Μεταγλωττιστές 2021-22

Εισαγωγή στη Συντακτική Ανάλυση

Συντακτική Ανάλυση (Parsing)

- Μετά τη λεκτική ανάλυση
  - Είσοδος: τα tokens του λεκτικού αναλυτή
- Σκοπός: η ανάλυση της δομής της εισόδου
  - Αν είναι σωστά δομημένη σύμφωνα με τους κανόνες σύνταξης της εκάστοτε γλώσσας προγραμματισμού
- Έξοδος;
  - Θεωρητικά, τίποτα αν η είσοδος είναι συντακτικά έγκυρη
  - Ή σφάλμα σύνταξης στην αντίθετη περίπτωση

Η συντακτική ανάλυση στην πράξη

- Κατά τη διαδικασία ανάλυσης
  - Κρατείται η πληροφορία της δομής της εισόδου
    - Π.χ. τι υπάρχει μέσα σε ένα if, for κλπ
- Συνδυάζεται ταυτόχρονα (ή διαδοχικά) με τη σημασιολογική ανάλυση
- Σύλληψη της «έννοιας» του προγράμματος εισόδου
    - Αυτό που θέλει να κάνει ο προγραμματιστής

Σημασιολογική ανάλυση - μια γρήγορη ματιά

- Context-sensitive analysis, τι δεν μπορεί να κάνει η συντακτική ανάλυση από μόνη της, μερικά παραδείγματα
  - Τι είναι αποθηκευμένο στο x;
    - Τύποι μεταβλητών
  - Παράμετροι συναρτήσεων
    - Καλείται μια συνάρτηση με τις παραμέτρους που πρέπει;
  - Διάρκεια ζωής μεταβλητών

Έξοδος συντακτικής+σημασιολογικής ανάλυσης

- «Ενδιάμεση αναπαράσταση» (intermediate representation)
  - Το «νόημα» του προγράμματος
  - Σε συστηματική αναπαράσταση
  - Έτοιμο για επεξεργασία από τα επόμενα στάδια του μεταγλωττιστή
    - Στάδια βελτιστοποίησης
    - Παραγωγή εκτελέσιμου κώδικα ή διερμηνεία (απ’ ευθείας εκτέλεση)

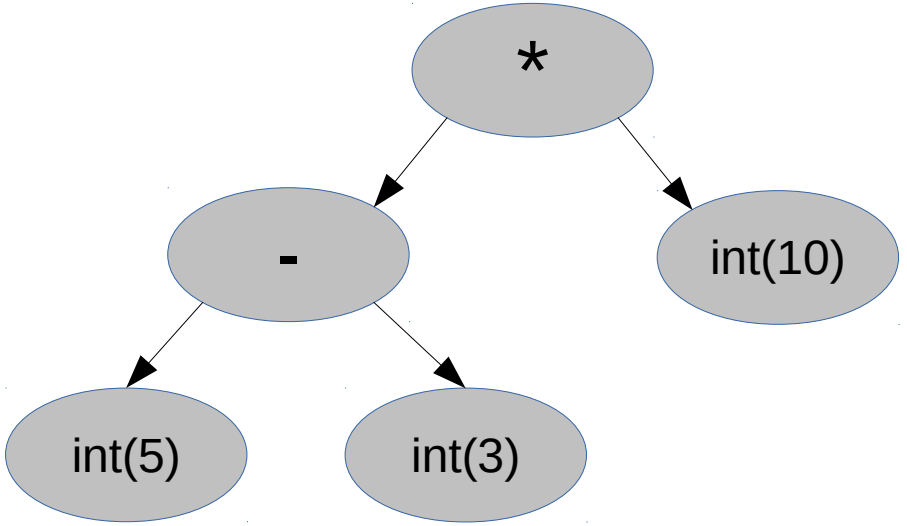
Παράδειγμα ενδιάμεσης αναπαράστασης (AST)

- Έστω η είσοδος: `(5-3)*10`
- Η λεκτική ανάλυση δίνει τα tokens:

  `lparen int(5) minus int(3) rparen mult int(10)`
- Abstract Syntax Tree (AST)
  - Το «νόημα» εδώ: η σωστή αριθμητική ερμηνεία

Παράδειγμα ενδιάμεσης αναπαράστασης (AST)

- Είσοδος: (5-3)*10



- Πώς θα ήταν στην περίπτωση του 5-3*10;
- Και στην περίπτωση του 5-3-2;

Γιατί δεν αρκεί η λεκτική ανάλυση

- Γιατί να μην κάνουμε τα πάντα με κανονικές εκφράσεις;
  - Δεν μπορούμε: οι κανονικές εκφράσεις δεν αναγνωρίζουν nested δομές (γενικά δεν υποστηρίζουν αναδρομή)
  - Π.χ. με τις κανονικές εκφράσεις δεν μπορούμε να αναγνωρίσουμε αν η είσοδος έχει ισορροπημένες παρενθέσεις
  - Ένα αυτόματο μπορεί να μας πει σε ποιο state είμαστε, όχι όμως πόσες φορές έχουμε περάσει από το state αυτό...

