

## Σύνταξη & Συντακτική Ανάλυση



Rene Magritte, Le Seize Septembre

Κωστής Σαγώνας &lt;kostis@cs.ntua.gr&gt;

### Μια γραμματική για τα Αγγλικά

Μια πρόταση αποτελείται από μια ουσιαστική φράση, ένα ρήμα, και μια ουσιαστική φράση

$$\langle S \rangle ::= \langle NP \rangle \langle V \rangle \langle NP \rangle$$

Μια ουσιαστική φράση αποτελείται από ένα άρθρο και ένα ουσιαστικό

$$\langle NP \rangle ::= \langle A \rangle \langle N \rangle$$

Ρήματα είναι τα εξής...

$$\langle V \rangle ::= \text{loves} \mid \text{hates} \mid \text{eats}$$

Άρθρα είναι τα εξής...

$$\langle A \rangle ::= \text{a} \mid \text{the}$$

Ουσιαστικά είναι τα εξής...

$$\langle N \rangle ::= \text{dog} \mid \text{cat} \mid \text{rat}$$

Σύνταξη και Συντακτική Ανάλυση

2

### Πως δουλεύει μια γραμματική

- Μια γραμματική είναι ένα σύνολο κανόνων που ορίζουν το πως κατασκευάζεται ένα **συντακτικό δένδρο**
- Ξεκινάμε βάζοντας το  $\langle S \rangle$  στη ρίζα του δένδρου
- Οι κανόνες της γραμματικής λένε πως μπορούμε να προσθέσουμε παιδιά σε κάθε σημείο του δένδρου
- Για παράδειγμα, ο κανόνας
 
$$\langle S \rangle ::= \langle NP \rangle \langle V \rangle \langle NP \rangle$$
 λέει ότι μπορούμε να προσθέσουμε κόμβους  $\langle NP \rangle$ ,  $\langle V \rangle$ , και  $\langle NP \rangle$ , με αυτή τη σειρά, ως παιδιά του κόμβου  $\langle S \rangle$

Σύνταξη και Συντακτική Ανάλυση

3

### Γραμματική για αριθμητικές εκφράσεις

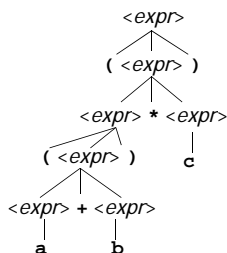
$$\begin{aligned} \langle \text{expr} \rangle ::= & \langle \text{expr} \rangle + \langle \text{expr} \rangle \\ & | \langle \text{expr} \rangle * \langle \text{expr} \rangle \\ & | ( \langle \text{expr} \rangle ) \\ & | \text{a} \mid \text{b} \mid \text{c} \end{aligned}$$

- Μια αριθμητική έκφραση μπορεί να είναι
  - το άθροισμα δύο άλλων εκφράσεων, ή
  - το γινόμενο δύο εκφράσεων, ή
  - μια έκφραση που περικλείεται από παρενθέσεις, ή
  - κάποια από τις μεταβλητές a, b, ή c

Σύνταξη και Συντακτική Ανάλυση

4

### Συντακτικό δένδρο



Σύνταξη και Συντακτική Ανάλυση

5

### Συστατικά μιας γραμματικής

αρχικό σύμβολο  $\langle S \rangle ::= \langle NP \rangle \langle V \rangle \langle NP \rangle$

κανόνας παραγωγής  $\langle NP \rangle ::= \langle A \rangle \langle N \rangle$

$\langle V \rangle ::= \text{loves} \mid \text{hates} \mid \text{eats}$

μη τερματικά σύμβολα  $\langle A \rangle ::= \text{a} \mid \text{the}$

$\langle N \rangle ::= \text{dog} \mid \text{cat} \mid \text{rat}$

τερματικά σύμβολα (λεκτικές μονάδες)

Σύνταξη και Συντακτική Ανάλυση

6

## Ορισμός γραμματικών σε μορφή Backus-Naur

- Μια γραμματική σε μορφή Backus-Naur αποτελείται από
  - Ένα σύνολο από **λεκτικές μονάδες (tokens)**
    - Συμβολοσειρές που αποτελούν τα μικρότερα αδιαίρετα κομμάτια της σύνταξης του προγράμματος
  - Ένα σύνολο από μη **τερματικά σύμβολα**
    - Συμβολοσειρές που εγκλείονται σε αγκύλες, π.χ.  $\langle NP \rangle$ , και αντιπροσωπεύουν κομμάτια του συντακτικού της γλώσσας
    - Δε συναντιούνται στο πρόγραμμα, αλλά είναι σύμβολα που βρίσκονται στο αριστερό μέρος κάποιων κανόνων της γραμματικής
  - Το **αρχικό σύμβολο** της γραμματικής
    - Ένα συγκεκριμένο μη τερματικό σύμβολο που αποτελεί τη ρίζα του συντακτικού δένδρου για κάθε αποδεκτό από τη γλώσσα πρόγραμμα
  - Ένα σύνολο από **κανόνες παραγωγής**

