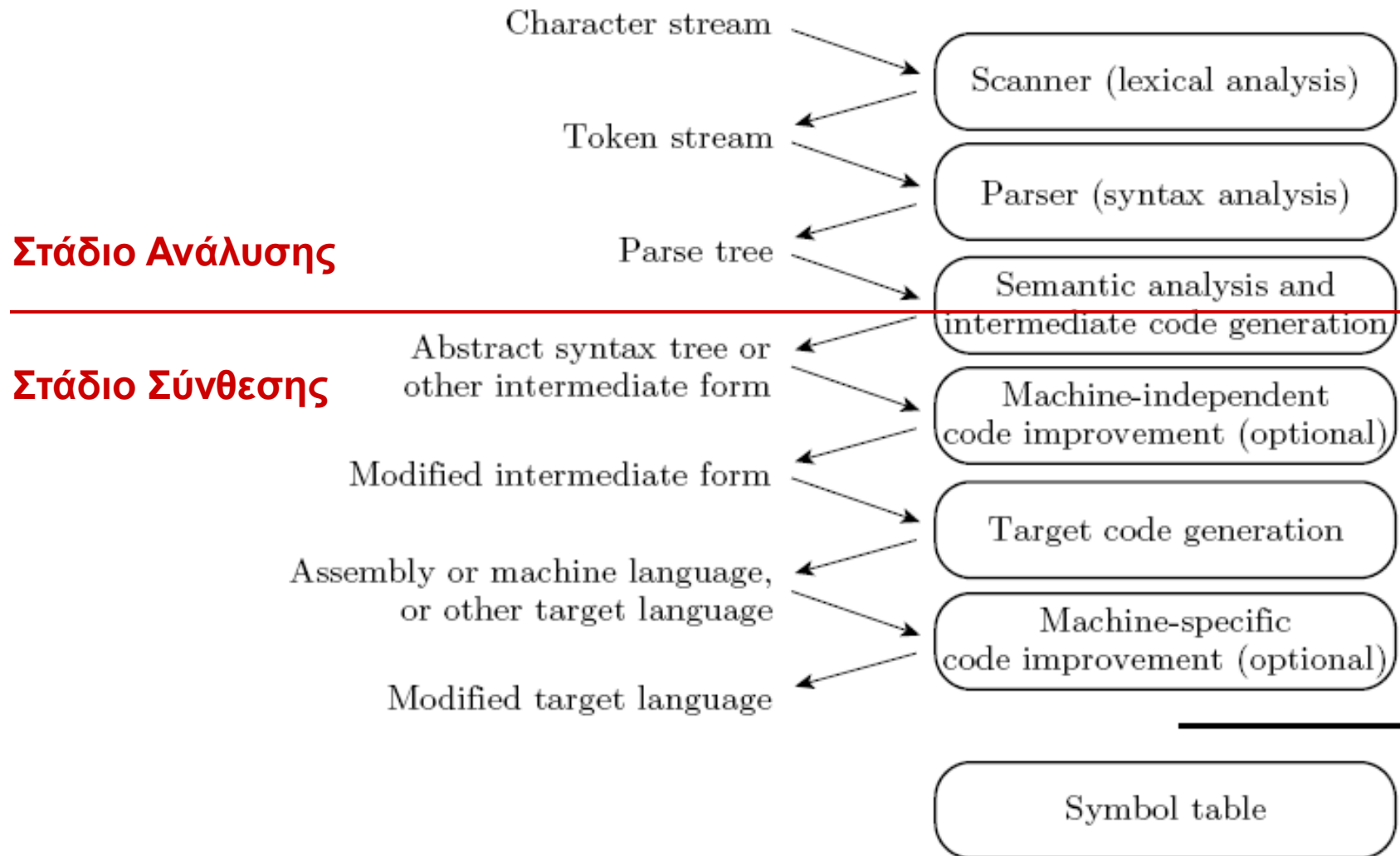
f. *Μη-χωριστοί Ορισμοί Υποπρογραμμάτων*

- BASIC, SNOBOL
- Καμία οργάνωση. Τα υπ/ματα μπορούν να χρησιμοποιηθούν και από άλλα τμήματα του προγράμματος

Φάσεις Μετάφρασης (1)

- Κάθε **Φάση** δέχεται ως είσοδο ένα πρόγραμμα *ισοδύναμο* με το αρχικό σε κάποια μορφή και παράγει ως έξοδο το *ίδιο* πρόγραμμα σε άλλη μορφή.
- Δύο **Στάδια**:
 - **Ανάλυση** πηγαίου προγράμματος (3 Φάσεις)
 - **Σύνθεση** εκτελέσιμου κώδικα (3-4 Φάσεις)
- Αριθμός **Περασμάτων** πηγαίου κώδικα:
 - Συνήθως: 2 περάσματα αντίστοιχα με τα 2 Στάδια
 - Πολλές φορές: 3 περάσματα:
 - Ανάλυση πηγαίου κώδικα
 - Ξαναγράψιμο πηγαίου κώδικα με αλγόριθμους βελτιστοποίησης
 - Δημιουργία εκτελέσιμου κώδικα

Φάσεις Μετάφρασης (2)



Φάσεις Ανάλυσης πηγαίου προγρ/τος (1)

- Λεξική Ανάλυση (lexical analysis – scanning)
 - Η ακολουθία χαρακτήρων του αρχικού προγράμματος χωρίζεται σε βασικά συντακτικά στοιχεία – *tokens* – όπως: αναγνωριστικά, αριθμοί, σύμβολα, λέξεις κλειδιά, ...
 - Μπαίνει TYPE TAG σε κάθε token
 - Εισάγονται τα tokens στον *Πίνακα Συμβόλων* (symbol table) αφού έχουν μετατραπεί σε κατάλληλη εσωτερική αναπαράσταση
 - Υπολογιστικό μοντέλο λεξικής ανάλυσης:
Πεπερασμένα Αυτόματα (finite-state automata)

Φάσεις *Ανάλυσης* πηγαίου προγρ/τος (2)

- Συντακτική Ανάλυση (syntax analysis – parsing)
 - Αναγνωρίζονται τα συντακτικά στοιχεία υψηλότερου επιπέδου (εκφράσεις, εντολές, υποπρογράμματα)
 - Συνήθως εναλλάσσεται με την επόμενη Φάση (σημασιολογική ανάλυση)
 - Οι δύο Φάσεις «επικοινωνούν» μέσω μιας stack, στην οποία ο συντακτικός αναλυτής τοποθετεί τα στοιχεία που αναγνωρίζει
 - Παράγεται το *Δέντρο Συντακτικής Ανάλυσης* (parse tree) που δίνει και την ιεραρχία των συντακτικών στοιχείων
 - Τύπος Γραμματικής για συντακτική ανάλυση:
Γλώσσες Χωρίς Συμφραζόμενα (context-free languages)

Φάσεις Ανάλυσης πηγαίου προγρ/τος (3)

- Σημασιολογική Ανάλυση (semantic analysis)
 - Κεντρικό τμήμα της μετάφρασης. Είναι η γέφυρα μεταξύ ανάλυσης και σύνθεσης
 - Αρχίζει και διαμορφώνεται η δομή εκτελέσιμου κώδικα
 - Στατικός έλεγχος τύπων και παραμέτρων υποπρ/των
 - Άλλες λειτουργίες:
 - *Συντήρηση Πίνακα Συμβόλων* (χρησιμοποιείται από γλώσσες και στο run time, π.χ. αν έχει δημιουργία μεταβλητών χωρίς ορισμό, ή σε debugging όπως στο dbx του UNIX)
 - *Εισαγωγή ενσωματωμένης (implicit) πληροφορίας* (π.χ. τύπος μεταβλητών της FORTRAN με βάση το αρχικό γράμμα τους)
 - *Εντοπισμός λαθών* (που δεν εντοπίζονται στη συντακτική ανάλυση)
 - *Macro processing* (αντικατάσταση macro με τον κώδικά του)
 - *Compile-time operations*

Φάσεις Σύνθεσης εκτελέσιμου κώδικα (1)

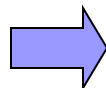
- Δημιουργία Ενδιάμεσου Κώδικα (intermediate code generation)
 - Παράγεται ακολουθία operators – ορισμάτων
- Βελτιστοποίηση κώδικα ανεξάρτητα από μηχανή (machine independent code improvement)
 - Υπολογισμός κοινών υπο-εκφράσεων μία φορά
 - Απομάκρυνση σταθερών λειτουργιών από loops
 - Βελτιστοποίηση της χρήσης registers

Φάσεις Σύνθεσης εκτελέσιμου κώδικα (2)

Παράδειγμα βελτιστοποίησης: Εντολή $A = B + C + D$

Σε «ενδιάμεσο» κώδικα:

- a) $Temp1 = B + C$
- b) $Temp2 = Temp1 + D$
- c) $A = Temp2$



Σε «τελικό» κώδικα:

- 1. Load register with B
- 2. Add C to register
- 3. Store register in Temp1
- 4. Load register with Temp1
- 5. Add D to register
- 6. Store register in Temp2
- 7. Load register with Temp2
- 8. Store register in A

Τα ζευγάρια εντολών 3-4 και 6-7 του «τελικού» κώδικα, μπορούν να αφαιρεθούν - βελτιστοποίηση

Τυπικός ορισμός Συντακτικού

- Στόχος → Ακριβείς ορισμοί συντακτικού των ΓΠ για τους χρήστες και υλοποιητές των ΓΠ
- Επιπλέον → Βάση για τη λεξική και συντακτική ανάλυση από τους μεταφραστές
- Ο Τυπικός (φορμαλιστικός) ορισμός του Συντακτικού μιας ΓΠ ονομάζεται **Γραμματική**
- Μια Γραμματική *αποτελείται* από:
 - Ένα πεπερασμένο σύνολο *συμβόλων* – **Αλφάβητο**
 - Ένα πεπερασμένο σύνολο *ορισμών* (**Κανόνες** ή **Παραγωγές**) που καθορίζουν τις ακολουθίες χαρακτήρων (ή tokens) που δομούν επιτρεπτά προγράμματα της ΓΠ

Τυπική Γραμματική (formal grammar)

- Γραμματική ορισμένη με τη χρήση αυστηρά καθορισμένης *σημειογραφίας* (notation)
- Π.χ. η σημειογραφία **BNF** (Backus Naur Form) για τον ορισμό Γραμματικών χωρίς συμφραζόμενα (context-free grammars)
 - Εμφάνιση στην αναφορά της ALGOL (1960)
 - Την ίδια περίπου εποχή (1959) ο γλωσσολόγος *Noam Chomsky* ανέπτυξε τη θεωρία του για τις Γραμματικές
 - Οι δύο προσεγγίσεις αποδείχτηκαν ισοδύναμες

Τυπικές Γλώσσες (formal languages)

- Το συντακτικό τους ορίζεται από Τυπικές Γραμματικές
- Για μια Τυπική Γλώσσα **L** χρειάζονται:
 1. Ένα Αλφάβητο **Σ** διακριτών συμβόλων
 2. Ένα σύνολο κανόνων **P** που καθορίζουν ποιες ακολουθίες (ή λέξεις) συμβόλων του Σ είναι αποδεκτές στην **L**

Τα παραπάνω 1 και 2 ονομάζονται *Γραμματική στο Σ* .

Δηλαδή, μια Γραμματική **G** είναι ένα ζευγάρι (Σ , **P**) που παράγει τη γλώσσα **L(G)**

Ένα παράδειγμα Τυπικής Γλώσσας

- Η γλώσσα $L(B) = \{0.00 \ 0.01 \ 0.10 \ 0.11\}$ με $B = (\Sigma, P)$ είναι όλοι οι μη-αρνητικοί δυαδικοί αριθμοί μικρότεροι από 1, με δύο υπο-δυαδικά ψηφία.
 - $\Sigma = \{0 \ . \ 1\}$
 - $P = \{$
 - R1: $S_0 \rightarrow 0S_1$
 - R2: $S_1 \rightarrow .S_2$
 - R3: $S_2 \rightarrow 0S_3$
 - R4: $S_2 \rightarrow 1S_3$
 - R5: $S_3 \rightarrow 0$
 - R6: $S_3 \rightarrow 1$ }
- $N = \{ S_0 \ S_1 \ S_2 \ S_3 \} : \text{Συντακτικές Κατηγορίες ή Μη-Τερματικά Σύμβολα}$
- Start** = S_0 : Αρχικό Σύμβολο (start symbol). Η συντακτική κατηγορία του πιο υψηλού επιπέδου
- $\rightarrow$: Ορίζεται ως (μετασύμβολο)

Ισοδύναμη Γραμματική

$$\mathbf{P} = \{ \begin{array}{l} \text{R1: } S_0 \rightarrow 0.S_2 \\ \text{R2: } S_2 \rightarrow 0S_3 \\ \text{R3: } S_2 \rightarrow 1S_3 \\ \text{R4: } S_3 \rightarrow 0 \\ \text{R5: } S_3 \rightarrow 1 \end{array} \}$$

Ένα παράδειγμα Τυπικής Γλώσσας (2)

- Παραγωγή $R: \alpha \rightarrow \beta$

Ένα νέο string παράγεται με τη χρήση της παραγωγής R , αντικαθιστώντας το α με το β

- Δημιουργία (derivation) του string 0.01:

$$S_0 \xRightarrow{R1} 0S_1 \xRightarrow{R2} 0.S_2 \xRightarrow{R3} 0.0S_3 \xRightarrow{R6} 0.01$$

$$R1: S_0 \rightarrow 0S_1$$

$$R2: S_1 \rightarrow .S_2$$

$$R3: S_2 \rightarrow 0S_3$$

$$R4: S_2 \rightarrow 1S_3$$

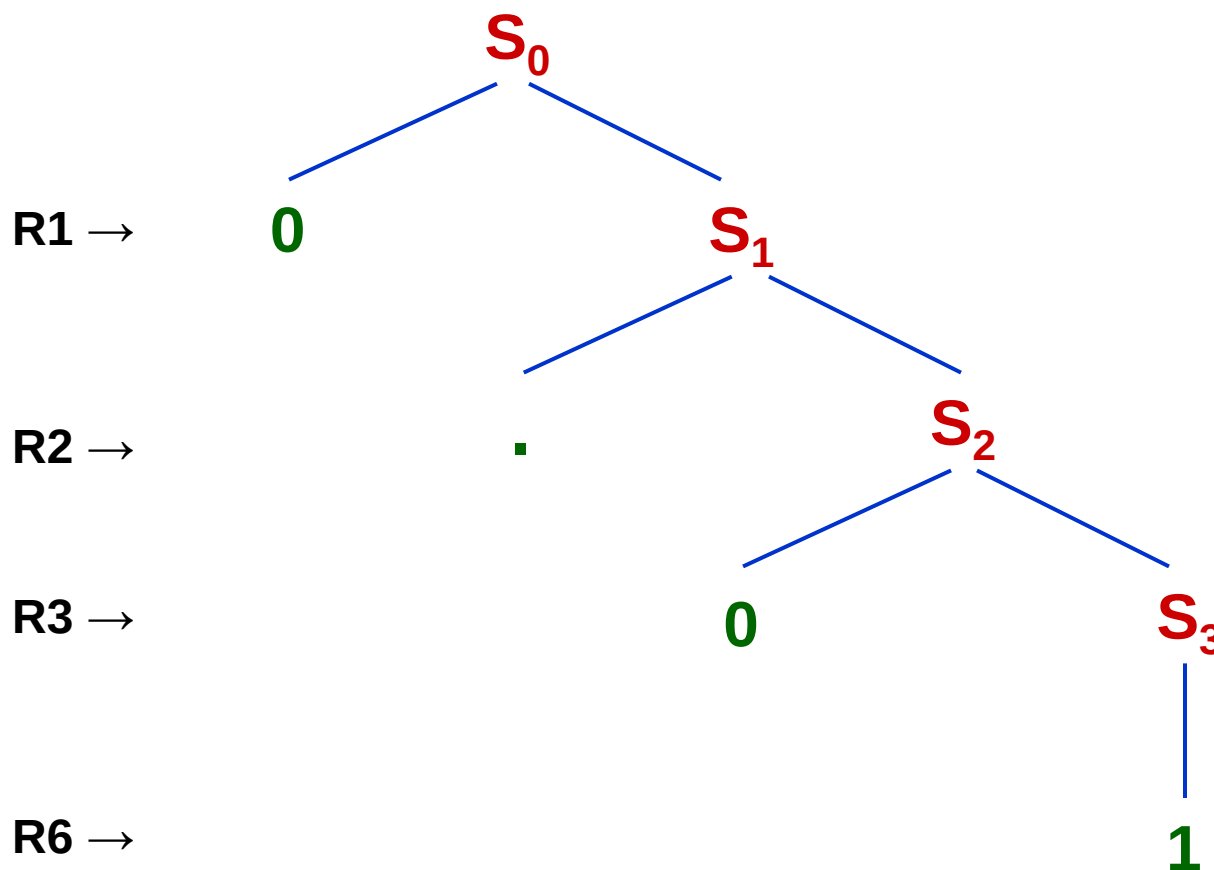
$$R5: S_3 \rightarrow 0$$

$$R6: S_3 \rightarrow 1$$

Πρέπει με διαδοχικές χρήσεις παραγωγών, ξεκινώντας από το start symbol, να καταλήγουμε σε αποδεκτό string της γλώσσας

