Κεφάλαιο 3 : Σύνταξη Γλωσσών Προγραμματισμού

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Γιάννης Γαροφαλάκης, Σπύρος Σιούτας

Εισαγωγή

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

  A(1) στις FORTRAN, COBOL, PL/1, Ada

  A[1] στις Pascal, C

  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;
```

Συντακτικά Στοιχεία 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 → 2^8 = 256 χαρακτήρες)
   b. ASCII (American Standard Code for Information Interchange) από ANSI (7 bit → 2^7 = 128 χαρακτήρες)

   Τώρα:

   UNICODE (Universal Character Set)

   16 bit → 2^{16} = 65.536 χαρακτήρες

   Συμβατή κωδικοποίηση με ASCII

Συντακτικά Στοιχεία μιας ΓΠ (2)

2. **Αναγνωριστικά** (identifiers)

   Ονόματα μεταβλητών, συναρτήσεων, …

3. **Σύμβολα πράξεων**

   + * - / ** && || AND OR NOT …

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

Συντακτικά Στοιχεία 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

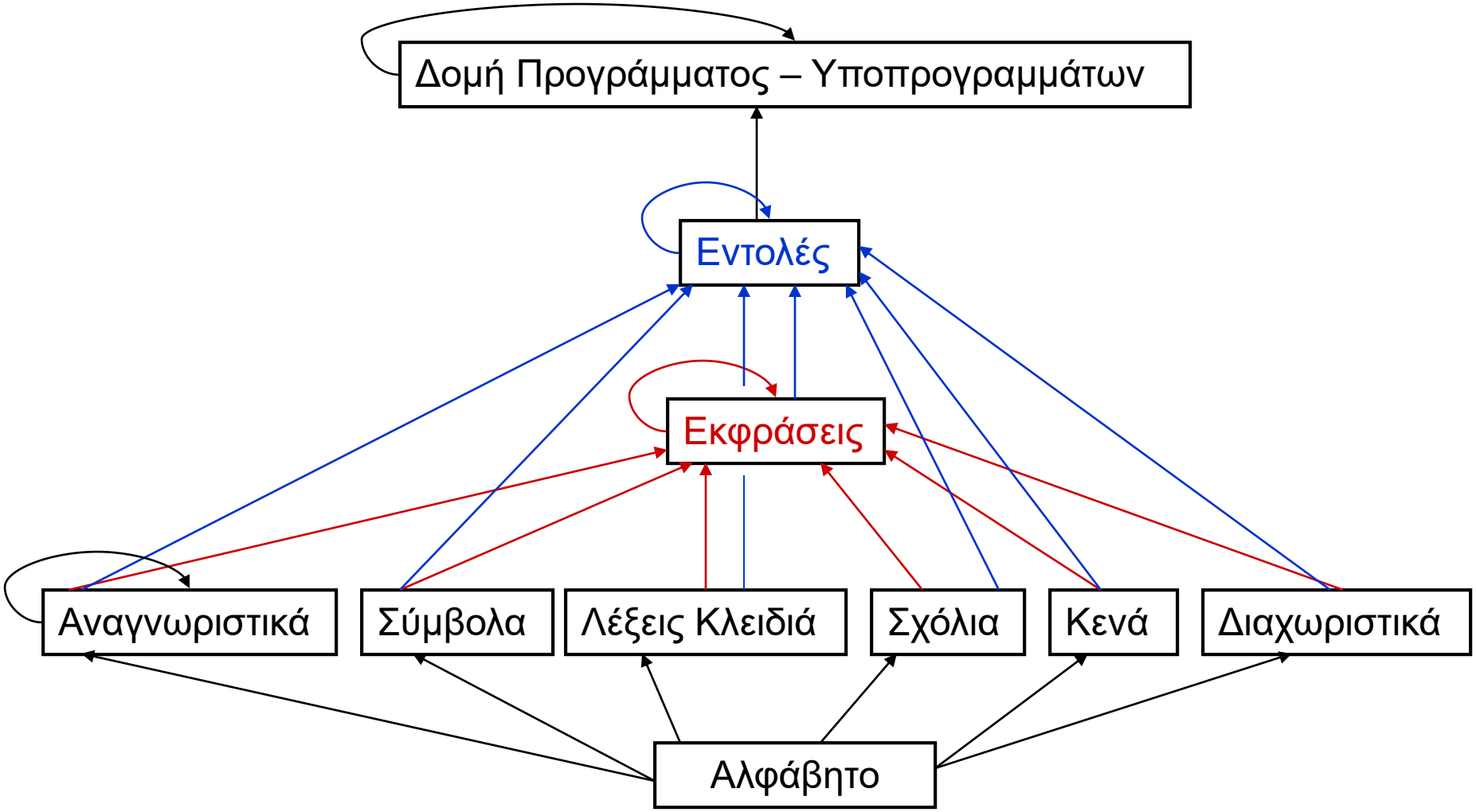
9. **Εντολές** (statements)

   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
   - Όλα τα τμήματα κώδικα είναι υποπρογράμματα
   - Κάθε υποπρόγραμμα θεωρείται διακριτή συντακτική μονάδα, μεταφράζεται χωριστά και συνδέονται όλα όταν γίνεται φόρτωση

b. Ξεχωριστός Ορισμός Δεδομένων
   - Java, C++, Smalltalk: Μηχανισμός κλάσεων
   - Ομαδοποίηση των λειτουργιών που χειρίζονται ένα δεδομένο data object

c. Φωλιασμένος Ορισμός Υποπρογράμματος
   - Pascal, ALGOL
   - Ιεραρχικός ορισμός. Δυνατότητα ορισμού εμβέλειας. Στατικός έλεγχος τύπων.

Δομή Προγράμματος – Υποπ/μάτων (2)

d. Ξεχωριστός Ορισμός Interface
   - C, Ada
   - Δυνατότητα χρήσης file operations του ΛΣ (.h και .c file specs του make στη C)

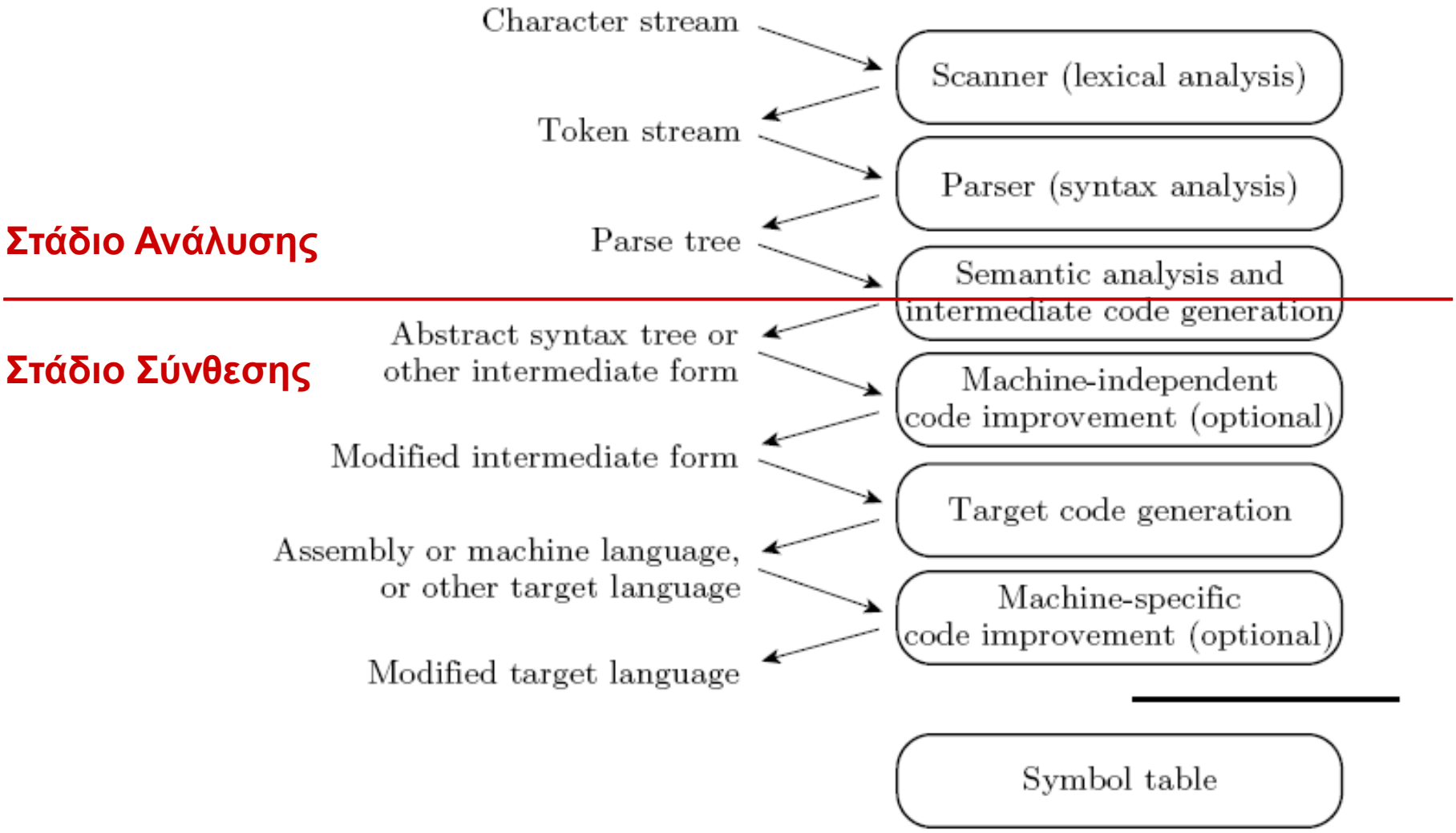
e. Περιγραφές δεδομένων χωριστές από τις εκτελέσιμες εντολές
   - COBOL
   - Όλα τα data είναι global. Data και procedure τμήματα

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, 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)

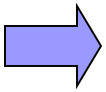
- Δημιουργία Ενδιάμεσου Κώδικα (intermediate code generation)
  - Παράγεται ακολουθία operators – ορισμάτων
- Βελτιστοποίηση κώδικα ανεξάρτητα από μηχανή (machine independent code improvement)
  - Υπολογισμός κοινών υπο-εκφράσεων μία φορά
  - Απομάκρυνση σταθερών λειτουργιών από loops
  - Βελτιστοποίηση 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 = (Σ, P) είναι όλοι οι μη-αρνητικοί δυαδικοί αριθμοί μικρότεροι από 1, με δύο υπο-δυαδικά ψηφία.

