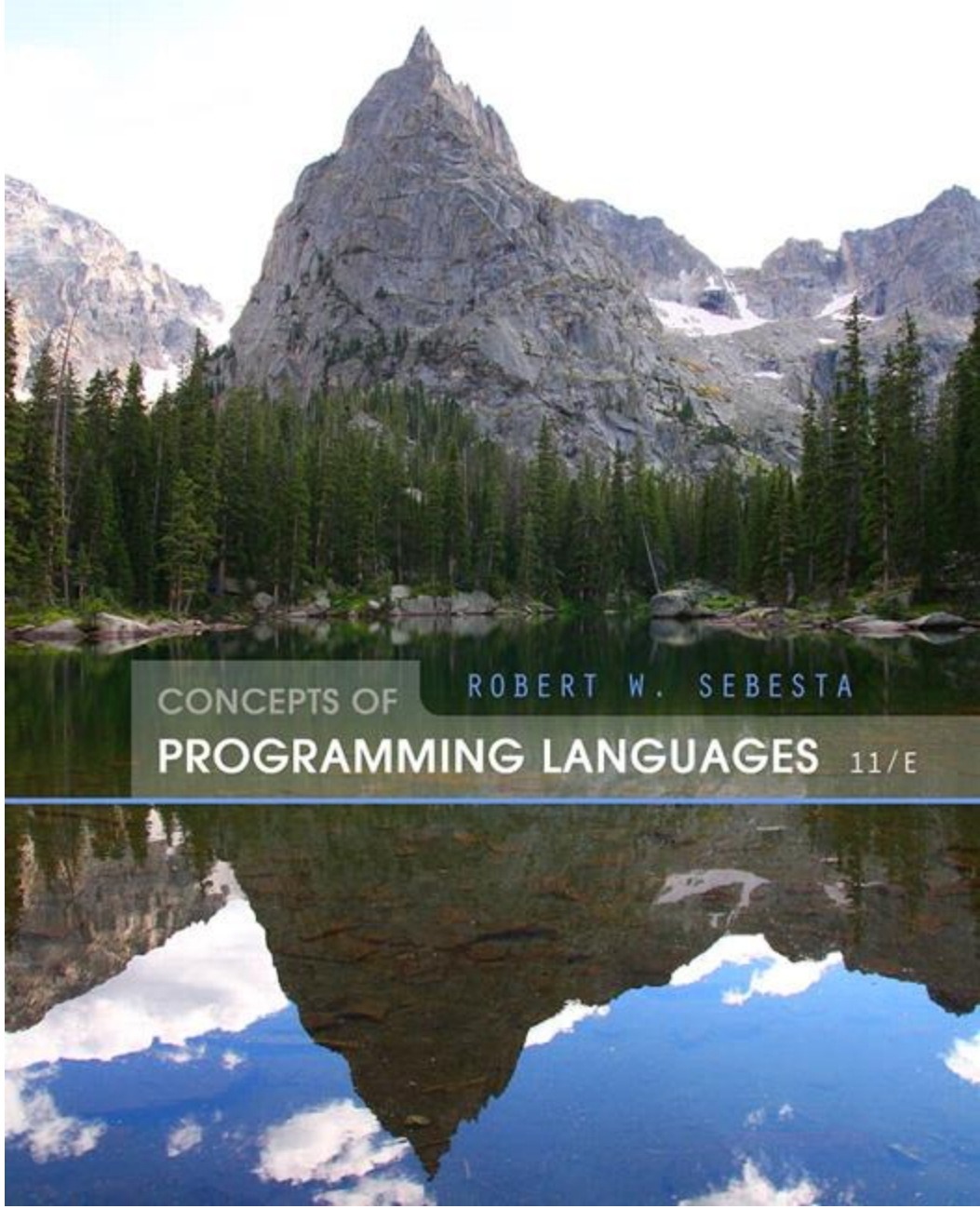
Κεφάλαιο 3

Περιγραφή Συντακτικού και Σημασιολογίας

Γκόγκος Χρήστος
Τμήμα Πληροφορικής και Τηλεπικοινωνιών (Άρτα)
Πανεπιστήμιο Ιωαννίνων



ISBN 0-321-49362-1

Κεφάλαιο 3 – Θεματικές Ενότητες

- Εισαγωγή
- Το Γενικό Πρόβλημα της Περιγραφής Συντακτικού
- Τυπικές Μέθοδοι Περιγραφής Συντακτικού
- Γραμματικές χαρακτηριστικών
- Περιγραφή της νοήματος των προγραμμάτων: Δυναμική σημασιολογία

Εισαγωγή

- **Συντακτικό**: η μορφή (δομή) των εκφράσεων, εντολών, και προγραμματιστικών μονάδων
- **Σημασιολογία**: η σημασία των εκφράσεων, εντολών, και προγραμματιστικών μονάδων
- Το συντακτικό και η σημασιολογία συνιστούν τον ορισμό μιας γλώσσας
  - Χρήστες του ορισμού μιας γλώσσας
    - Σχεδιαστές γλωσσών
    - Κατασκευαστές γλωσσών προγραμματισμού
    - Προγραμματιστές (οι χρήστες της γλώσσας)

Το Γενικό Πρόβλημα της Περιγραφής του Συντακτικού: Ορολογία

- Μια *πρόταση* είναι ένα σύνολο χαρακτήρων κάποιου αλφαβήτου
- Μια *γλώσσα* είναι ένα σύνολο προτάσεων
- *Ένα λέξημα (lexeme)* είναι η συντακτική μονάδα χαμηλότερου επιπέδου της γλώσσας (π.χ., `*`, `sum`, `begin`)
- Μια *λεκτική μονάδα (token)* είναι μια κατηγορία λεξημάτων (π.χ., αναγνωριστικά)

Τυπικός Ορισμός Γλωσσών

- **Μηχανισμοί αναγνώρισης (Recognizers)**
  - Διαβάζουν από την είσοδο συμβολοσειρές του αλφαβήτου της γλώσσας και αποφασίζουν το εάν ανήκουν ή όχι στη γλώσσα
  - Παράδειγμα: Ο συντακτικός αναλυτής ενός μεταγλωττιστή
    - Η συντακτική ανάλυση αναλύεται στο Κεφάλαιο 4
- **Μηχανισμοί παραγωγής (Generators)**
  - Παράγουν έγκυρες προτάσεις της γλώσσας
  - Μπορεί να εξεταστεί αν η σύνταξη μιας συγκεκριμένης πρότασης είναι ορθή πραγματοποιώντας σύγκριση με τη δομή του μηχανισμού παραγωγής

BNF και Γραμματικές Χωρίς Συμφραζόμενα (CFGs)