Σύνταξη και Συντακτική Ανάλυση

7

## Κανόνες παραγωγής

- Οι κανόνες παραγωγής χρησιμοποιούνται για την κατασκευή του συντακτικού δένδρου
- Κάθε κανόνας έχει τη μορφή  $A ::= \Delta$ 
  - Το αριστερό μέρος  $A$  αποτελείται από ένα μη τερματικό σύμβολο
  - Το δεξί μέρος  $\Delta$  είναι μια ακολουθία από τερματικά (λεκτικές μονάδες) και μη τερματικά σύμβολα
- Κάθε κανόνας προσδιορίζει έναν πιθανό τρόπο κατασκευής του συντακτικού υποδένδρου που
  - έχει ως ρίζα του το μη τερματικό σύμβολο στο αριστερό μέρος  $A$  του κανόνα και
  - ως παιδιά αυτής της ρίζας (με την ίδια σειρά εμφάνισης) έχει τα σύμβολα στο δεξί μέρος  $\Delta$  του κανόνα

Σύνταξη και Συντακτική Ανάλυση

8

## Επιλογές στη γραφή των κανόνων παραγωγής

- Όταν υπάρχουν περισσότεροι από ένας κανόνες παραγωγής με το ίδιο αριστερό μέρος, μπορούμε να κάνουμε χρήση της παρακάτω συντομογραφίας
- Στη BNF γραμματική μπορούμε να δώσουμε το αριστερό μέρος, το διαχωριστή  $::=$ , και μετά μια ακολουθία από δεξιά μέρη που διαχωρίζονται από το ειδικό σύμβολο  $|$
- Οι δύο γραμματικές είναι ίδιες

```
<expr> ::= <expr> + <expr>
        | <expr> * <expr>
        | ( <expr> )
        | a | b | c
```

```
<expr> ::= <expr> + <expr>
<expr> ::= <expr> * <expr>
<expr> ::= ( <expr> )
<expr> ::= a
<expr> ::= b
<expr> ::= c
```

Σύνταξη και Συντακτική Ανάλυση

9

## Κανόνες παραγωγής του κενού

- Το ειδικό μη τερματικό  $\langle empty \rangle$  χρησιμοποιείται σε περιπτώσεις που θέλουμε κάποιους κανόνες να μην παράγει τίποτα
- Για παράδειγμα, οι παρακάτω κανόνες ορίζουν τη δομή **if-then** των περισσότερων γλωσσών, η οποία επιτρέπει την ύπαρξη ενός προαιρετικού **else**

```
<if-stmt> ::= if <expr> then <stmt> <else-part>
<else-part> ::= else <stmt> | <empty>
```

Σύνταξη και Συντακτική Ανάλυση

10

## Κατασκευή συντακτικών δένδρων

- Αρχίζουμε την κατασκευή βάζοντας το αρχικό σύμβολο της γραμματικής στη ρίζα του δένδρου
- Προσθέτουμε παιδιά σε κάθε μη τερματικό σύμβολο, χρησιμοποιώντας *κάποιον* από τους *κανόνες παραγωγής* της γλώσσας για το συγκεκριμένο μη τερματικό
- Η διαδικασία τερματίζει όταν όλα τα φύλλα του δένδρου αποτελούνται από λεκτικές μονάδες
- Η συμβολοσειρά που αντιστοιχεί στο δένδρο βρίσκεται διαβάζοντας τα φύλλα του δένδρου από αριστερά προς τα δεξιά

Σύνταξη και Συντακτική Ανάλυση

11

## Παραδείγματα

- Τα συντακτικά δένδρα για τις παρακάτω εκφράσεις

$a+b$        $(a+b)$        $(a+(b))$        $a*b+c$

- Η κατασκευή των συντακτικών δένδρων είναι η δουλειά του συντακτικού αναλυτή ενός μεταγλωττιστή
- Υπάρχουν διάφοροι αποδοτικοί αλγόριθμοι και εργαλεία για ημιαυτόματη κατασκευή του συντακτικού αναλυτή

Σύνταξη και Συντακτική Ανάλυση

12

## Ορισμός γλωσσών

- Για να ορίσουμε τη σύνταξη των γλωσσών προγραμματισμού χρησιμοποιούμε γραμματικές
- Η γλώσσα που ορίζεται από μια γραμματική είναι το σύνολο των συμβολοσειρών για τα οποία η γραμματική μπορεί να παράξει συντακτικά δένδρα
- Τις περισσότερες φορές το σύνολο αυτό είναι άπειρο (παρόλο που η γραμματική είναι πεπερασμένη)
- Η κατασκευή μιας γραμματικής για μια γλώσσα μοιάζει λίγο με προγραμματισμό...

