Μεταγλωττιστές 2024-25

Προσθήκη λογικών τελεστών
και τελεστών σύγκρισης

Ο στόχος

- Προσθήκη λογικών τελεστών
  - and, or, not
- Και τελεστών σύγκρισης
  - ==, !=, >, >=, <, <=
- Για χρήση σε συνθήκες εντολών if και while

Οι νέοι τελεστές

Τελεστής (υψηλότερη προτεραιότητα πρώτα)	**Σημείωση**
* /	διόρθωση για αριστερή προσεταιριστικότητα
+ -	διόρθωση για αριστερή προσεταιριστικότητα
== != > >= < <=	
not	
and	“short-circuiting”
or	“short-circuiting”

Αναπαράσταση λογικών (boolean) τιμών

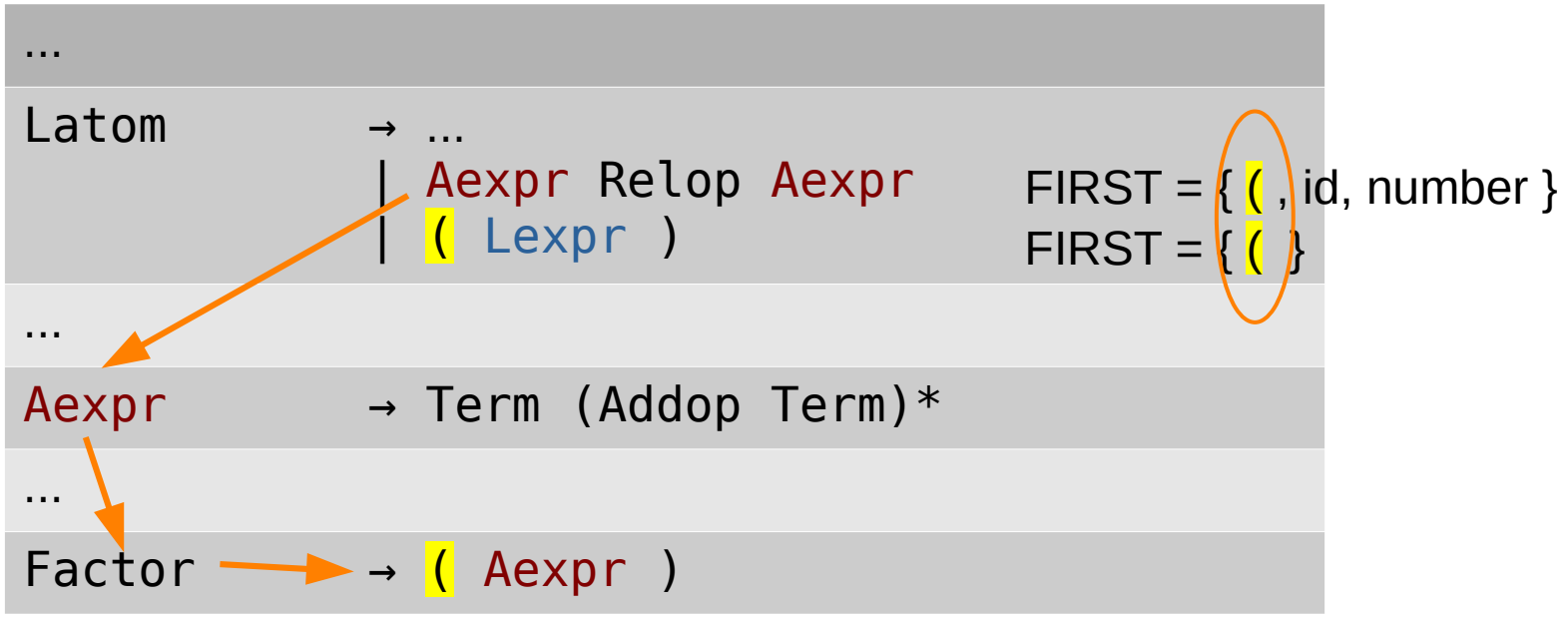
- Το αποτέλεσμα μιας λογικής πράξης είναι boolean (true/false)
  - Τελεστές and, or, not
- Το ίδιο ισχύει για το αποτέλεσμα μιας σύγκρισης
  - Τελεστές ==, !=, >, >=, <, <=
- Σχεδιαστική απόφαση: πώς θα αναπαρασταθούν οι λογικές τιμές στη γλώσσα/διερμηνευτή;
  - Αυτή τη στιγμή υπάρχουν μόνο floats