- Γραμματικές Χωρίς Συμφραζόμενα (CFGs = Context-Free Grammars)
  - Αναπτύχθηκαν από τον Noam Chomsky στα μέσα της δεκαετίας του 1950
  - Πρόκειται για generators, που είχαν ως στόχο την περιγραφή του συντακτικού των φυσικών γλωσσών
  - Ορίζουν μια κατηγορία γλωσσών που ονομάζονται γλώσσες χωρίς συμφραζόμενα
- Backus-Naur Form (1959)
  - Προτάθηκαν από τον John Backus για να περιγράψουν το συντακτικό της Algol 58
  - Η BNF είναι ισοδύναμη με τις CFGs

Βασικές έννοιες BNF

- Στη BNF, χρησιμοποιούνται αφαιρέσεις για να αναπαραστήσουν κατηγορίες συντακτικών δομών – οι αφαιρέσεις αυτές συμπεριφέρονται ως συντακτικές μεταβλητές (ονομάζονται επίσης *μη-τερματικά σύμβολα* ή *μη-τερματικά*)

- Τα *τερματικά* είναι λεξήματα ή λεκτικές μονάδες

- Ένας κανόνας έχει ένα αριστερό μέρος (LHS = Left-Hand Side), που είναι ένα μη-τερματικό, και ένα δεξιό μέρος (RHS = Right-Hand Side), που είναι μια συμβολοσειρά από τερματικά και/ή μη-τερματικά

Βασικές έννοιες BNF (συνέχεια)

- Τα μη-τερματικά συχνά εσωκλείονται σε αγκύλες της μορφής: <>
  - Παραδείγματα κανόνων BNF:

```
<ident_list> → identifier | identifier, <ident_list>
<if_stmt> → if <logic_expr> then <stmt>
```

- Γραμματική: ένα πεπερασμένο μη κενό σύνολο κανόνων
- Το *σύμβολο έναρξης (start symbol)* είναι ένα ειδικό στοιχείο από τα μη-τερματικά της γλώσσας

Κανόνες BNF

- Μια αφαίρεση (ή μη-τερματικό σύμβολο) μπορεί να έχει περισσότερα από ένα RHS

```
<stmt> → <single_stmt>
       | begin <stmt_list> end
```

Περιγραφή Λιστών

- Οι συντακτικές λίστες περιγράφονται χρησιμοποιώντας αναδρομή

```
<ident_list> → ident
             | ident, <ident_list>
```

- Μια παραγωγή (derivation) είναι η επαναλαμβανόμενη εφαρμογή κανόνων, που ξεκινά από το αρχικό σύμβολο και τερματίζει σε μια πρόταση που περιέχει μόνο τερματικά σύμβολα

Ένα παράδειγμα Γραμματικής

<program> → <stmts>

<stmts> → <stmt> | <stmt> ; <stmts>

<stmt> → <var> = <expr>

<var> → a | b | c | d

<expr> → <term> + <term> | <term> - <term>

<term> → <var> | const

Ένα παράδειγμα παραγωγής

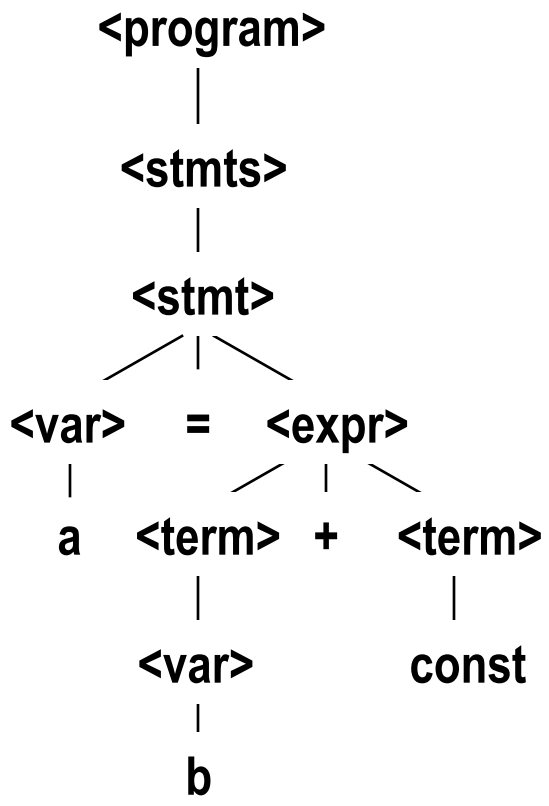
```
<program> => <stmts> => <stmt>
                     => <var> = <expr>
                     => a = <expr>
                     => a = <term> + <term>
                     => a = <var> + <term>
                     => a = b + <term>
                     => a = b + const
```

Παραγωγές

- Κάθε συμβολοσειρά σε μια παραγωγή βρίσκεται στη λεγόμενη *προτασιακή μορφή* (*sentential form)*
- Μια πρόταση *(sentence)* είναι μια προτασιακή μορφή που περιέχει μόνο τερματικά σύμβολα
- Μια αριστερότερη παραγωγή είναι μια παραγωγή στην οποία το αριστερότερο μη-τερματικό σε κάθε προτασιακή μορφή είναι αυτό και το οποίο επεκτείνεται
- Μια παραγωγή μπορεί να μην είναι ούτε αριστερότερη ούτε δεξιότερη, αλλά συνδυασμός αριστερών και δεξιών παραγωγών

Δένδρο Συντακτικής Ανάλυσης (Parse Tree)