Μέθοδοι συντακτικής ανάλυσης

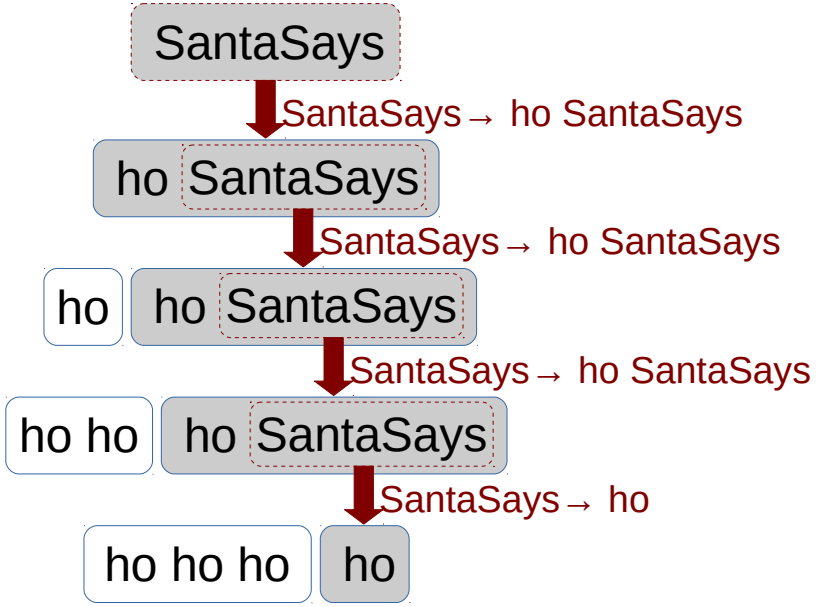
- Βασισμένες σε γραμματικές
  - Με τη μεγαλύτερη μαθηματική τυπική θεμελίωση (1960s-1970s)
- Άλλες μέθοδοι
  - Λιγότερο «τυπικές», περισσότερο πρακτικές
  - Δεν υστερούν όμως σε απόδοση από τις μεθόδους με γραμματικές

Γραμματικές

- Ως «συνταγές» για την παραγωγή προτάσεων

  `SantaSays → ho SantaSays | ho`

- Παραγωγή (production) προτάσεων (sentences)