- **Σ** = {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 } : *Συντακτικές Κατηγορίες* ή *Μη-Τερματικά Σύμβολα*

Start = S_0 : *Αρχικό Σύμβολο* (start symbol). Η συντακτική κατηγορία του πιο υψηλού επιπέδου

$\rightarrow$: Μετασύμβολο

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:

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

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

Σύστημα Παραγωγής

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

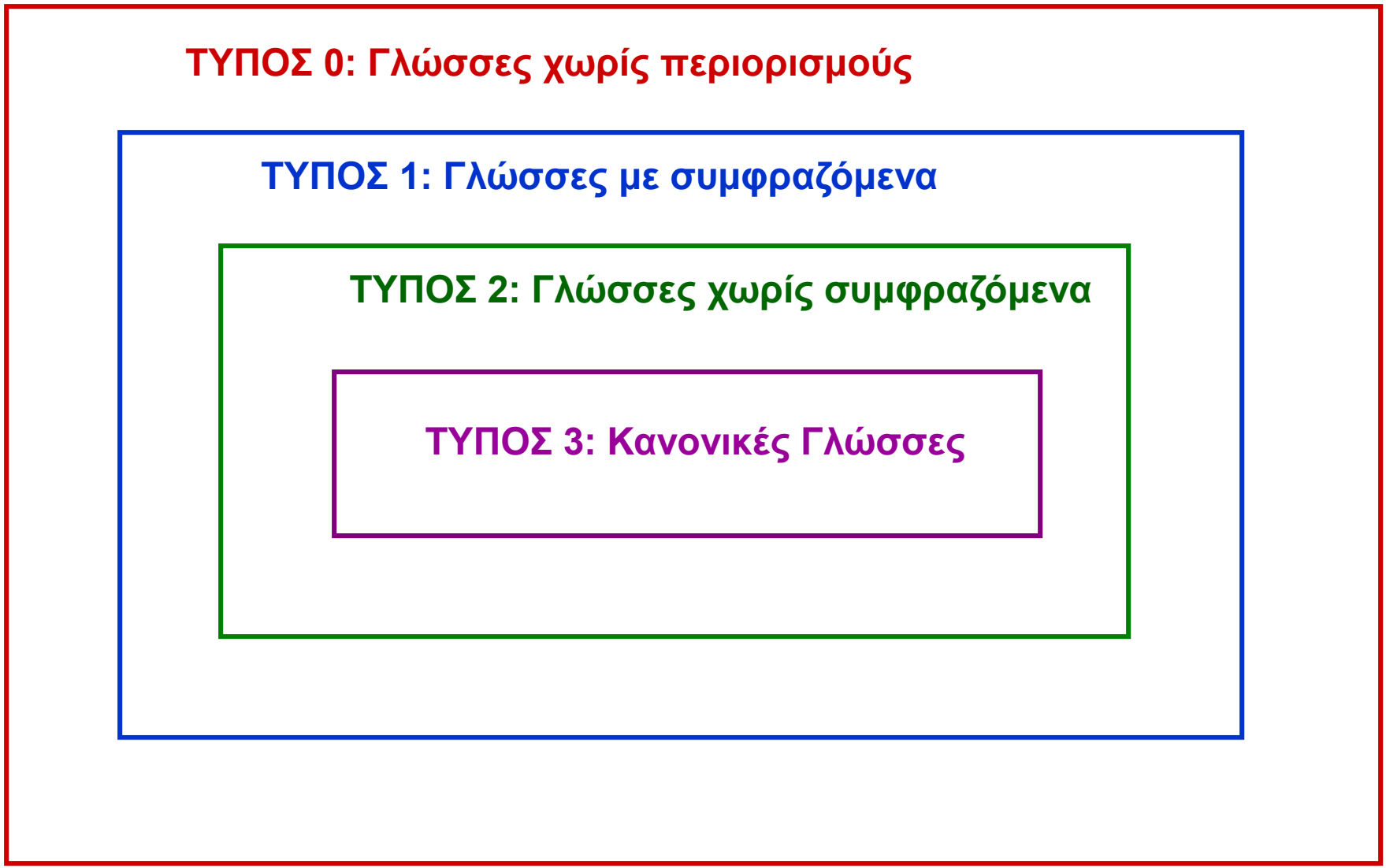
όπου $\mathbf{Start \in \Sigma \cup N}$

Ιεραρχία Γραμματικών

- Ανάλογα με το είδος των παραγωγών, υπάρχουν τελικά μόνο 4 τύποι τυπικών γλωσσών
- ΤΥΠΟΣ ΓΛΩΣΣΑΣ ↔ 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



Τύπος 3: Κανονικές Γλώσσες (regular languages)

- Η Γραμματική **G = (Σ, N, P, Start)** είναι κανονική, αν οι παραγωγές της είναι της μορφής:

  A → **α** ή **A** → **αB** όπου **A**, **B** ϵ N και **α** ϵ Σ

- Δηλαδή, το πρώτο σύμβολο δεξιά του → είναι τερματικό και μπορεί να ακολουθείται από μη-τερματικό σύμβολο
- Κατάλληλες για *λεξική ανάλυση*
- Αναγνωρίζονται από *πεπερασμένα αυτόματα*

Παράδειγμα Κανονικής Γραμματικής

X → a | … | z | aL | … | zL | aD | … | zD

L → aL | … | zL | aD | … | zD | a | … | z

D → 0L | … | 9L | 0D | … | 9D | 0 | … | 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**

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

- Η Γραμματική **G = (Σ, N, P, Start)** είναι χωρίς συμφραζόμενα, αν οι παραγωγές της είναι της μορφής:

  A → **s** όπου **A** ϵ N και **s** ϵ Σ ∪ N

- Ονομάζονται έτσι, διότι αντικαταστάσεις μπορούν να γίνουν οπουδήποτε εμφανίζεται μη-τερματικό σύμβολο, χωρίς να λαμβάνονται υπόψη τα περιβάλλοντα σύμβολα
- Κατάλληλες για *συντακτική ανάλυση*
- Αναγνωρίζονται από *push-down αυτόματα*

Το Παράδειγμα

X → a | … | z | aL | … | zL | aD | … | zD

L → aL | … | zL | aD | … | zD | a | … | z

D → 0L | … | 9L | 0D | … | 9D | 0 | … | 9

μπορεί να γραφεί ως:

X → L | XL | XD

L → a | … | z

D → 0 | … | 9

Αλλά τώρα δεν είναι κανονική γραμματική…

Είναι όμως context-free.

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

- Η Γραμματική **G = (Σ, N, P, Start)** είναι με συμφραζόμενα, αν οι παραγωγές είναι της μορφής:

  α → β όπου:

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

  2. (Μήκος του **α**) ≤ (Μήκος του **β**)

- *Παράδειγμα:* **aB → ab, cB → bc**

  (τα **a** και **c** είναι το context)

- Αναγνωρίζονται από *linear-bounded αυτόματα*

Τύπος 0: Γλώσσες χωρίς περιορισμούς

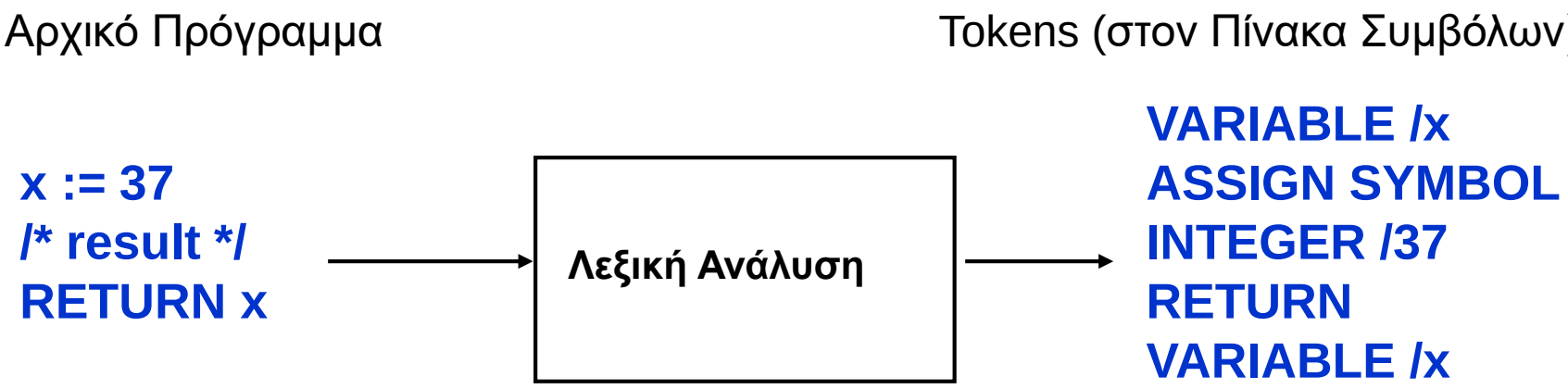
- Η Γραμματική **G = (Σ, N, P, Start)** είναι χωρίς περιορισμούς, αν οι παραγωγές είναι της μορφής:

  α → **β** όπου:

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

- Αναγνωρίζονται από *Μηχανές Turing*

Λεξική Ανάλυση (1)



- 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)

Χρησιμοποιούνται επίσης, για συντομία, τα σύμβολα:

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

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)

$$
\begin{aligned}
digit &\longrightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
unsigned\_integer &\longrightarrow digit\ digit* \\
unsigned\_number &\longrightarrow unsigned\_integer\ (\ (\ \texttt{.}\ unsigned\_integer\ ) \mid \epsilon\ ) \\
&\qquad (\ (\ (\ \texttt{e} \mid \texttt{E}\ )\ (\ \texttt{+} \mid \texttt{-} \mid \epsilon\ )\ unsigned\_integer\ ) \mid \epsilon\ )
\end{aligned}
$$

είναι αντίστοιχη της συνοπτικής κανονικής έκφρασης:

(0 – 9)$^+$ (.(0 – 9)$^+$)? ((e | E)(+ | -)? (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*.

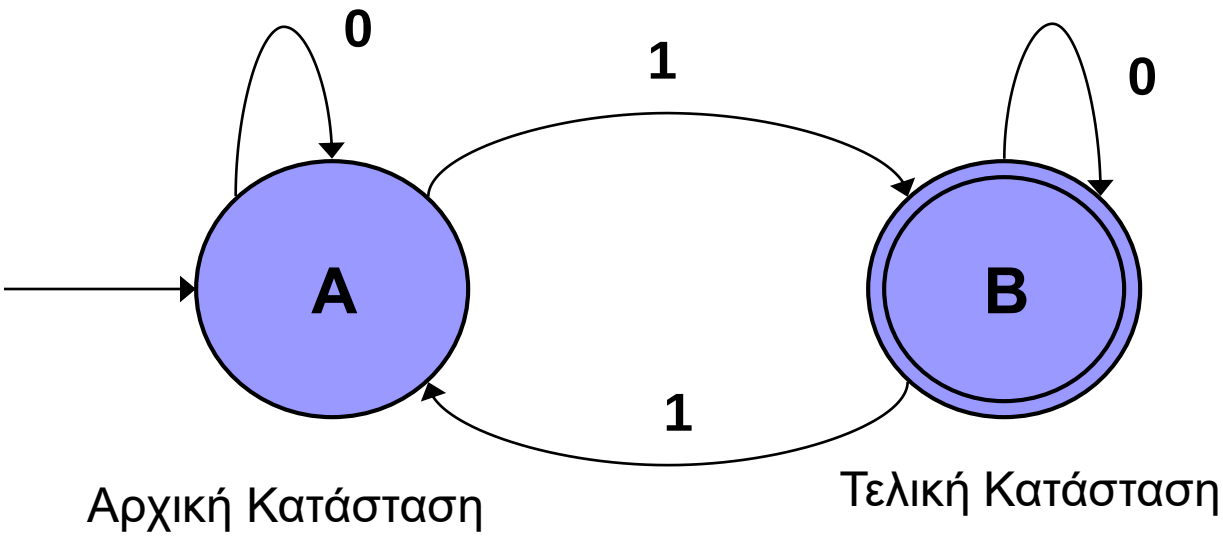
Λεξική Ανάλυση (6)

- **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.