- Πρόκειται για ιεραρχική αναπαράσταση μιας παραγωγής



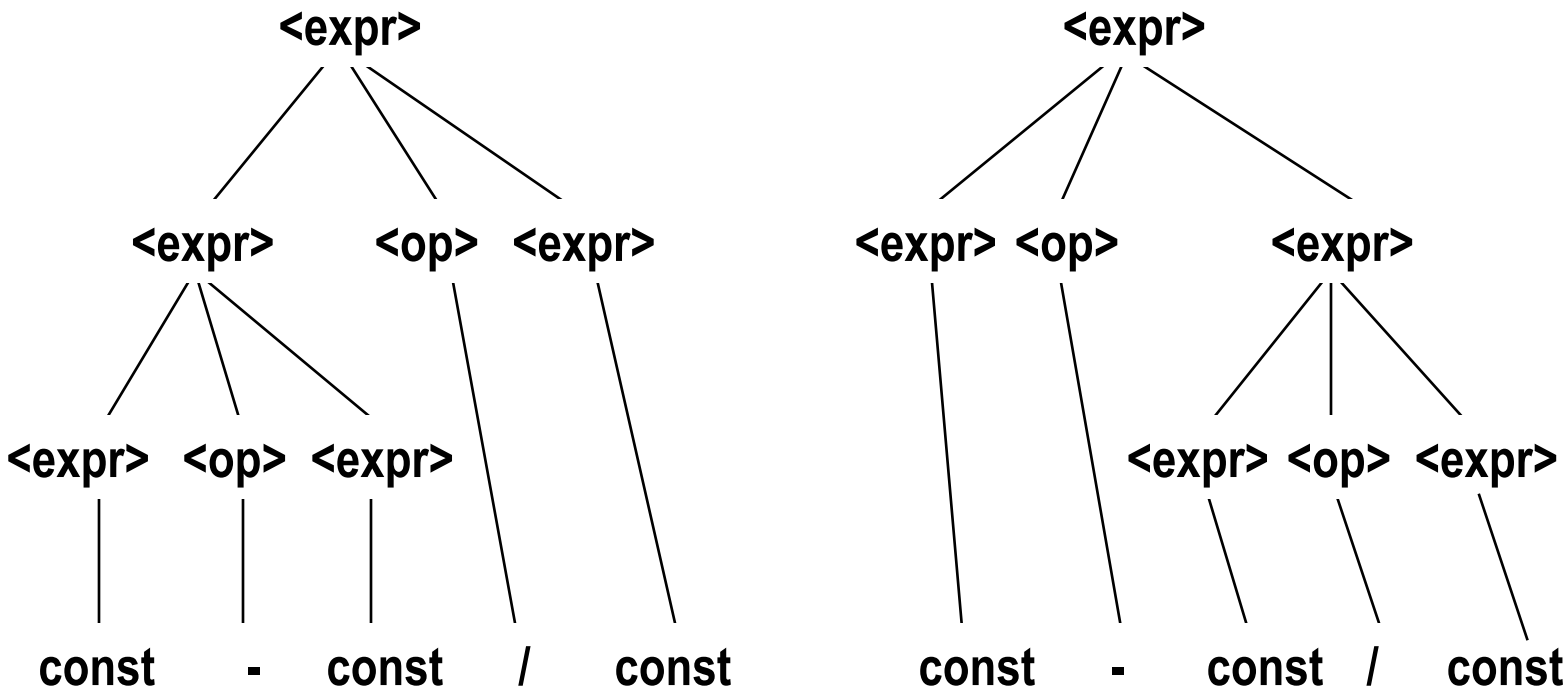
Ασάφεια (ambiguity) στις Γραμματικές

- Μια γραμματική είναι ασαφής (ambiguous) αν και μόνο αν παράγει μια προτασιακή μορφή με δύο ή περισσότερα διακριτά δένδρα συντακτικής ανάλυσης

Μια Ασαφής Γραμματική Εκφράσεων

<expr> → <expr> <op> <expr> | const

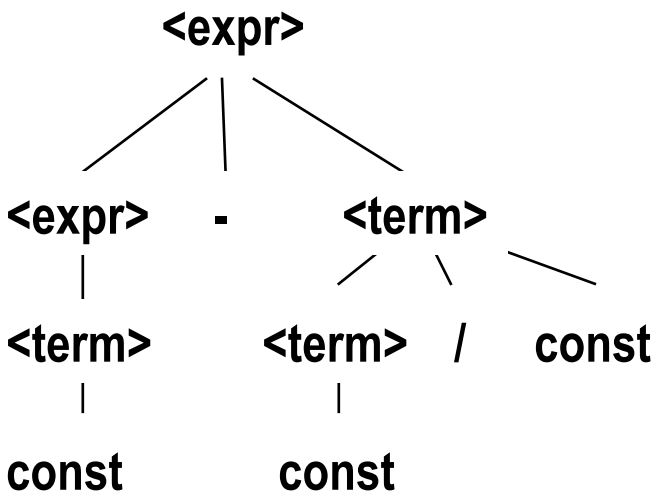
<op> → / | -



Μια **σαφής** (unambiguous) Γραμματική Εκφράσεων

- Αν χρησιμοποιούμε ένα δένδρο συντακτικής ανάλυσης για να υποδηλώσουμε τα επίπεδα προτεραιότητας των τελεστών, τότε δεν μπορεί να υπάρχει ασάφεια

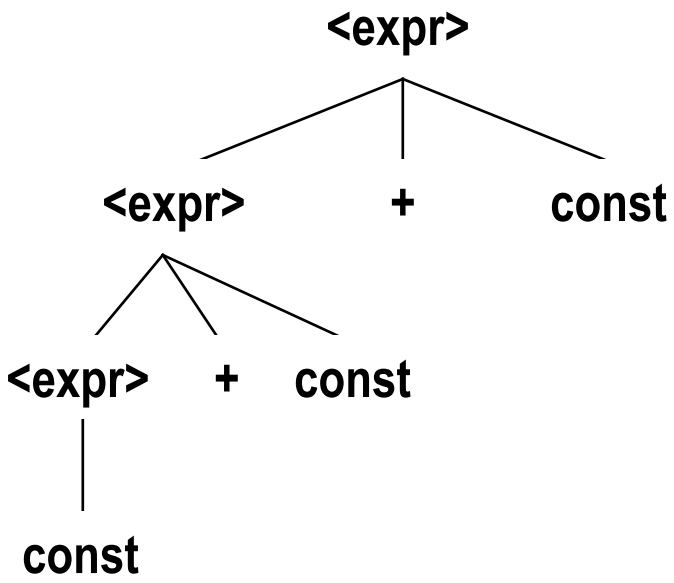
```
<expr> → <expr> - <term> | <term>
<term> → <term> / const | const
```



Προσεταιριστικότητα τελεστών

- Η προσεταιριστικότητα τελεστών μπορεί να καθορίζεται μέσω της γραμματικής

```
<expr> -> <expr> + <expr> |  const  (ambiguous)
<expr> -> <expr> + const  |  const  (unambiguous)
```



Ασαφής Γραμματική για το if-else

- Η γραμματική για την `if-then-else` της Java

```
<if_stmt> -> if (<logic_expr>) <stmt>
           | if (<logic_expr>) <stmt> else <stmt>
<stmt> -> <if_stmt>
```

Είναι ασαφής!

