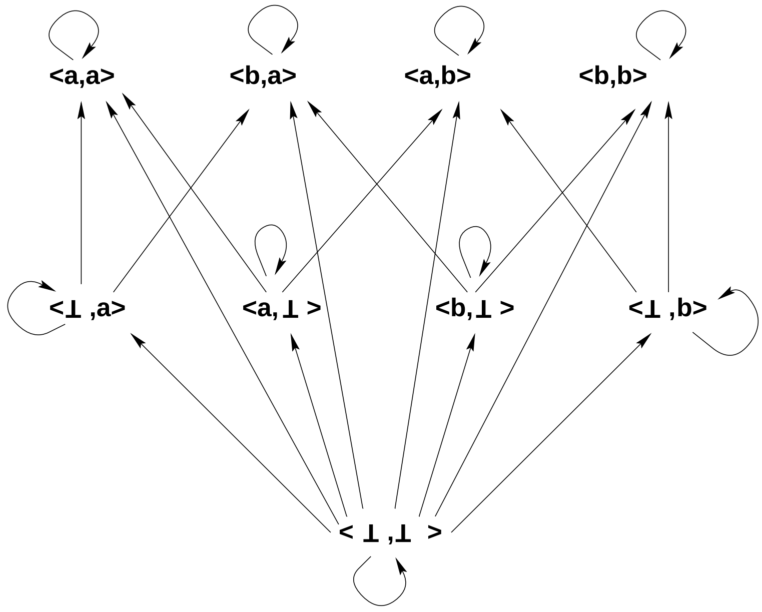

ΕΙΔΙΚΑ ΘΕΜΑΤΑ ΘΕΩΡΗΤΙΚΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ:
Σημασιολογία Γλωσσών Προγραμματισμού

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
Φεβρουάριος 2002

Παναγιώτης Ροντογιάννης

Contents

1 Σημασιολογία **2**

1.1 Εισαγωγή 2

1.2 Μέθοδοι Σημασιολογίας 3

1.2.1 Η Μηχανιστική Μεθοδολογία (Operational Semantics) 3

1.2.2 Η Αξιωματική Μεθοδολογία (Axiomatic Semantics) 4

1.2.3 Η Ενδεικτική Μεθοδολογία (Denotational Semantics) 4

1.2.4 Παρατηρήσεις 4

1.3 Ενα Παράδειγμα: Δυαδικές Ακολουθίες 5

1.4 Συνθεσιμότητα 6

1.5 Ασκήσεις 7

2 Μια Απλή Προστακτική Γλώσσα **8**

2.1 Εισαγωγή 8

2.2 Εκφράσεις και Εντολές 8

2.2.1 Συντακτικό 8

2.2.2 Σημασιολογία 10

2.2.3 Παρατηρήσεις 12

2.3 Εντολές Ανάθεσης 13

2.4 Αόριστες Επαναλήψεις 14

2.5 Ασκήσεις 18

Chapter 1

Σημασιολογία

1.1 Εισαγωγή

Οσοι σχεδιάζουν, υλοποιούν ή ακόμη και χρησιμοποιούν μιά γλώσσα προγραμματισμού, χρειάζεται να γνωρίζουν τη γλώσσα από δύο τελείως διαφορετικές σκοπιές:

- Από τη σκοπιά του *συντακτικού* (syntax)
- Από τη σκοπιά της *σημασιολογίας* (semantics)

Για την αποτελεσματική περιγραφή του συντακτικού, υπάρχει η θεωρία των τυπικών γλωσσών (π.χ. regular languages, context-free languages, κλπ).

Για την περιγραφή της σημασιολογίας όμως, για πολύ καιρό οι ερευνητές κατέφευγαν σε άτυπες μεθόδους (όπως για παράδειγμα είναι η χρησιμοποίηση της φυσικής γλώσσας).

Παράδειγμα 1.1 Ας θεωρήσουμε την περίπτωση του βρόχου `while` ο οποίος έχει τη μορφή

`while` ⟨λογική συνθήκη⟩ `do` ⟨εντολές⟩

Μιά άτυπη περιγραφή της σημασιολογίας του `while` μπορεί να είναι η ακόλουθη:

> "Αρχικά γίνεται ο έλεγχος της λογικής συνθήκης. Αν το αποτέλεσμα είναι αληθές, τότε γίνεται είσοδος στο βρόχο και οι εντολές που βρίσκονται μέσα στον βρόχο εκτελούνται μία φορά. Στη συνέχεια η συνθήκη ελέγχεται και πάλι. Οταν η συνθήκη γίνει ψευδής, ο βρόχος παρακάμπτεται και ο έλεγχος μεταφέρεται στην πρώτη εντολή που ακολουθεί τη δομή του βρόχου".