Ξεχωριστές αριθμητικές και λογικές εκφράσεις

- Ακριβέστερη αλλά συνθετότερη γραμματική
  - Δεν είναι LL(1)
  - Δεν αναλύεται με προσθήκη custom κώδικα στην LL(1) μέθοδο
  - Παράδειγμα: έστω Lexpr οι λογικές εκφράσεις και Aexpr οι αριθμητικές

Lexpr	→ ...
…	
Latom	→ Aexpr Relop Aexpr | lparen Lexpr rparen
Relop	→ eq | ne | gt | ge | lt | le
Aexpr	→ Term (Addop Term)*
…	
Factor	→ (Aexpr) | id | number

Σύγκρουση FIRST sets



- Πρακτικά: όταν είμαστε στο Latom και το επόμενο token είναι το (τί διαλέγουμε;
  - Ανοίγει παρένθεση λογικής ή αριθμητικής έκφρασης;
  - Αν αποφασίσουμε λάθος δεν μπορούμε να αλλάξουμε επιλογή…
- Πότε θα ξέρουμε ποιο είναι το σωστό;

```
((((....(((5 + …
```

Μπορούμε με 1 lookahead token (ή γενικότερα με k tokens) να ξέρουμε αν η παρένθεση ανήκει σε αριθμητική ή λογική έκφραση;

Ενιαίος τύπος εκφράσεων

- Στη γραμματική ένα και μοναδικό Expr παντού
- Αριθμητική αναπαράσταση του true και false
  - ίσο με 0.0 → false,
  - διάφορο του 0.0 → true
    - Όπως και σε πολλές «πραγματικές» γλώσσες
- Δυναμική χρήση κατά την εκτέλεση
  - Ανάλογα με τη θέση που βρίσκεται η έκφραση
    - Ως μέρος αριθμητικής πράξης
    - Ως μέρος λογικής συνθήκης

Η γραμματική τώρα

```
Stmt_list   → Stmt Stmt_list | ε
Stmt        → id = Expr | print Expr
Expr        → Term (Addop Term)*
~~Term_tail → Addop Term Term_tail | ε~~
Term        → Factor (Multop Factor)*
~~Factor_tail → Multop Factor Factor_tail | ε~~
Factor      → (Expr) | id | number
Addop       → + | -
Multop      → * | /
```

- Για τους νέους τελεστές θα πρέπει να προστεθούν τόσοι κανόνες όσα τα νέα επίπεδα προτεραιότητας