Ένα παράδειγμα Τυπικής Γλώσσας (3)

Δέντρο Συντακτικής Ανάλυσης του string **0.01**



R1: $S_0 \rightarrow 0S_1$

R2: $S_1 \rightarrow .S_2$

R3: $S_2 \rightarrow 0S_3$

R4: $S_2 \rightarrow 1S_3$

R5: $S_3 \rightarrow 0$

R6: $S_3 \rightarrow 1$

Σύστημα Παραγωγής

$$G = (\Sigma, N, P, \text{Start})$$

όπου $\text{Start} \in \Sigma \cup N$

Ιεραρχία Γραμματικών

- Ανάλογα με το είδος των παραγωγών, υπάρχουν τελικά μόνο 4 τύποι τυπικών γλωσσών
- ΤΥΠΟΣ ΓΛΩΣΣΑΣ $\leftrightarrow$ ΘΕΩΡΗΤΙΚΗ ΜΗΧΑΝΗ
(αναγνωρίζεται από)
- Ερωτήματα:
 - Ποιοι περιορισμοί είναι βασικοί και ποιοι όχι;
 - Με ένα είδος κανόνων, τι προβλήματα λύνονται;
 - Με ένα τύπο γλώσσας, ποιες θεωρητικές μηχανές αναγνωρίζουν τα «νόμιμα» strings;
 - Μπορεί η θεωρητική μηχανή να αναγνωρίσει strings άπειρου μήκους;

Γλώσσες του Chomsky

ΤΥΠΟΣ 0: Γλώσσες χωρίς περιορισμούς

ΤΥΠΟΣ 1: Γλώσσες με συμφραζόμενα

ΤΥΠΟΣ 2: Γλώσσες χωρίς συμφραζόμενα

ΤΥΠΟΣ 3: Κανονικές Γλώσσες

Τύπος 3: Κανονικές Γλώσσες (regular languages)

- Η Γραμματική $G = (\Sigma, N, P, \text{Start})$ είναι **κανονική**, αν οι παραγωγές της είναι της μορφής:
 $A \rightarrow \alpha$ ή $A \rightarrow \alpha B$ όπου $A, B \in N$ και $\alpha \in \Sigma$
- Δηλαδή, το πρώτο σύμβολο δεξιά του $\rightarrow$ είναι τερματικό και μπορεί να ακολουθείται από μη-τερματικό σύμβολο
- Κατάλληλες για λεξική ανάλυση
- Αναγνωρίζονται από πεπερασμένα αυτόματα

Παράδειγμα Κανονικής Γραμματικής

$$X \rightarrow a \mid \dots \mid z \mid aL \mid \dots \mid zL$$
$$L \rightarrow aL \mid \dots \mid zL \mid 0L \mid \dots \mid 9L \mid a \mid \dots \mid z \mid 0 \mid \dots \mid 9$$

| : εναλλακτικός κανόνας – ή (μετασύμβολο)

Περιγράφει ονόματα **X** (π.χ. μεταβλητών) που είναι ένα μικρό γράμμα, ή ένα μικρό γράμμα που ακολουθείται από ακολουθία μικρών γραμμάτων ή/και αριθμητικών ψηφίων.

Π.χ. **a**, **h7**, **kds09u7**

Εναλλακτική Ισοδύναμη Κανονική Γραμματική:

$$X \rightarrow aL \mid \dots \mid zL$$
$$L \rightarrow aL \mid \dots \mid zL \mid 0L \mid \dots \mid 9L \mid a \mid \dots \mid z \mid 0 \mid \dots \mid 9 \mid \epsilon$$

Τύπος 2: Γλώσσες χωρίς συμφραζόμενα (context-free languages)

- Η Γραμματική $G = (\Sigma, N, P, \text{Start})$ είναι **χωρίς συμφραζόμενα**, αν οι παραγωγές της είναι της μορφής:
 $A \rightarrow s$ όπου $A \in N$ και $s \in \Sigma \cup N$
- Ονομάζονται έτσι, διότι αντικαταστάσεις μπορούν να γίνουν οπουδήποτε εμφανίζεται μη-τερματικό σύμβολο, χωρίς να λαμβάνονται υπόψη τα περιβάλλοντα σύμβολα (context)
- Κατάλληλες για *συντακτική ανάλυση*
- Αναγνωρίζονται από *push-down αυτόματα*

Το Παράδειγμα

$X \rightarrow a \mid \dots \mid z \mid aL \mid \dots \mid zL$

$L \rightarrow aL \mid \dots \mid zL \mid 0L \mid \dots \mid 9L \mid a \mid \dots \mid z \mid 0 \mid \dots \mid 9$

μπορεί να γραφεί ως:

$X \rightarrow L \mid XL \mid XD$

$L \rightarrow a \mid \dots \mid z$

$D \rightarrow 0 \mid \dots \mid 9$

Αλλά τώρα δεν είναι κανονική γραμματική...

Είναι όμως context-free.

(Γιατί πρέπει να υπάρχει τώρα το **D**;)

Τύπος 1: Γλώσσες με συμφραζόμενα (context-sensitive languages)

- Η Γραμματική $G = (\Sigma, N, P, \text{Start})$ είναι με συμφραζόμενα, αν οι παραγωγές είναι της μορφής:
 $\alpha \rightarrow \beta$ όπου:
 1. Το α μπορεί να περιέχει περισσότερα από 1 σύμβολα (τουλάχιστον 1 μη-τερματικό)
 2. (Μήκος του α) $\leq$ (Μήκος του β)
- Παράδειγμα: $aB \rightarrow ab, cB \rightarrow bc$
(τα a και c είναι το context)
- Αναγνωρίζονται από *linear-bounded* αυτόματα

Τύπος 0: Γλώσσες χωρίς περιορισμούς

- Η Γραμματική $G = (\Sigma, N, P, \text{Start})$ είναι **χωρίς περιορισμούς**, αν οι παραγωγές είναι της μορφής:

$\alpha \rightarrow \beta$ όπου:

Το α μπορεί να περιέχει περισσότερα από 1 σύμβολα
(τουλάχιστον 1 μη-τερματικό)

- Αναγνωρίζονται από *Μηχανές Turing*

Λεξική Ανάλυση (1)

Αρχικό Πρόγραμμα

Tokens (στον Πίνακα Συμβόλων)

```
x := 37  
/* result */  
RETURN x
```

Λεξική Ανάλυση

```
VARIABLE /x  
ASSIGN SYMBOL  
INTEGER /37  
RETURN  
VARIABLE /x
```

- Μηχανή αναγνώρισης tokens:
Πεπερασμένα Αυτόματα ↔ *Κανονικές Εκφράσεις*
Finite State Automata Regular Expressions
- Ένα Πεπερασμένο Αυτόματο για κάθε είδος token
(μεταβλητές, σύμβολα, λέξεις κλειδιά, ...)

Λεξική Ανάλυση (2)

- Μία **Κανονική Έκφραση** (ΚΕ) είναι ένα από τα:
 - Ένας χαρακτήρας
 - Το **ε** σύμβολο (τίποτα – empty)
 - Δύο ΚΕ η μία δίπλα στην άλλη
 - Δύο ΚΕ χωρισμένες με **|** (ή)
 - Μία ΚΕ ακολουθούμενη από το Kleene Star: **X***
(συνένωση 0 ή περισσότερων Κανονικών Εκφράσεων X)

Π.χ.: **00*11*** ή **0⁺1⁺**

(a|ε)bb* ή **a?b⁺**

Χρησιμοποιούνται επίσης, για συντομία, τα σύμβολα:

- **X⁺** $\Leftrightarrow$ **XX*** (1 ή περισσότερα X)
- **X?** $\Leftrightarrow$ **X|ε** (κανένα ή 1 X)
- **c – g** $\Leftrightarrow$ **c|d|e|f|g**
- **(X)** $\Leftrightarrow$ για καθορισμό προτεραιότητας της ΚΕ

Π.χ.: **Γαρ(ο|ου|υ|ε)φαλλ?άκης**

Λεξική Ανάλυση (3)

- **Κανονικές Γλώσσες:**

Δημιουργούνται από Κανονικές Εκφράσεις και αναγνωρίζονται από *Πεπερασμένα Αυτόματα*

- Για τη δημιουργία *Γλωσσών Χωρίς Συμφραζόμενα*, αρκεί η προσθήκη δυνατότητας *Αναδρομής*.

Λεξική Ανάλυση (4)

■ Παράδειγμα **Κανονικών Εκφράσεων**:

Στην Pascal για την παραγωγή και αναγνώριση αριθμών (π.χ.: 34 8.76 54e22 61.7E-89), οι *Κανονικοί Ορισμοί* (τα παρακάτω ονόματα δεν είναι συντακτικές κατηγορίες, ίσως θα μπορούσαν να γίνουν)

$digit \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$unsigned_integer \longrightarrow digit\ digit^*$

$unsigned_number \longrightarrow unsigned_integer\ ((.\ unsigned_integer) \mid \epsilon)$
 $(((e \mid E) (+ \mid - \mid \epsilon) unsigned_integer) \mid \epsilon)$

είναι αντίστοιχη της συνοπτικής κανονικής έκφρασης:

$(0 - 9)^+ (. (0 - 9)^+)? ((e \mid E)(+ \mid -)? (0 - 9)^+)?$

Λεξική Ανάλυση (5)

- Η αντίστοιχη τυπική γραμματική, έχει τους παραπάνω κανόνες σε αντίθετη σειρά (το *unsigned_number* θα είναι το *start symbol*)
- Αποδεικνύεται ότι ένα σύνολο κανονικών εκφράσεων, είναι αντίστοιχο με *τυπική κανονική γραμματική*.

Λεξική Ανάλυση (6)

■ Πεπερασμένα Αυτόματα

Finite State Automata – FSA

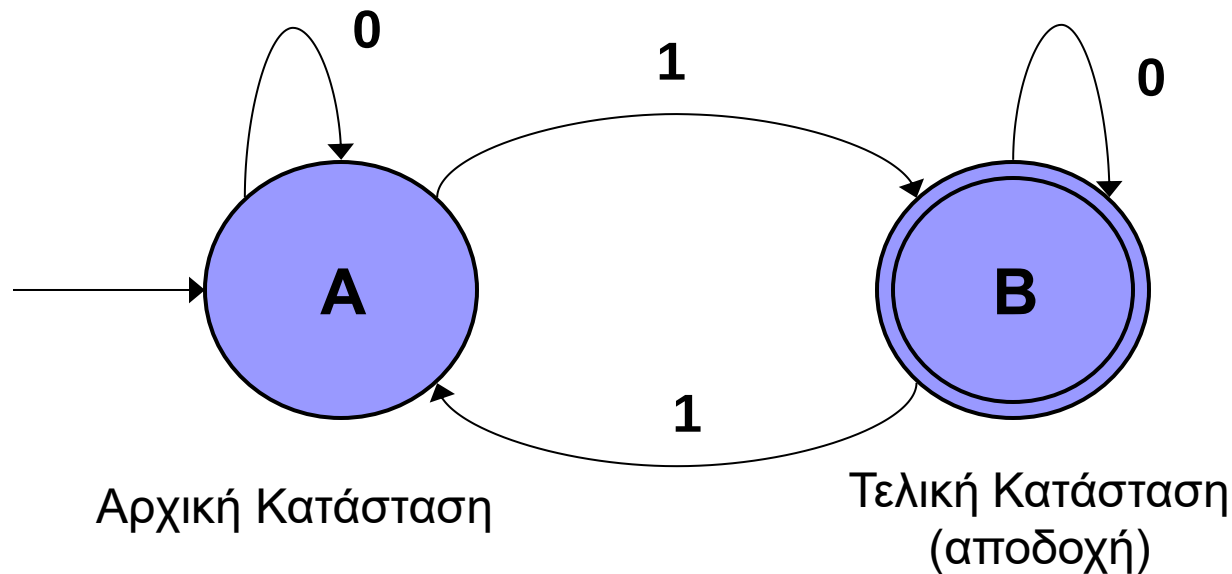
■ Τρόπος λειτουργίας:

- Διαβάζει κάθε φορά ένα χαρακτήρα από αριστερά.
- «Αποφασίζει» **αν το string που έχει διαβάσει μέχρι τώρα**, είναι αποδεκτό ως token. Η απόφαση αυτή καθορίζεται από την «κατάσταση» στην οποία βρίσκεται το αυτόματο μετά την ανάγνωση του τελευταίου χαρακτήρα.
- Όταν αναγνωριστεί ότι τελείωσε η λέξη, το token είναι αποδεκτό αν το αυτόματο βρεθεί σε «τελική κατάσταση». Αν βρεθεί σε άλλη κατάσταση, δεν γίνεται αποδεκτό.