– Μια σαφής γραμματική για το `if-then-else`

```
<stmt> -> <matched> | <unmatched>
<matched> -> if (<logic_expr>) <stmt>
             | a non-if statement
<unmatched> -> if (<logic_expr>) <stmt>
               | if (<logic_expr>) <matched> else
                   <unmatched>
```

Εκτεταμένη BNF (Extended BNF)

- Προαιρετικά τμήματα τοποθετούνται σε αγκύλες [ ]

  `<proc_call> -> ident [(<expr_list>)]`

- Εναλλακτικά τμήματα των RHS τοποθετούνται σε παρενθέσεις και χωρίζονται με κάθετες γραμμές

  `<term> → <term> (+|-) const`

- Επαναλήψεις (0 ή περισσότερες) τοποθετούνται μέσα σε αγκύλες { }

  `<ident> → letter {letter|digit}`

BNF and EBNF

- BNF

```
<expr> → <expr> + <term>
       | <expr> - <term>
       | <term>
<term> → <term> * <factor>
       | <term> / <factor>
       | <factor>
```

- EBNF

```
<expr> → <term> {(+ | -) <term>}
<term> → <factor> {(* | /) <factor>}
```

Πρόσφατες παραλλαγές στην EBNF

- Εναλλακτικά RHS τοποθετούνται σε ξεχωριστές γραμμές
- Χρήση : αντί για `=>`
- Χρήση `opt` για προαιρετικά μέρη
- Χρήση `oneof` για επιλογές

Στατική Σημασιολογία

- Η στατική σημασιολογία δεν έχει σχέση με το νόημα του προγράμματος
- Οι γραμματικές χωρίς συμφραζόμενα (CFGs) δεν μπορούν να περιγράψουν όλους τους κανόνες συντακτικού των γλωσσών προγραμματισμού
- Υπάρχουν κατηγορίες κατασκευών που παρουσιάζουν πρόβλημα στην περιγραφή τους με CFG:
  - Είτε μπορούν να οριστούν σε CFGs, αλλά με δύσχρηστο τρόπο (π.χ. κατάλληλοι τύποι τελεστέων σε εκφράσεις)
  - Δεν είναι δυνατόν να οριστούν σε CFGs (π.χ. μεταβλητές που πρέπει να δηλωθούν πριν χρησιμοποιηθούν)

Γραμματικές χαρακτηριστικών

- Οι γραμματικές χαρακτηριστικών (AGs=Attribute Grammars) αποτελούν επεκτάσεις των CFGs στις οποίες προστίθεται σημασιολογική πληροφορία στους κόμβους του δένδρου συντακτικής ανάλυσης

- Προστιθέμενη αξία των AGs:
  - Δυνατότητα καθορισμού της στατικής σημασιολογίας
  - Σχεδίαση μεταγλωττιστών (έλεγχος στατικής σημασιολογίας)

Γραμματικές Χαρακτηριστικών: Ορισμός

- Ορισμός: Μια γραμματική χαρακτηριστικών είναι μια γραμματική χωρίς συμφραζόμενα G=(S, N, T, P) με τις ακόλουθες προσθήκες:
  - Για κάθε σύμβολο της γραμματικής *x* υπάρχει ένα σύνολο *A(x)* από τιμές χαρακτηριστικών
  - Κάθε κανόνας έχει ένα σύνολο από συναρτήσεις που ορίζουν συγκεκριμένα χαρακτηριστικά για τα μη-τερματικά του κανόνα
  - Κάθε κανόνας έχει ένα (πιθανά άδειο) σύνολο κατηγορημάτων τα οποία πρέπει να ελέγχονται για να διασφαλιστεί η συνέπεια των χαρακτηριστικών

Γραμματικές Χαρακτηριστικών: Ορισμός

- Έστω $X_0 \rightarrow X_1 \ldots X_n$ ένας κανόνας
- Συναρτήσεις της μορφής
  $S(X_0) = f(A(X_1), \ldots, A(X_n))$
  ορίζουν τα συντιθέμενα χαρακτηριστικά *(synthesized attributes)*
- Συναρτήσεις της μορφής
  $I(X_j) = f(A(X_0), \ldots, A(X_n))$, για $i <= j <= n$, ορίζουν τα κληρονομούμενα χαρακτηριστικά *(inherited attributes)*
- Αρχικά, λαμβάνονται τιμές των λεγόμενων εσωτερικών χαρακτηριστικών (*intrinsic attributes)* στα φύλλα του δένδρου συντακτικής ανάλυσης

Γραμματικές Χαρακτηριστικών: Παράδειγμα

- Συντακτικό

  <assign> -> <var> = <expr>

  <expr> -> <var> + <var> | <var>

  <var> A | B | C

- actual_type: συντιθέμενο χαρακτηριστικό για <var> και <expr>
- expected_type: κληρονομούμενο χαρακτηριστικό για <expr>

- Ζητείται η κατασκευή του επισημειωμένου δένδρου συντακτικής ανάλυσης για την πρόταση

  A = A + B

  με A να έχει τύπο real και B να έχει τύπο int

Γραμματικές Χαρακτηριστικών: παράδειγμα (συνέχεια)

- Κανόνας σύνταξης 1: `<assign> → <var> = <expr>`
  Σημασιολογικός κανόνας: `<expr>.expected_type = <var>.actual_type`

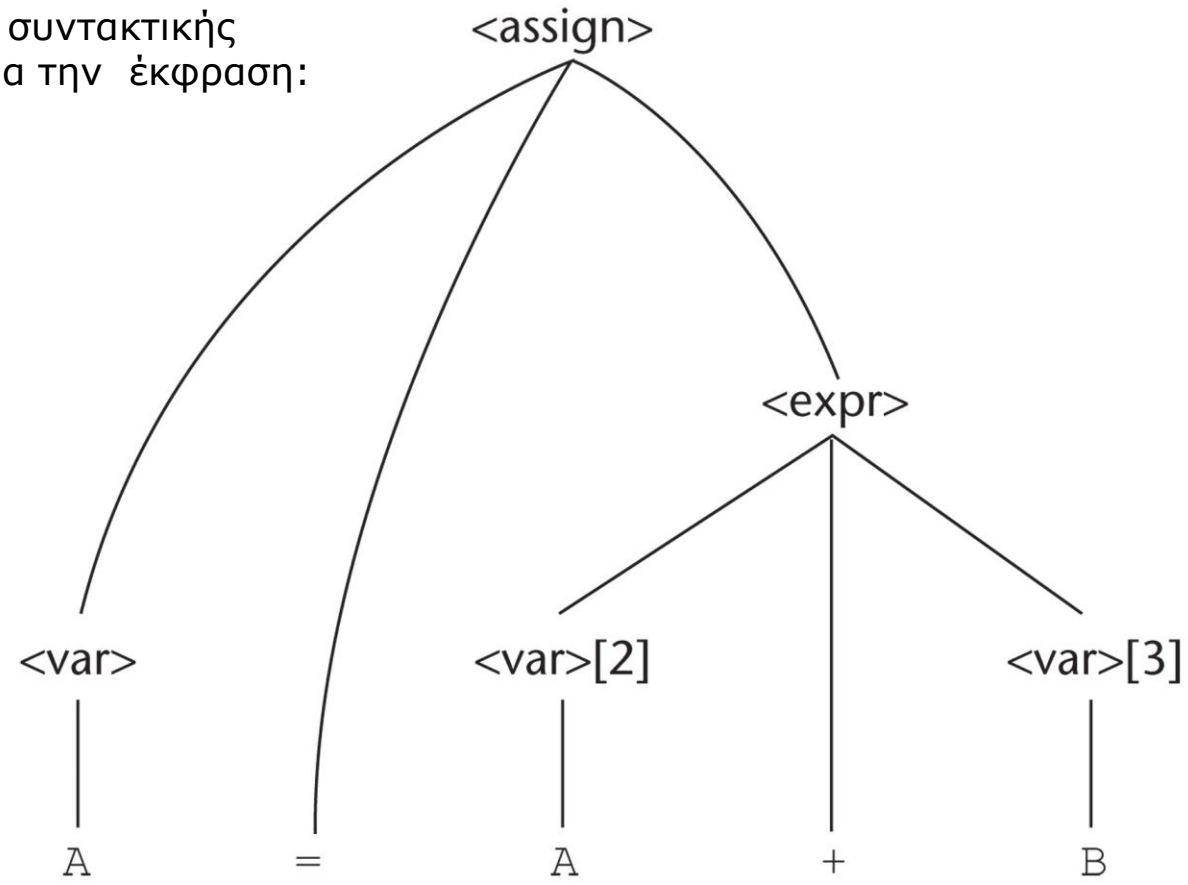
- Κανόνας σύνταξης 2: `<expr> → <var>[2] + <var>[3]`
  Σημασιολογικοί κανόνες: `<expr>.actual_type =`
  ```
  if <var>[2].actual_type == int_type and <var>[3].actual_type == int_type
  then int_type
  else real_type
  ```
  Κατηγόρημα: `<expr>.actual_type == <expr>.expected_type`