Σύνταξη και Συντακτική Ανάλυση

13

## Παράδειγμα κατασκευής γραμματικής (1)

Συνήθως γίνεται με χρήση της τεχνικής “διαίρει και βασίλευε” (divide and conquer)

- **Παράδειγμα:** κατασκευή της γλώσσας των δηλώσεων της Java:

- αρχικά, η δήλωση έχει ένα όνομα τύπου
- στη συνέχεια μια ακολουθία από μεταβλητές που διαχωρίζονται με κόμματα (όπου κάθε μεταβλητή μπορεί να πάρει μια αρχική τιμή)
- και στο τέλος ένα ερωτηματικό (semicolon)

```
float a;
boolean a, b, c;
int a = 1, b, c = 1 + 2;
```

Σύνταξη και Συντακτική Ανάλυση

14

## Παράδειγμα κατασκευής γραμματικής (2)

- Αρχικά ας αγνοήσουμε την πιθανή ύπαρξη αρχικοποιητών:

```
<var-decl> ::= <type-name> <declarator-list> ;
```

- Ο κανόνας για τα ονόματα των πρωτόγονων τύπων (primitive types) είναι απλούστατος:

```
<type-name> ::= boolean | byte | short | int
              | long | char | float | double
```

**Σημείωση:** δεν παίρνουμε υπόψη κατασκευασμένους τύπους (constructed types): ονόματα κλάσεων, ονόματα interfaces, και τύπους πινάκων

Σύνταξη και Συντακτική Ανάλυση

15

## Παράδειγμα κατασκευής γραμματικής (3)

- Η ακολουθία των μεταβλητών που διαχωρίζονται με κόμματα έχει ως εξής:

```
<declarator-list> ::= <declarator>
                  | <declarator> , <declarator-list>
```

- Όπου ξανά, έχουμε προς το παρόν αγνοήσει τους πιθανούς αρχικοποιητές των μεταβλητών

Σύνταξη και Συντακτική Ανάλυση

16

## Παράδειγμα κατασκευής γραμματικής (4)

- Οι δηλωτές μεταβλητών, με ή χωρίς αρχικοποιήσεις, ορίζονται ως:

```
<declarator> ::= <variable-name>
              | <variable-name> = <expr>
```

- Για ολόκληρη τη Java:

- Πρέπει να επιτρέψουμε και ζεύγη από αγκύλες μετά το όνομα των μεταβλητών για τη δήλωση των πινάκων
- Πρέπει επίσης να ορίσουμε και τη σύνταξη των αρχικοποιητών πινάκων
- (Φυσικά θέλουμε και ορισμούς για τα μη τερματικά σύμβολα <variable-name> και <expr>)

Σύνταξη και Συντακτική Ανάλυση

17

## Πως ορίζονται οι λεκτικές μονάδες;

- Κομμάτια της γλώσσας που επιλέγουμε να τα θεωρούμε ως μη κατασκευαζόμενα από μικρότερα κομμάτια
- Μεταβλητές (i), λέξεις κλειδιά (i.f), τελεστές (==, ++), σταθερές (123.4), κ.λπ.
- Οι γραμματικές που έχουμε ορίσει δίνουν τη δομή των φράσεων (*phrase structure*): πως το πρόγραμμα κατασκευάζεται από μια σειρά λεκτικών μονάδων
- Πρέπει επιπλέον να ορίσουμε και τη λεκτική δομή (*lexical structure*): πως ένα αρχείο χωρίζεται σε λεκτικές μονάδες



Σύνταξη και Συντακτική Ανάλυση

18

## Δύο επιλογές ορισμού της λεκτικής δομής

- Μια κοινή γραμματική
  - Οι χαρακτήρες είναι οι μοναδικές λεκτικές μονάδες
  - Συνήθως δεν ακολουθείται αυτή η επιλογή: κενά και σχόλια περιπλέκουν αρκετά τη γραμματική και την καθιστούν μη αναγνώσιμη
- Ξεχωριστές γραμματικές
  1. Μία που ορίζει πώς προκύπτουν οι λεκτικές μονάδες από ένα αρχείο με χαρακτήρες
    - Η γραμματική αυτή συνήθως είναι μια κανονική γραμματική (regular grammar) και χρησιμοποιείται από το **λεκτικό αναλυτή (scanner)**
  2. Μία που ορίζει πώς προκύπτουν τα συντακτικά δένδρα από μία ακολουθία λεκτικών μονάδων
    - Η γραμματική αυτή συνήθως είναι μια γραμματική ελεύθερη συμφραζομένων (context-free grammar) και χρησιμοποιείται από το **συντακτικό αναλυτή (parser)**