Παραγωγή προτάσεων

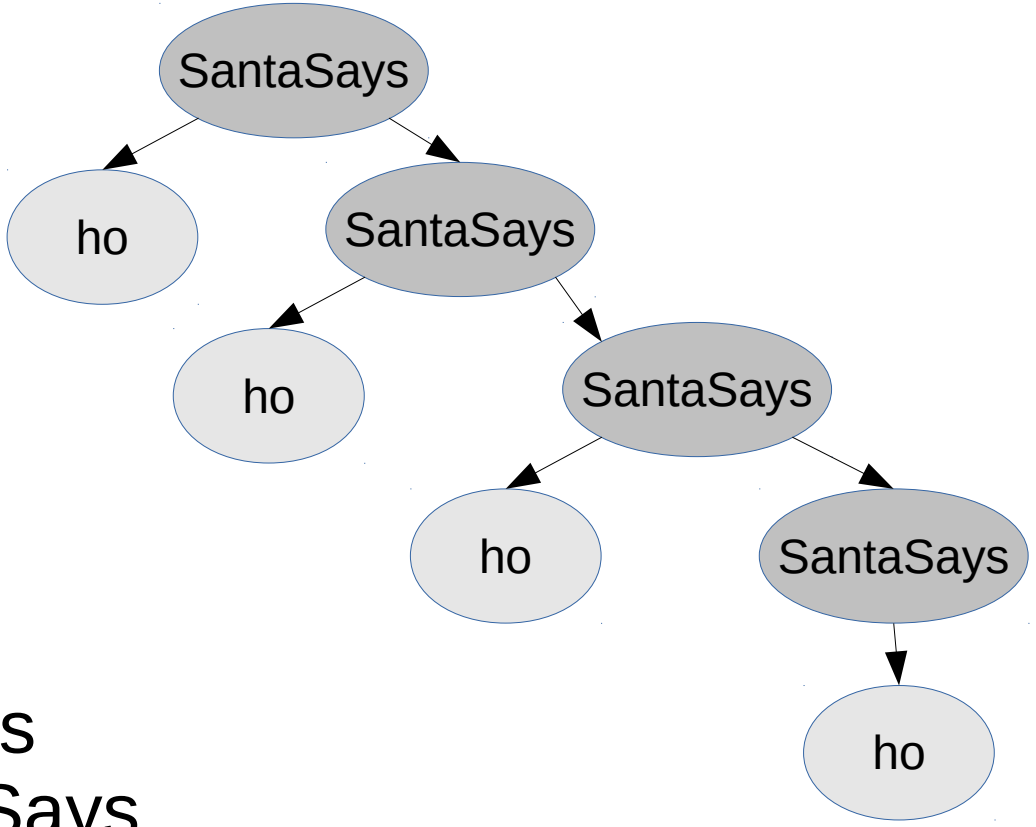
- Μη τερματικά σύμβολα: ενδιάμεσες μορφές που πρέπει να αντικατασταθούν (π.χ. SantaSays)
- Τερματικά σύμβολα: οι τελικές «λέξεις» της γλώσσας που παράγει η γραμματική (π.χ. ho)
  - Τα tokens του λεκτικού αναλυτή
- Σύμφωνα με τους κανόνες της γραμματικής, αντικαθιστούμε τα μη τερματικά σε βήματα παραγωγής (productions)
  - Το αποτέλεσμα είναι σειρές από τερματικά και μη τερματικά σύμβολα (sentential forms)
  - Στο τελευταίο βήμα μένουν μόνο σειρές από τερματικά σύμβολα, οι τελικές προτάσεις (sentences)
- Για γραμματική G, η γλώσσα της γραμματικής L(G) είναι το σύνολο των τελικών προτάσεων που παράγει η G

Ο φορμαλισμός της γραμματικής

- Μια γραμματική είναι η τετράδα (T, NT, S, P)
  - T το σύνολο των τερματικών συμβόλων
  - NT το σύνολο των μη τερματικών
    - $T \cap NT = \varnothing$
  - S το αρχικό σύμβολο
  - P σύνολο κανόνων για παραγωγές
    - $(T \cup NT)+ \rightarrow (T \cup NT)*$

Δένδρο (ή γράφος) παραγωγής

- Προσοχή: μην το συγχέετε με ένα AST!



SantaSays
ho SantaSays
ho ho SantaSays
ho ho ho SantaSays
ho ho ho ho

Τεχνικές συντακτικής ανάλυσης

- Top-down
  - Ξεκινώ από το αρχικό σύμβολο (ρίζα δένδρου) και προσπαθώ να διαλέξω κανόνες παραγωγής που θα με οδηγήσουν στα tokens εισόδου (φύλλα δένδρου)
- Bottom-up
  - Ξεκινώ από τα tokens εισόδου (φύλλα δένδρου) και με βάση τους κανόνες παραγωγής προσπαθώ να φτάσω στο αρχικό σύμβολο (ρίζα δένδρου)

Γραμματικές κατά Chomsky

- Ιεραρχία εκφραστικής δύναμης (τι μπορούν να περιγράψουν)
- Type 0: χωρίς περιορισμούς μετασχηματισμοί

  `comma Name End → and Name`
- Type 1: ένα μόνο σύμβολο στο αριστερό μέρος μετασχηματίζεται

  `Name comma Name End → Name and Name End`
  - Ποια η διαφορά από το απλό `comma → and`;
  - Context-sensitive μετασχηματισμός

Γραμματικές κατά Chomsky

- Δυστυχώς, για τα Type 0 και 1 η συντακτική ανάλυση (αν γίνεται) απαιτεί εκθετικό χρόνο
  - μη πρακτικό
- Type 2: Στο αριστερό μέρος μόνο ένα μη-τερματικό

  `List → Name comma List`
  - Η αντικατάσταση ενός μη τερματικού δεν εξαρτάται από τα γειτονικά του (context-free)
  - Υποστηρίζεται αναδρομή και nesting
    - Μπορούμε να θυμόμαστε τι πρέπει να γίνει «μετά»