- Κανόνας σύνταξης 3: `<expr> → <var>`
  Σημασιολογικός κανόνας: `<expr>.actual_type = <var>.actual_type`
  Κατηγόρημα: `<expr>.actual_type == <expr>.expected_type`

- Κανόνας σύνταξης 4: `<var> → A  // ομοίως και για <var> → B, <var> → C`
  Σημασιολογικός κανόνας: `<var>.actual_type = lookup(A.value)`

Γραμματικές Χαρακτηριστικών: παράδειγμα (συνέχεια)

Ένα δένδρο συντακτικής ανάλυσης για την έκφραση:
A = A + B

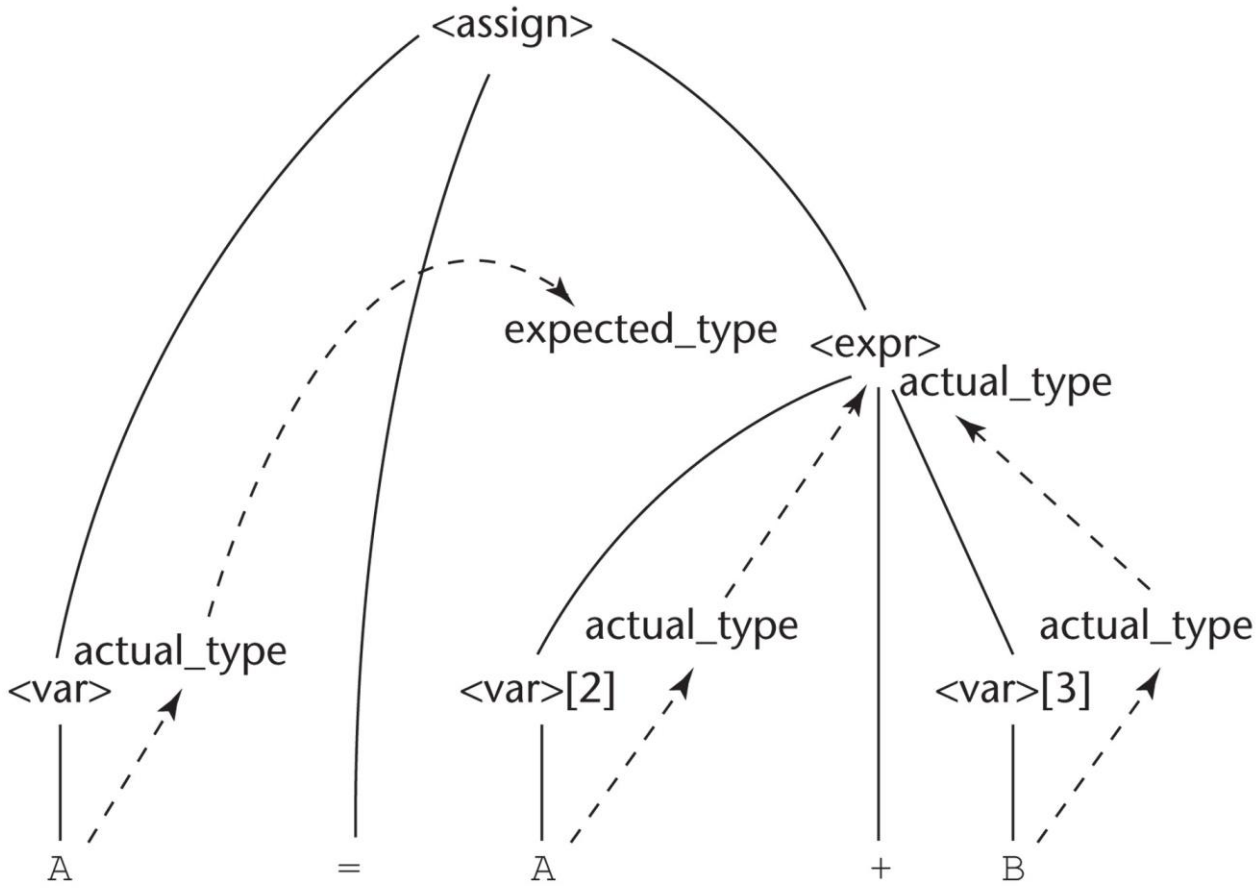


Γραμματικές Χαρακτηριστικών: παράδειγμα (συνέχεια)

- Πως υπολογίζονται οι τιμές των χαρακτηριστικών;
  - Αν όλα τα χαρακτηριστικά ήταν κληρονομούμενα, το δένδρο θα επισημειώνονταν (decorated) από πάνω προς τα κάτω (top-down).
  - Αν όλα τα χαρακτηριστικά ήταν συντιθέμενα, το δένδρο θα επισημειώνονταν από κάτω προς τα πάνω (bottom-up).
  - Σε πολλές περιπτώσεις, χρησιμοποιούνται και οι δύο τύποι χαρακτηριστικών, και κάποιος συνδυασμός από top-down και bottom-up πρέπει να χρησιμοποιηθεί.