Λεξική Ανάλυση (7)

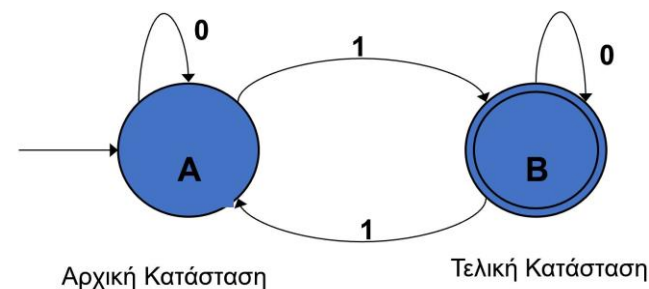
■ Παράδειγμα:

Πεπερασμένο Αυτόματο που αναγνωρίζει
δυαδικούς αριθμούς οι οποίοι έχουν **περιττό**
αριθμό από 1



Αντίστοιχη ΚΕ: $0^*10^*(10^*10^*)^*$ ή $(0^*10^*1)^*0^*10^*$

Λεξική Ανάλυση (8)

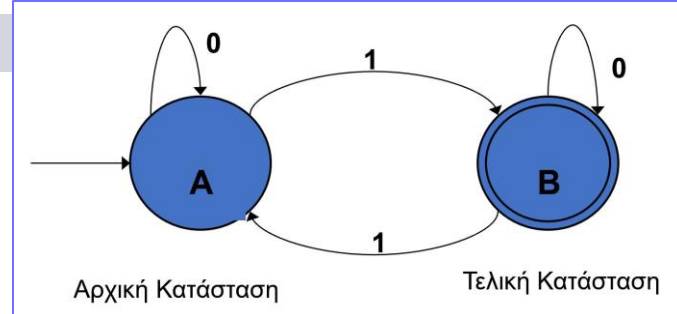


■ Ισοδύναμη αναπαράσταση:

Πίνακας Καταστάσεων – Μεταβάσεων

Τρέχουσα Κατάσταση	Χαρακτήρας που διαβάζεται	Νέα Κατάσταση	Αποδοχή token
A	0	A	ΟΧΙ
A	1	B	ΝΑΙ
B	0	B	ΝΑΙ
B	1	A	ΟΧΙ

Λεξική Ανάλυση (9)



- Λειτουργία του FSA για την είσοδο **100101**:

Αναγνωσμένοι χαρακτήρες	Νέα Κατάσταση	Αποδοχή token
	A	ΟΧΙ
1	B	ΝΑΙ
10	B	ΝΑΙ
100	B	ΝΑΙ
1001	A	ΟΧΙ
10010	A	ΟΧΙ
100101	B	ΝΑΙ

Λεξική Ανάλυση (10)

- Ένα Πεπερασμένο Αυτόματο έχει:
 - Μία **αρχική** κατάσταση
 - Μία ή περισσότερες **τελικές** καταστάσεις
 - Ένα σύνολο **μεταβάσεων**
- Κάθε string που ξεκινάει το Πεπερασμένο Αυτόματο από την αρχική κατάσταση, και τελειώνει σε μια τελική κατάσταση, είναι **αποδεκτό** token.
- Τα Πεπερασμένα Αυτόματα που χρησιμοποιούμε είναι **ντετερμινιστικά**.
- **Μη-ντετερμινιστικά** είναι αυτά που έχουν περισσότερες από μία μεταβάσεις με το ίδιο label.

Λεξική Ανάλυση (11)

■ Υλοποίηση:

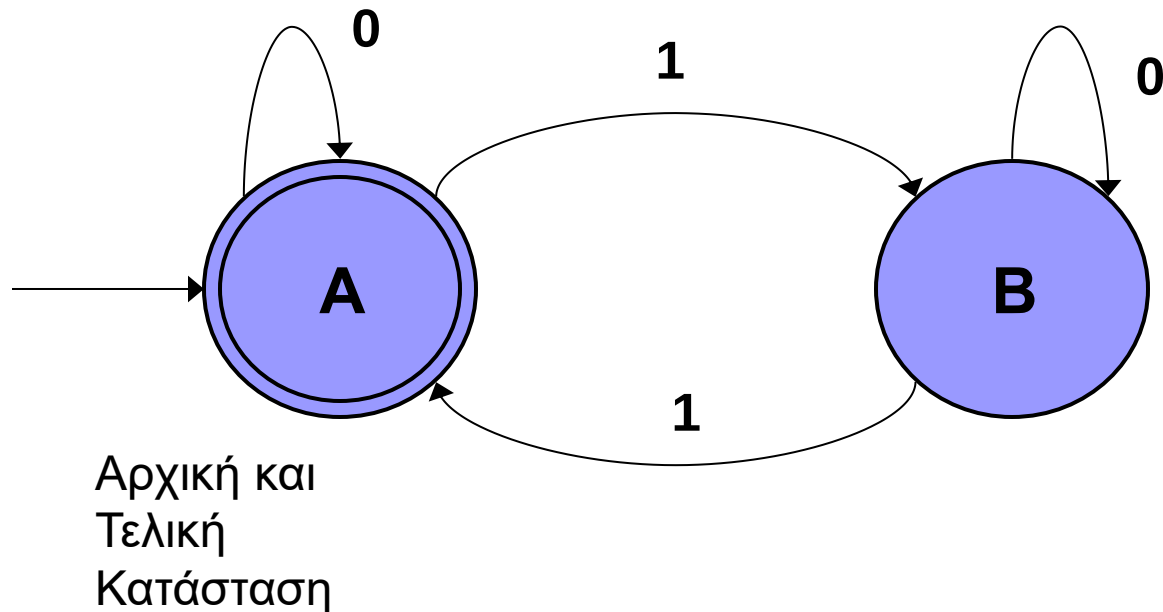
- Κάθε κατάσταση είναι μια ρουτίνα του προγράμματος που υλοποιεί το FSA
- Ο λεξικός αναλυτής ξεκινά με GETCHAR από αριστερά, καλώντας ένα FSA
- Αν ο λεξικός αναλυτής δεν φτάσει σε τελική κατάσταση του τρέχοντος FSA, επιστρέφει στον αρχικό χαρακτήρα και ξεκινά ένα άλλο FSA

■ Στην Άσκηση θα χρησιμοποιήσουμε το γεννήτορα λεξικών αναλυτών **flex** που βασίζεται σε FSAs

Λεξική Ανάλυση (12)

■ 2ο Παράδειγμα:

Πεπερασμένο Αυτόματο που αναγνωρίζει
δυαδικούς αριθμούς οι οποίοι έχουν **άρτιο** αριθμό
από 1



Αντίστοιχη ΚΕ: $0^*(10^*10^*)^*$ ή $(0^*10^*1)^*0^*$

Λεξική Ανάλυση (13)

Πίνακας Καταστάσεων – Μεταβάσεων

Τρέχουσα Κατάσταση	Χαρακτήρας που διαβάζεται	Νέα Κατάσταση	Αποδοχή token
A	0	A	NAI
A	1	B	OXI
B	0	B	OXI
B	1	A	NAI

Λεξική Ανάλυση (14)

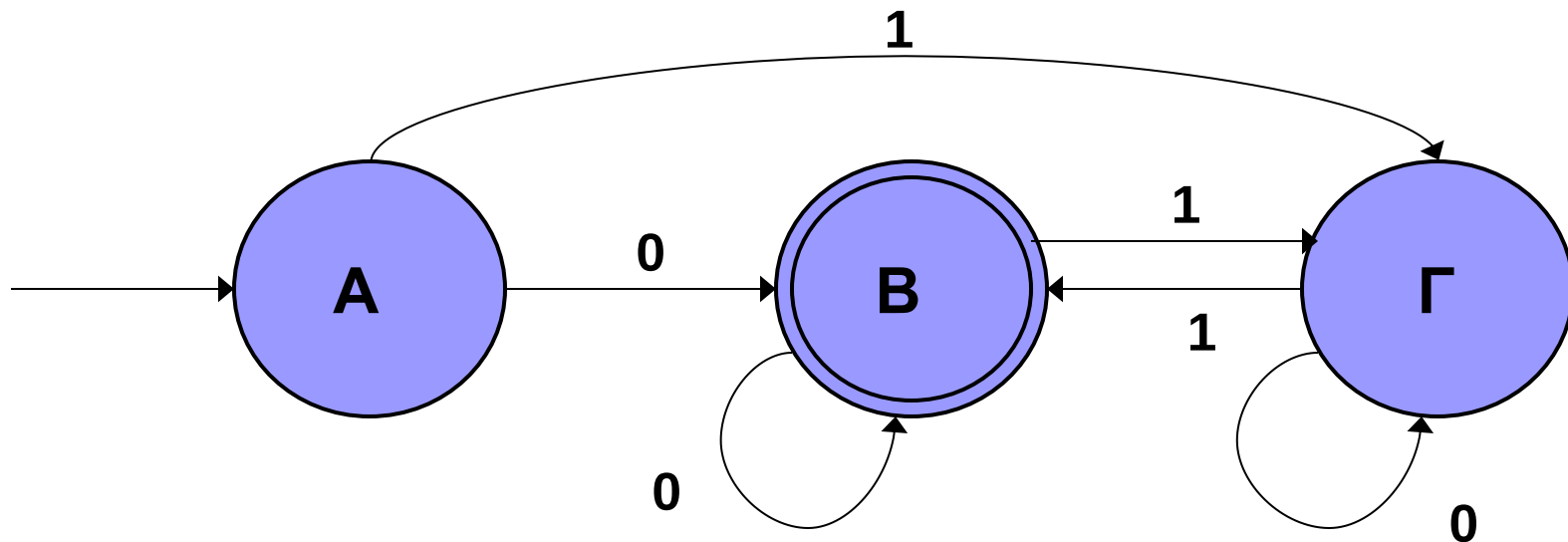
- Λειτουργία του FSA για την είσοδο **100101**:

Αναγνωσμένοι χαρακτήρες	Νέα Κατάσταση	Αποδοχή token
	A	ΝΑΙ
1	B	ΟΧΙ
10	B	ΟΧΙ
100	B	ΟΧΙ
1001	A	ΝΑΙ
10010	A	ΝΑΙ
100101	B	ΟΧΙ

Λεξική Ανάλυση (15)

■ 3ο Παράδειγμα:

Πεπερασμένο Αυτόματο που αναγνωρίζει
δυαδικούς αριθμούς οι οποίοι έχουν **άρτιο** αριθμό
από 1, χωρίς το ϵ :

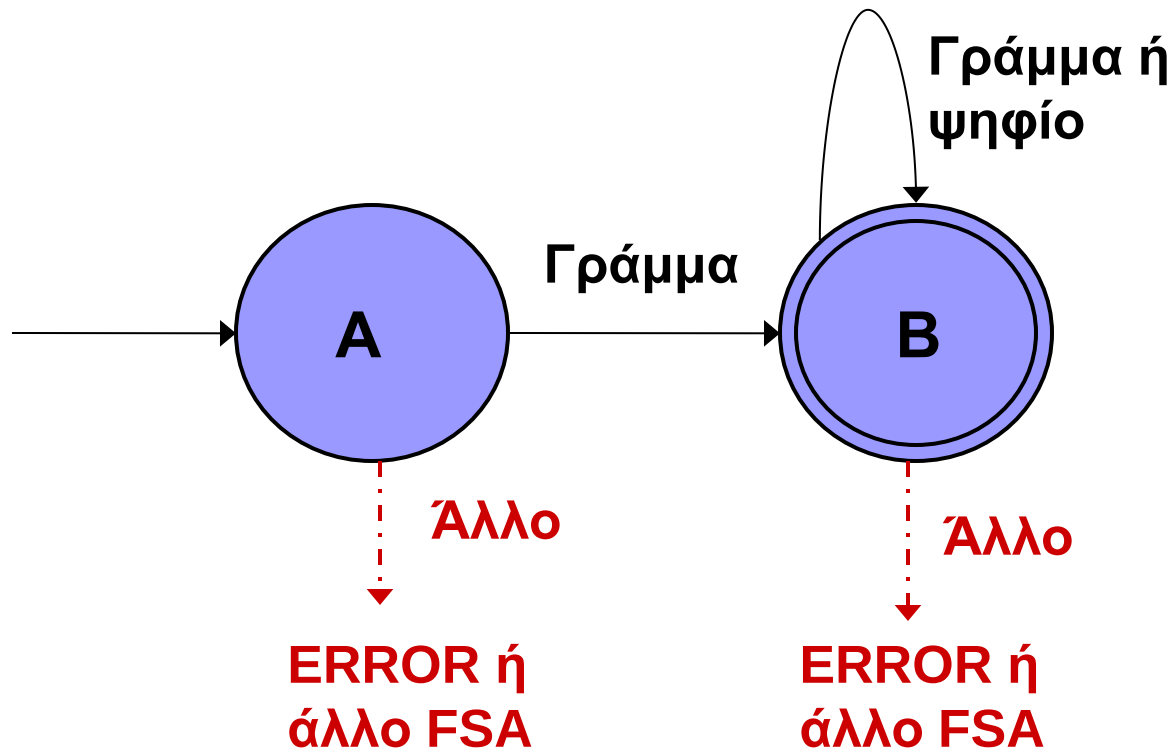


Αντίστοιχη ΚΕ: $0^+ \mid (0^*10^*10^*)^+$

Λεξική Ανάλυση (16)

■ 4ο Παράδειγμα:

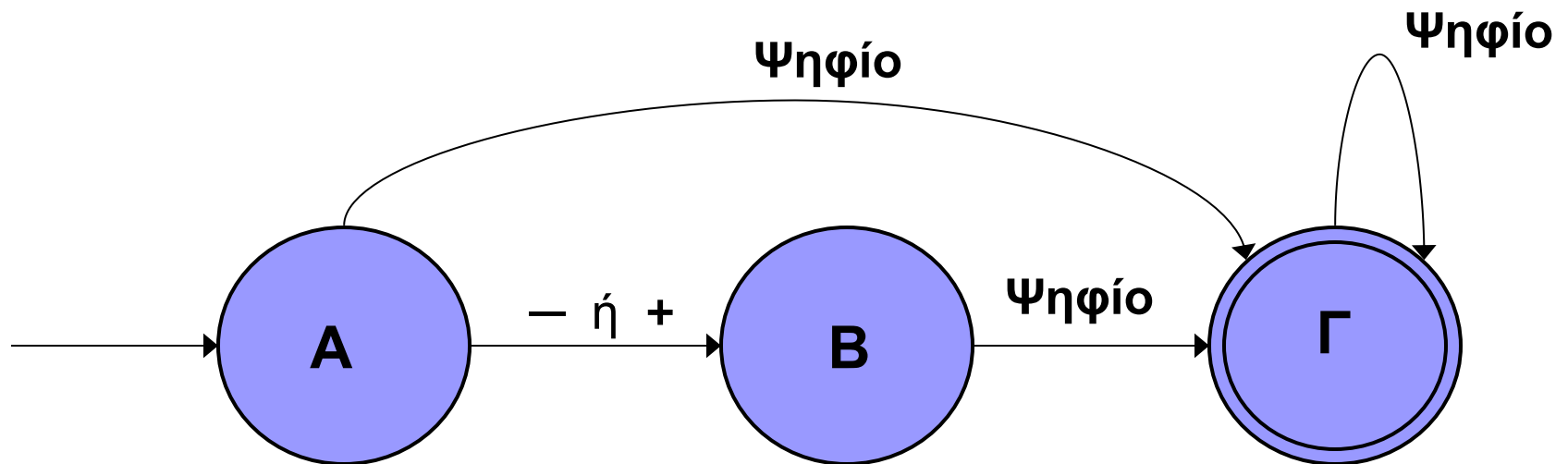
Πεπερασμένο Αυτόματο που αναγνωρίζει **ονόματα μεταβλητών** που αρχίζουν με γράμμα και αποτελούνται από γράμματα και αριθμητικά ψηφία