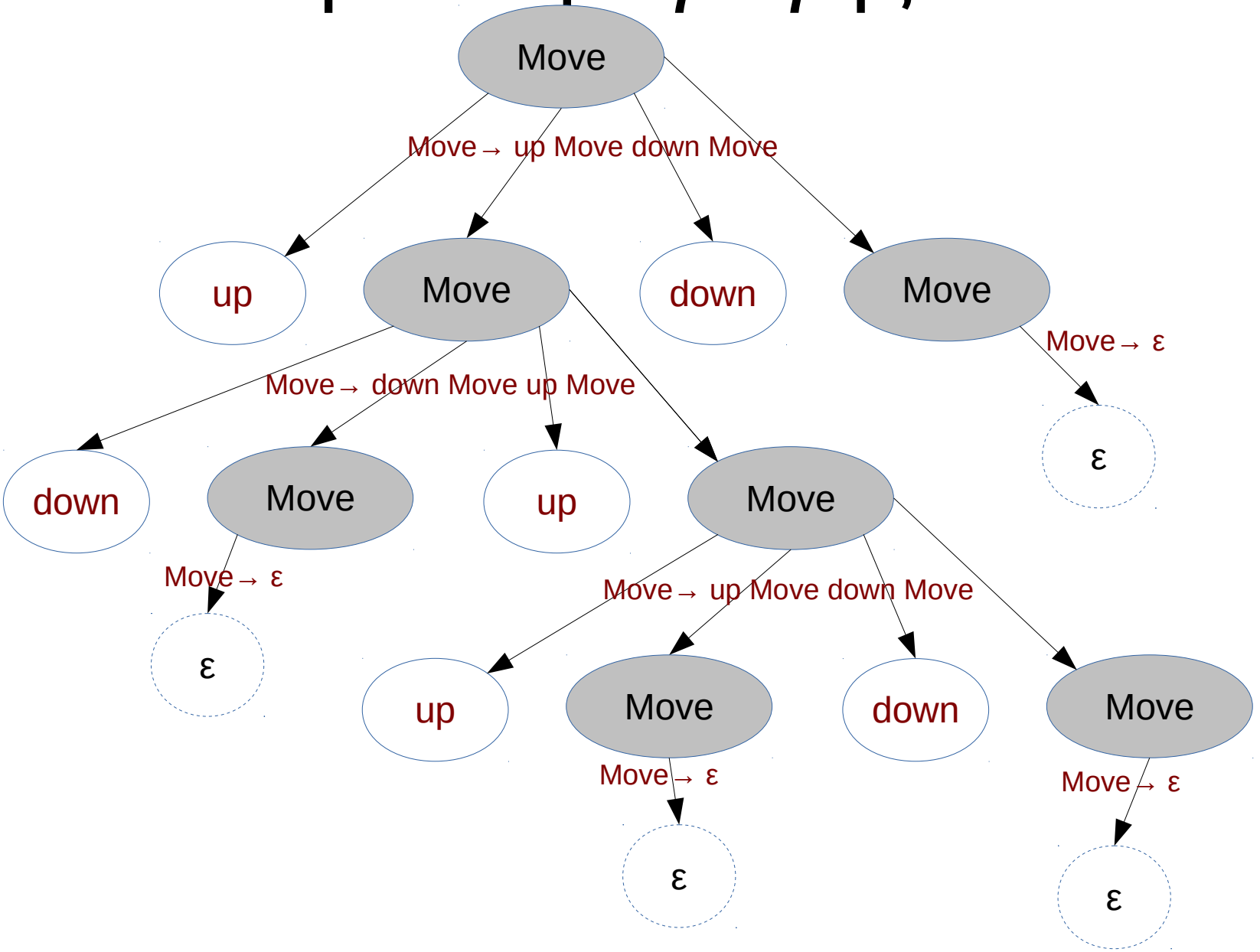
Παράδειγμα

- Η γραμματική «του ασανσέρ»

```
Move → up Move down Move
     | down Move up Move
     | ε
```

- Σχεδιάστε το δέντρο παραγωγής για την είσοδο: up down up up down down
- Πού θα βρίσκεται το ασανσέρ σε σχέση με την αρχική του θέση για κάθε πρόταση που παράγει η γραμματική;
  - γιατί;

Δένδρο παραγωγής



Γραμματικές κατά Chomsky

- Γενικά, η συντακτική ανάλυση για Type 2 γραμματικές (context-free grammars, CFG) απαιτεί χρόνο $O(n^3)$
  - μη πρακτικό
  - Υπάρχουν όμως υποκατηγορίες γραμματικών που απαιτούν χρόνο O(n) = πρακτική λύση!
  - Καλύπτουν σχεδόν το σύνολο των απαιτήσεων ανάλυσης των γλωσσών προγραμματισμού
- Type 3: στο δεξί μέρος μόνο ένα μη τερματικό (και συνήθως αυτό στο τέλος)
  - Αδυναμία αναδρομής και nesting
  - Αντιστοιχία με τις γνωστές μας κανονικές εκφράσεις