Γραμματικές Χαρακτηριστικών: παράδειγμα (συνέχεια)

A = A + B



Γραμματικές Χαρακτηριστικών: παράδειγμα (συνέχεια)

A = A + B

Έστω ότι το A είναι real_type και το B είναι int_type

Σ.Κ. = Σημασιολογικός Κανόνας
Κ.Σ.Κ. = Κατηγόρημα Σημασιολογικού Κανόνα

Σ.Κ.4 <var>.actual_type ← look-up(A)
το <var>.actual_type γίνεται real_type

Σ.Κ.1 <expr>.expected_type ← <var>.actual_type
το <expr>.expected_type γίνεται real_type

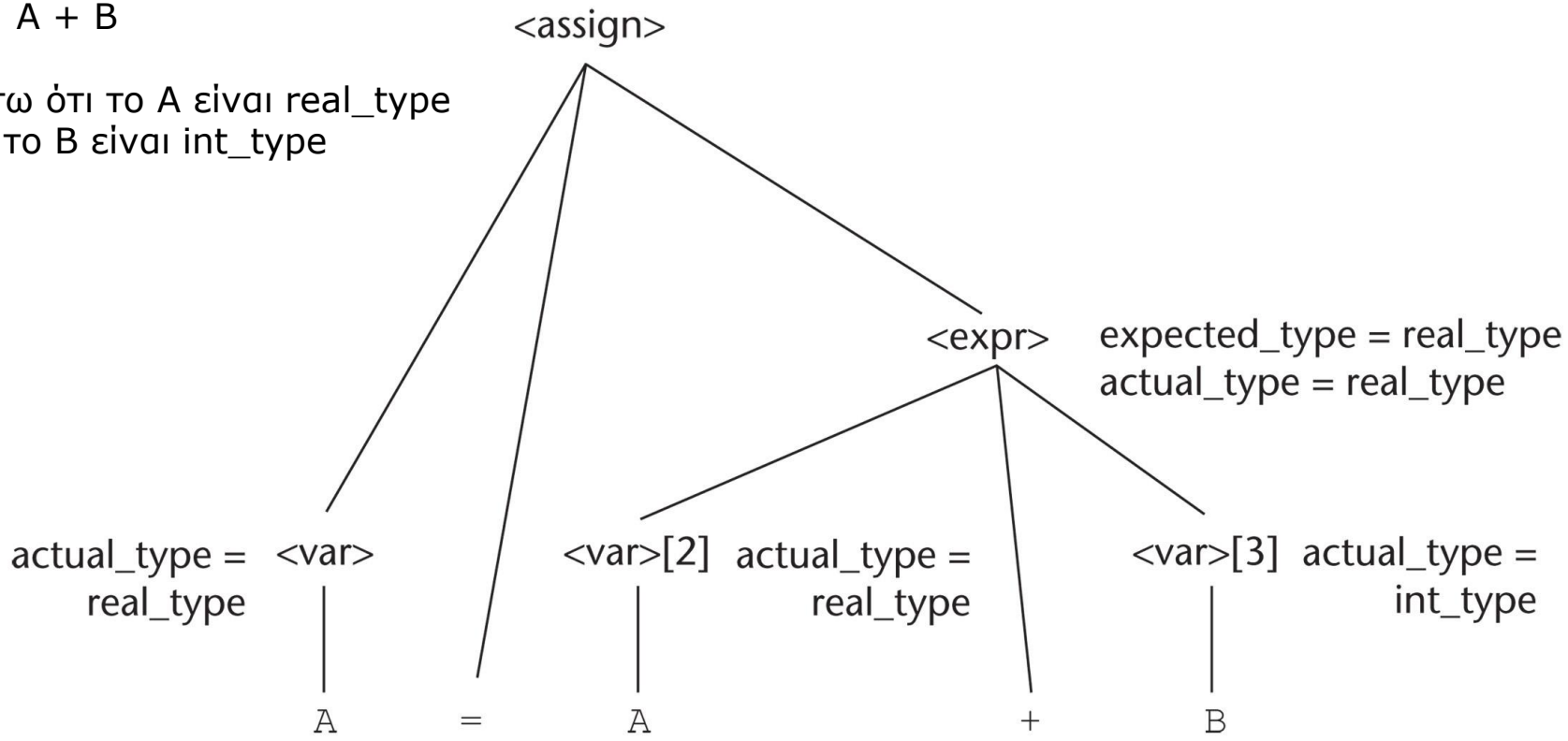
Σ.Κ.4 <var>[2].actual_type ← lookup(A)
το <var>[2].actual_type γίνεται real_type
Σ.Κ.4 <var>[3].actual_type ← lookup(B)
το <var>[3].actual_type γίνεται int_type

Σ.Κ.2 <expr>.actual_type ← είτε int_type είτε real_type
το <expr>.actual_type γίνεται real_type
Κ.Σ.Κ.2 <expr>.expected_type == <expr>.actual_type
το κατηγόρημα αποτιμάται ως TRUE

Γραμματικές Χαρακτηριστικών: παράδειγμα (συνέχεια)

A = A + B

Έστω ότι το A είναι real_type
και το B είναι int_type



Σημασιολογία

- Δεν υπάρχει ένας κοινά αποδεκτός συμβολισμός ή φορμαλισμός για την περιγραφή της σημασιολογίας των γλωσσών προγραμματισμού
- Η ύπαρξη μεθοδολογίας και συμβολισμού για τη σημασιολογία καλύπτει πολλές ανάγκες:
  - Οι προγραμματιστές χρειάζεται να γνωρίζουν το τι σημαίνουν οι εντολές
  - Οι συγγραφείς μεταγλωττιστών πρέπει να γνωρίζουν ακριβώς τι κάνουν οι επιμέρους δομές της γλώσσας
  - Να υπάρχει η δυνατότητα κατασκευής μηχανισμών απόδειξης ορθότητας
  - Να υπάρχει η δυνατότητας κατασκευής γεννητριών μεταγλωττιστών
  - Να μπορούν οι σχεδιαστές να ανιχνεύουν ασάφειες και ασυνέπειες

Λειτουργική Σημασιολογία

- Λειτουργική σημασιολογία
  - Περιγράφει το νόημα ενός προγράμματος εκτελώντας τις εντολές του προγράμματος σε μια μηχανή είτε σε προσομοίωση είτε πραγματικά. Η αλλαγή στην κατάσταση της μηχανής (μνήμη, καταχωρητές, κ.λπ.) ορίζει τη σημασία των εντολών
- Προκειμένου να χρησιμοποιηθεί η λειτουργική σημασιολογία, απαιτείται μια ιδεατή μηχανή

Λειτουργική Σημασιολογία

- Ένας καθαρός διερμηνευτής *υλικού* θα ήταν πολύ ακριβός
- Ένα καθαρός διερμηνευτής *λογισμικού* επίσης παρουσιάζει προβλήματα
  - Οι λεπτομέρειες των χαρακτηριστικών ενός συγκεκριμένου υπολογιστή καθιστά τις ενέργειες δύσκολο να γίνουν κατανοητές
  - Ένας τέτοιος ορισμός σημασιολογίας θα ήταν εξαρτώμενος από τη μηχανή

Λειτουργική Σημασιολογία (συνέχεια)