Σύνταξη και Συντακτική Ανάλυση

19

## Ιστορικές Σημειώσεις (1)

- Κάποιες παλιές γλώσσες δε διαχωρίζαν τη λεκτική από την φραστική δομή
  - Παλιές εκδόσεις της Fortran και της Algol επέτρεπαν κενά σε οποιοδήποτε σημείο, ακόμα και στο μέσο μιας λέξης κλειδί!
  - Άλλες γλώσσες, π.χ. η PL/I, επιτρέπουν τη χρήση λέξεων κλειδιών ως μεταβλητές
    - (Το ίδιο συμβαίνει και στην ML, αλλά εκεί δεν αποτελεί πρόβλημα.)
- Τα παραπάνω
  - προσθέτουν δυσκολία στην λεκτική και συντακτική ανάλυση και
  - μειώνουν την αναγνωσιμότητα των προγραμμάτων

Σύνταξη και Συντακτική Ανάλυση

20

## Ιστορικές Σημειώσεις (2)

- Κάποιες γλώσσες έχουν **λεκτική δομή σταθερής μορφής (fixed-format)**—τα κενά είναι σημαντικά
  - Μία εντολή ανά γραμμή (π.χ. της διάτρητης κάρτας)
  - Οι πρώτες 7 θέσεις κάθε γραμμής για την ταμπέλα (label)
- Οι πρώτες διάλεκτοι της Fortran, Cobol, και της Basic
- Σχεδόν οι περισσότερες μοντέρνες γλώσσες είναι **ελεύθερης μορφής (free-format)**: τα κενά αγνοούνται
  - Π.χ. Algol, Pascal, Java
  - Μερικές άλλες (C, C++) διαφέρουν λίγο λόγω του προεπεξεργαστή



Σύνταξη και Συντακτική Ανάλυση

21

## Άλλες μορφές γραμματικών

- Μικρές διαφοροποιήσεις της μορφής Backus-Naur (BNF)
  - Χρήση  $\rightarrow$  ή  $=$  αντί για  $::=$
  - Όχι  $\langle \rangle$  αλλά κάποιο ειδικό font ή χρήση αποστρόφων για τις λεκτικές μονάδες ώστε να ξεχωρίζονται εύκολα από τα μη τερματικά σύμβολα
- Επεκτάσεις της μορφής Backus-Naur (EBNF)
  - Πρόσθετος συμβολισμός για την απλοποίηση κάποιων κανόνων:
    - $\{x\}$  υποδηλώνει μηδέν ή περισσότερες επαναλήψεις του  $x$
    - $[x]$  υποδηλώνει ότι το  $x$  είναι προαιρετικό (δηλαδή  $x \mid \langle \text{empty} \rangle$ )
    - $()$  για ομαδοποίηση
    - $|$  οπουδήποτε για να υποδηλώσει επιλογή
    - Αποστρόφους γύρω από τις λεκτικές μονάδες ούτως ώστε να ξεχωρίζονται από τα παραπάνω μετασύμβολα
- Συντακτικά διαγράμματα

Σύνταξη και Συντακτική Ανάλυση

22

## Παραδείγματα EBNF

```
<if-stmt> ::= if <expr> then <stmt> [else <stmt>]
```

```
<stmt-list> ::= {<stmt> ;}
```

```
<thing-list> ::= { (<stmt> | <declaration>); }
```

Σύνταξη και Συντακτική Ανάλυση

23

## Συντακτικά διαγράμματα (1)

- Έστω ότι έχουμε μια γραμματική σε EBNF
- Ο κάθε κανόνας παραγωγής μετατρέπεται σε μια σειρά από κουτιά
  - Ορθογώνια για τα μη τερματικά σύμβολα
  - Οβάλ για τα τερματικά σύμβολα
  - Τα παραπάνω ενώνονται με βέλη
  - (Πιθανώς κάποια βέλη να παρακάμπτουν κάποια από τα κουτιά.)

```
<if-stmt> ::= if <expr> then <stmt> [ else <stmt> ]
```

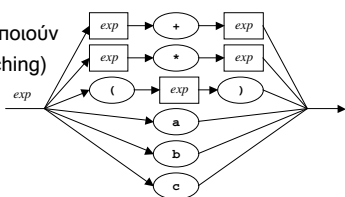


Σύνταξη και Συντακτική Ανάλυση

24

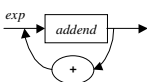
### Συντακτικά διαγράμματα (2)

- Πολλαπλοί κανόνες παραγωγής χρησιμοποιούν διακλαδώσεις (branching)



- Η επανάληψη υποδηλώνεται με χρήση βρόγχων

`<exp> ::= <addend> {+ <addend>}`



### Παράδειγμα διαφορετικού συμβολισμού EBNF

*WhileStatement.*

`while ( Expression ) Statement`

*DoStatement.*

`do Statement while ( Expression );`

*ForStatement.*

`for ( ForInitopt ; Expressionopt ; ForUpdateopt ) Statement`

από το βιβλίο *The Java™ Language Specification*, James Gosling et al.