■

Τα πράγματα όμως δεν είναι πάντα τόσο απλά όσο στην παραπάνω περίπτωση. Ο αρχικός ορισμός της ALGOL 60 γιά παράδειγμα, περιείχε διφορούμενους ορισμούς, και έτσι ένας νέος "διορθωμένος" ορισμός προτάθηκε το 1963. Τα προβλήματα συνεχίστηκαν και το 1967 ο D. Knuth έγραφε μιά αναφορά με τα λάθη των ορισμών του 1963. Εγραφε:

> "Οταν η ALGOL 60 πρωτοδημοσιεύτηκε το 1960, πολλά νέα στοιχεία δημιουργήθηκαν γιά τις γλώσσες προγραμματισμού... Ηταν δύσκολο αρχικά για οποιονδήποτε να καταλάβει την σημασία του κάθε νέου στοιχείου σε σχέση με όλα τα άλλα στοιχεία τα οποία διέθετε η γλώσσα, και πολλοί άνθρωποι όταν διάβαζαν την αναφορά της γλώσσας συχνά ανακάλυπταν πράγματα τα οποία πριν κανείς δεν γνώριζε οτι ήταν δυνατά. Συχνά τα νεοανακαλυφθέντα αυτά στοιχεία έδιναν αντιπαραδείγματα στις μέχρι τότε γνωστές τεχνικές για την υλοποίηση της γλώσσας, και σε πολλές περιπτώσεις μπορούσε κανείς να κατασκευάσει προγράμματα τα οποία ήταν δυνατό να υλοποιηθούν με διαφορετικούς τρόπους (δίνοντας διαφορετικά αποτελέσματα)".

Παράδειγμα 1.2 Είναι στην ALGOL 60 ισοδύναμες οι εκφράσεις `x+f(x)` και `f(x)+x`; Η απάντηση εξαρτάται από το εάν η `f` αλλάζει την τιμή του `x`. Εάν συμβαίνει κάτι τέτοιο, τότε η τιμή των δύο παραπάνω εκφράσεων μπορεί να είναι διαφορετική ανάλογα με τη σειρά που η υλοποίηση της γλώσσας υπολογίζει τις τιμές των εκφράσεων `x` και `f(x)`. ■

Μιά αυστηρή μαθηματική θεωρία της σημασιολογίας των γλωσσών προγραμματισμού είναι επομένως αναγκαία έτσι ώστε να εξασφαλιστούν:

- Η ορθή περιγραφή και υλοποίηση των γλωσσών προγραμματισμού
- Η συστηματική ανάπτυξη και απόδειξη ορθότητας των προγραμμάτων.
- Η ανάλυση και αξιολόγηση των υπαρχόντων γλωσσών προγραμματισμού
- Ο σχεδιασμός νέων, απλούστερων αλλά και πιό ισχυρών γλωσσών προγραμματισμού.

Η θεωρία αυτή της σημασιολογίας και οι εφαρμογές της είναι το αντικείμενο του μαθήματος αυτού.

1.2 Μέθοδοι Σημασιολογίας

Τρείς είναι οι κύριες μέθοδοι που έχουν αναπτυχθεί για την περιγραφή της σημασιολογίας των γλωσσών προγραμματισμού:

1.2.1 Η Μηχανιστική Μεθοδολογία (Operational Semantics)

Στη μεθοδολογία αυτή ορίζεται αρχικά ένα "αφηρημένο μοντέλο μηχανής", το οποίο αποτελείται από την *κατάσταση (state)* και από ένα σύνολο *βασικών εντολών*. Κατόπιν, η σημασιολογία μιας γλώσσας προγραμματισμού μπορεί να οριστεί με βάση το μοντέλο αυτό. Η βασική ιδέα είναι ότι αν και το μοντέλο το οποίο χρησιμοποιείται είναι πολλές φορές μη-ρεαλιστικό από πρακτική σκοπιά, είναι τόσο απλό ώστε να μην αφήνει περιθώρια για παρανοήσεις. Η σημασιολογία της γλώσσας καθορίζεται ως μετάφραση των βασικών της εντολών στην γλώσσα του αφηρημένου μοντέλου. Γλώσσες που έχουν οριστεί με τη μέθοδο αυτή είναι η PL/1 (χρησιμοποιώντας τη μέθοδο VDM), όπως επίσης και η ALGOL 68.