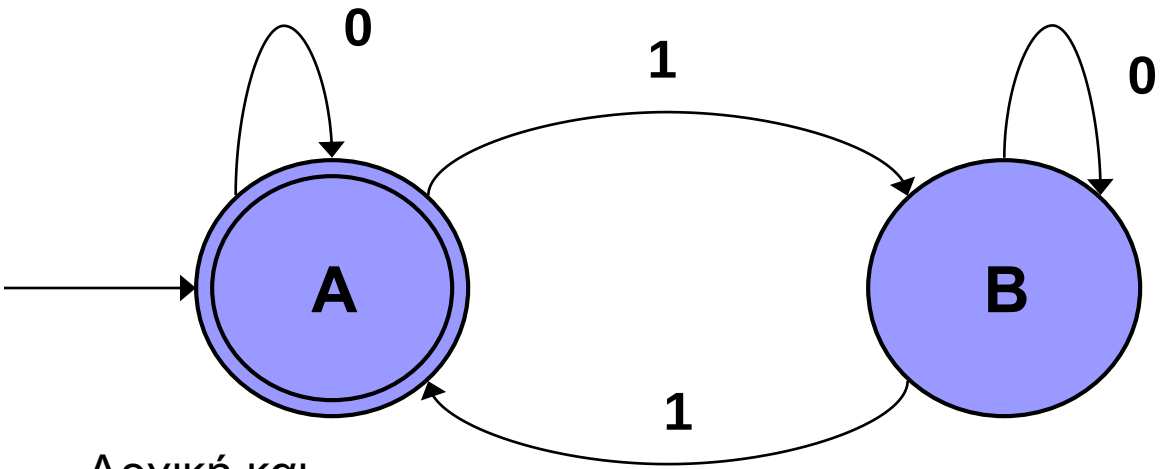
Λεξική Ανάλυση (11)

- 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	ΝΑΙ
A	1	B	ΟΧΙ
B	0	B	ΟΧΙ
B	1	A	ΝΑΙ

Λεξική Ανάλυση (14)

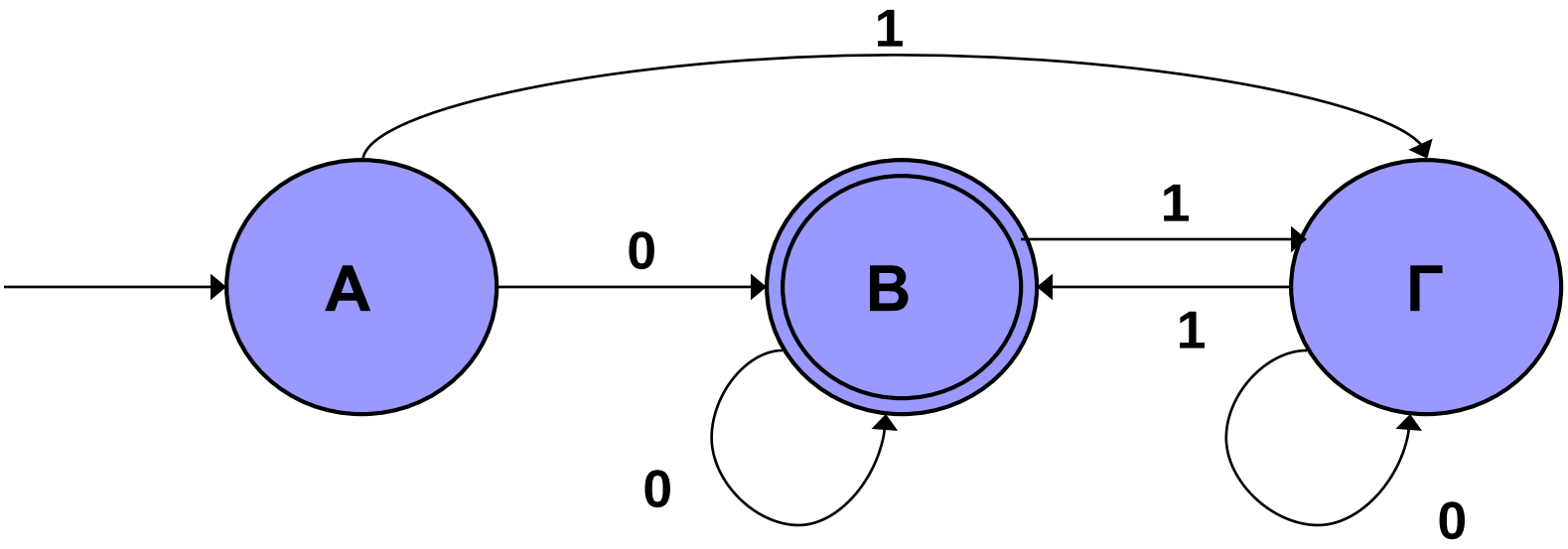
- Λειτουργία του FSA για την είσοδο **100101**:

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:

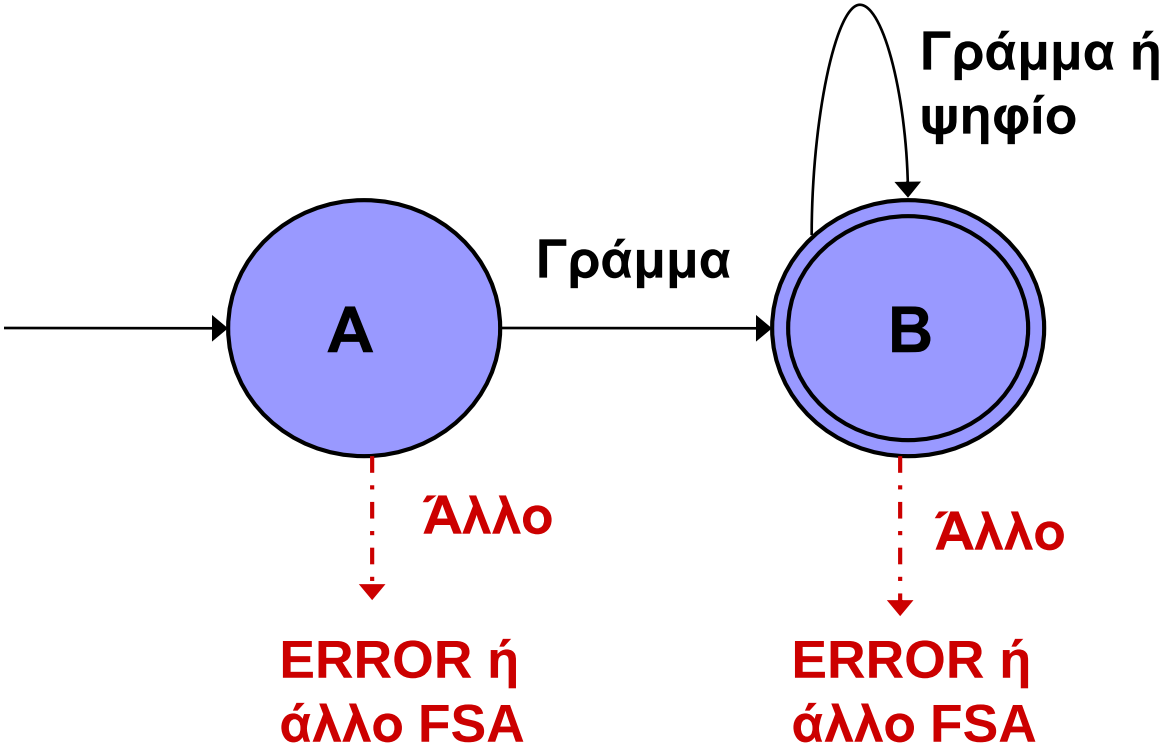


Αντίστοιχη ΚΕ: $0^+ \mid (0*10*10*)^+$

Λεξική Ανάλυση (16)

- 4ο Παράδειγμα:

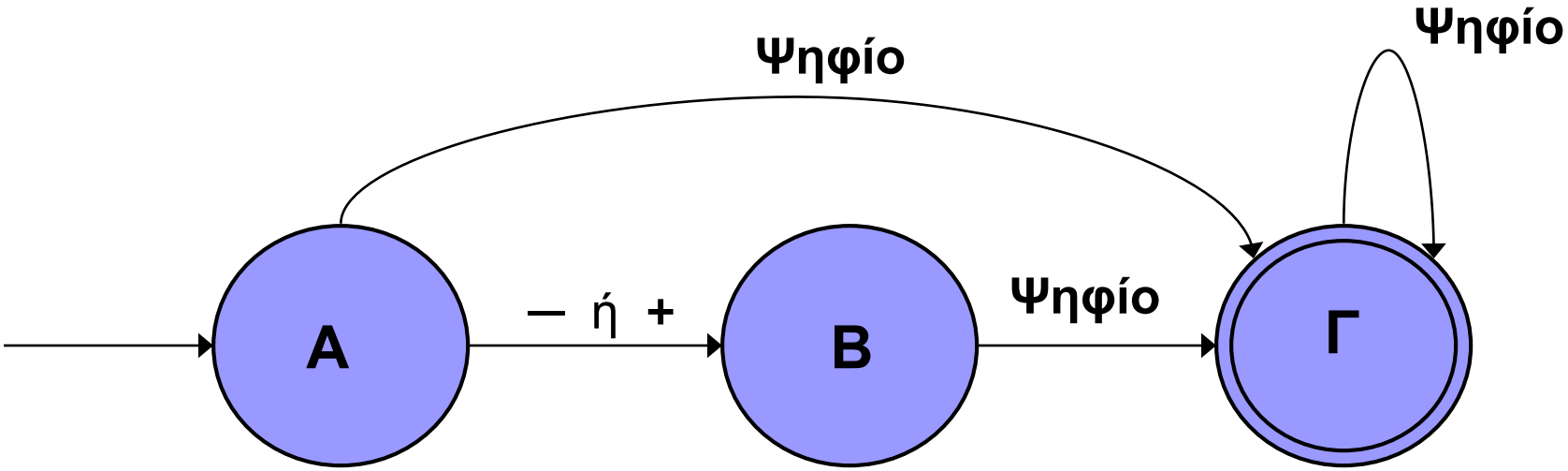
  Πεπερασμένο Αυτόματο που αναγνωρίζει **ονόματα μεταβλητών** που αρχίζουν με γράμμα και αποτελούνται από γράμματα και αριθμητικά ψηφία



Λεξική Ανάλυση (17)

- 5ο Παράδειγμα:

  Πεπερασμένο Αυτόματο που αναγνωρίζει προσημασμένους ή μη-προσημασμένους **ακέραιους αριθμούς**



Αντίστοιχη ΚΕ: $(+ \mid -)? \; (0 – 9)^{+}$

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
< … >	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*:

<μη-προσ. ακέραιος> ::= <ψηφίο> | <μη-προσ. ακέραιος> <ψηφίο>

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

- Μία μικρή BNF Γραμματική:

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

<πράξη> ::= **+** | **-** | ***** | **/** | ↑

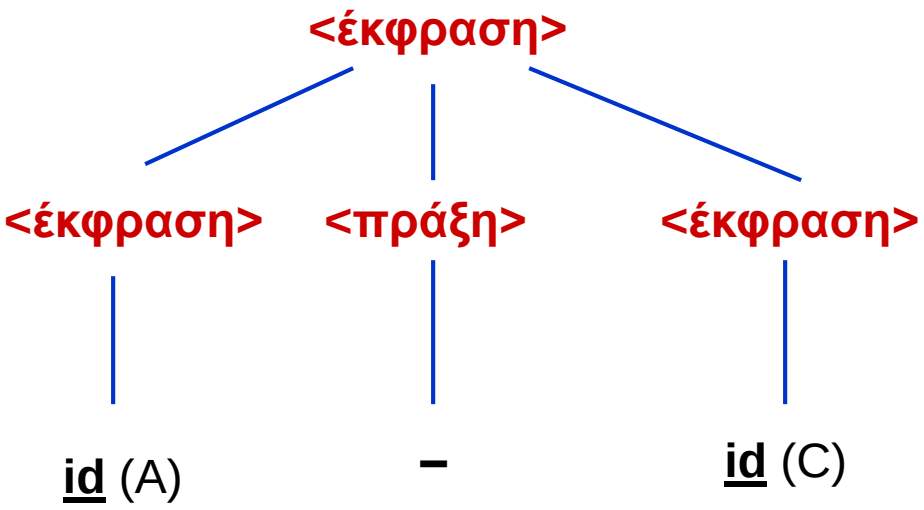
όπου: **id** **αρ** είναι είδη token αναγνωρισμένα στη *λεξική ανάλυση*
(id = αναγνωριστικό, αρ = αριθμός)
<έκφραση> = start symbol

Συντακτική Ανάλυση (4)

- Αναγνώριση σωστών προγραμμάτων:

  Top – down Parsing → Δέντρο Συντακτικής Ανάλυσης (Parse Tree)

- Π.χ. για την έκφραση A – C



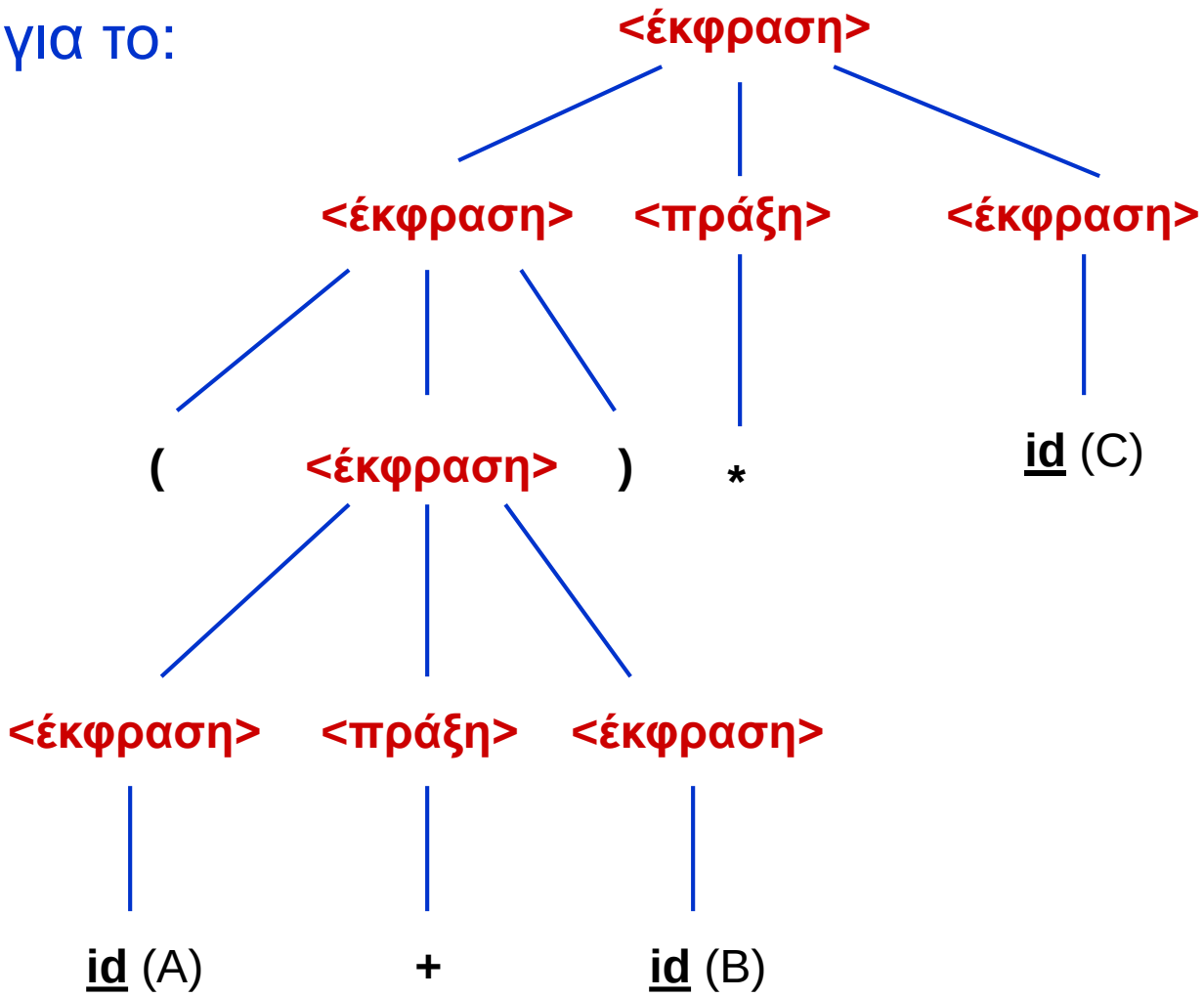
Συντακτική Ανάλυση (5)

- 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 (6)

Parse Tree για το:

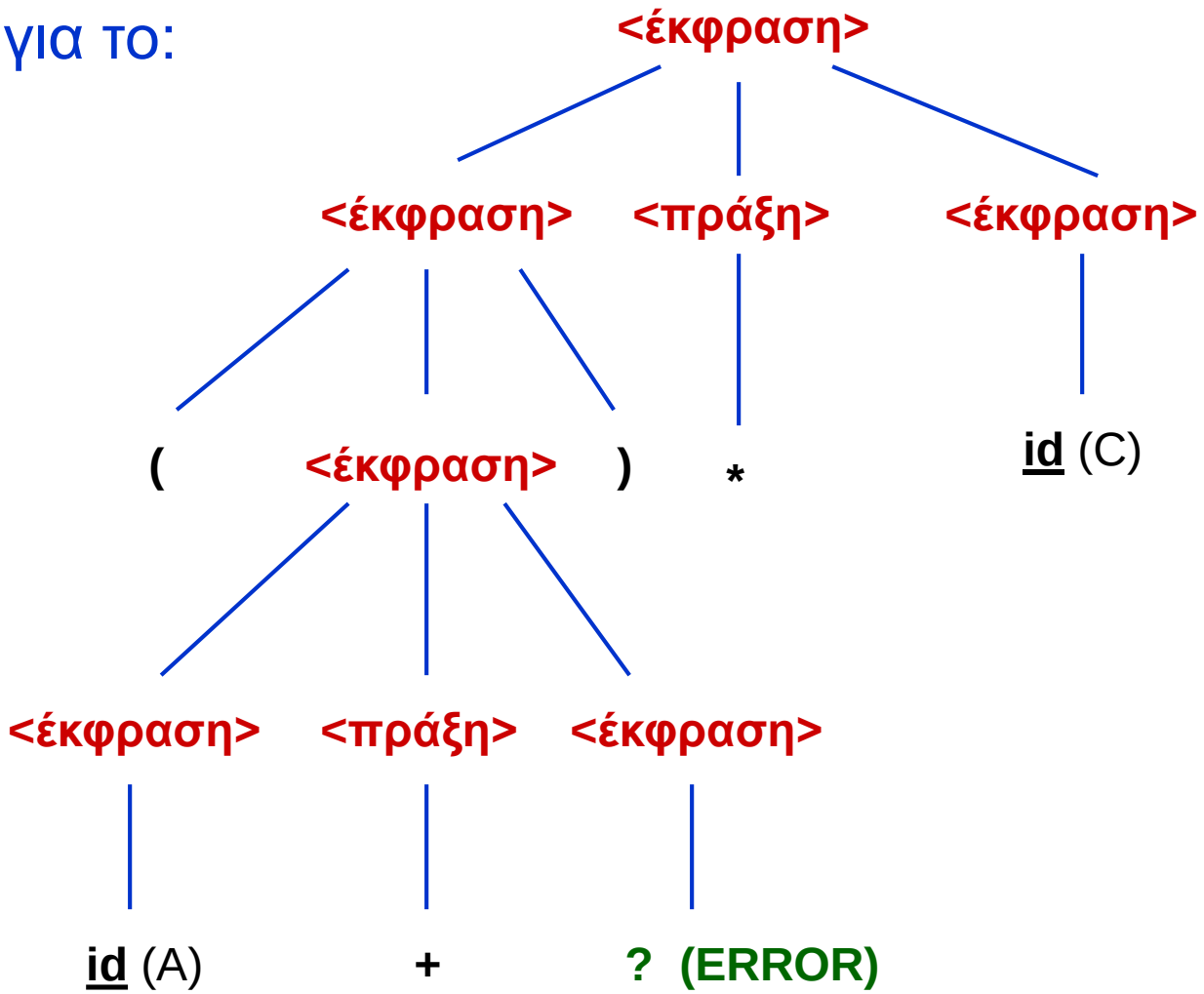
(A + B) * C



Συντακτική Ανάλυση (7)

Parse Tree για το:

(A +) * C



Συντακτική Ανάλυση (8)

Μια πιο σύνθετη Γραμματική:

<A> ::= <E> **=** <B>

<B> ::= <C> | <B> **+** <C> | <B> **-** <C>

<C> ::= <D> | <C> ***** <D> | <C> **/** <D>

<D> ::= <E> | **αρ** | **(**<B>**)**

<E> ::= **id** | **id** **[**<F>**]**

<F> ::= <B> | <F> **,** <B>

Συντακτική Ανάλυση (8)

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>**]**

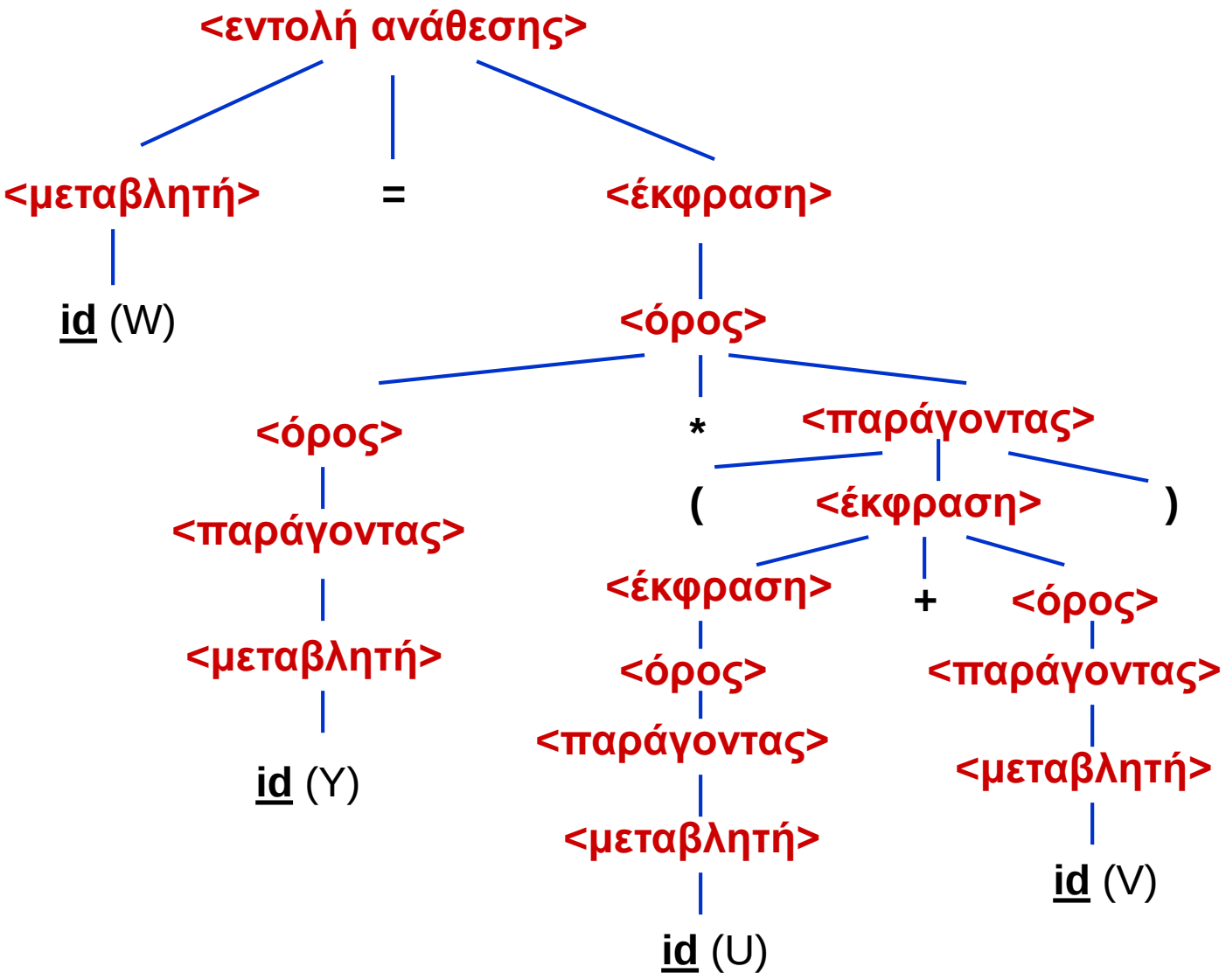
<λίστα δεικτών> ::= <έκφραση> | <λίστα δεικτών> **,** <έκφραση>

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

Συντακτική Ανάλυση (9)

Δέντρο Συντακτικής Ανάλυσης για το

W=Y*(U+V)



Συντακτική Ανάλυση (10)

- 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 (11)

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

  […] : Προαιρετικό
  { … } : 0 ή περισσότερες εμφανίσεις
  (…) : Ομαδοποίηση

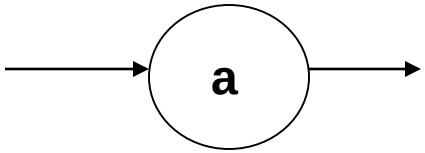
<εντολή ανάθεσης> ::= <μεταβλητή> **=** <έκφραση>
<έκφραση> ::= <όρος> { (+ | -) <όρος> }
<όρος> ::= <παράγοντας> { (* | /) <παράγοντας> }
<παράγοντας> ::= <μεταβλητή> | **αρ** | **(**<έκφραση>**)**
<μεταβλητή> ::= **id** | **id** **[**<λίστα δεικτών>**]**
<λίστα δεικτών> ::= <έκφραση> { **,** <έκφραση>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

Μειονέκτημα: Μερικές φορές κρύβεται η αναδρομή

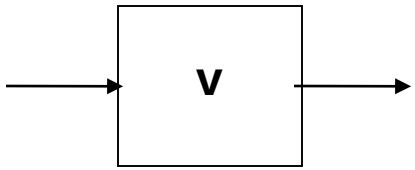
Συντακτική Ανάλυση (12)

- 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**

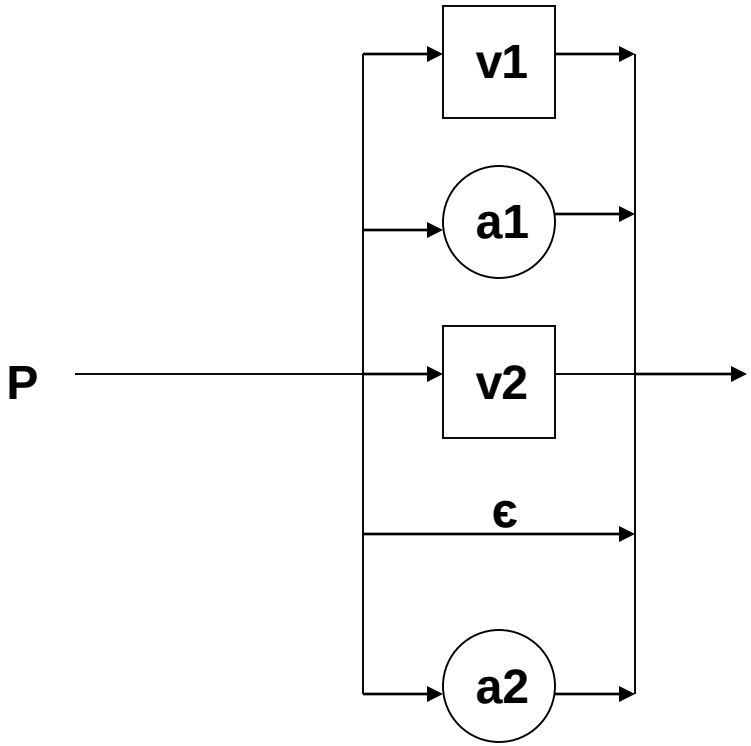


2. Μη-τερματικό Σύμβολο **<v>**



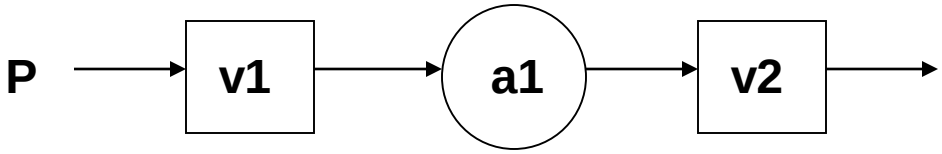
Συντακτική Ανάλυση (13)

3. **<P> ::= <v1> | a1 | <v2> | ϵ | a2**



Συντακτική Ανάλυση (14)

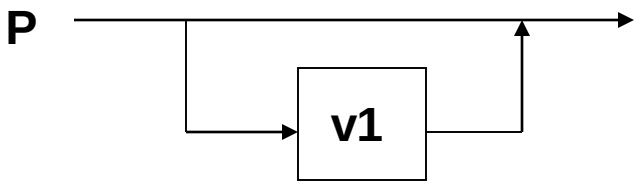
4. **<P> ::= <v1> a1 <v2>**



5. **<P> ::= [<v1>]**

0 ή 1 φορές το <v1>

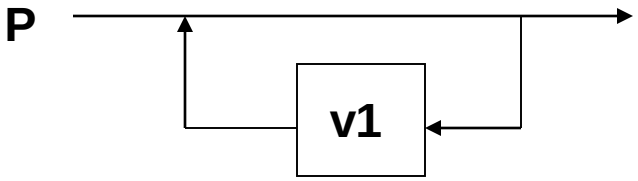