Λεξική Ανάλυση (17)

■ 5ο Παράδειγμα:

Πεπερασμένο Αυτόματο που αναγνωρίζει
προσημασμένους ή μη-προσημασμένους
ακέραιους αριθμούς



Αντίστοιχη ΚΕ: $(+ | -)? (0 - 9)^+$

Γλώσσες του Chomsky

ΤΥΠΟΣ 0: Γλώσσες χωρίς περιορισμούς

ΤΥΠΟΣ 1: Γλώσσες με συμφραζόμενα

ΤΥΠΟΣ 2: Γλώσσες χωρίς συμφραζόμενα

ΤΥΠΟΣ 3: Κανονικές Γλώσσες

Τύπος 2: Γλώσσες χωρίς συμφραζόμενα (context-free languages)

- Η Γραμματική $G = (\Sigma, N, P, \text{Start})$ είναι **χωρίς συμφραζόμενα**, αν οι παραγωγές της είναι της μορφής:
$$A \rightarrow s \quad \text{όπου } A \in N \text{ και } s \in \Sigma \cup N$$
- Ονομάζονται έτσι, διότι αντικαταστάσεις μπορούν να γίνουν οπουδήποτε εμφανίζεται μη-τερματικό σύμβολο, χωρίς να λαμβάνονται υπόψη τα περιβάλλοντα σύμβολα (context)
- Κατάλληλες για *συντακτική ανάλυση*
- Αναγνωρίζονται από *push-down αυτόματα*

Συντακτική Ανάλυση (1)

- Για τη Συντακτική Ανάλυση είναι κατάλληλες οι Γραμματικές *Χωρίς Συμφραζόμενα*
- Τυπική σημειογραφία **BNF** (Backus Naur Form) για τον ορισμό Γραμματικών Χωρίς Συμφραζόμενα (context-free grammars)
- **Μετασύμβολα** της σημειογραφίας BNF, για τον ορισμό των κανόνων της Γλώσσας:
 - $::=$ ορίζεται ως
 - $|$ εναλλακτικός κανόνας – ή
 - $< \dots >$ Μη-τερματικό σύμβολο (συντακτική κατηγορία)

Συντακτική Ανάλυση (2)

- Και η *Λεξική Ανάλυση* θα μπορούσε να είναι μέρος της Συντακτικής. Π.χ. η BNF Γραμματική:

<ψηφίο> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

η οποία ορίζει τον τύπο token «αριθμητικό ψηφίο».

- Διότι οι Κανονικές Γλώσσες (λεξική ανάλυση), είναι υποσύνολο των Γλωσσών Χωρίς Συμφραζόμενα (συντακτική ανάλυση).
- Με τις Γλώσσες Χωρίς Συμφραζόμενα, μπορούμε να περιγράψουμε πιο σύνθετες γλώσσες. Π.χ.

<υπο-συνθήκη εντολή> ::= if <λογική έκφραση> then <εντολή> else <εντολή> | if <λογική έκφραση> then <εντολή>

Συντακτική Ανάλυση (3)

- Χρήσιμη μορφή κανόνα, η αναδρομή:

$\langle \text{μη-προσ. ακέραιος} \rangle ::= \langle \text{ψηφίο} \rangle \mid \langle \text{μη-προσ. ακέραιος} \rangle \langle \text{ψηφίο} \rangle$
 $\langle \text{ψηφίο} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

- Μία ακόμα BNF Γραμματική:

$\langle A \rangle ::= 0 \mid \langle B \rangle \mid 0 \langle B \rangle$

$\langle B \rangle ::= \langle B \rangle \langle C \rangle \mid \langle D \rangle$

$\langle C \rangle ::= 0 \mid \langle D \rangle$

$\langle D \rangle ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Τώρα επιτρέπεται μόνο ένα 0 στην αρχή του αριθμού...

(Υπάρχει ΚΕ που εκφράζει το τελευταίο: $0 \mid (1-9)(0-9)^* \mid 0(1-9)(0-9)^*$
Άρα υπάρχει και ισοδύναμη κανονική γραμματική...)

Ισοδύναμη Κανονική Γραμματική

$\langle S \rangle ::= 0 \mid 0\langle A \rangle \mid 1\langle D \rangle \mid 2\langle D \rangle \mid \dots \mid 9\langle D \rangle$

$\langle A \rangle ::= 1\langle D \rangle \mid 2\langle D \rangle \mid \dots \mid 9\langle D \rangle$

$\langle D \rangle ::= 1\langle D \rangle \mid 2\langle D \rangle \mid \dots \mid 9\langle D \rangle \mid 0\langle D \rangle \mid \epsilon$

Και εναλλακτική ΚΕ:

$0((1-9)(0-9)^*)? \mid (1-9)(0-9)^*$

Ισοδύναμη Κανονική Γραμματική της αρχικής:

$\langle S \rangle ::= 0\langle D \rangle \mid 1\langle D \rangle \mid 2\langle D \rangle \mid \dots \mid 9\langle D \rangle$

$\langle D \rangle ::= 0\langle D \rangle \mid 1\langle D \rangle \mid 2\langle D \rangle \mid \dots \mid 9\langle D \rangle \mid \epsilon$

Συντακτική Ανάλυση (4)

- Μία μικρή BNF Γραμματική:

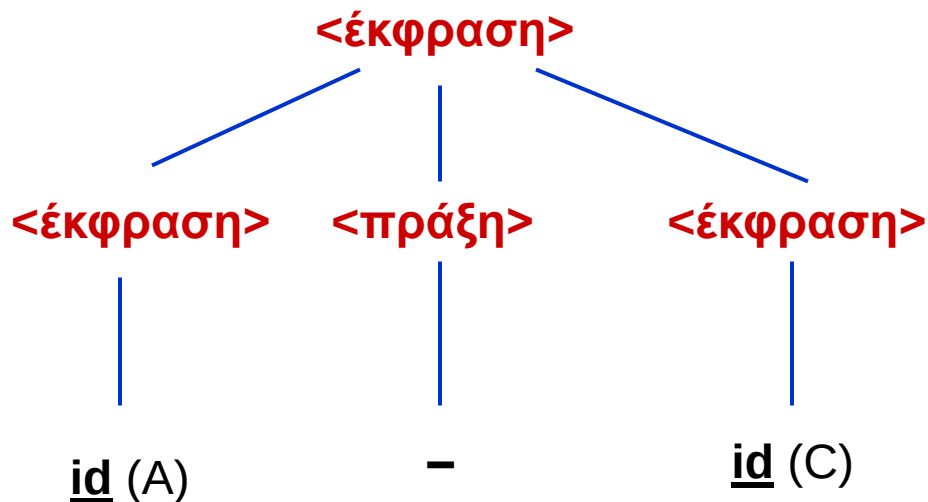
$\langle \text{έκφραση} \rangle ::= \underline{\text{id}} \mid \underline{\text{αρ}} \mid \langle \text{έκφραση} \rangle \langle \text{πράξη} \rangle \langle \text{έκφραση} \rangle \mid$
 $\mid (\langle \text{έκφραση} \rangle) \mid - \langle \text{έκφραση} \rangle$
 $\langle \text{πράξη} \rangle ::= + \mid - \mid * \mid / \mid \uparrow$

όπου: id αρ είναι είδη token αναγνωρισμένα στη λεξική ανάλυση
(id = αναγνωριστικό/όνομα, αρ = αριθμός)
 $\langle \text{έκφραση} \rangle$ = start symbol

Συντακτική Ανάλυση (5)

- Αναγνώριση σωστών προγραμμάτων:
Top – down Parsing → Δέντρο Συντακτικής
Ανάλυσης (Parse Tree)
- Π.χ. για την έκφραση $A - C$

$\langle \text{έκφραση} \rangle ::= \underline{\text{id}} \mid \underline{\text{ap}} \mid \langle \text{έκφραση} \rangle \langle \text{πράξη} \rangle \langle \text{έκφραση} \rangle \mid$
 $\mid (\langle \text{έκφραση} \rangle) \mid - \langle \text{έκφραση} \rangle$
 $\langle \text{πράξη} \rangle ::= + \mid - \mid * \mid / \mid \uparrow$



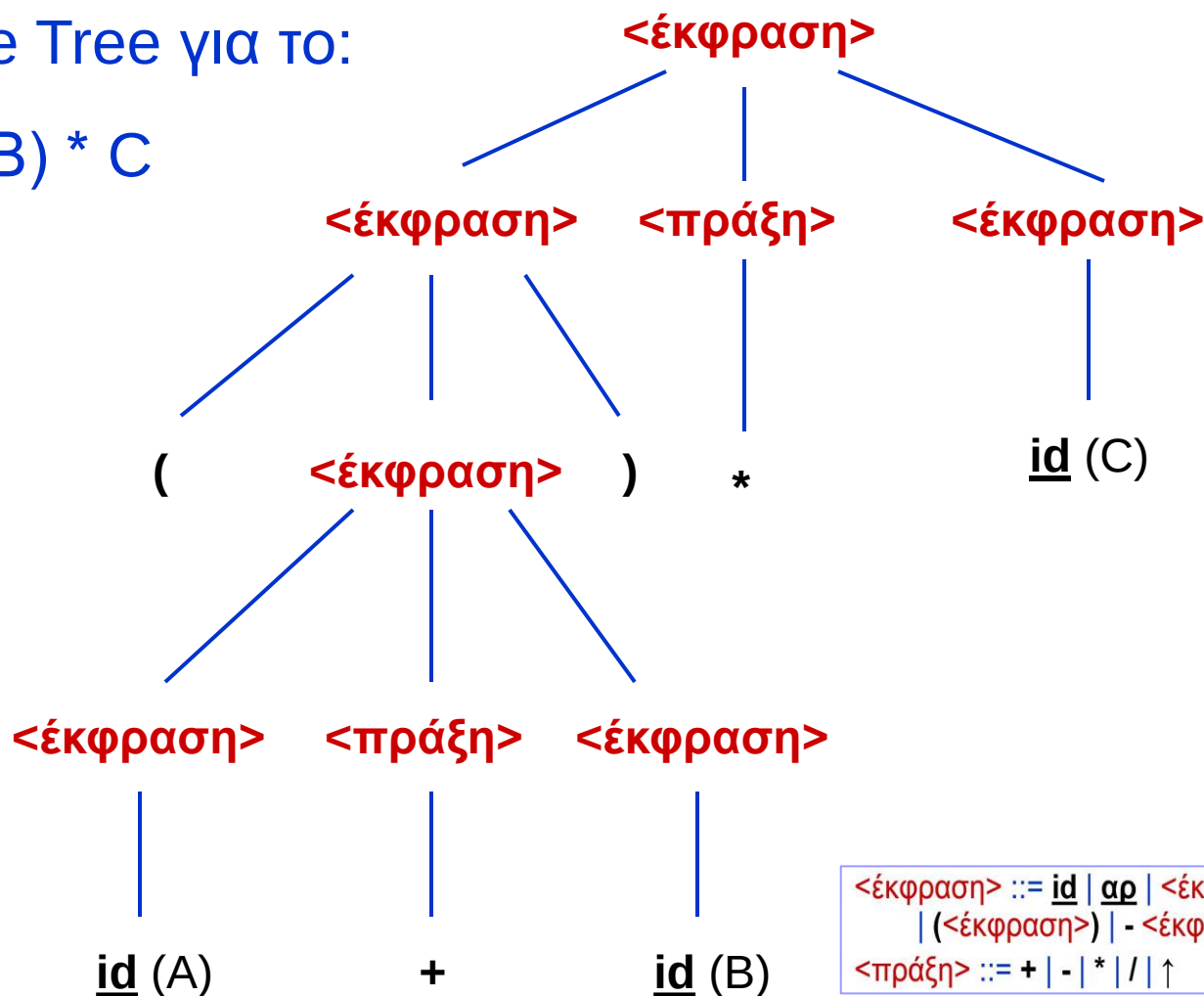
Συντακτική Ανάλυση (6)

- Πρέπει, ξεκινώντας με ρίζα το start symbol, να δημιουργείται ένα δέντρο, που φύλλα του είναι η ακολουθία χαρακτήρων που θέλουμε να αναγνωριστεί.
- Οι ακολουθίες χαρακτήρων που μπορούν να δημιουργηθούν από το start symbol με κάποιο parse tree, συγκροτούν τη Γλώσσα που ορίζει η Γραμματική.
- Το Δέντρο Συντακτικής Ανάλυσης για την αναγνώριση της έκφρασης $(A + B) * C$:

Συντακτική Ανάλυση (7)

Parse Tree για το:

$(A + B) * C$



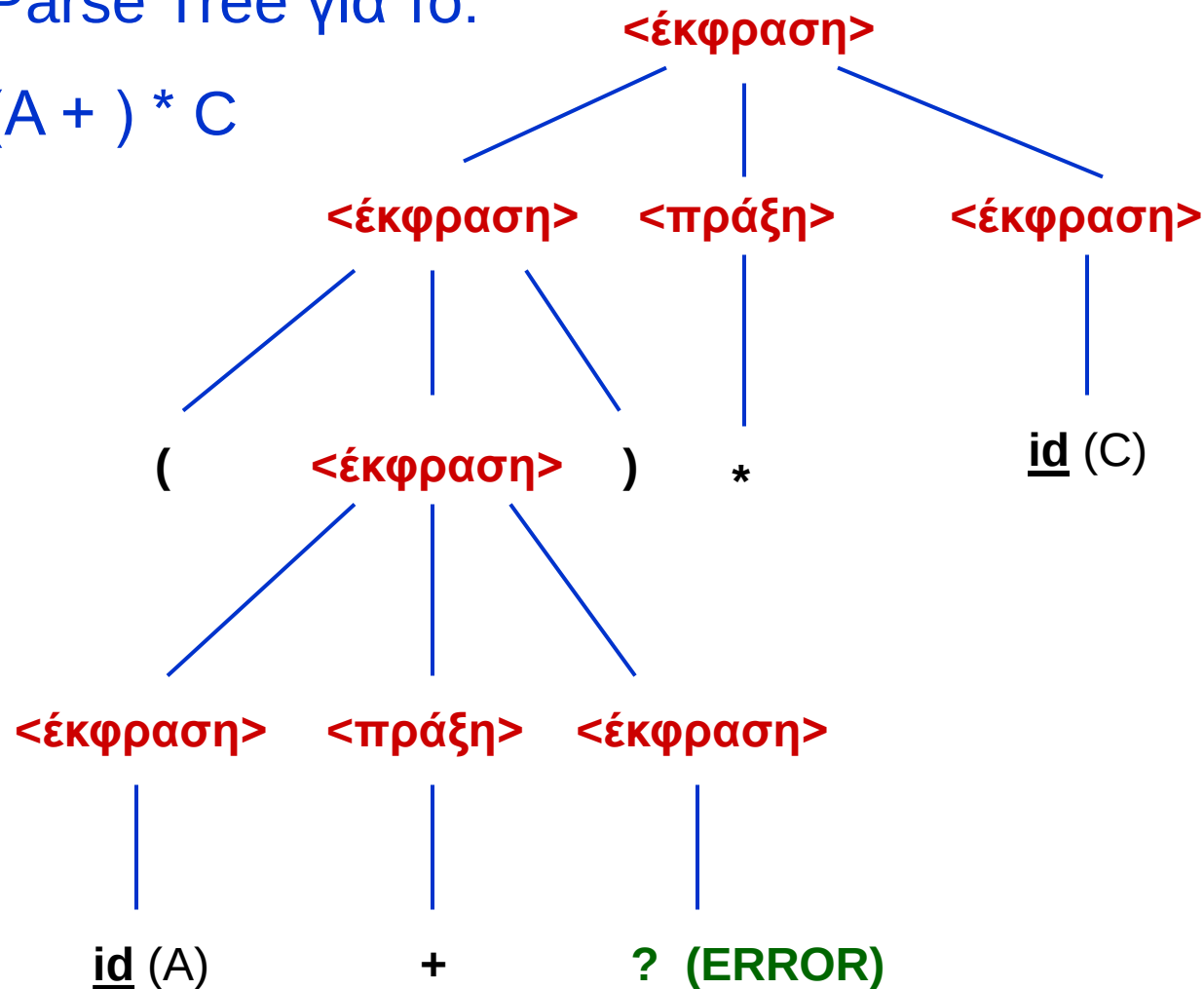
**<έκφραση> ::= id | αρ | <έκφραση> <πράξη> <έκφραση> |
| (<έκφραση>) | - <έκφραση>
<πράξη> ::= + | - | * | / | ↑**