### Από τη Σύνταξη προς τη Σημασιολογία

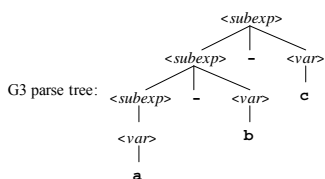
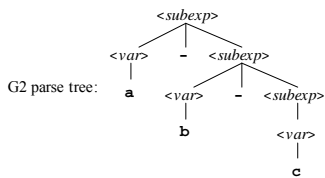
### Τρεις “Ισοδύναμες” Γραμματικές

G1: `<subexp> ::= a | b | c | <subexp> - <subexp>`

G2: `<subexp> ::= <var> - <subexp> | <var>`  
`<var> ::= a | b | c`

G3: `<subexp> ::= <subexp> - <var> | <var>`  
`<var> ::= a | b | c`

- Και οι τρεις γραμματικές ορίζουν την ίδια γλώσσα: τη γλώσσα όλων των συμβολοσειρών που περιλαμβάνουν ένα ή περισσότερα a, b, ή c που διαχωρίζονται από ένα μείον. Αλλά...



### Γιατί είναι σημαντικά τα συντακτικά δένδρα;

- Θέλουμε η δομή του συντακτικού δένδρου να αντικατοπτρίζει τη σημασιολογία της συμβολοσειράς που αντιπροσωπεύει
- Αυτό κάνει το σχεδιασμό της γλώσσας πιο δύσκολο:
  - ενδιαφερόμαστε για τη δομή του κάθε συντακτικού δένδρου
  - όχι μόνο για την συμβολοσειρά των φύλλων του

Τα συντακτικά δένδρα είναι το μέρος που η σύνταξη αρχίζει να συναντά τη σημασιολογία

### Τελεστές (operators)

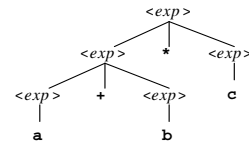
- Τελεστές χρησιμοποιούνται για λειτουργίες που γίνονται συχνά, π.χ. πρόσθεση, αφαίρεση, πολλαπλασιασμό, ...
- Ο όρος τελεστής αναφέρεται τόσο στη λεκτική μονάδα (π.χ. +, \*) όσο και στη λειτουργία αυτή καθ' αυτή
  - **Μοναδιαίοι** τελεστές δέχονται ένα ορίσμα: -1
  - **Διαδικοί** τελεστές δέχονται δύο ορίσματα: 1+2
  - **Τριαδικοί** τελεστές δέχονται τρία ορίσματα: a?b:c
- Στις περισσότερες γλώσσες
  - Οι διαδικοί τελεστές γράφονται σε infix μορφή: π.χ. 1+2
  - Οι μοναδιαίοι τελεστές γράφονται σε prefix (-2) ή σε postfix μορφή (1++)

### Προτεραιότητα (precedence) τελεστών

- Έστω η γραμματική G4:
  - Ένα συντακτικό δένδρο για τη συμβολοσειρά **a+b\*c** είναι το

```

<exp> ::= <exp> + <exp>
        | <exp> * <exp>
        | ( <exp> )
        | a | b | c
    
```



- Στο δένδρο αυτό η πρόσθεση γίνεται πριν από τον πολλαπλασιασμό, κάτι που δεν είναι σε αρμονία με τις συνηθεις προτεραιότητες των τελεστών + και \*

### Προτεραιότητα τελεστών στη γραμματική

- Για να έχουμε τη σωστή προτεραιότητα τελεστών, αλλάζουμε τη γραμματική με τέτοιο τρόπο ώστε ο τελεστής με την μεγαλύτερη προτεραιότητα να καταλήγει "πιο κάτω" στο συντακτικό δένδρο

```

G5: <exp> ::= <exp> + <exp> | <mulexp>
     <mulexp> ::= <mulexp> * <mulexp>
             | ( <exp> )
             | a | b | c
    
```