δηλ. <P> ::= ϵ | <v1>



6. **<P> ::= { <v1> }**

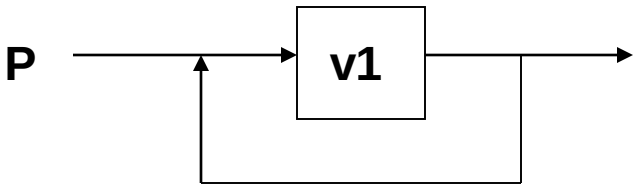
0 ή περισσότερες φορές το <v1>

δηλ. <P> ::= ϵ | <v1><P>



7. **<P> ::= <v1> | <v1><P>**

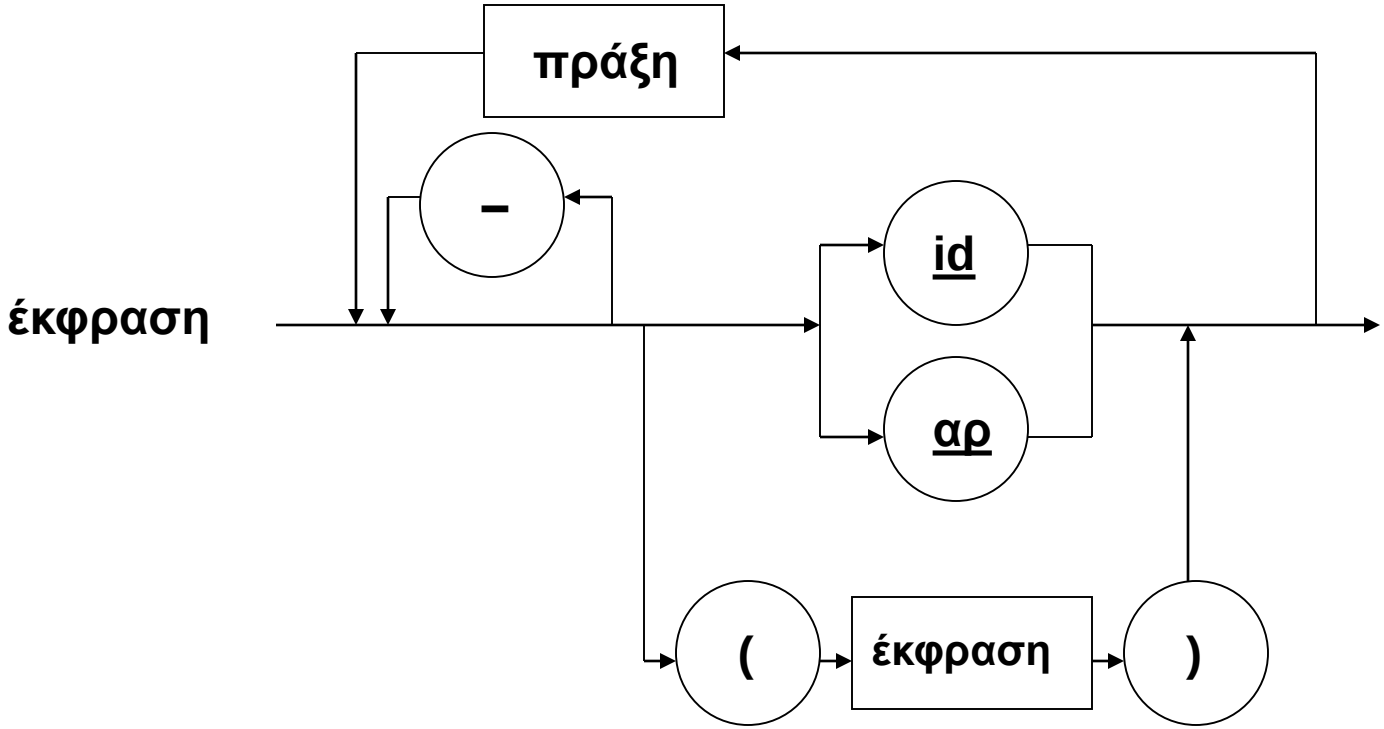
1 ή περισσότερες φορές το <v1>



Συντακτική Ανάλυση (15)

- Το παράδειγμα

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



Συντακτική Ανάλυση (16)

Η σύνθετη Γραμματική για εντολές ανάθεσης:

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

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

| <έκφραση> **-** <όρος>

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

| <όρος> **/** <παράγοντας>

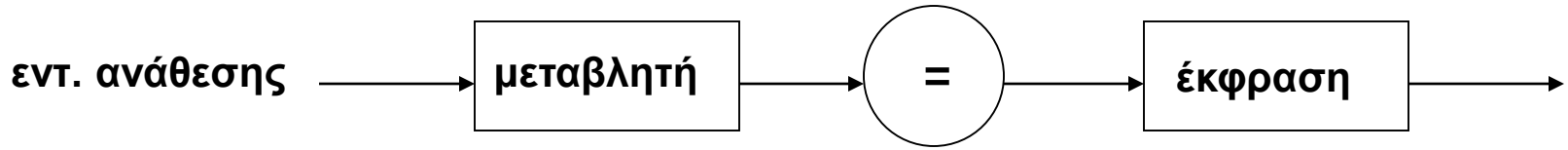
<παράγοντας> ::= <μεταβλητή> | **αρ** | **(**<έκφραση>**)**

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

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

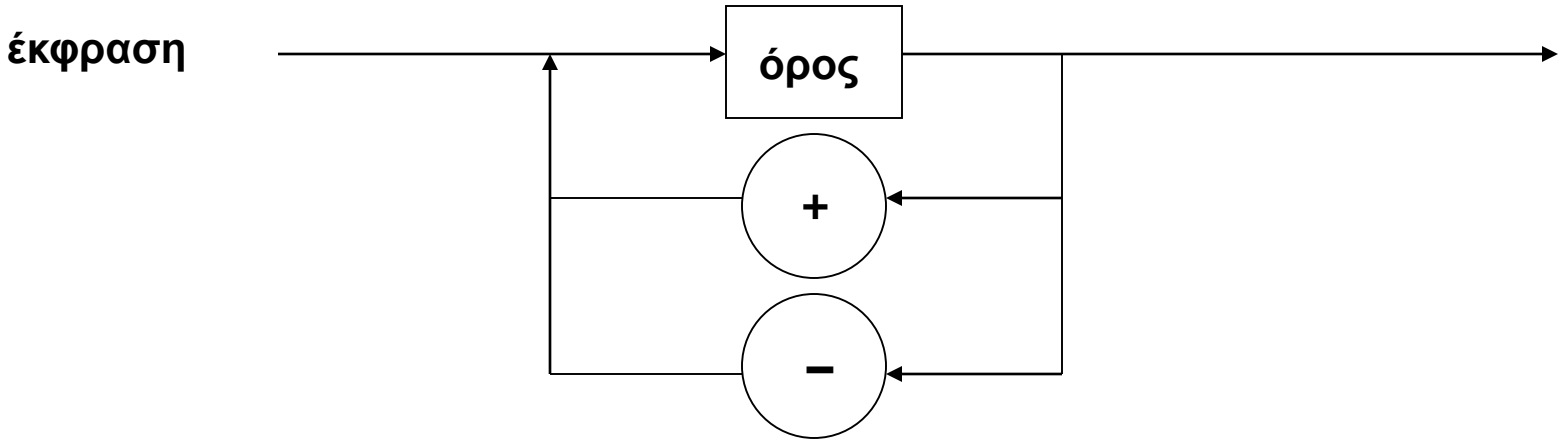
Συντακτική Ανάλυση (17)

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



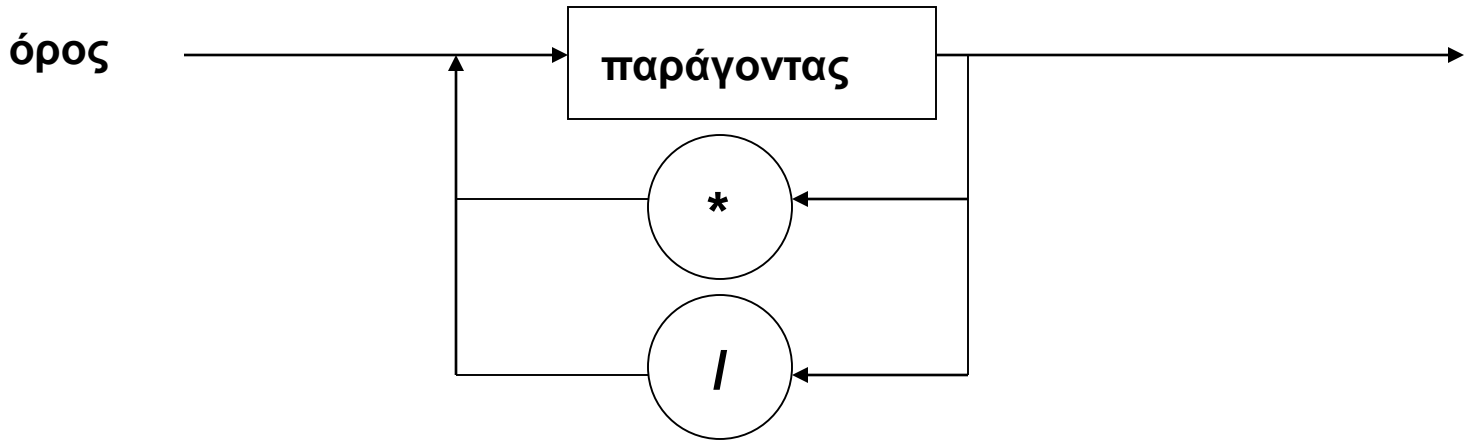
<έκφραση> ::= <όρος> | <έκφραση> **+** <όρος> |

| <έκφραση> **-** <όρος>



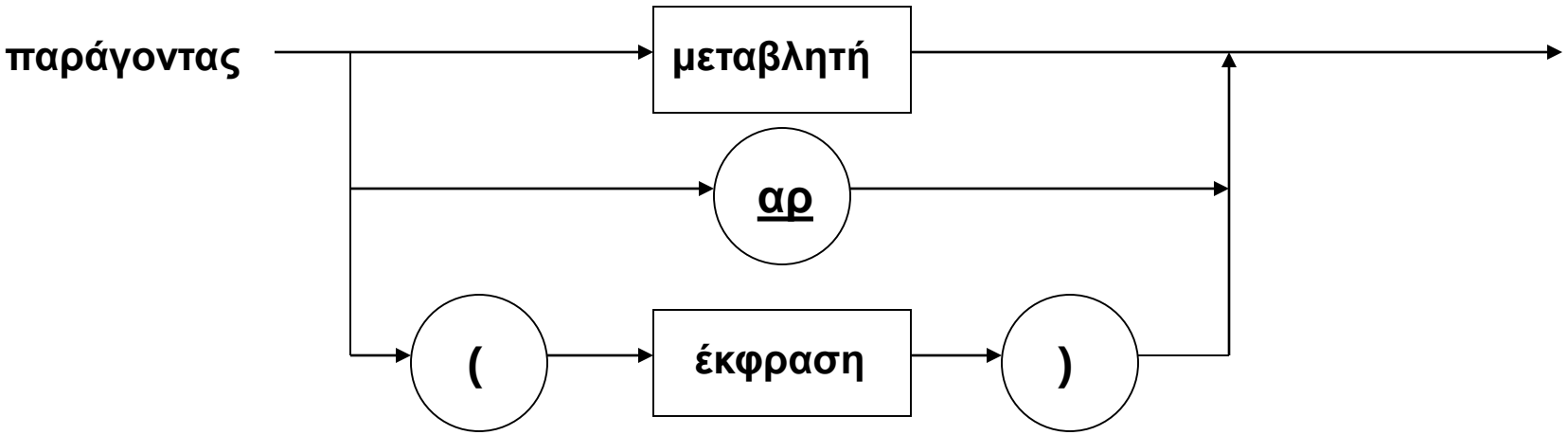
Συντακτική Ανάλυση (18)

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



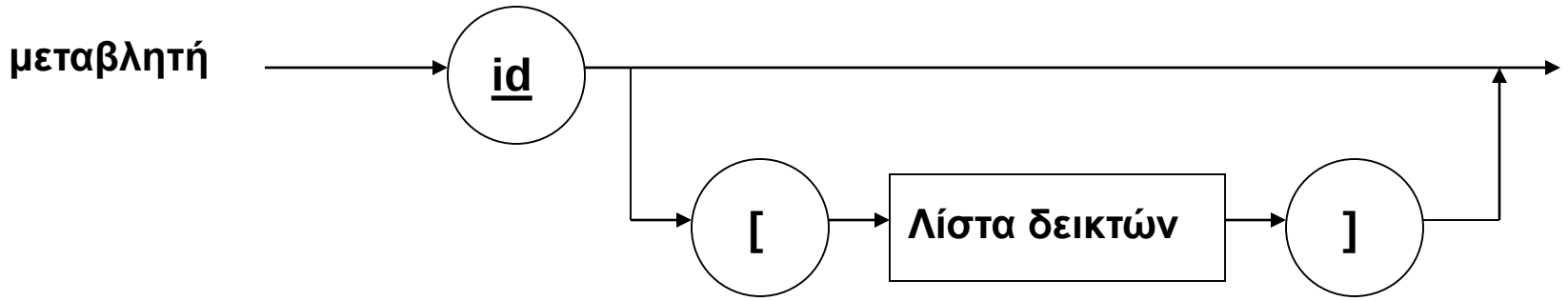
Συντακτική Ανάλυση (19)

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

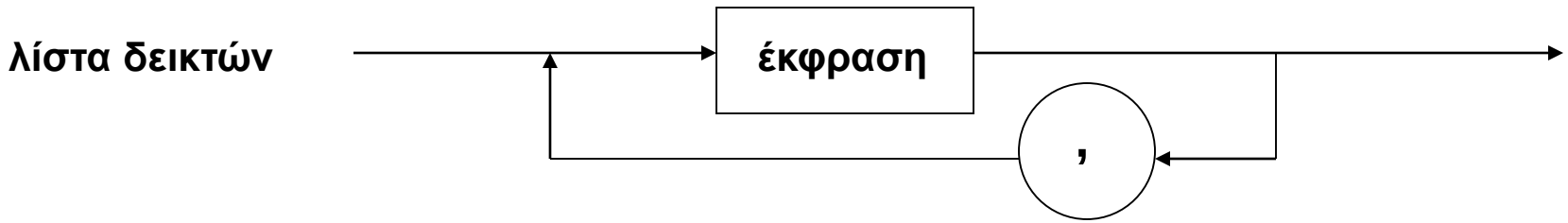


Συντακτική Ανάλυση (20)

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



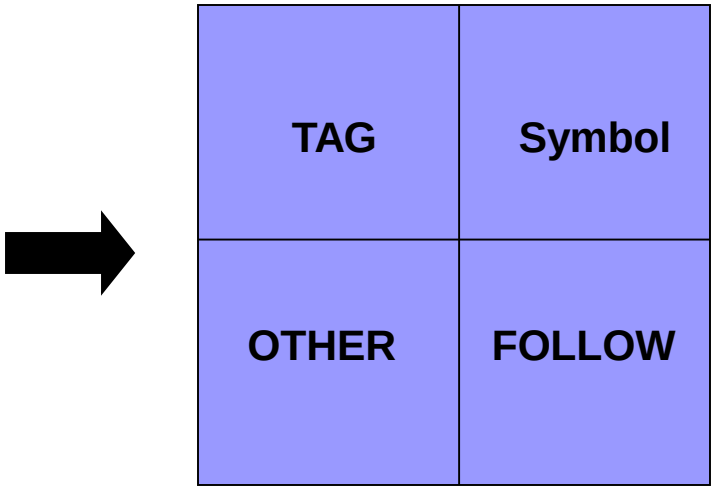
<λίστα δεικτών> ::= <έκφραση> | <λίστα δεικτών> , <έκφραση>



Συντακτική Ανάλυση (21)

- Υλοποίηση Συντακτικών Διαγραμμάτων με **Ειδικές Δομές Δεδομένων**:

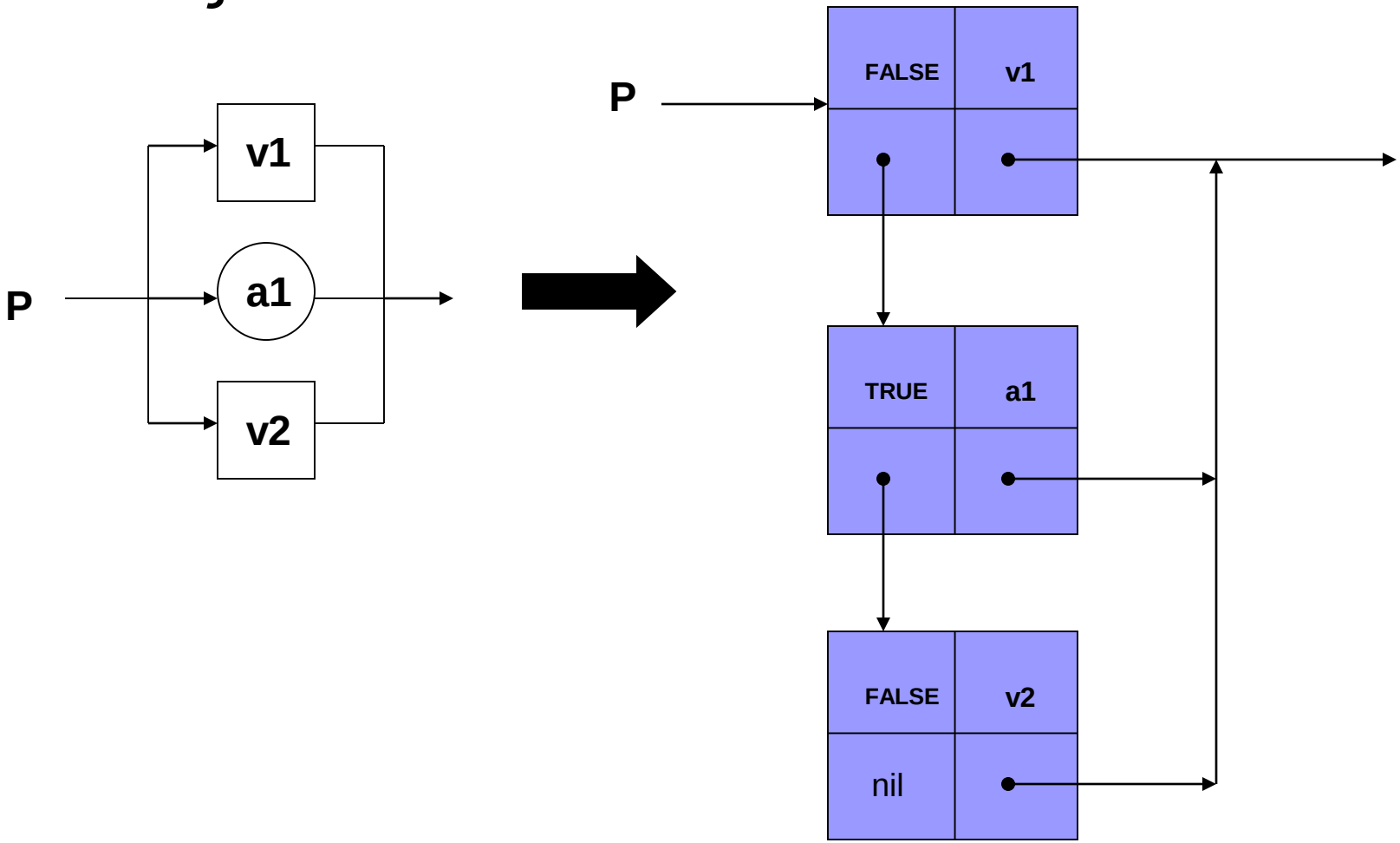
Κόμβος Συντακτικού Διαγράμματος