Συντακτική Ανάλυση (8)

$\langle \text{έκφραση} \rangle ::= \underline{\text{id}} \mid \underline{\text{op}} \mid \langle \text{έκφραση} \rangle \langle \text{πράξη} \rangle \langle \text{έκφραση} \rangle \mid (\langle \text{έκφραση} \rangle) \mid - \langle \text{έκφραση} \rangle$
 $\langle \text{πράξη} \rangle ::= + \mid - \mid * \mid / \mid \uparrow$

Parse Tree για το:

$(A +) * C$



Συντακτική Ανάλυση (9)

Μια πιο σύνθετη Γραμματική:

$\langle A \rangle ::= \langle E \rangle = \langle B \rangle$

$\langle B \rangle ::= \langle C \rangle \mid \langle B \rangle + \langle C \rangle \mid \langle B \rangle - \langle C \rangle$

$\langle C \rangle ::= \langle D \rangle \mid \langle C \rangle * \langle D \rangle \mid \langle C \rangle / \langle D \rangle$

$\langle D \rangle ::= \langle E \rangle \mid \underline{\alpha\rho} \mid (\langle B \rangle)$

$\langle E \rangle ::= \underline{id} \mid \underline{id} [\langle F \rangle]$

$\langle F \rangle ::= \langle B \rangle \mid \langle F \rangle , \langle B \rangle$

Συντακτική Ανάλυση (10)

Είναι μια Γραμματική για εντολές ανάθεσης:

<εντολή ανάθεσης> ::= <μεταβλητή> = <έκφραση>

**<έκφραση> ::= <όρος> | <έκφραση> + <όρος> |
| <έκφραση> - <όρος>**

**<όρος> ::= <παράγοντας> | <όρος> * <παράγοντας> |
| <όρος> / <παράγοντας>**

<παράγοντας> ::= <μεταβλητή> | αρ | (<έκφραση>)

<μεταβλητή> ::= id | id [<λίστα δεικτών>]

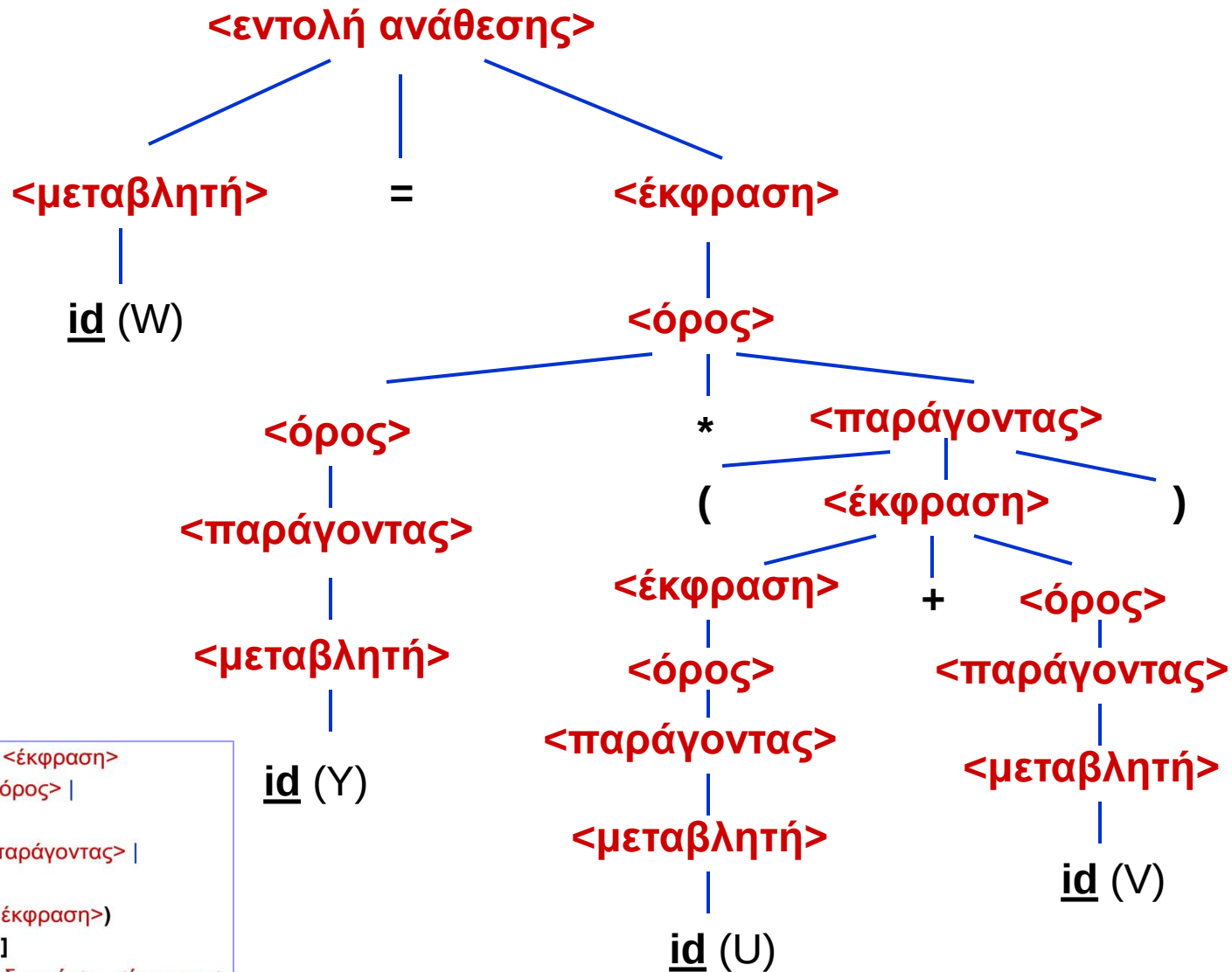
<λίστα δεικτών> ::= <έκφραση> | <λίστα δεικτών> , <έκφραση>

Π.χ. $A = B + K - 3$ $C[3, 2N - 4] = L[(3 * P[X] - H) + 67 / D[9]] + 4$

Συντακτική Ανάλυση (11)

Δέντρο
Συντακτικής
Ανάλυσης
για το

$W=Y*(U+V)$



<εντολή ανάθεσης> ::= <μεταβλητή> = <έκφραση>
 <έκφραση> ::= <όρος> | <έκφραση> + <όρος> |
 | <έκφραση> - <όρος>
 <όρος> ::= <παράγοντας> | <όρος> * <παράγοντας> |
 | <όρος> / <παράγοντας>
 <παράγοντας> ::= <μεταβλητή> | ap | (<έκφραση>)
 <μεταβλητή> ::= id | id [<λίστα δεικτών>]
 <λίστα δεικτών> ::= <έκφραση> | <λίστα δεικτών> , <έκφραση>

Συντακτική Ανάλυση (12)

<εντολή ανάθεσης> ::= <μεταβλητή> = <έκφραση>
<έκφραση> ::= <όρος> | <έκφραση> + <όρος> |
| <έκφραση> - <όρος>
<όρος> ::= <παράγοντας> | <όρος> * <παράγοντας> |
| <όρος> / <παράγοντας>
<παράγοντας> ::= <μεταβλητή> | αρ | (<έκφραση>)
<μεταβλητή> ::= id | id [<λίστα δεικτών>]
<λίστα δεικτών> ::= <έκφραση> | <λίστα δεικτών> , <έκφραση>

Εναλλακτική Γραμματική για την ίδια Γλώσσα:

<εντολή ανάθεσης> ::= <μεταβλητή> = <έκφραση>
<έκφραση> ::= <όρος> | <έκφραση> * <όρος> |
| <έκφραση> + <όρος>
<όρος> ::= <παράγοντας> | <όρος> - <παράγοντας> |
| <όρος> / <παράγοντας>
<παράγοντας> ::= <μεταβλητή> | αρ | (<έκφραση>)
<μεταβλητή> ::= id | id [<λίστα δεικτών>]
<λίστα δεικτών> ::= <έκφραση> | <λίστα δεικτών> , <έκφραση>

Συντακτική Ανάλυση (13)

- **Extended BNF** σημειογραφία.
- Επιπλέον *μετασύμβολα*:

[...] : Προαιρετικό

{ ... } : 0 ή περισσότερες εμφανίσεις

(...) : Ομαδοποίηση

```
<εντολή ανάθεσης> ::= <μεταβλητή> = <έκφραση>  
<έκφραση> ::= <όρος> | <έκφραση> + <όρος> |  
| <έκφραση> - <όρος>  
<όρος> ::= <παράγοντας> | <όρος> * <παράγοντας> |  
| <όρος> / <παράγοντας>  
<παράγοντας> ::= <μεταβλητή> | αρ | (<έκφραση>)  
<μεταβλητή> ::= id | id [<λίστα δεικτών>]  
<λίστα δεικτών> ::= <έκφραση> | <λίστα δεικτών> , <έκφραση>
```

<εντολή ανάθεσης> ::= <μεταβλητή> = <έκφραση>

<έκφραση> ::= <όρος> { (+ | -) <όρος> }

<όρος> ::= <παράγοντας> { (* | /) <παράγοντας> }

<παράγοντας> ::= <μεταβλητή> | αρ | (<έκφραση>)

<μεταβλητή> ::= id | id [<λίστα δεικτών>]

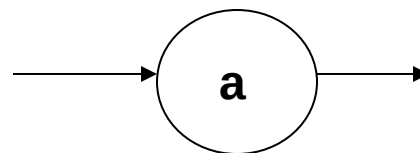
<λίστα δεικτών> ::= <έκφραση> { , <έκφραση> }

Μειονέκτημα: Μερικές φορές κρύβεται η αναδρομή

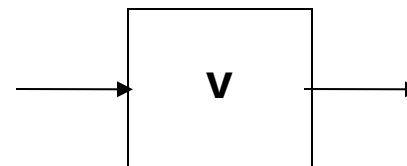
Συντακτική Ανάλυση (14)

- Ισοδύναμη μέθοδος περιγραφής του συντακτικού με την BNF: **Συντακτικά Διαγράμματα.**
- Κανόνες μετατροπής BNF σε Συντακτικά Διαγράμματα, στην περιγραφή της Pascal από τον Wirth, 1976:

1. Τερματικό Σύμβολο **a**

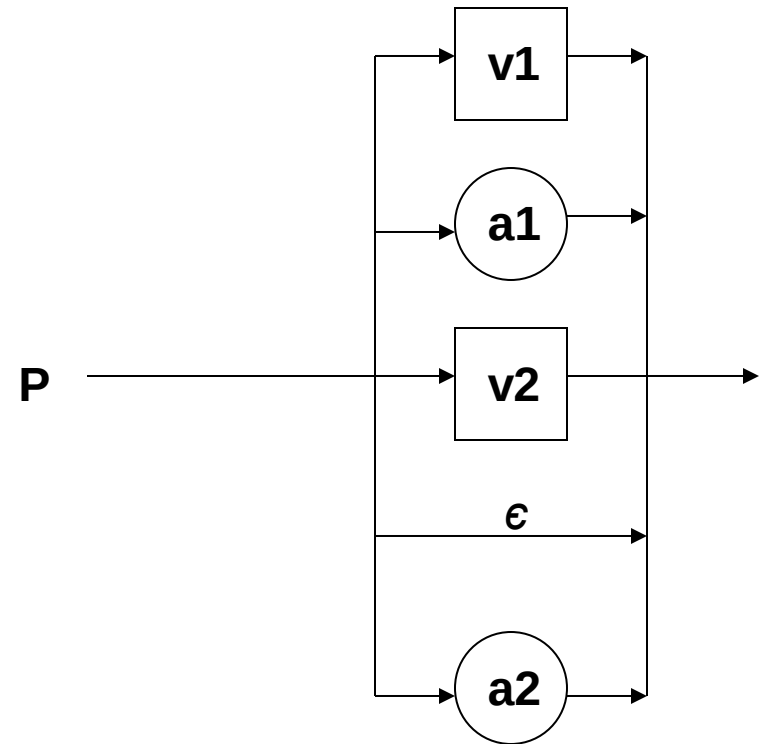


2. Μη-τερματικό Σύμβολο **<v>**



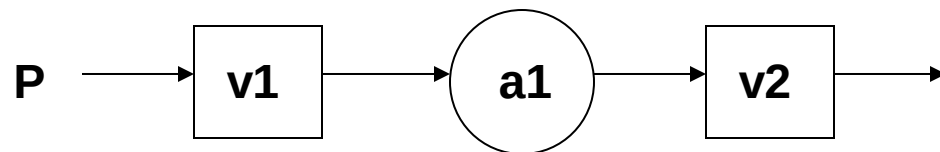
Συντακτική Ανάλυση (15)

3. $\langle P \rangle ::= \langle v1 \rangle \mid a1 \mid \langle v2 \rangle \mid \epsilon \mid a2$



Συντακτική Ανάλυση (16)

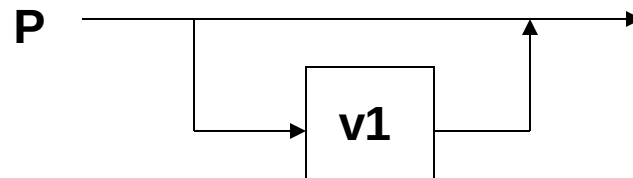
4. $\langle P \rangle ::= \langle v1 \rangle a1 \langle v2 \rangle$



5. $\langle P \rangle ::= [\langle v1 \rangle]$

0 ή 1 φορές το $\langle v1 \rangle$

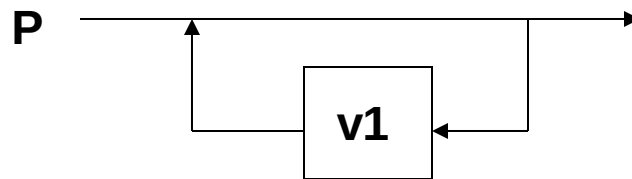
δηλ. $\langle P \rangle ::= \epsilon \mid \langle v1 \rangle$