- Μια καλύτερη εναλλακτική: Μια πλήρης προσομοίωση σε υπολογιστή
- Η διαδικασία:
  - Κατασκευή ενός μεταφραστή (μεταφράζει τον πηγαίο κώδικα σε κώδικα μηχανής ενός εξιδανικευμένου υπολογιστή)
  - Κατασκευή ενός προσομοιωτή για τον εξιδανικευμένο υπολογιστή

Λειτουργική Σημασιολογία (συνέχεια)

- Χρήσεις της λειτουργικής σημασιολογίας:
  - Εγχειρίδια γλώσσας και βιβλία
  - Διδασκαλία γλωσσών προγραμματισμού

- Δύο διαφορετικά επίπεδα χρήσης της λειτουργικής σημασιολογίας:
  - Φυσική λειτουργική σημασιολογία
  - Δομική λειτουργική σημασιολογία

- Αποτίμηση της λειτουργικής σημασιολογίας:
  - Καλή αν χρησιμοποιείται χωρίς προσκόλληση στην τυπικότητα (εγχειρίδια γλώσσας, κλπ)
  - Εξαιρετικά πολύπλοκη αν χρησιμοποιηθεί τυπικά (π.χ., VDL), όπως χρησιμοποιήθηκε για την περιγραφή της σημασιολογίας της PL/I

Δηλωτική Σημασιολογία

- Βασίζεται σε θεωρία αναδρομικών συναρτήσεων
- Πρόκειται για την πλέον αφηρημένη μέθοδο περιγραφής σημασιολογίας
- Αρχικά αναπτύχθηκε από του Scott και Strachey (1970)

Δηλωτική Σημασιολογία – συνέχεια

- Διαδικασία κατασκευής της δηλωτικής σημασιολογίας μιας γλώσσας:
  - Ορισμός ενός μαθηματικού αντικειμένου για κάθε οντότητα της γλώσσας
  - Ορισμός μιας συνάρτηση που αντιστοιχεί στιγμιότυπα των οντοτήτων της γλώσσας σε στιγμιότυπα των αντίστοιχων μαθηματικών αντικειμένων
- Το νόημα των δομών της γλώσσας ορίζεται μόνο από τις τιμές των μεταβλητών του προγράμματος

Δηλωτική σημασιολογία:
κατάσταση προγράμματος

- Η κατάσταση ενός προγράμματος είναι οι τιμές όλων των τρεχουσών μεταβλητών του

$$s = \{<i_1, v_1>, <i_2, v_2>, \ldots, <i_n, v_n>\}$$

- Έστω **VARMAP** μια συνάρτηση, που δεδομένου ενός ονόματος μεταβλητής και μιας κατάστασης, επιστρέφει την τρέχουσα τιμή της μεταβλητής

$$\text{VARMAP}(i_j, s) = v_j$$

Δεκαδικοί αριθμοί

```
<dec_num> → '0' | '1' | '2' | '3' | '4' | '5' |
            '6' | '7' | '8' | '9' |
            <dec_num> ('0' | '1' | '2' | '3' |
                       '4' | '5' | '6' | '7' |
                       '8' | '9')
```

M_{dec}('0') = 0, M_{dec} ('1') = 1, …, M_{dec} ('9') = 9

M_{dec} (<dec_num> '0') = 10 * M_{dec} (<dec_num>)

M_{dec} (<dec_num> '1') = 10 * M_{dec} (<dec_num>) + 1

…

M_{dec} (<dec_num> '9') = 10 * M_{dec} (<dec_num>) + 9

Εκφράσεις

- Αντιστοιχεί εκφράσεις στο Z $\cup$ {error}
- Υποθέτουμε ότι οι εκφράσεις είναι δεκαδικοί αριθμοί, μεταβλητές, ή δυαδικές εκφράσεις που έχουν έναν αριθμητικό τελεστή και δύο τελεστέους, καθένας από τους οποίους μπορεί να είναι μια έκφραση

Εκφράσεις

```
Me(<expr>, s) Δ=
    case <expr> of
      <dec_num> => Mdec(<dec_num>, s)
      <var> =>
           if VARMAP(<var>, s) == undef
                then error
                else VARMAP(<var>, s)
     <binary_expr> =>
          if (Me(<binary_expr>.<left_expr>, s) == undef
                OR Me(<binary_expr>.<right_expr>, s) =
                                 undef)
              then error
          else
          if (<binary_expr>.<operator> == '+' then
             Me(<binary_expr>.<left_expr>, s) +
                    Me(<binary_expr>.<right_expr>, s)
          else Me(<binary_expr>.<left_expr>, s) *
               Me(<binary_expr>.<right_expr>, s)
       ...
```

Εντολές Ανάθεσης

- Αντιστοιχούν σύνολα καταστάσεων σε σύνολα καταστάσεων U {error}

M_a(x := E, s) Δ=
    if M_e(E, s) == error
       then error
       else s′ =
      {<i_1,v_1′>,<i_2,v_2′>,...,<i_n,v_n′>},
                where for j = 1, 2, ..., n,
                    if i_j == x
                      then v_j′ = M_e(E, s)
                      else v_j′ = VARMAP(i_j, s)

Λογικοί προέλεγχοι Βρόχων

- Αντιστοιχούν σύνολα καταστάσεων σε σύνολα καταστάσεων U {error}

```
M_l(while B do L, s) Δ=
    if M_b(B, s) == undef
        then error
        else if M_b(B, s) == false
            then s
            else if M_sl(L, s) == error
                 then error
                 else M_l(while B do L, M_sl(L, s))
```

Σημασία Βρόχου

- Η σημασία του βρόχου είναι η τιμή των μεταβλητών του προγράμματος μετά από όταν οι εντολές του βρόχου έχουν εκτελεστεί τον ορισμένο αριθμό φορών, υποθέτοντας ότι δεν έχουν συμβεί σφάλματα
- Στην πραγματικότητα, ο βρόχος έχει μετατραπεί από επανάληψη σε αναδρομή, όπου ο έλεγχος της αναδρομής ορίζεται μαθηματικά από άλλες αναδρομικές συναρτήσεις αντιστοίχισης καταστάσεων

  – Συγκρίνοντας την αναδρομή,με την επανάληψης, είναι ευκολότερο να περιγραφεί η αναδρομή παρά η επανάληψη με μαθηματική αυστηρότητα

Αποτίμηση της Δηλωτικής Σημασιολογίας

- Μπορεί να χρησιμοποιηθεί για αποδείξεις ορθότητας προγραμμάτων
- Παρέχει έναν μαθηματικά αυστηρό τρόπο σκέψης για τα προγράμματα
- Μπορεί να βοηθήσει στη σχεδίαση γλωσσών
- Έχει χρησιμοποιηθεί στη σχεδίαση συστημάτων γεννητριών μεταγλωττιστών
- Λόγω πολυπλοκότητάς, παρουσιάζει μικρή χρησιμότητα για τους χρήστες της γλώσσας