### Παραδείγματα προτεραιότητας τελεστών

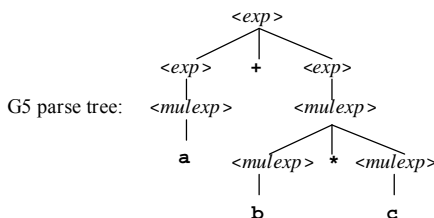
- C (15 επίπεδα προτεραιότητας—πάρα πολλά!)
- Pascal (5 επίπεδα—όχι αρκετά!)
- Smalltalk (1 επίπεδο για όλους τους δυαδικούς τελεστές)

`a = b < c ? * p + b * c : 1 << d ( )`

`a <= 0 or 100 <= a` **Συντακτικό λάθος!**

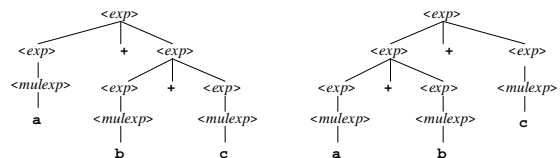
`a + b * c`

### Συντακτικό δένδρο με σωστή προτεραιότητα



Η γραμματική G5 παράγει μόνο αυτό το δένδρο για **a+b\*c**. Αναγνωρίζει την ίδια γλώσσα με τη G4, αλλά δεν παράγει πλέον δένδρα με λάθος προτεραιότητα τελεστών.

### Προσεταιριστικότητα (associativity) τελεστών



- Η γραμματική G5 παράγει τα παραπάνω δένδρα για **a+b+c**. Το πρώτο από αυτά δεν αντικατοπτρίζει τη συνηθη προσεταιριστικότητα του τελεστή +.

### Προσεταιριστικότητα τελεστών

- Έχουν χρησιμότητα όταν η σειρά της αποτίμησης δεν καθορίζεται από παρενθέσεις ή προτεραιότητα τελεστών
- Οι **αριστερά προσεταιριστικοί** τελεστές ομαδοποιούν από αριστερά προς τα δεξιά:  $a+b+c+d = ((a+b)+c)+d$
- Οι **δεξιά προσεταιριστικοί** τελεστές ομαδοποιούν από δεξιά προς τα αριστερά:  $a+b+c+d = a+(b+(c+d))$
- Στις περισσότερες γλώσσες, οι περισσότεροι τελεστές είναι αριστερά προσεταιριστικοί, αλλά υπάρχουν και εξαιρέσεις

Σύνταξη και Συντακτική Ανάλυση

37

### Παραδείγματα προσεταιριστικότητας

- C
  - $a<<b<<c$  — τελεστές είναι αριστερά προσεταιριστικοί
  - $a=b=0$  — δεξιά προσεταιριστικός (ανάθεση)
- ML
  - $3-2-1$  — τελεστές είναι αριστερά προσεταιριστικοί
  - $1::2::nil$  — δεξιά προσεταιριστικός (κατασκευή λίστας)
- Fortran
  - $a/b*c$  — τελεστές είναι αριστερά προσεταιριστικοί
  - $a**b**c$  — δεξιά προσεταιριστικός (ύψωση σε δύναμη)

Σύνταξη και Συντακτική Ανάλυση

38

### Προσεταιριστικότητα στη γραμματική

G5:  $\langle expr \rangle ::= \langle expr \rangle + \langle expr \rangle \mid \langle mulexp \rangle$   
 $\langle mulexp \rangle ::= \langle mulexp \rangle * \langle mulexp \rangle$   
 $\mid ( \langle expr \rangle )$   
 $\mid a \mid b \mid c$

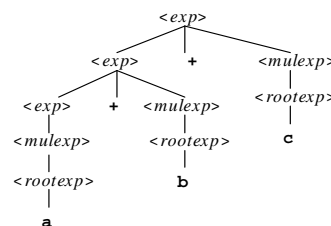
- Για να διορθώσουμε το πρόβλημα, τροποποιούμε τη γραμματική ώστε να κάνουμε δένδρα με τελεστές + να μεγαλώνουν προς τα αριστερά (παρόμοια για τον \*)

G6:  $\langle expr \rangle ::= \langle expr \rangle + \langle mulexp \rangle \mid \langle mulexp \rangle$   
 $\langle mulexp \rangle ::= \langle mulexp \rangle * \langle rootexp \rangle \mid \langle rootexp \rangle$   
 $\langle rootexp \rangle ::= ( \langle expr \rangle )$   
 $\mid a \mid b \mid c$

Σύνταξη και Συντακτική Ανάλυση

39

### Δένδρο με σωστή προσεταιριστικότητα



Η γραμματική G6 παράγει μόνο αυτό το δένδρο για  $a+b+c$ . Παράγει την ίδια γλώσσα με τη G5, αλλά δεν παράγει πλέον δένδρα με λάθος προσεταιριστικότητα τελεστών.