Η σημασία της πολυπλοκότητας

- Έστω 10.000 tokens στο πρόγραμμα εισόδου
- O(n) σημαίνει 1 «ενέργεια» ανά token
  - μια ανάθεση, μια προσπέλαση κλπ
- Αν η «ενέργεια» απαιτεί 10ns, με πολυπλοκότητα O(n) απαιτείται χρόνος $10*10^{-9} * 10^{4} = 10^{-4}$ sec ή 100μsec.
- Με πολυπλοκότητα $O(n^3)$, δηλ. 10^{4*3} «ενέργειες», πόσος θα είναι ο χρόνος συντακτικής ανάλυσης;

Ποιο μη τερματικό αντικαθιστούμε

- Σε κανόνες με πολλά μη τερματικά σύμβολα, στο δεξί μέρος των κανόνων, ποιο διαλέγουμε για την επόμενη αντικατάσταση;
  - Αριστερότερη ακολουθία παραγωγών (leftmost derivation): αντικαθιστούμε σε κάθε βήμα το αριστερότερο μη τερματικό
  - Δεξιότερη ακολουθία παραγωγών (rightmost derivation): αντικαθιστούμε σε κάθε βήμα το δεξιότερο μη τερματικό

Παράδειγμα

- Έστω η γραμματική

  `Expr →Expr Op num | num`

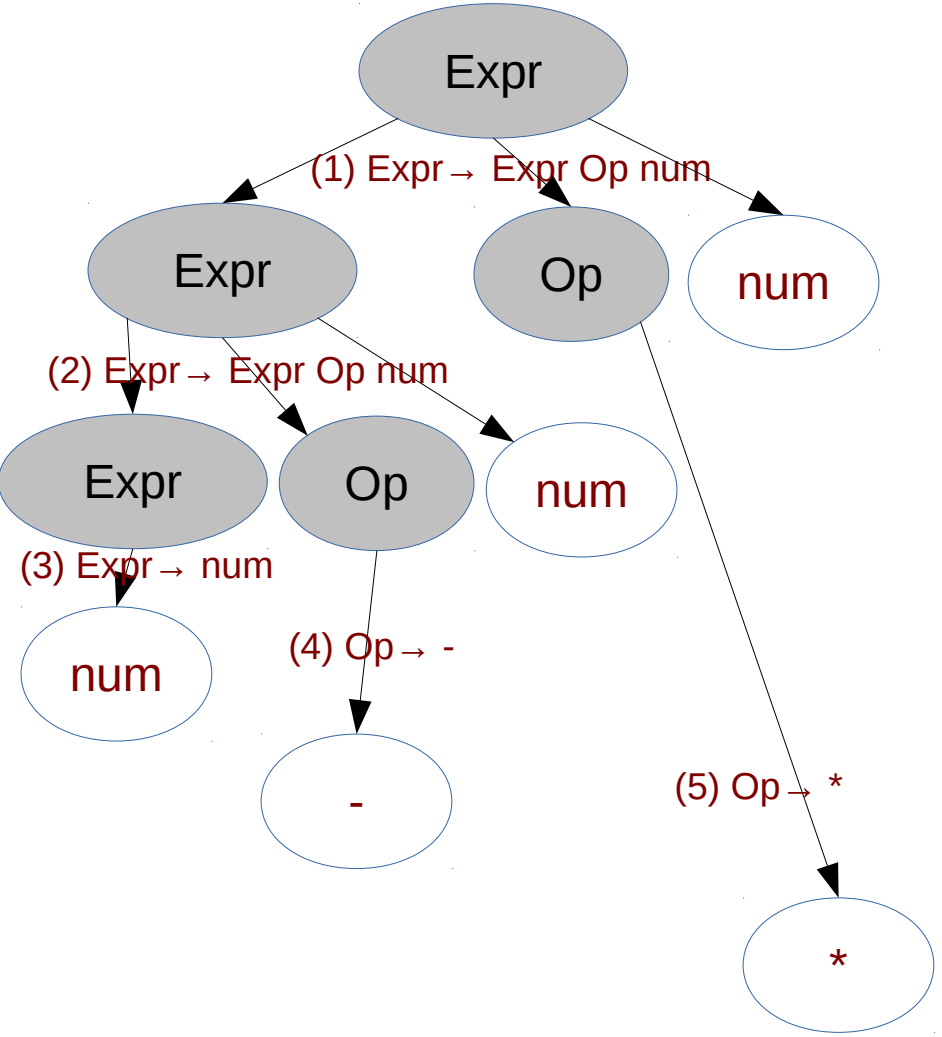
  `Op → + | - | * | /`

- Και έστω η είσοδος (tokens)