6. $\langle P \rangle ::= \{ \langle v1 \rangle \}$

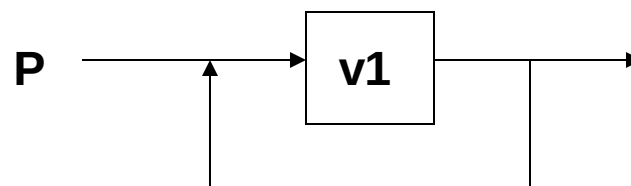
0 ή περισσότερες φορές το $\langle v1 \rangle$

δηλ. $\langle P \rangle ::= \epsilon \mid \langle v1 \rangle \langle P \rangle$



7. $\langle P \rangle ::= \langle v1 \rangle \mid \langle v1 \rangle \langle P \rangle$

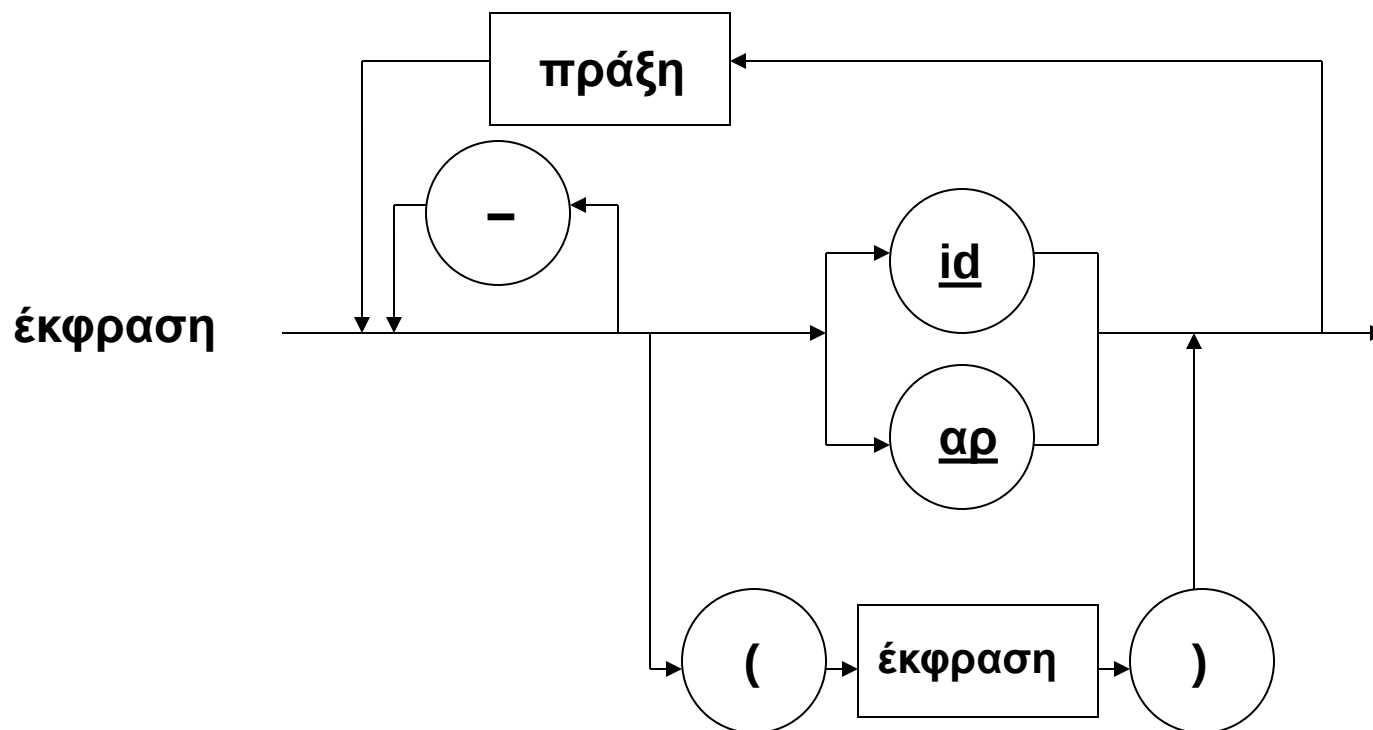
1 ή περισσότερες φορές το $\langle v1 \rangle$



Συντακτική Ανάλυση (17)

■ Το παράδειγμα

$\langle \text{έκφραση} \rangle ::= \underline{\text{id}} \mid \underline{\text{αρ}} \mid \langle \text{έκφραση} \rangle \langle \text{πράξη} \rangle \langle \text{έκφραση} \rangle \mid (\langle \text{έκφραση} \rangle) \mid - \langle \text{έκφραση} \rangle$



Συντακτική Ανάλυση (18)

Η σύνθετη Γραμματική για εντολές ανάθεσης:

<εντολή ανάθεσης> ::= <μεταβλητή> = <έκφραση>

**<έκφραση> ::= <όρος> | <έκφραση> + <όρος> |
| <έκφραση> - <όρος>**

**<όρος> ::= <παράγοντας> | <όρος> * <παράγοντας> |
| <όρος> / <παράγοντας>**

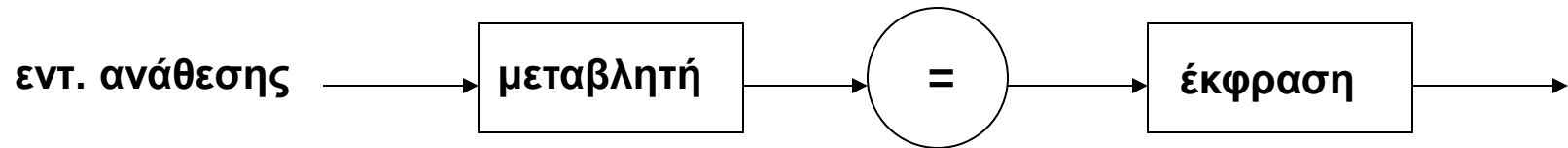
<παράγοντας> ::= <μεταβλητή> | αρ | (<έκφραση>)

<μεταβλητή> ::= id | id [<λίστα δεικτών>]

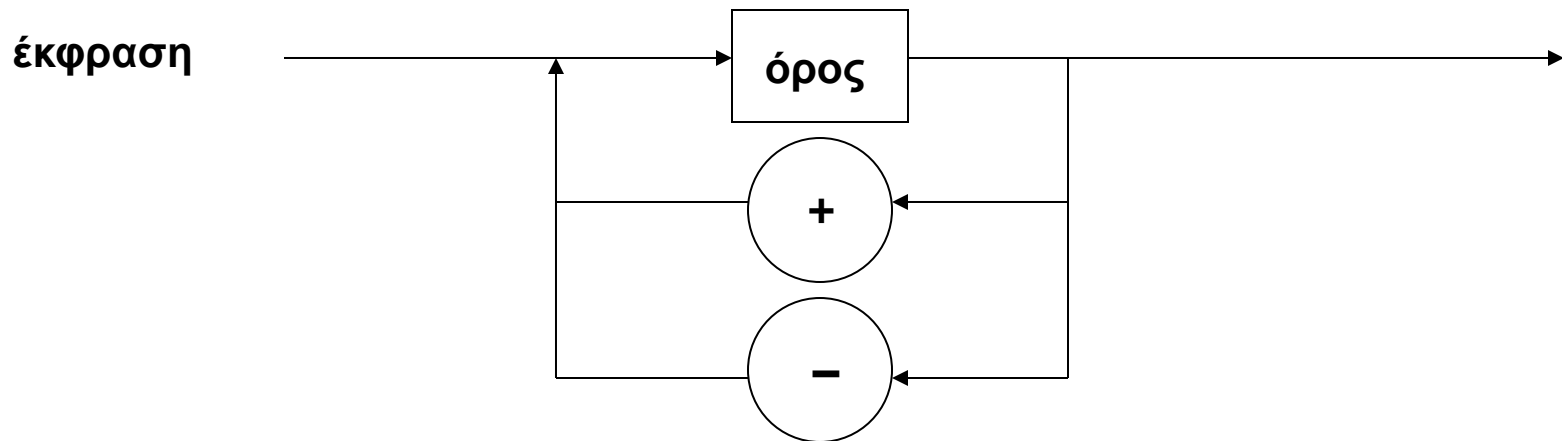
<λίστα δεικτών> ::= <έκφραση> | <λίστα δεικτών> , <έκφραση>

Συντακτική Ανάλυση (19)

$\langle \text{εντολή ανάθεσης} \rangle ::= \langle \text{μεταβλητή} \rangle = \langle \text{έκφραση} \rangle$

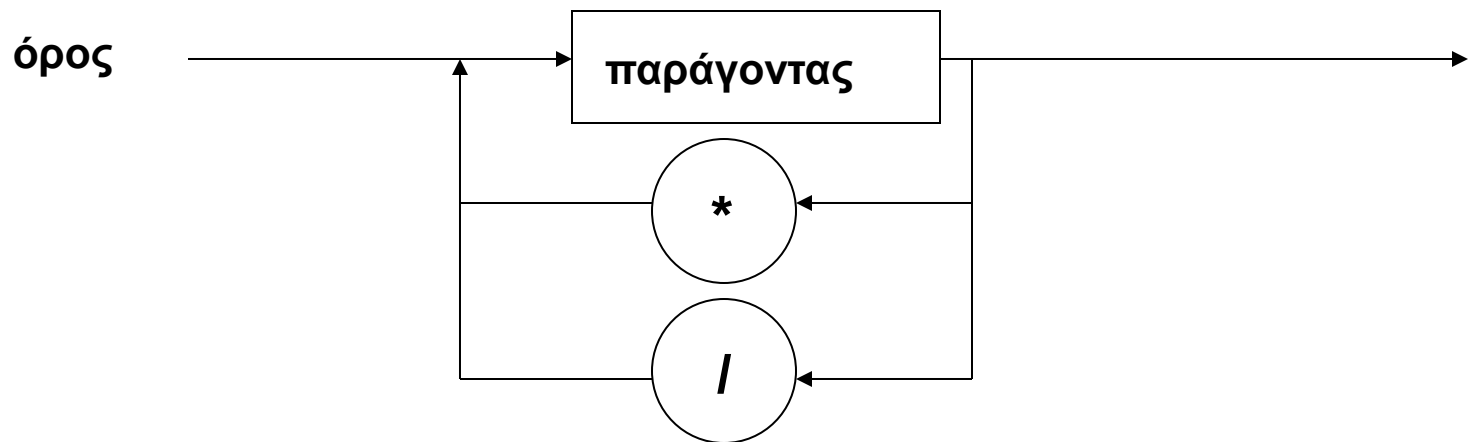


$\langle \text{έκφραση} \rangle ::= \langle \text{όρος} \rangle \mid \langle \text{έκφραση} \rangle + \langle \text{όρος} \rangle \mid \langle \text{έκφραση} \rangle - \langle \text{όρος} \rangle$



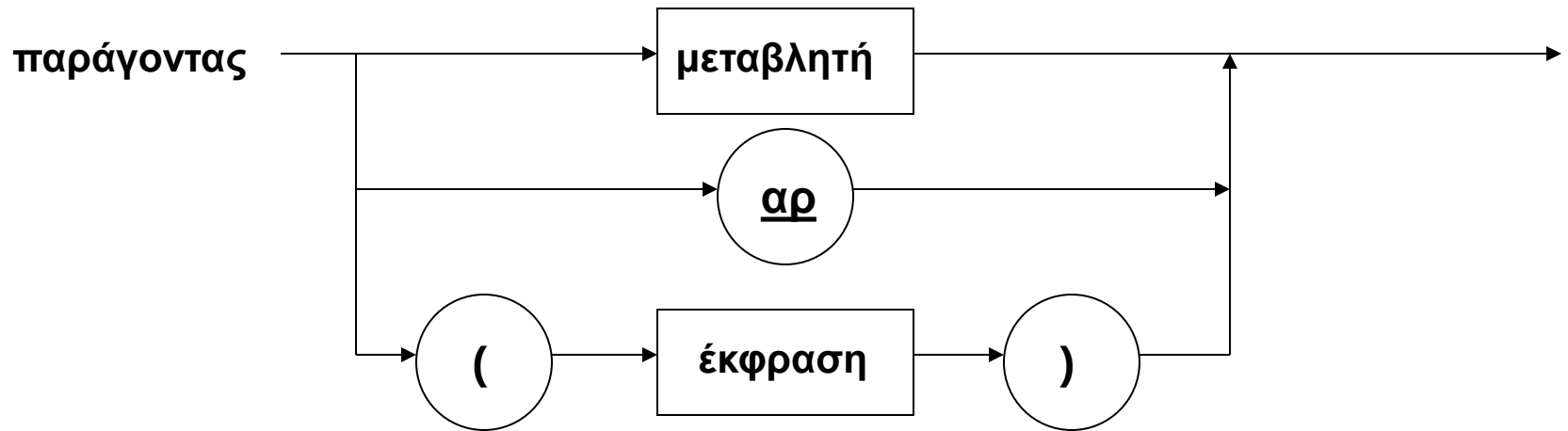
Συντακτική Ανάλυση (20)

$\langle \text{όρος} \rangle ::= \langle \text{παράγοντας} \rangle \mid \langle \text{όρος} \rangle * \langle \text{παράγοντας} \rangle \mid \langle \text{όρος} \rangle / \langle \text{παράγοντας} \rangle$



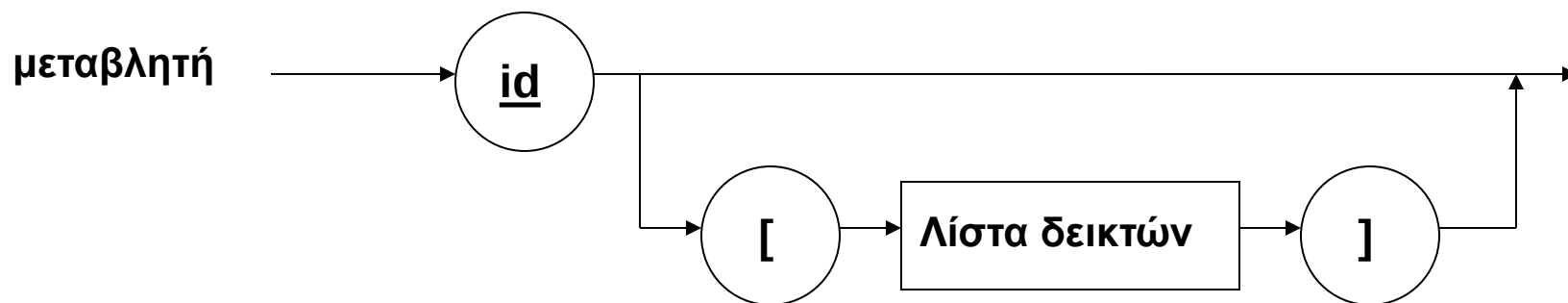
Συντακτική Ανάλυση (21)

<παράγοντας> ::= <μεταβλητή> | αρ | (<έκφραση>)

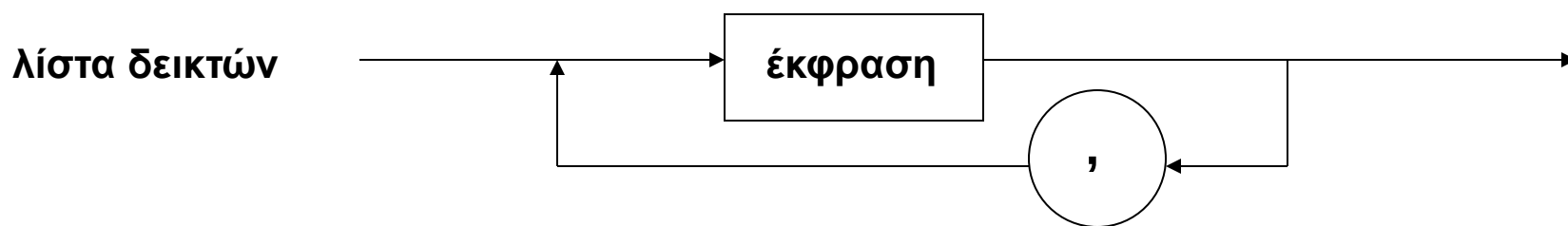


Συντακτική Ανάλυση (22)

<μεταβλητή> ::= id | id [<λίστα δεικτών>]



<λίστα δεικτών> ::= <έκφραση> | <λίστα δεικτών> , <έκφραση>



Ισοδύναμες Γραμματικές (1)

■ Ισοδύναμες Γραμματικές:

Γραμματικές που παράγουν την ίδια γλώσσα.