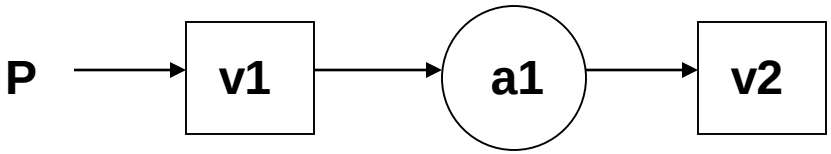
- **TAG**: TRUE αν το Symbol είναι τερματικό σύμβολο
  FALSE αν το Symbol είναι μη-τερματικό σύμβολο
- **OTHER**: Pointer που δείχνει τον επόμενο *εναλλακτικό* (παράλληλο) κόμβο («nil» αν ο κόμβος είναι η τελευταία επιλογή)
- **FOLLOW**: Pointer που δείχνει τον επόμενο *υποχρεωτικό* (σειριακό) κόμβο («nil» αν ο κόμβος είναι ο τελευταίος στη σειρά)

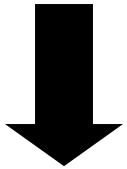
Συντακτική Ανάλυση (22)

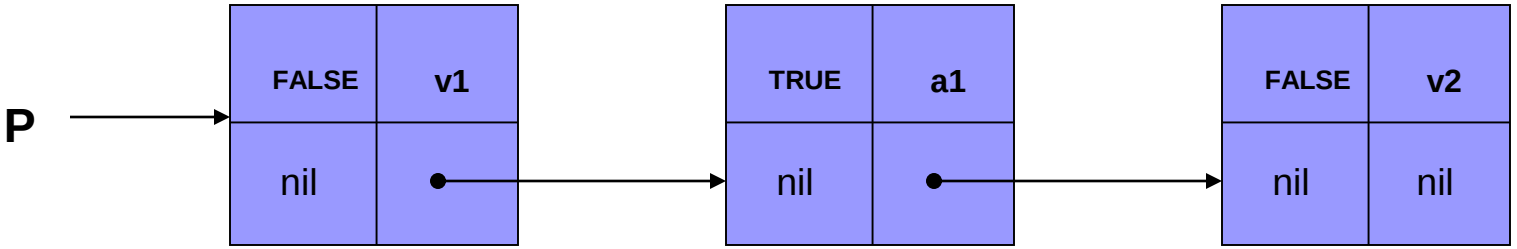
Κανόνες:



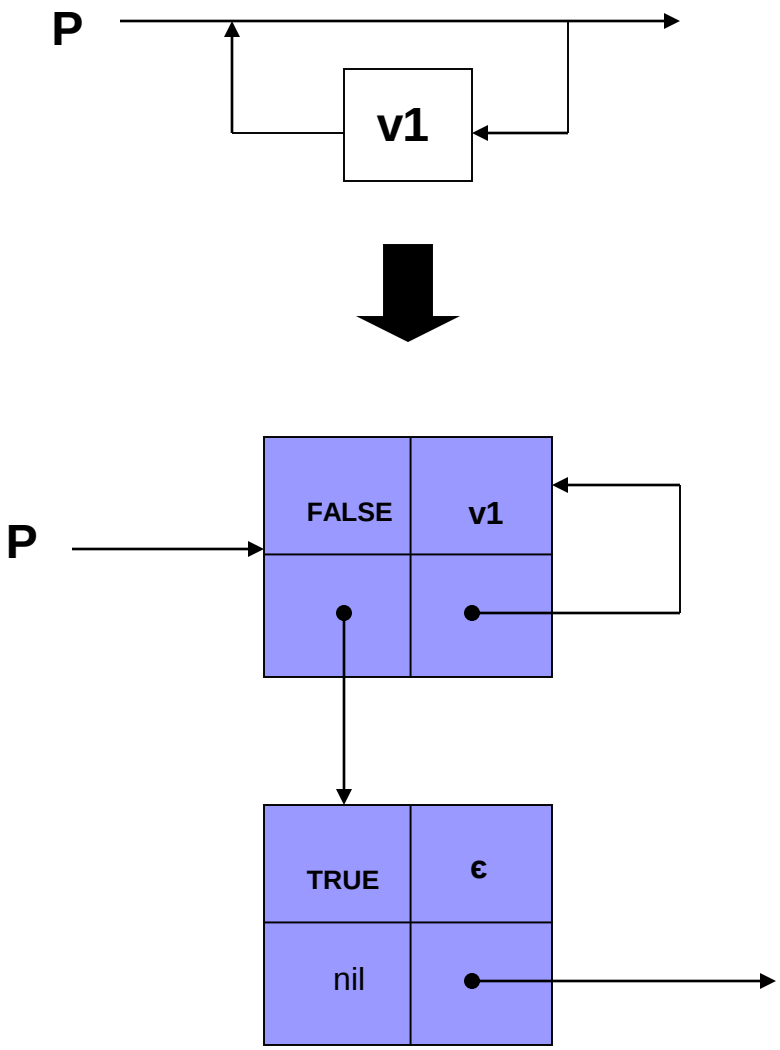
Συντακτική Ανάλυση (23)







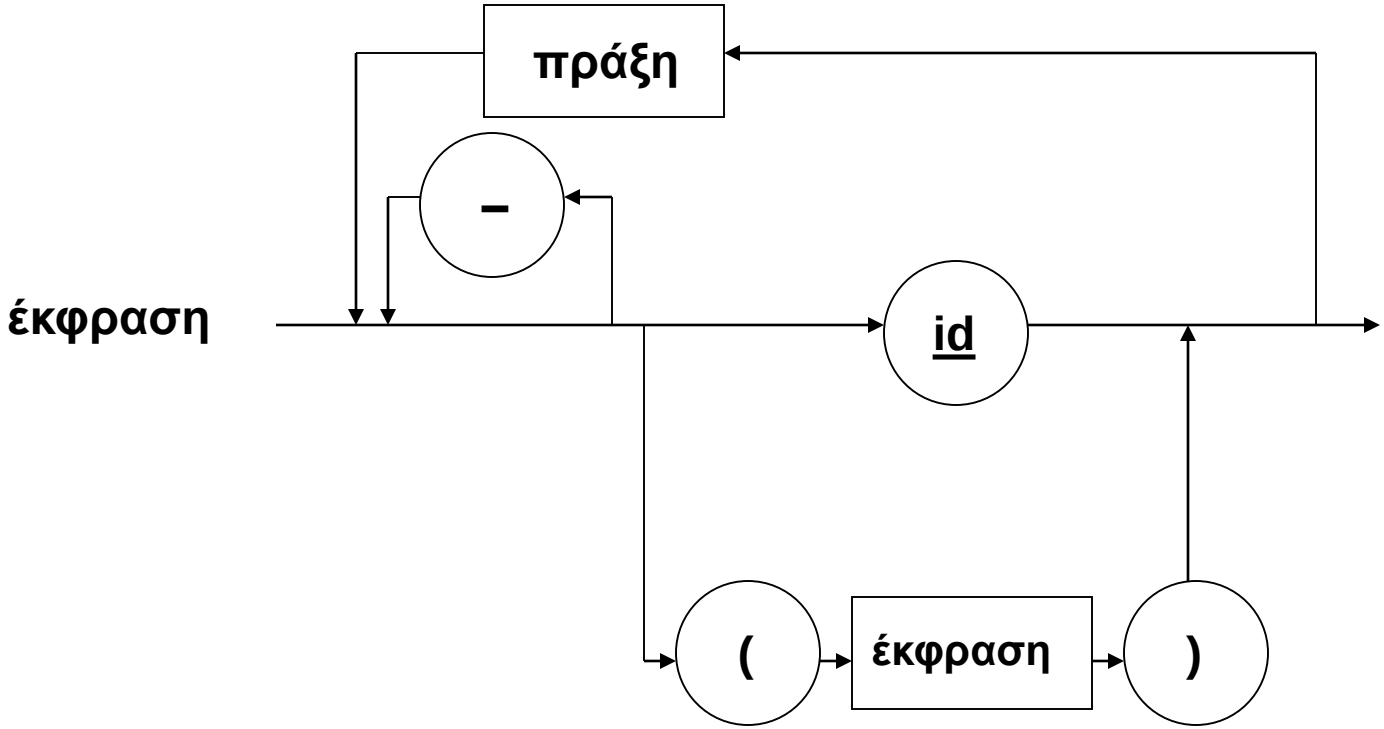
Συντακτική Ανάλυση (24)



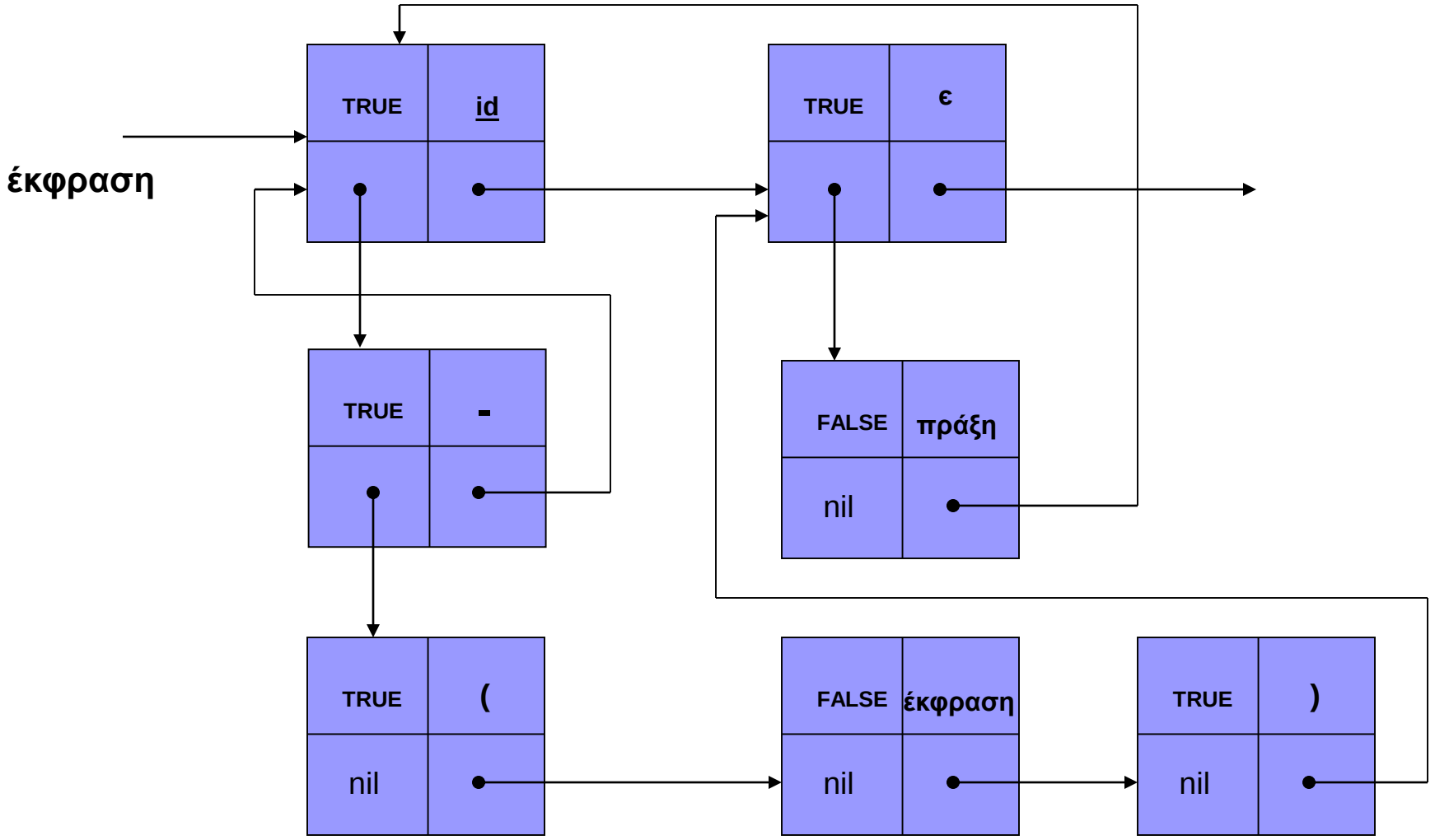
Συντακτική Ανάλυση (25)

- Το παράδειγμα

<έκφραση> ::= **id** | <έκφραση> <πράξη> <έκφραση> |
| **(**<έκφραση>**)** | **-** <έκφραση>



Συντακτική Ανάλυση (26)



Ισοδύναμες Γραμματικές (1)

- **Ισοδύναμες** Γραμματικές:

  Γραμματικές που παράγουν την ίδια γλώσσα.
  - Δηλαδή, οι γραμματικές $\mathbf{G_1}$ και $\mathbf{G_2}$ είναι ισοδύναμες, αν και μόνο αν $\mathbf{L(G_1) = L(G_2)}$
  - 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 την ίδια παραγόμενη συμβολοσειρά.

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

Ισοδύναμες Γραμματικές (3)

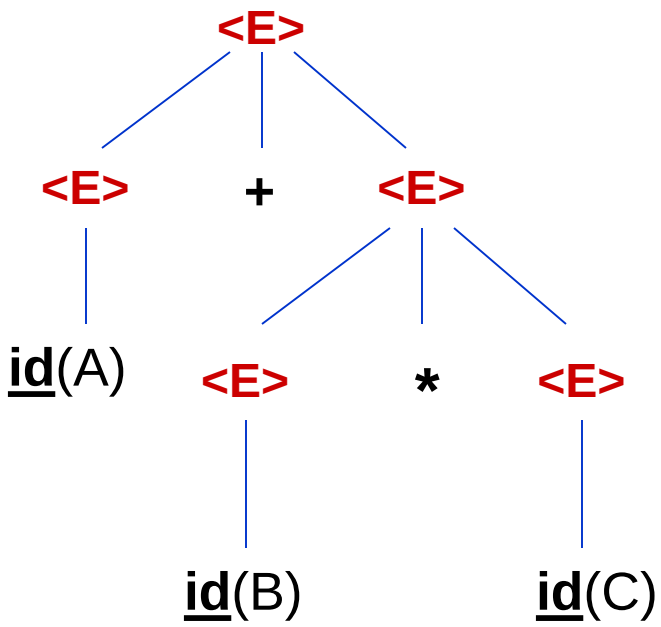
- Παράδειγμα διφορούμενης γραμματικής:

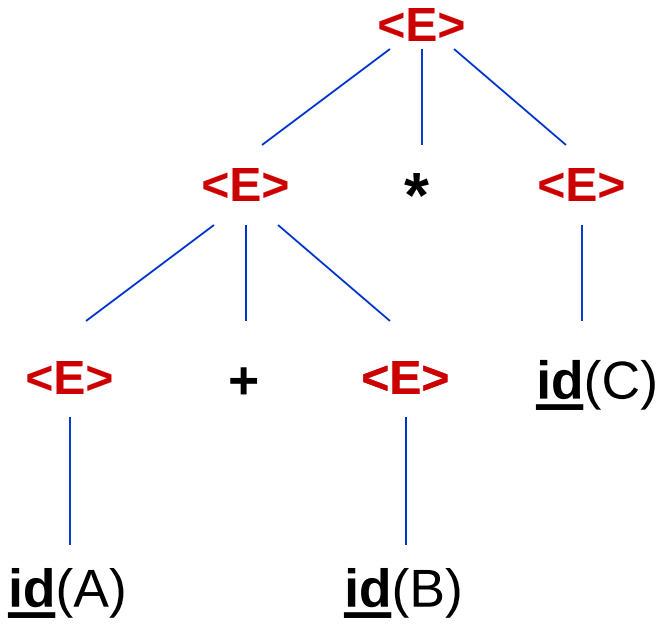
  <E> ::= **id** | <E> **+** <E> | <E> ***** <E> | **(**<E>**)**

  Δέντρα Συντακτικής Ανάλυσης για το:

  A + B * C

Ισοδύναμες Γραμματικές (4)





Ισοδύναμες Γραμματικές (5)

- Ισοδύναμη Γραμματική μη-διφορούμενη:

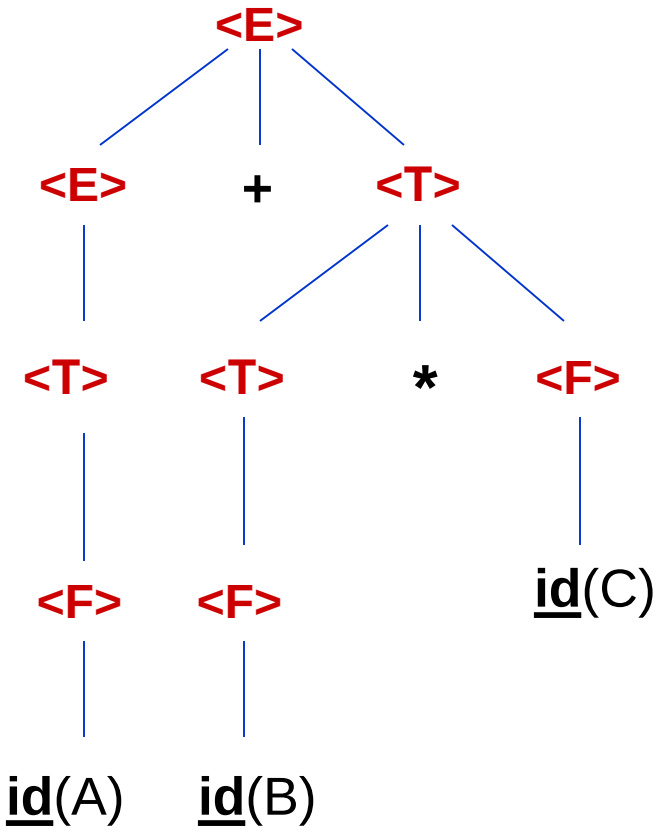
<E> ::= <E> **+** <T> | <T>

<T> ::= <T> ***** <F> | <F>

<F> ::= **(**<E>**)** | **id**