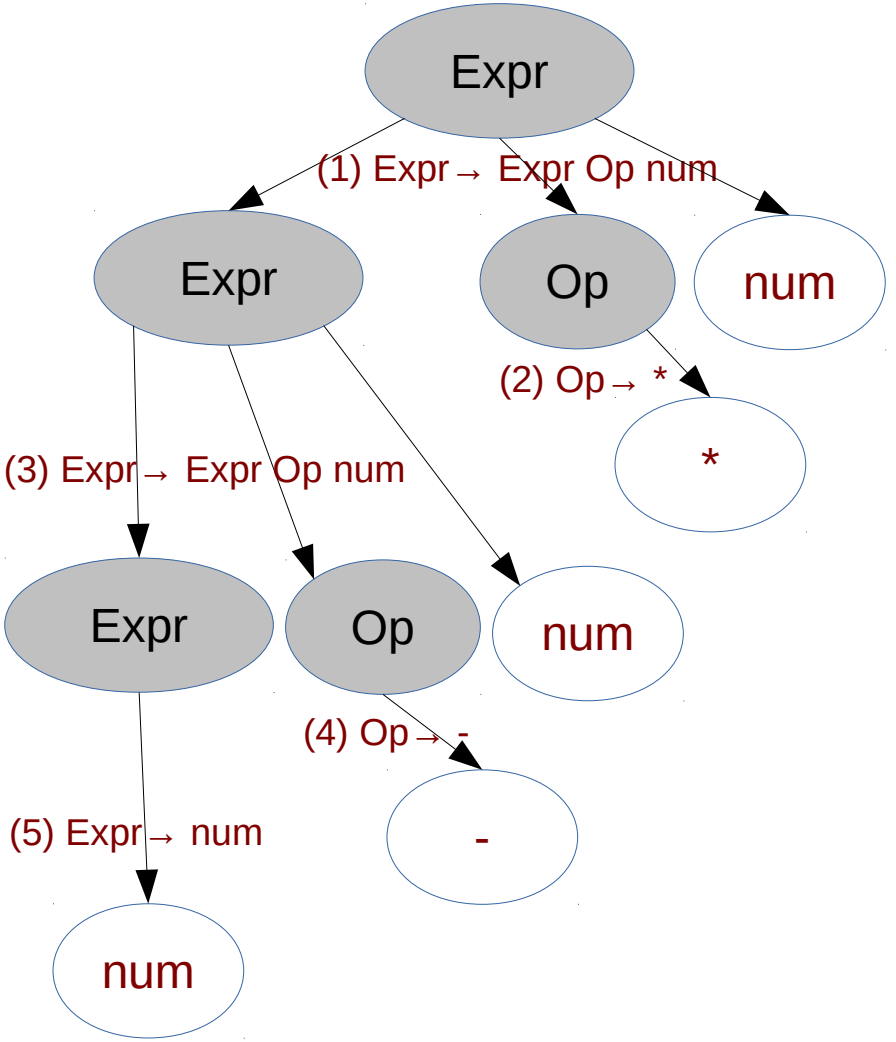
  `num - num * num`

- Βρείτε αριστερότερη και δεξιότερη ακολουθία παραγωγών και τα αντίστοιχα δένδρα
  - Τι παρατηρείτε;
  - Αν χρησιμοποιούσαμε τα δένδρα κατευθείαν ως AST, θα είχαμε σωστή αριθμητική ερμηνεία της εισόδου;

Δενδρα παραγωγής



Αριστερότερη ακολουθία παραγωγής



Δεξιότερη ακολουθία παραγωγής

Μια καλύτερη γραμματική

```
Expr → Expr + Term | Expr - Term | Term
Term →Term * num | Term / num | num
```

- Ποιο το δένδρο παραγωγής για την προηγούμενη είσοδο tokens;
- Αν θέλαμε να προσθέσουμε τη δυνατότητα να υπάρχουν εκφράσεις μέσα σε παρενθέσεις δηλ. (Expr), πώς θα μπορούσαμε να το κάνουμε;
  - Υπόδειξη: σκεφτείτε τα επίπεδα προτεραιότητας, κάθε επίπεδο εισάγει νέο κανόνα στη γραμματική

Αμφισημία (ambiguity)

- Όταν υπάρχει πάνω από μία αριστερότερη (ή δεξιότερη) ακολουθία παραγωγής για την ίδια είσοδο
  - Αυτό σημαίνει ότι ο μεταγλωττιστής ενδεχομένως δεν θα καταλάβει αυτό που θέλουμε να πούμε!