- Δηλαδή, οι γραμματικές G_1 και G_2 είναι ισοδύναμες, αν και μόνο αν $L(G_1) = L(G_2)$
- Χρήσιμη ιδιότητα στην κατασκευή compilers:
 - Κατά τη σχεδίαση, χρησιμοποιούμε απλές γραμματικές που είναι εύκολα κατανοητές
 - Στη συνέχεια, ίσως χρειαστεί να τις μετασχηματίσουμε σε ισοδύναμες, με ιδιότητες που τις κάνουν πιο εύκολα υλοποιήσιμες
- Το πρόβλημα της διαπίστωσης της ισοδυναμίας δύο γραμματικών είναι μη-υπολογίσιμο.

Ισοδύναμες Γραμματικές (2)

■ **Διφορούμενες** (ambiguous) Γραμματικές Χωρίς Συμφραζόμενα:

Όταν υπάρχουν 2 ή περισσότερα δέντρα συντακτικής ανάλυσης για κάποια/ες παραγόμενη συμβολοσειρά.

- Μια διφορούμενη γραμματική *μπορεί* να είναι ισοδύναμη με άλλη γραμματική που δεν είναι διφορούμενη.
- Στους compilers, αποφεύγουμε τη χρήση διφορούμενων γραμματικών, γιατί οδηγούν σε σημασιολογικές ασάφειες.
- Οι γραμματικές που δεν μπορούν να μετασχηματιστούν σε μη-διφορούμενες, ονομάζονται **εγγενώς διφορούμενες** (inherently ambiguous).

Ισοδύναμες Γραμματικές (3)

- Παράδειγμα διαφορούμενης γραμματικής:

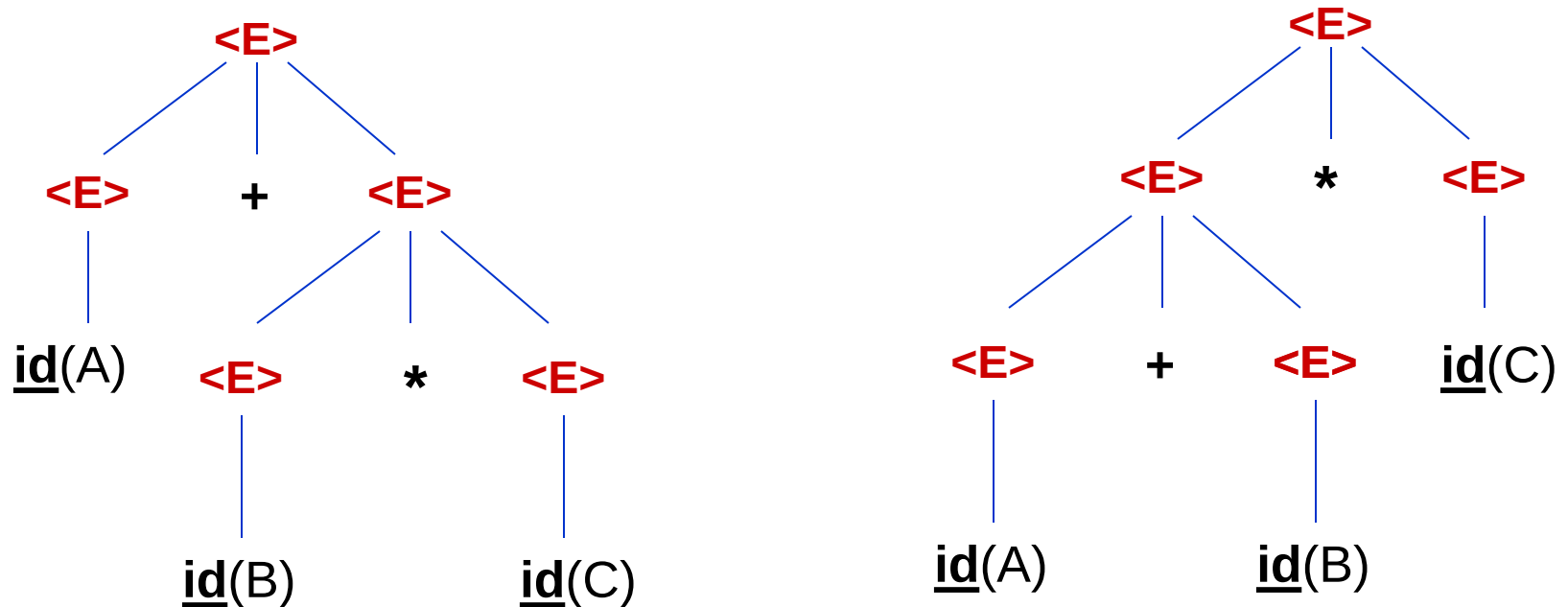
$\langle E \rangle ::= \underline{\text{id}} \mid \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid (\langle E \rangle)$

Δέντρα Συντακτικής Ανάλυσης για το:

$A + B * C$

Ισοδύναμες Γραμματικές (4)

$A + B * C$



$\langle E \rangle ::= \underline{\text{id}} \mid \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid (\langle E \rangle)$

Ισοδύναμες Γραμματικές (5)

- Ισοδύναμη Γραμματική μη-διφορούμενη:

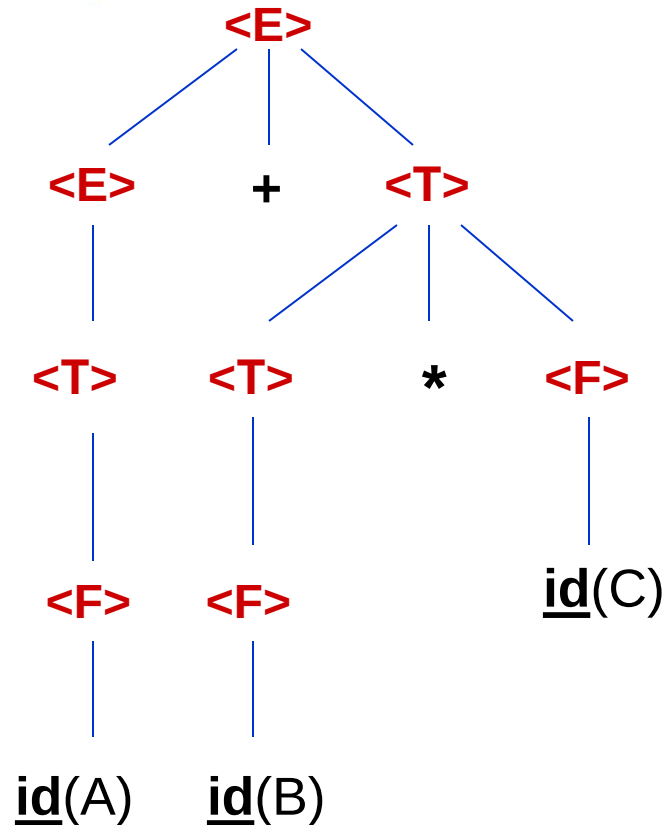
$\langle E \rangle ::= \underline{\text{id}} \mid \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid (\langle E \rangle)$

$$\langle E \rangle ::= \langle E \rangle + \langle T \rangle \mid \langle T \rangle$$
$$\langle T \rangle ::= \langle T \rangle * \langle F \rangle \mid \langle F \rangle$$
$$\langle F \rangle ::= (\langle E \rangle) \mid \underline{\text{id}}$$

Δηλαδή, «επιβάλλουμε» τη χρήση από τη ρίζα, του κανόνα $\langle E \rangle + \langle E \rangle$. Αυτό ταιριάζει με τη σημασιολογία που θα θέλαμε: Η πρόσθεση είναι υψηλότερου επιπέδου από τον πολλαπλασιασμό στην ιεραρχία των αριθμητικών τελεστών.

Ισοδύναμες Γραμματικές (6)

$A + B * C$



Δηλαδή, δημιουργήσαμε μια ισοδύναμη γραμματική, αλλά πιο πολύπλοκη, που παράγει και πιο πολύπλοκα δέντρα συντακτικής ανάλυσης.

Όμως, δεν είναι διαφορούμενη...

$\begin{aligned} \langle E \rangle &::= \langle E \rangle + \langle T \rangle \mid \langle T \rangle \\ \langle T \rangle &::= \langle T \rangle * \langle F \rangle \mid \langle F \rangle \\ \langle F \rangle &::= (\langle E \rangle) \mid \underline{id} \end{aligned}$

Συντακτική Ανάλυση Top-Down (1)

- **Top-Down** Συντακτική Ανάλυση (Καθοδική)

- Ο Συντακτικός Αναλυτής (ΣΑ) πρέπει να αποφασίσει για τα εξής:

1. Ποιος κανόνας παραγωγής θα χρησιμοποιηθεί για να δημιουργηθούν τα «παιδιά» ενός κόμβου;

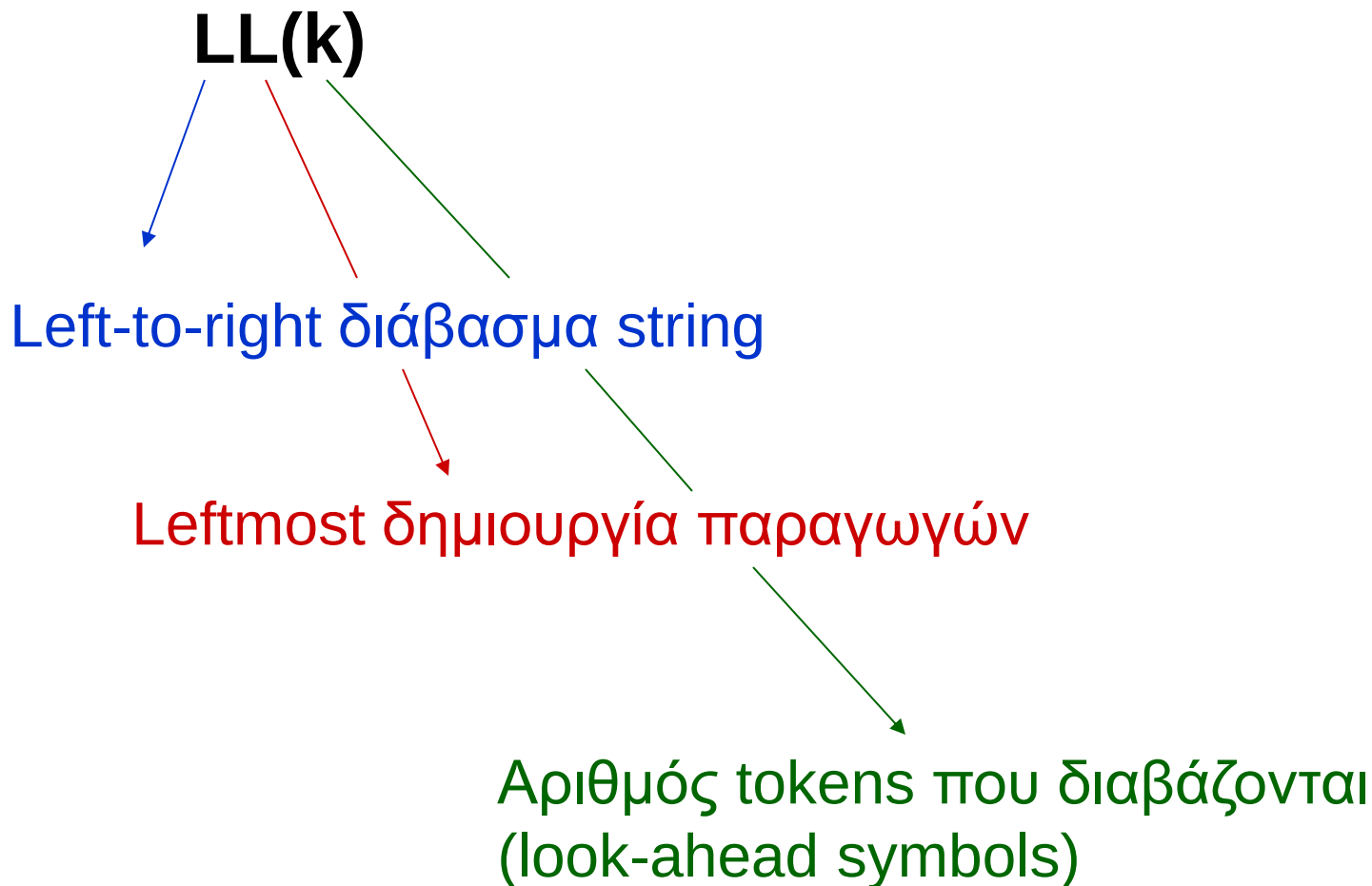
ΑΠΑΝΤΗΣΗ: Δύσκολη... Διαβάζει ένα αριθμό k tokens για να αποφασίσει, χρησιμοποιεί την παραγωγή που είναι πιο αριστερά, ...

1. Στη συνέχεια, με ποια σειρά θα γίνει η επεξεργασία των νέων κόμβων που προκύπτουν;

ΑΠΑΝΤΗΣΗ: Οι περισσότεροι ΣΑ τους επεξεργάζονται από αριστερά προς τα δεξιά.

Συντακτική Ανάλυση Top-Down (2)

- Οι συνήθειες top-down ΣΑ αναφέρονται ως:



Συντακτική Ανάλυση Top-Down (3)

- Γραμματικές **LL(1)**:

Οι γραμματικές χωρίς συμφραζόμενα που αναγνωρίζονται από LL(1) Συντακτικούς Αναλυτές.

- Αποδεικνύεται ότι για να είναι μια γραμματική LL(1), όλες οι παραγωγές της πρέπει να ικανοποιούν τα εξής:

1. Να μην έχουν *αριστερή αναδρομή*, άμεση ή έμμεση.
2. Να μην έχουν 2 εναλλακτικούς κανόνες, τα δεξιά μέλη των οποίων αρχίζουν με το *ίδιο σύμβολο*.
3. Να μην έχουν 2 εναλλακτικούς κανόνες, τα δεξιά μέλη των οποίων παράγουν την *κενή συμβολοσειρά* (ϵ).