Σύνταξη και Συντακτική Ανάλυση

40

### Πρακτική άσκηση

- Ξεκινώντας από τη γραμματική

G6:  $\langle expr \rangle ::= \langle expr \rangle + \langle mulexp \rangle \mid \langle mulexp \rangle$   
 $\langle mulexp \rangle ::= \langle mulexp \rangle * \langle rootexp \rangle \mid \langle rootexp \rangle$   
 $\langle rootexp \rangle ::= ( \langle expr \rangle )$   
 $\mid a \mid b \mid c$

1. Προσθέστε έναν αριστερά προσεταιριστικό τελεστή &, με προτεραιότητα μικρότερη από όλους τους άλλους τελεστές
2. Στη συνέχεια προσθέστε έναν δεξιά προσεταιριστικό τελεστή \*\*, με προτεραιότητα μεγαλύτερη από όλους τους άλλους τελεστές

Σύνταξη και Συντακτική Ανάλυση

41

### Πολλαπλή ερμηνεία (Ambiguity)

- Η γραμματική G4 είναι **διφορούμενη (ambiguous)**: παράγει περισσότερα από ένα συντακτικά δένδρα για την ίδια συμβολοσειρά
- Όμως η επιδιόρθωση των προβλημάτων προτεραιότητας και προσεταιριστικότητας των τελεστών εξαφάνισε όλη την ασάφεια στη συγκεκριμένη γραμματική
- Αυτό είναι επιθυμητό: το συντακτικό δένδρο υποδηλώνει την ερμηνεία του προγράμματος και δε θέλουμε αυτή να είναι διφορούμενη
- Όμως υπάρχουν και άλλοι λόγοι που μια γραμματική είναι διφορούμενη, όχι μόνο λόγοι σχετικοί με τους τελεστές

Σύνταξη και Συντακτική Ανάλυση

42

## Το πρόβλημα του “ξεκρέμαστου else”

```

<stmt> ::= <if-stmt> | s1 | s2
<if-stmt> ::= if <expr> then <stmt> else <stmt>
           | if <expr> then <stmt>
<expr> ::= e1 | e2
  
```

- Η γραμματική αυτή είναι διφορούμενη ως προς το “ξεκρέμαστο else” (“dangling-else ambiguity”).

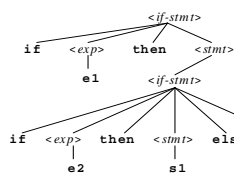
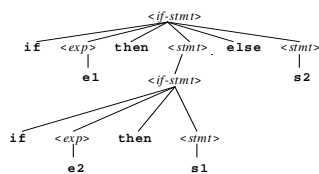
- Η παρακάτω εντολή

```
if e1 then if e2 then s1 else s2
```

έχει δύο συντακτικά δένδρα

Σύνταξη και Συντακτική Ανάλυση

43



Οι περισσότερες γλώσσες που έχουν αυτό το πρόβλημα επιλέγουν το κάτω συντακτικό δένδρο: το `else` πηγαίνει με το κοντινότερο αταίριαστο `then`

Σύνταξη και Συντακτική Ανάλυση

44

## Διόρθωση του προβλήματος (1)

```

<stmt> ::= <if-stmt> | s1 | s2
<if-stmt> ::= if <expr> then <stmt> else <stmt>
           | if <expr> then <full-stmt>
<expr> ::= e1 | e2
  
```

- Θέλουμε να επιβάλλουμε ότι εάν αυτό επεκταθεί σε ένα `if`, τότε το `if` πρέπει ήδη να έχει το δικό του `else`. Πρώτα, δημιουργούμε ένα νέο μη τερματικό `<full-stmt>` που παράγει όλες τις εντολές που παράγονται από το `<stmt>`, αλλά απαγορεύει τις `if` εντολές χωρίς `else`

```

<full-stmt> ::= <full-if> | s1 | s2
<full-if> ::= if <expr> then <full-stmt> else <full-stmt>
  
```

Σύνταξη και Συντακτική Ανάλυση

45

## Διόρθωση του προβλήματος (2)

- Μετά χρησιμοποιούμε το νέο μη τερματικό εδώ

```

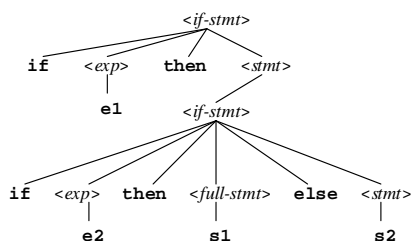
<stmt> ::= <if-stmt> | s1 | s2
<if-stmt> ::= if <expr> then <full-stmt> else <stmt>
           | if <expr> then <stmt>
<expr> ::= e1 | e2
  
```

- Το αποτέλεσμα είναι ότι η παραπάνω γραμματική μπορεί να ταιριάξει ένα `else` με ένα `if` μόνο όταν όλα τα κοντινά `if` έχουν ήδη κάποιο `else` ως ταίρι τους.