Δηλαδή, «επιβάλλουμε» τη χρήση από τη ρίζα, του δεύτερου αρχικού κανόνα:

Ισοδύναμες Γραμματικές (6)



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

Όμως, δεν είναι διφορούμενη…

Συντακτική Ανάλυση Top-Down (1)

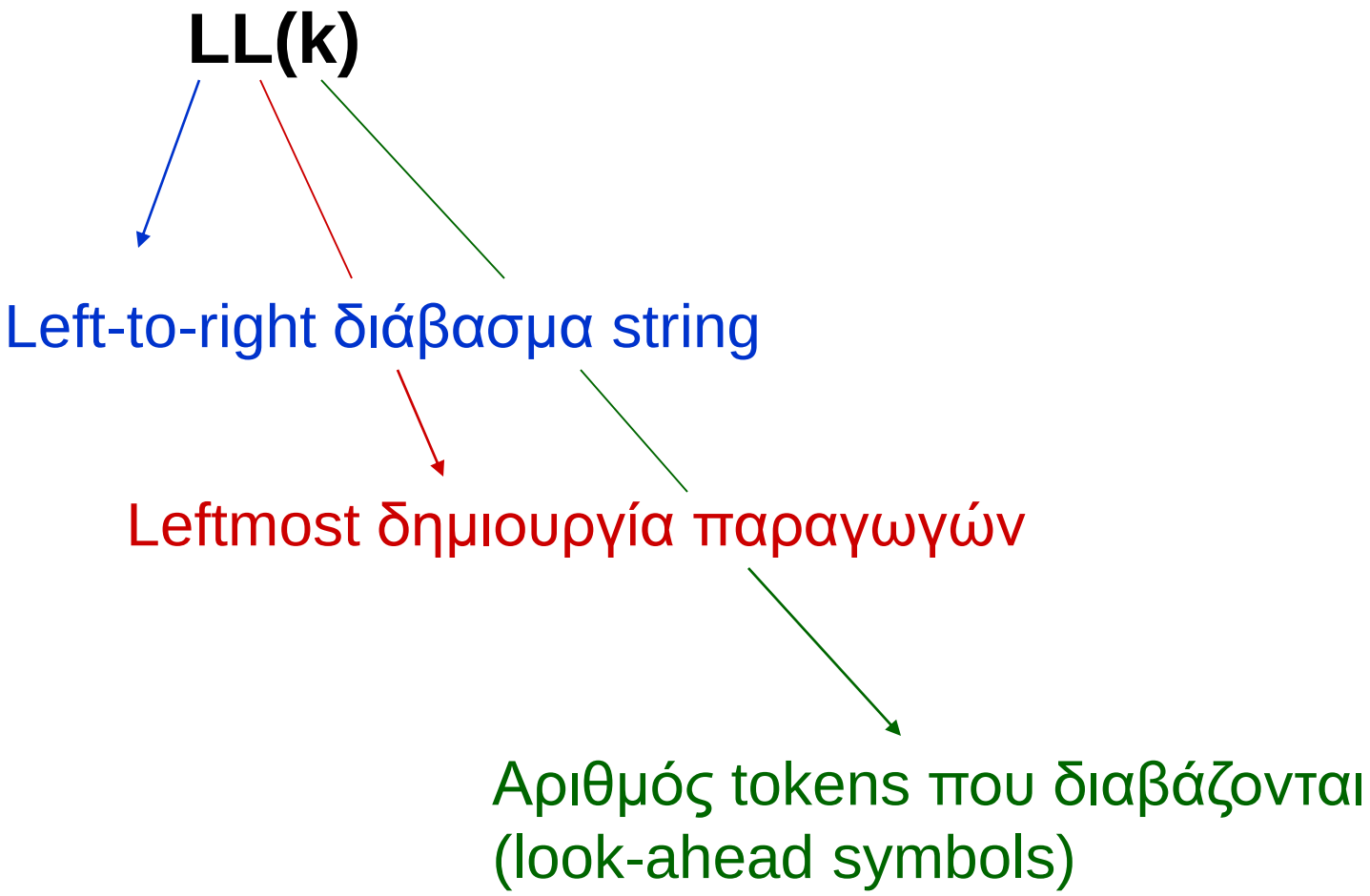
- **Top-Down** Συντακτική Ανάλυση (*Καθοδική*)
- Ο Συντακτικός Αναλυτής (ΣΑ) πρέπει να αποφασίσει για τα εξής:
  1. Ποιος κανόνας παραγωγής θα χρησιμοποιηθεί για να δημιουργηθούν τα «παιδιά» ενός κόμβου;

     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 συνήθεις 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: **<S> ::= (<S>)<S> | ϵ**

String εισόδου: **()**

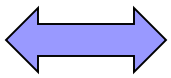
Βήμα	Στοίβα	Είσοδος	Πράξη
0	**<S>**	**()** EOF	Πρόβλεψη <S> ::= (<S>)<S>
1	<S>)<S>(	**()** EOF	Ταίριασμα συμβόλου **(**
2	<S>)<S>	**)** EOF	Πρόβλεψη <S> ::= ϵ
3	<S>)	**)** EOF	Ταίριασμα συμβόλου **)**
4	<S>	EOF	Πρόβλεψη <S> ::= ϵ
5	**ϵ**	EOF	**Αναγνώριση**

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**:

Α. Αντικατάσταση

$$A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n \qquad B \rightarrow \beta_1 A \beta_2$$



$$A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n \qquad B \rightarrow \beta_1\alpha_1\beta_2 \mid \ldots \mid \beta_1\alpha_n\beta_2$$

Συντακτική Ανάλυση Top-Down (7)

Β. Αριστερή Παραγοντοποίηση

$$\mathbf{A \rightarrow \alpha\beta_1 \mid \ldots \mid \alpha\beta_n} \quad \Leftrightarrow \quad \mathbf{A \rightarrow \alpha B}, \quad \mathbf{B \rightarrow \beta_1 \mid \ldots \mid \beta_n}$$

Γ. Απαλοιφή Αριστερής Αναδρομής (ΑΑ)

Άμεση ΑΑ: $A \rightarrow A\alpha$

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

$$\mathbf{A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m} \quad \Leftrightarrow \quad \mathbf{A \rightarrow \beta_1 B \mid \ldots \mid \beta_m B}, \quad \mathbf{B \rightarrow \alpha_1 B \mid \ldots \mid \alpha_n B \mid \epsilon}$$