Συντακτική Ανάλυση Top-Down (4)

Χρησιμοποιείται μια *Στοίβα* και δύο *Πράξεις*:

- **Ταίριασμα συμβόλου**: Αν στην κορυφή της στοίβας βρίσκεται το *τερματικό* σύμβολο **a** και το τρέχον σύμβολο του string εισόδου είναι επίσης **a**, τότε το **a** αφαιρείται από τη στοίβα και διαβάζεται το επόμενο σύμβολο του string εισόδου.
- **Πρόβλεψη**: Αν στην κορυφή της στοίβας βρίσκεται το *μη-τερματικό* σύμβολο **<A>**, το αντικαθιστούμε με το δεξιό μέρος κάποιου κανόνα ορισμού του **<A>**, με τα σύμβολα σε *αντίθετη σειρά*.

Αν καμία από τις δύο πράξεις δεν μπορεί να εφαρμοστεί, τότε υπάρχει συντακτικό σφάλμα.

Συντακτική Ανάλυση Top-Down (5)

Παράδειγμα υλοποίησης LL(1) Συντακτικού Αναλυτή:

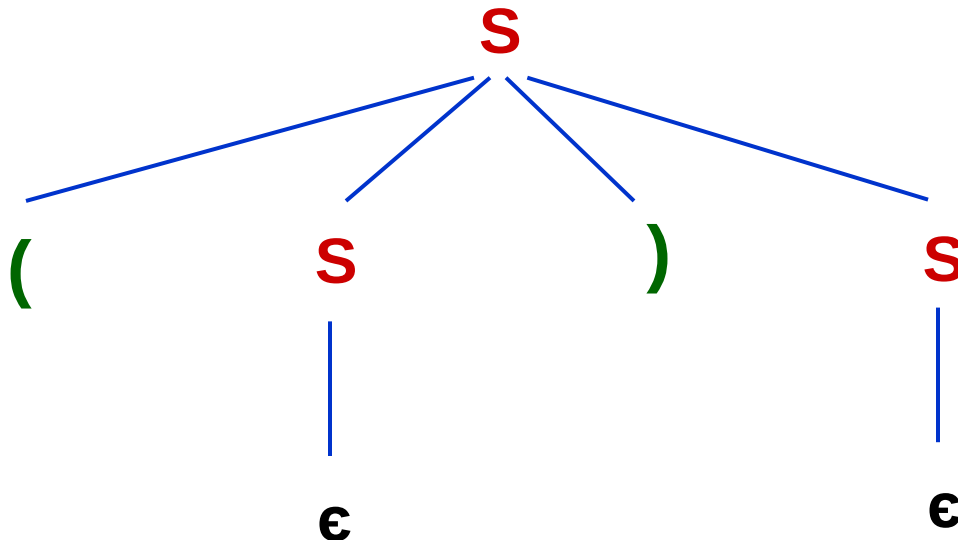
Γραμματική: $\langle S \rangle ::= (\langle S \rangle)\langle S \rangle \mid \epsilon$

String εισόδου: $()$

Βήμα	Στοίβα	Είσοδος	Πράξη
0	$\langle S \rangle$	$()$ EOF	Πρόβλεψη $\langle S \rangle ::= (\langle S \rangle)\langle S \rangle$
1	$\langle S \rangle)\langle S \rangle($	$()$ EOF	Ταίριασμα συμβόλου $($
2	$\langle S \rangle)\langle S \rangle$	$)$ EOF	Πρόβλεψη $\langle S \rangle ::= \epsilon$
3	$\langle S \rangle)$	$)$ EOF	Ταίριασμα συμβόλου $)$
4	$\langle S \rangle$	EOF	Πρόβλεψη $\langle S \rangle ::= \epsilon$
5	ϵ	EOF	Αναγνώριση

Το δέντρο συντακτικής ανάλυσης

$\langle S \rangle ::= (\langle S \rangle) \langle S \rangle \mid \epsilon$



Βήμα	Στοίβα	Είσοδος	Πράξη
0	$\langle S \rangle$	() EOF	Πρόβλεψη $\langle S \rangle ::= (\langle S \rangle) \langle S \rangle$
1	$\langle S \rangle \langle S \rangle ($	() EOF	Ταίριασμα συμβόλου (
2	$\langle S \rangle \langle S \rangle$	) EOF	Πρόβλεψη $\langle S \rangle ::= \epsilon$
3	$\langle S \rangle)$	) EOF	Ταίριασμα συμβόλου)
4	$\langle S \rangle$	EOF	Πρόβλεψη $\langle S \rangle ::= \epsilon$
5	ϵ	EOF	Αναγνώριση

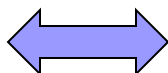
Συντακτική Ανάλυση Top-Down (6)

- Σε αρκετές περιπτώσεις μπορούμε να μετατρέψουμε μια γραμματική σε ισοδύναμη LL(1).
- Για τη μετατροπή αυτή, χρησιμοποιούμε τρία είδη μετασχηματισμών:

A. Αντικατάσταση

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

$$B \rightarrow \beta_1 A \beta_2$$



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

$$B \rightarrow \beta_1 \alpha_1 \beta_2 \mid \dots \mid \beta_1 \alpha_n \beta_2$$

Συντακτική Ανάλυση Top-Down (7)

B. Αριστερή Παραγοντοποίηση

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

Γ. Απαλοιφή Αριστερής Αναδρομής (AA)

Άμεση AA: $A \rightarrow A\alpha$

Έμμεση AA: $A \rightarrow B\alpha_1, B \rightarrow C\alpha_2, C \rightarrow A\alpha_3$ (δηλ. $A \rightarrow A\alpha_3\alpha_2\alpha_1$)

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

Συντακτική Ανάλυση Top-Down (8)

■ Παράδειγμα:

$\langle X \rangle ::= \langle B \rangle a \langle A \rangle b$

$\langle A \rangle ::= a \langle A \rangle \mid a$

$\langle B \rangle ::= \langle B \rangle b \mid b$

Η γραμματική αυτή δεν είναι LL(1), διότι:

- Στη 2^η παραγωγή υπάρχουν 2 εναλλακτικοί κανόνες που αρχίζουν με το ίδιο σύμβολο (a).
- Η 3^η παραγωγή έχει άμεση αριστερή αναδρομή.

■ Για να δημιουργήσουμε μια ισοδύναμη LL(1) γραμματική, θα κάνουμε τις εξής 2 μετατροπές:

Συντακτική Ανάλυση Top-Down (9)

- Στη 2^η παραγωγή θα χρησιμοποιήσουμε το μετασχηματισμό της *Αριστερής Παραγοντοποίησης*:

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

$$\langle A \rangle ::= a \langle A \rangle \mid a \quad \longleftrightarrow \quad \begin{array}{l} \langle A \rangle ::= a \langle C \rangle \\ \langle C \rangle ::= \langle A \rangle \mid \epsilon \end{array}$$

- Στην 3^η παραγωγή θα χρησιμοποιήσουμε το μετασχηματισμό της *Απαλοιφής Αριστερής Αναδρομής*:

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

$$\langle B \rangle ::= \langle B \rangle b \mid b \quad \longleftrightarrow \quad \begin{array}{l} \langle B \rangle ::= b \langle D \rangle \\ \langle D \rangle ::= b \langle D \rangle \mid \epsilon \end{array}$$

Συντακτική Ανάλυση Top-Down (10)

- Τελικά, η ισοδύναμη LL(1) γραμματική, είναι η:

$\langle X \rangle ::= \langle B \rangle a \langle A \rangle b$

$\langle A \rangle ::= a \langle C \rangle$

$\langle C \rangle ::= \langle A \rangle \mid \epsilon$

$\langle B \rangle ::= b \langle D \rangle$

$\langle D \rangle ::= b \langle D \rangle \mid \epsilon$

Συντακτική Ανάλυση Bottom-Up (1)

- **Bottom-Up** Συντακτική Ανάλυση (Ανοδική)
- Ο Συντακτικός Αναλυτής (ΣΑ) ξεκινά την κατασκευή του δέντρου συντακτικής ανάλυσης από τα **φύλλα** (τα τερματικά σύμβολα του προς έλεγχο string).
- Στη συνέχεια προσπαθεί να βρει τον αριστερότερο κόμβο που δεν έχει ακόμα φτιαχτεί, ενώ όλα τα παιδιά του έχουν φτιαχτεί.
- Μέχρι να κατασκευαστεί η ρίζα του δέντρου με το Start.
- Κάθε στιγμή ο ΣΑ πρέπει να επιλέξει ποιους από τους ήδη υπάρχοντες κόμβους θα χρησιμοποιήσει ως παιδιά του νέου κόμβου που θα φτιάξει: **Ελάττωση** (reducing).

Συντακτική Ανάλυση Bottom-Up (2)

Παράδειγμα:

$\langle S \rangle ::= r \langle B \rangle$

$\langle B \rangle ::= \langle D \rangle \mid \langle B \rangle , \langle D \rangle$

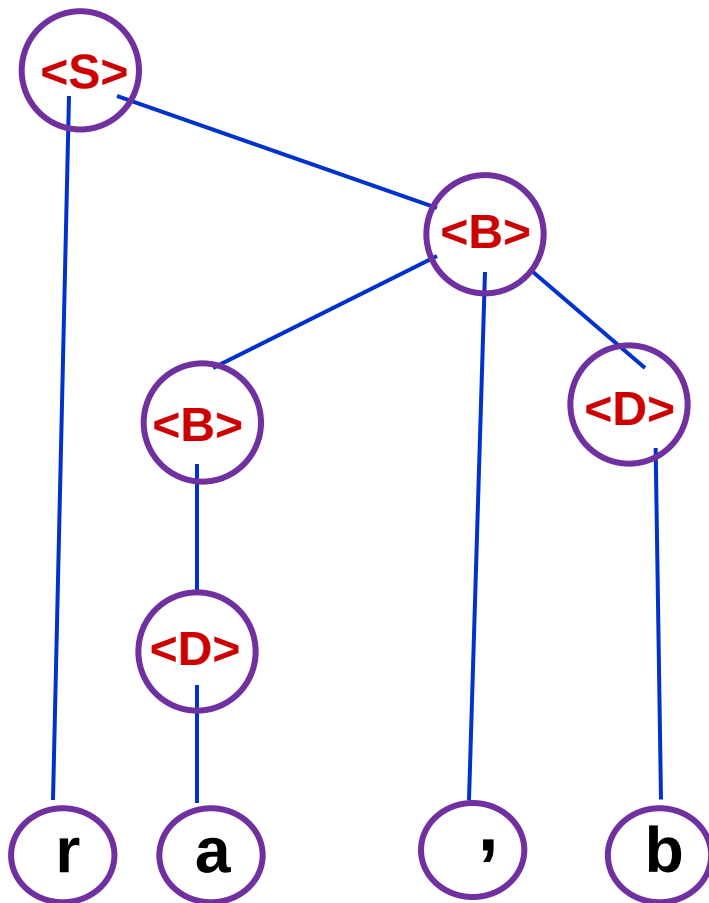
$\langle D \rangle ::= a \mid b$

Κατασκευή του δέντρου συντακτικής ανάλυσης για το string **ra,b** :

Συντακτική Ανάλυση Bottom-Up (3)

$\langle S \rangle ::= r \langle B \rangle$
 $\langle B \rangle ::= \langle D \rangle \mid \langle B \rangle , \langle D \rangle$
 $\langle D \rangle ::= a \mid b$

ra,b



Συντακτική Ανάλυση Bottom-Up (4)

Συντακτικοί Αναλυτές **ολίσθησης-ελάττωσης** (shift-reduce parsers).

Χρησιμοποιούν μια *Στοίβα* και δύο *Πράξεις*:

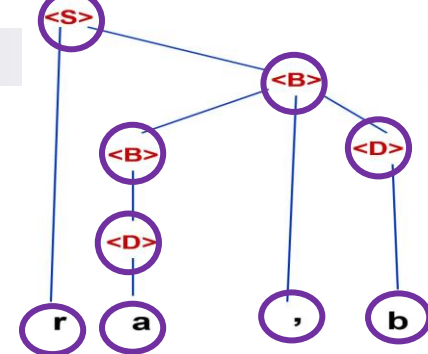
- **Ολίσθηση** (shift): Αφαιρεί ένα σύμβολο από την αρχή του string και το βάζει στην κορυφή της στοίβας.
- **Ελάττωση** (reduce): Όταν στην κορυφή της στοίβας υπάρχει το δεξί μέλος παραγωγής. Αφαιρούνται αυτά τα σύμβολα από τη στοίβα και αντικαθίστανται από το αριστερό μέλος.

Συντακτική Ανάλυση Bottom-Up (5)

$\langle S \rangle ::= r \langle B \rangle$

$\langle B \rangle ::= \langle D \rangle \mid \langle B \rangle , \langle D \rangle$

$\langle D \rangle ::= a \mid b$



Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	ra,b EOF	Ολίσθηση
①	r	a,b EOF	Ολίσθηση
②	ra	,b EOF	Ελάττωση $\langle D \rangle ::= a$
③	r<D>	,b EOF	Ελάττωση $\langle B \rangle ::= \langle D \rangle$
④	r	,b EOF	Ολίσθηση
⑤	r ,	b EOF	Ολίσθηση
⑥	r ,b	EOF	Ελάττωση $\langle D \rangle ::= b$
⑦	r ,<D>	EOF	Ελάττωση $\langle B \rangle ::= \langle B \rangle , \langle D \rangle$
⑧	r	EOF	Ελάττωση $\langle S \rangle ::= r \langle B \rangle$
⑨	<S>	EOF	Αναγνώριση

Συντακτική Ανάλυση Bottom-Up (6)

Προβλήματα:

- Στο Βήμα 4 υπήρχαν δύο επιλογές: ολίσθηση ή ελάττωση με τον κανόνα $\langle S \rangle ::= r \langle B \rangle$. Η 2^η επιλογή καταλήγει σε αδιέξοδο, γιατί η ανάλυση τελειώνει χωρίς να έχει σαρωθεί όλο το string εισόδου: **Σύγκρουση ολίσθησης-ελάττωσης** (shift-reduce conflict)
- Στο Βήμα 7 υπήρχαν δύο επιλογές: ελάττωση με τον κανόνα $\langle B \rangle ::= \langle B \rangle$, $\langle D \rangle$ ή ελάττωση με τον κανόνα $\langle B \rangle ::= \langle D \rangle$. Η 2^η επιλογή δεν είναι καλή, γιατί δεν μπορεί να προχωρήσει η ανάλυση: **Σύγκρουση ελάττωσης-ελάττωσης** (reduce-reduce conflict)

Τέτοια προβλήματα λύνονται από ντετερμινιστικούς ΣΑ ολίσθησης-ελάττωσης, τους **LR(k)**.

Συντακτική Ανάλυση Bottom-Up (7)

Bottom-up ΣΑ

LR(k)

Left-to-right διάβασμα string

Rightmost δημιουργία παραγωγών

Αριθμός tokens που διαβάζονται
(look-ahead symbols)

Συντακτική Ανάλυση Bottom-Up (8)

- Στην πράξη χρησιμοποιούνται οι LR(1).
- Η λειτουργία τους είναι περισσότερο πολύπλοκη από αυτή των LL(1).
- Δεν αντιμετωπίζουν πρόβλημα με αριστερή αναδρομή.
- Το εργαλείο **bison** είναι γεννήτορας συντακτικών αναλυτών για **γραμματικές χωρίς συμφραζόμενα** τύπου LR(1)...