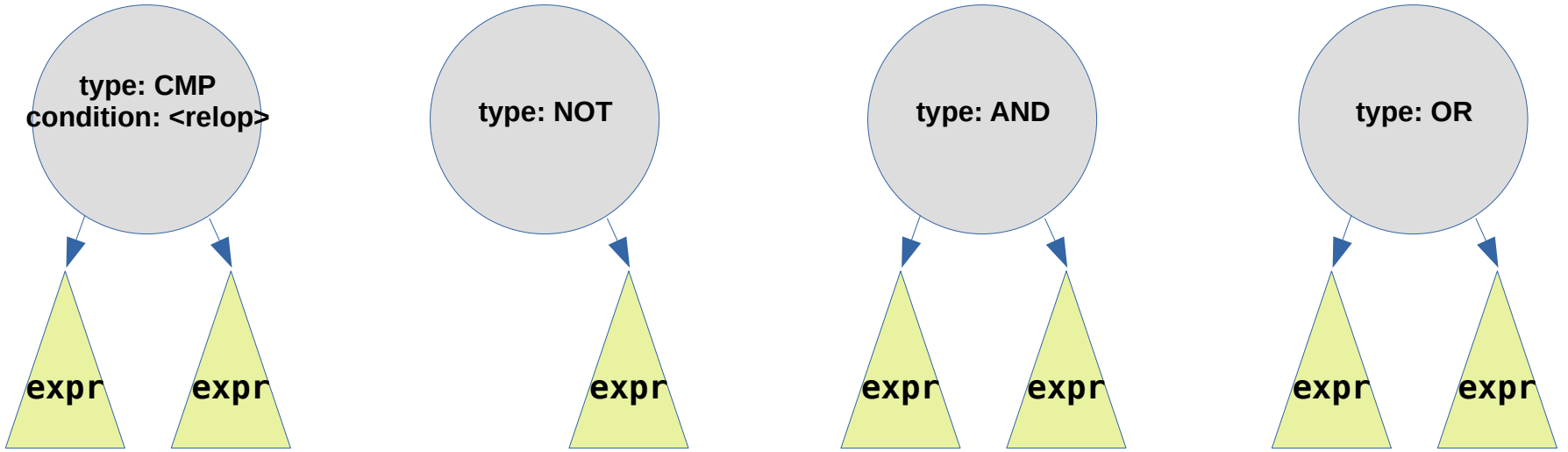
Η νέα γραμματική

```
Stmt_list    → Stmt Stmt_list | ε
Stmt         → id = Lexpr | print Lexpr
Lexpr        → Lterm Lterm_tail
Lterm_tail   → or Lexpr | ε          # or Lterm Lterm_tail | ε
Lterm        → Lfactor Lfactor_tail
Lfactor_tail → and Lterm | ε     # and Lfactor Lfactor_tail | ε
Lfactor      → not Lfactor | Expr Rest
Rest         → Relop Expr | ε
Expr         → Term (Addop Term)*
~~Term_tail  → Addop Term Term_tail | ε~~
Term         → Factor (Multop Factor)*
~~Factor_tail → Multop Factor Factor_tail | ε~~
Factor       → ( Lexpr ) | id | number
Addop        → + | -
Multop       → * | /
Relop        → == | != | > | >= | < | <=
```

FIRST & FOLLOW sets

nonterminal	first set	follow set
Stmt_list	id print	#
Stmt	id print	
Lexpr	not (id number	
Lterm_tail	or	) id print #
Lterm	not (id number	
Lfactor_tail	and	) or id print #
Lfactor	not (id number	
Rest	== != > >= < <=	) and or id print #
Expr	(id number	
Term	(id number	
Factor	(id number	
Multop	* /	
Addop	+ -	
Relop	== != > >= < <=	

Κατασκευή AST: οι νέοι κόμβοι



“Short-circuiting”

- Σε όλες τις γλώσσες προγραμματισμού

  `X and Y and Z and ..`

  - Εάν το X είναι false δεν υπολογίζονται τα Y, Z …
    - Μόνο εάν το X είναι true, τότε προχωράμε στον υπολογισμό του Y, κ.λ.π. προς τα δεξιά
    - Το ίδιο για το or: αν το X είναι true δεν υπολογίζουμε τα Y, Z...
  - Προκύπτει από τις ιδιότητες των and/or
- Προσοχή: δεν είναι (μόνο) θέμα βελτιστοποίησης
  - Ο υπολογισμός των Y, Z .. μπορεί να προκαλεί σφάλματα όταν το X είναι false
  - Για την ακρίβεια σε πολλές γλώσσες προγραμματισμού γίνεται ακριβώς αυτό
    - Ο προγραμματιστής προστατεύει από την εκτέλεση ενός άκυρου Y μέσω της κατάλληλης συνθήκης στο X

Η λειτουργία του short-circuitng

```
if type==’AND’ {
  a = evaluate(left_child)
  if a is false
    return false
  else
    return evaluate(right_child)
}
```