Αξιωματική Σημασιολογία

- Βασίζεται στην τυπική λογική (κατηγορηματικό λογισμό)
- Αρχικός σκοπός: τυπική επαλήθευση προγραμμάτων
- Ορίζονται αξιώματα ή κανόνες συμπερασμού για κάθε τύπο εντολής της γλώσσας (προκειμένου να επιτραπούν μετασχηματισμοί λογικών εκφράσεων σε περισσότερο τυπικές λογικές εκφράσεις)
- Αυτές οι λογικές εκφράσεις ονομάζονται *ισχυρισμοί (assertions)*

Αξιωματική Σημασιολογία (συνέχεια)

- Ένας ισχυρισμός πριν μια εντολή *(προσυνθήκη – precondition)* ορίζει τις σχέσεις και τους περιορισμούς μεταξύ μεταβλητών που είναι αληθείς στο συγκεκριμένο σημείο εκτέλεσης
- Ένας ισχυρισμός που ακολουθεί μια εντολή ονομάζεται *μετασυνθήκη postcondition*
- Μια ασθενέστατη προσυνθήκη (*weakest precondition)* είναι η λιγότερο περιοριστική προσυνθήκη που εγγυάται τη μετασυνθήκη

Μορφή Αξιωματικής Σημασιολογίας

- Pre-, post μορφή: `{P} statement {Q}`

- Ένα παράδειγμα
  - `a = b + 1  {a > 1}`
  - Μια πιθανή προσυνθήκη: `{b > 10}`
  - Ασθενέστερη προσυνθήκη: `{b > 0}`

Διαδικασία Απόδειξης Προγράμματος

- Η μετασυνθήκη για όλο το πρόγραμμα είναι το επιθυμητό αποτέλεσμα
  - Εργασία προς τα πίσω στο πρόγραμμα μέχρι την πρώτη εντολή. Αν η προσυνθήκη στην πρώτη εντολή είναι ίδια με τις προδιαγραφές του προγράμματος, τότε το πρόγραμμα είναι ορθό.

Αξιωματική Σημασιολογία: Ανάθεση

- Ένα αξίωμα για εντολές ανάθεσης (x = E): $\{Q_{x->E}\}$ x = E {Q}

- Ο κανόνας συνέπειας:

$$\frac{\{P\}S\{Q\}, P' \Rightarrow P, Q \Rightarrow Q'}{\{P'\}S\{Q'\}}$$

Αξιωματική Σημασιολογία: Ακολουθίες

- Ένας κανόνας συμπερασμού για ακολουθίες της μορφής S1; S2

{P1} S1 {P2}

{P2} S2 {P3}

$$\frac{\{P1\}S1\{P2\},\{P2\}S2\{P3\}}{\{P1\}S1;S2\{P3\}}$$

Αξιωματική Σημασιολογία : Επιλογή

- Ένας κανόνας συμπερασμού για την if
  - **`if`** B **`then`** S1 **`else`** S2

$$\frac{\{B \text{ and } P\}\ S1\ \{Q\},\ \{(\text{not } B) \text{ and } P\}\ S2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ S1\ \texttt{else}\ S2\ \{Q\}}$$

Αξιωματική Σημασιολογία: Βρόχοι

- Ένας κανόνας συμπερασμού για λογικούς προελέγχους σε βρόχους

  {P} **`while`** B **`do`** S **`end`** {Q}

$$\frac{(\mathrm{I}\ \mathrm{and}\ \mathrm{B})\ \mathrm{S}\ \{\mathrm{I}\}}{\{\mathrm{I}\}\ \mathrm{while}\ \mathrm{B}\ \mathrm{do}\ \mathrm{S}\ \{\mathrm{I}\ \mathrm{and}\ (\mathrm{not}\ \mathrm{B})\}}$$

  όπου I είναι η αναλλοίωτη βρόχου (η επαγωγική υπόθεση)

Αξιωματική Σημασιολογία: Αξιώματα

- Χαρακτηριστικά του loop invariant: το I θα πρέπει να ικανοποιεί τις ακόλουθες συνθήκες:
  - P => I -- το loop invariant θα πρέπει να είναι αληθές αρχικά
  - {I} B {I} -- η αποτίμηση του Boolean δεν θα πρέπει να αλλάζει την εγκυρότητα του I
  - {I and B} S {I} -- το I δεν αλλάζει εκτελώντας το σώμα του βρόχου
  - (I and (not B)) => Q -- αν το I είναι true και το B είναι false, τότε Q
  - Ο βρόχος τερματίζει -- μπορεί να είναι δύσκολο να αποδειχθεί

Αναλλοίωτη Βρόχου (Loop Invariant)

- Το loop invariant I είναι μια ασθενής μορφή της μετασυνθήκης βρόχου, και είναι επίσης και μετασυνθήκη
- Το I θα πρέπει να είναι επαρκώς ασθενές έτσι ώστε να ικανοποιείται πριν την εκκίνηση του βρόχου, αλλά όταν θα συνδυάζεται με τη συνθήκη εξόδου από το βρόχο, θα πρέπει να είναι επαρκώς ισχυρό έτσι ώστε να επιβάλλει στη μετασυνθήκη να είναι αληθής

Αποτίμηση Αξιωματικής Σημασιολογίας

- Είναι δύσκολη η ανάπτυξη αξιωμάτων ή κανόνων συμπερασμού για όλες τις εντολές μιας γλώσσας
- Είναι καλό εργαλείο για αποδείξεις ορθότητας, και εξαιρετικό πλαίσιο συλλογισμού για τα προγράμματα, αλλά δεν είναι το ίδιο χρήσιμο για χρήστες γλωσσών και συγγραφείς μεταγλωττιστών
- Έχει περιορισμένη χρησιμότητα ως μέσο περιγραφής του νοήματος μιας γλώσσας προγραμματισμού σε ότι αφορά χρήστες της γλώσσας και συγγραφείς μεταγλωττιστών

Δηλωτική σημασιολογία vs Λειτουργική σημασιολογία

- Στη λειτουργική σημασιολογία, οι αλλαγές κατάστασης ορίζονται με αλγορίθμους σε κώδικα
- Στη δηλωτική σημασιολογία, οι αλλαγές κατάστασης ορίζονται από αυστηρές μαθηματικές συναρτήσεις

Σύνοψη

- Η BNF και οι γραμματικές χωρίς συμφραζόμενα είναι ισοδύναμες μετα-γλώσσες
  - Κατάλληλες για την περιγραφή του συντακτικού των γλωσσών προγραμματισμού
- Μια γραμματική χαρακτηριστικών είναι ένας περιγραφικός φορμαλισμός που μπορεί να περιγράψει τόσο το συντακτικό όσο και τη σημασιολογία μιας γλώσσας
- Υπάρχουν τρεις κύριες μέθοδοι περιγραφής της σημασιολογίας
  - Λειτουργική, δηλωτική, αξιωματική