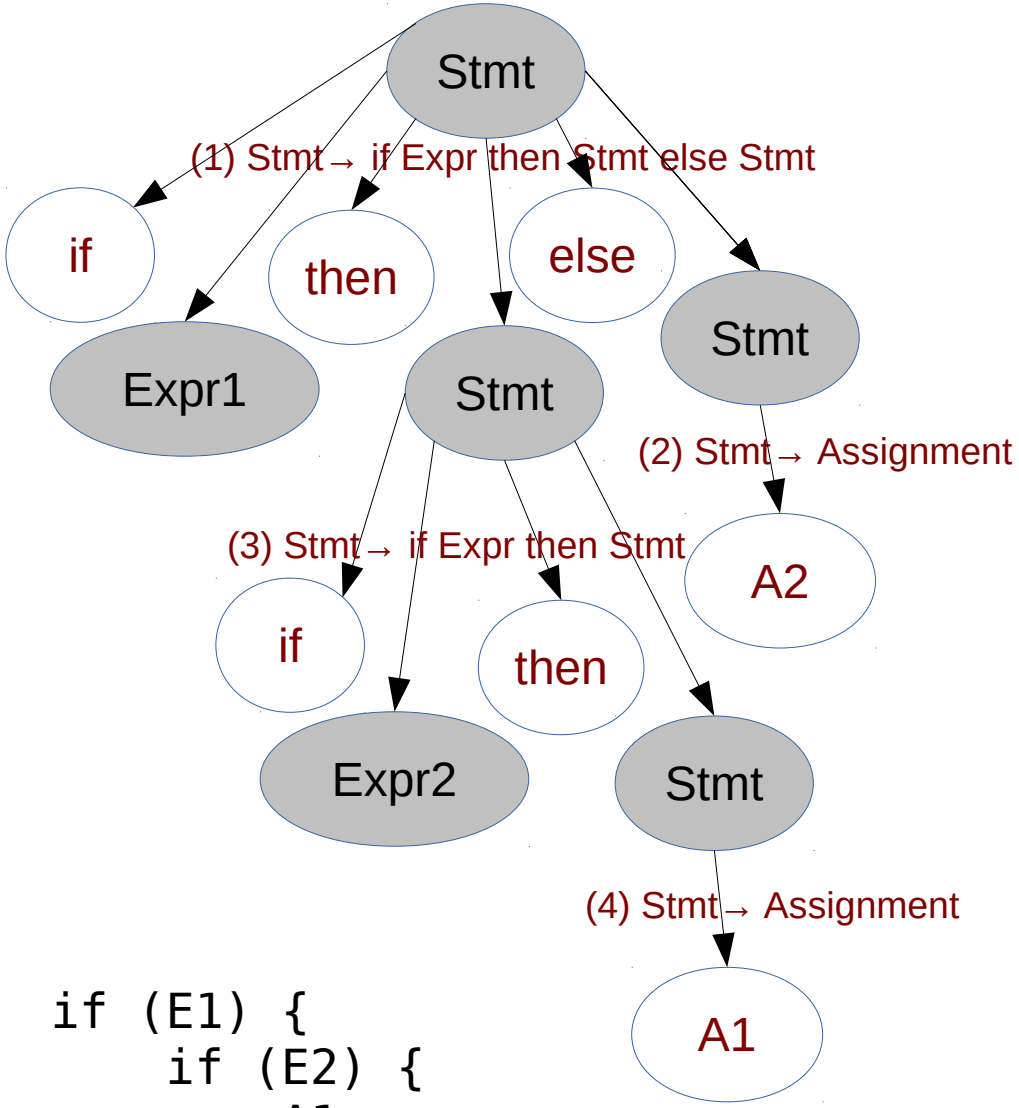
Παράδειγμα

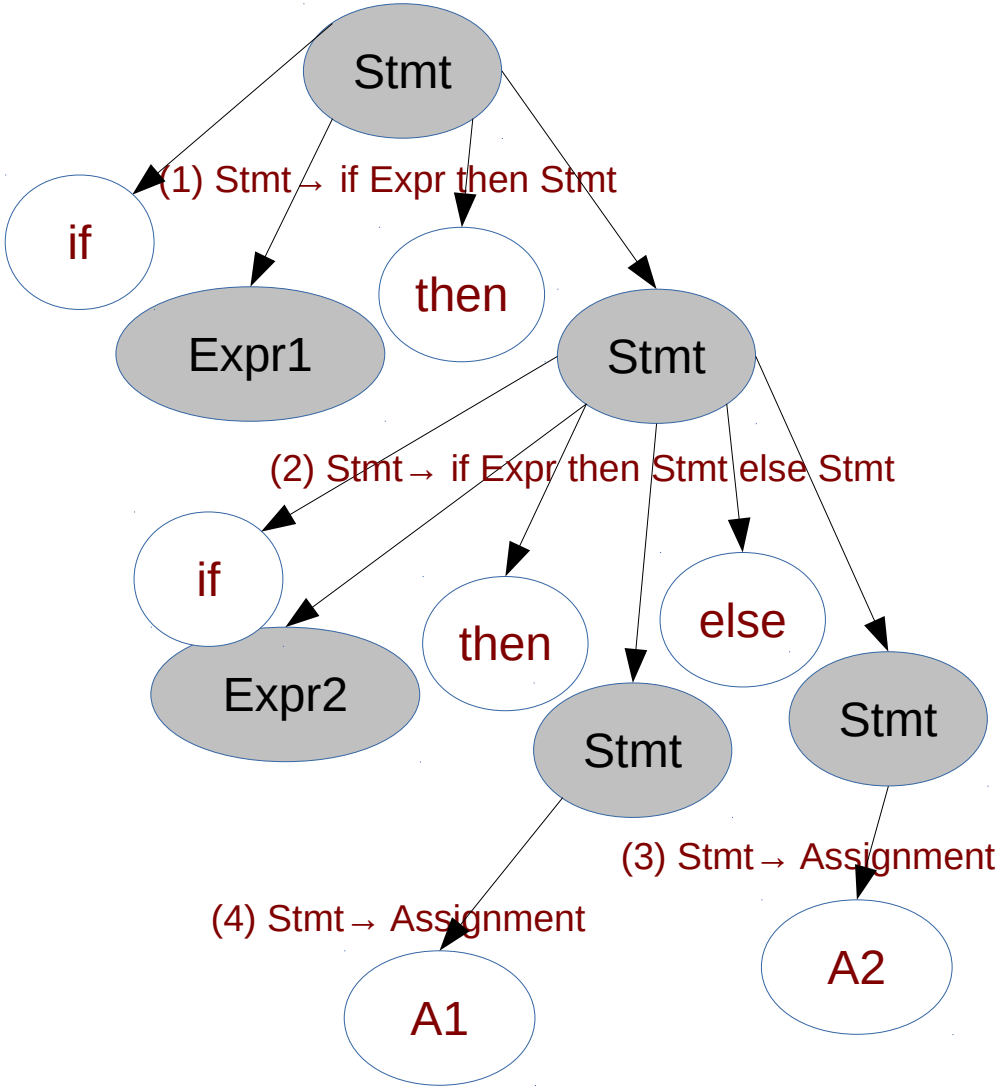
```
Stmt → if Expr then Stmt else Stmt
     | if Expr then Stmt
     | Assignment | ..
```

- Είσοδος tokens

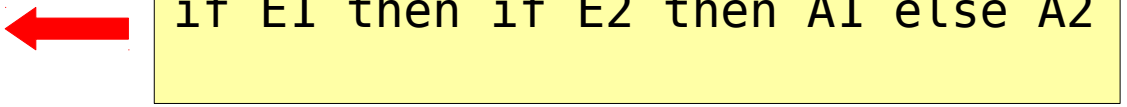
```
if E1 then if E2 then A1 else A2
```

- Βρείτε δύο διαφορετικές δεξιότερες παραγωγές
  - Τι σημαίνει η κάθε μία, αν το γράφατε με τη σημερινή C;






```
if (E1) {
    if (E2) {
        A1;
    }
}
else {
  A2;
}
```




```
if (E1) {
    if (E2) {
        A1;
    }
    else {
        A2;
    }
}
```

```
…
if reactor_on then
    if control_rods_in_place then
        normal_operation
    else
        emergency_shutdown
…
```

- Σημ: τα κενά στοίχισης δεν παίζουν ρόλο στη σύνταξη...