Συντακτική Ανάλυση Top-Down (8)

- Παράδειγμα:

  <X> ::= <B> a <A> b

  <A> ::= a <A> | a

  <B> ::= <B> b | b

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

  - Στη 2η παραγωγή υπάρχουν 2 εναλλακτικοί κανόνες που αρχίζουν με το ίδιο σύμβολο (a).
  - Η 3η 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 2η παραγωγή θα χρησιμοποιήσουμε το μετασχηματισμό της *Αριστερής Παραγοντοποίησης*:

<A> ::= a <A> | a ⟺ <A> ::= a <C>
<C> ::= <A> | ϵ

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

<B> ::= <B> b | b ⟺ <B> ::= b <D>
<D> ::= b<D> | ϵ

Συντακτική Ανάλυση Top-Down (10)

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

<X> ::= <B> a <A> b

<A> ::= a <C>

<C> ::= <A> | ϵ

<B> ::= b <D>

<D> ::= b<D> | ϵ

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:

<S> ::= r <B>

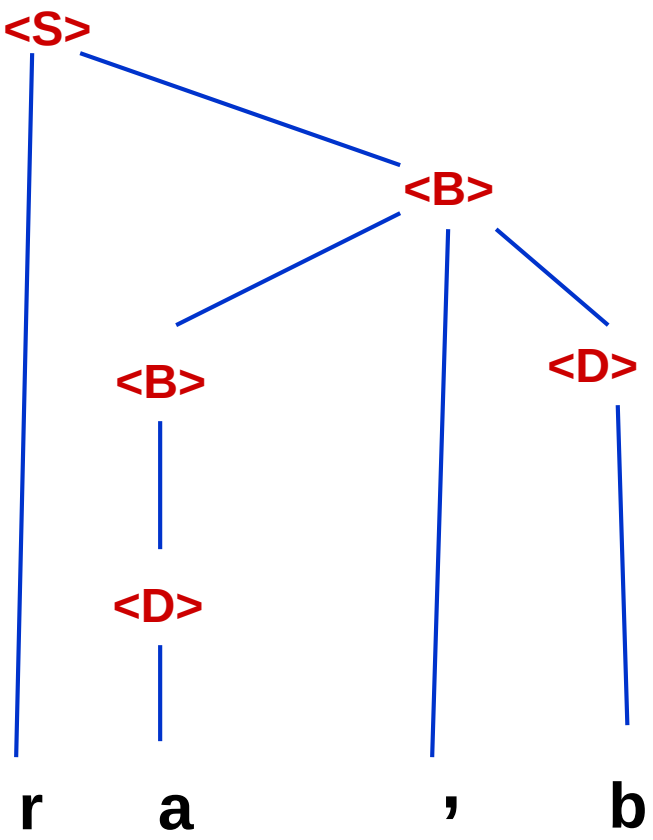
<B> ::= <D> | <B> **,** <D>

<D> ::= a | b

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

Συντακτική Ανάλυση Bottom-Up (3)

<S> ::= r <B>
<B> ::= <D> | <B> , <D>
<D> ::= a | 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)

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

Συντακτική Ανάλυση Bottom-Up (6)

Προβλήματα:

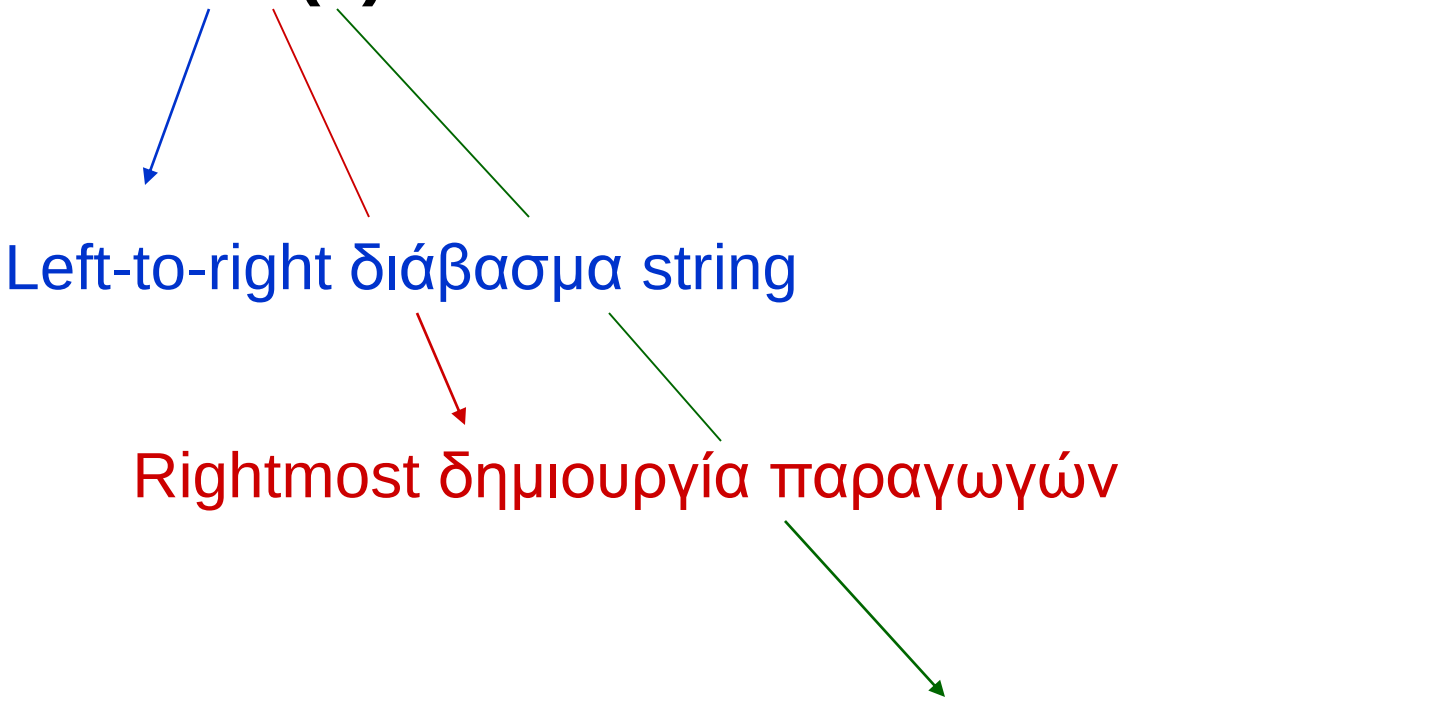
- Στο Βήμα 4 υπήρχαν δύο επιλογές: ολίσθηση ή ελάττωση με τον κανόνα <S> ::= r <B>. Η 2η επιλογή καταλήγει σε αδιέξοδο, γιατί η ανάλυση τελειώνει χωρίς να έχει σαρωθεί όλο το string εισόδου: **Σύγκρουση ολίσθησης-ελάττωσης** (shift-reduce conflict)
- Στο Βήμα 7 υπήρχαν δύο επιλογές: ελάττωση με τον κανόνα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



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)…