Σύνταξη και Συντακτική Ανάλυση

46

## Τώρα παράγουμε μόνο το συντακτικό δένδρο



Σύνταξη και Συντακτική Ανάλυση

47

## Ξεκρέμαστα else και αναγνωσιμότητα

- Διορθώσαμε τη γραμματική, αλλά...
- Το πρόβλημα στη γραμματική αντικατοπτρίζει κάποιο πρόβλημα στη γλώσσα, την οποία δεν αλλάξαμε
- Μια ακολουθία από `if-then-else` δεν είναι εύκολα αναγνώσιμη, ειδικά εάν κάποια από τα `else` λείπουν

```

int a=0;
if (0==0)
    if (0==1) a=1;
else a=2;
  
```

Ποια είναι η τιμή του `a` μετά την εκτέλεση του προγράμματος;

Σύνταξη και Συντακτική Ανάλυση

48



## Καλύτερα στυλ προγραμματισμού

```
int a=0;
if (0==0)
  if (0==1) a=1;
  else a=2;
```

Καλύτερο: σωστή στοίχιση

```
int a=0;
if (0==0) {
  if (0==1) a=1;
  else a=2;
}
```

Ακόμα καλύτερο: η χρήση μπλοκ δείχνει καθαρά τη δομή του κώδικα

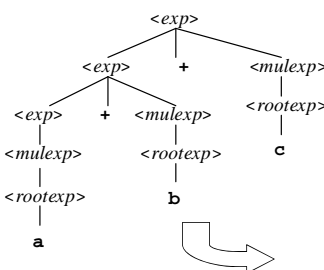
## Γλώσσες χωρίς ξεκρέμαστα else

- Μερικές γλώσσες ορίζουν τα `if-then-else` με τρόπο που επιβάλλει τη σαφήνεια στη χρήση τους
- Η Algol δεν επιτρέπει μέσα στο `then` να αρχίζει άμεσα κάποιο άλλο `if`
  - (αλλά μπορεί να αρχίζει ένα μπλοκ με ένα άλλο `if`)
- Η Algol 68 επιβάλλει σε κάθε `if` να τερματίζεται με ένα `fi` (υπάρχουν επίσης `do-od` και `case-esac`)
- Η Ada επιβάλλει σε κάθε `if` να τερματίζεται με ένα `end if`

## Γραμματική για ολόκληρη τη γλώσσα

- Κάθε ρεαλιστική γλώσσα περιέχει πολλά μη τερματικά σύμβολα
- Ειδικά εάν από τη γραμματική έχει εξαλειφθεί η ασάφεια
- Τα επιπλέον μη τερματικά καθορίζουν τον τρόπο με τον οποίο προκύπτει ένα μοναδικό συντακτικό δένδρο
- Όταν κατασκευαστεί το συντακτικό δένδρο, τα επιπλέον μη τερματικά δεν έχουν κάποια χρησιμότητα πλέον
- Οι υλοποιήσεις των γλωσσών συνήθως αποθηκεύουν μια συνεπτυγμένη μορφή του συντακτικού δένδρου, η οποία ονομάζεται **αφηρημένο συντακτικό δένδρο**

### Συγκεκριμένο συντακτικό δένδρο



### Αφηρημένο συντακτικό δένδρο

## Συμπερασματικά

- Για τον ορισμό του συντακτικού (και όχι μόνο!) των γλωσσών προγραμματισμού χρησιμοποιούμε γραμματικές
- Οι γραμματικές ορίζουν
  - το ποια είναι τα επιτρεπτά προγράμματα μιας γλώσσας, αλλά και
  - το συντακτικό δένδρο για αυτά τα προγράμματα
  - το δένδρο με τη σειρά του ορίζει τη σειρά εκτέλεσης των εντολών
  - και κατά συνέπεια συνεισφέρει στον ορισμό της σημασιολογίας
- Υπάρχει ισχυρή σύνδεση μεταξύ θεωρίας και πράξης
  - Δύο γραμματικές, δύο μέρη του μεταγλωττιστή (compiler)
  - **Σημείωση:** Υπάρχουν προγράμματα, γεννήτριες συντακτικών αναλυτών (parser generators), που μπορούν να δημιουργήσουν αυτόματα τον κώδικα του λεκτικού και του συντακτικού αναλυτή από τη γραμματική μιας γλώσσας