1.2.2 Η Αξιωματική Μεθοδολογία (Axiomatic Semantics)

Στη μεθοδολογία αυτή, ένα αξίωμα συνδέεται με κάθε είδος εντολής που υπάρχει στη γλώσσα προγραμματισμού. Το αξίωμα αυτό προσδιορίζει τι ακριβώς μπορεί να θεωρηθεί ως αληθές μετά την εκτέλεση της εντολής, σε σχέση με αυτά που ίσχυαν πρίν από την εκτέλεση της εντολής. Γιά παράδειγμα, αν θεωρήσουμε το βρόχο

`while` **B** `do` **C**

όπου **B** είναι μιά έκφραση και **C** μιά εντολή, γνωρίζουμε ότι όταν ο βρόχος ολοκληρωθεί (εάν ποτέ ολοκληρωθεί) η τιμή της έκφρασης **B** θα είναι ψευδής.

Το προτέρημα της μεθόδου είναι ότι πολλές φορές τα αξιώματα που εισάγονται μπορούν να εκφραστούν στην ίδια τη γλώσσα προγραμματισμού (με ελάχιστες αλλαγές στο συντακτικό της). Αυτό είναι σημαντικό διότι εάν η απόδειξη ορθότητας ενός προγράμματος γίνει ποτέ μέλημα του ίδιου του προγραμματιστή, είναι λογικό να περιμένει κανείς οτι τόσο η απόδειξη όσο και ο προγραμματισμός θα γίνονται στην ίδια γλώσσα.

1.2.3 Η Ενδεικτική Μεθοδολογία (Denotational Semantics)

Στη μεθοδολογία αυτή ορίζονται "συναρτήσεις υπολογισμού" (valuation functions), οι οποίες αντιστοιχίζουν τα συντακτικά στοιχεία της γλώσσας σε γνωστά μαθηματικά αντικείμενα (π.χ. αριθμούς, τιμές αληθείας, συναρτήσεις, κλπ). Οι συναρτήσεις υπολογισμού ορίζονται συνήθως αναδρομικά: η τιμή που θα λάβει μιά έκφραση της γλώσσας καθορίζεται με βάση τις τιμές των υποεκφράσεων που την αποτελούν.

1.2.4 Παρατηρήσεις

Και οι τρεις παραπάνω μέθοδοι αποτελούν αποδεκτές τεχνικές για τον καθορισμό της σημασιολογίας μιας γλώσσας προγραμματισμού. Διαφέρουν όμως σημαντικά ως προς τη φιλοσοφία τους. Ανακεφαλαιώνοντας, μπορεί κανείς να πει οτι η Μηχανιστική μέθοδος ταιριάζει περισσότερο σε αυτούς που υλοποιούν μια γλώσσα, η Αξιωματική είναι πιό αποδεκτή από τους προγραμματιστές ενώ η Ενδεικτική είναι απαραίτητο να χρησιμοποιείται από τους σχεδιαστές της γλώσσας. Στο μεγαλύτερο μέρος του μαθήματος αυτού θα επικεντρώσουμε το ενδιαφέρον μας στις τεχνικές που χρησιμοποιούνται από την Ενδεικτική μεθοδολογία.

1.3 Ενα Παράδειγμα: Δυαδικές Ακολουθίες

Σαν ένα εισαγωγικό παράδειγμα των μεθόδων που θα ακολουθήσουμε στο μάθημα αυτό, θεωρούμε τη στοιχειώδη "γλώσσα" των δυαδικών ακολουθιών, για την οποία δίνουμε το συντακτικό και τη σημασιολογία της.

<u>Συντακτικό</u>

Το συντακτικό της γλώσσας των δυαδικών ακολουθιών μπορεί να περιγραφεί με τη βοήθεια μιας context-free γραμματικής, ως εξής:

$$\begin{array}{rcl} \mathbf{S} & ::= & \mathtt{0} \\ & | & \mathtt{1} \\ & | & \mathbf{S}\mathtt{0} \\ & | & \mathbf{S}\mathtt{1} \end{array}$$

Εστω Seq το σύνολο των ακολουθιών που παράγονται με βάση τον παραπάνω ορισμό. Ο ορισμός αυτός μας λέει ότι το Seq αποτελείται από πεπερασμένες ακολουθίες των ψηφίων 0 και 1. Ομως, προς το παρόν δεν γνωρίζουμε τι αναπαριστούν οι ακολουθίες αυτές, δεν ξέρουμε τίποτα δηλαδή γιά το *νόημα* τους.

<u>Ενδεικτική Σημασιολογία:</u>

Μπορούμε να δώσουμε νόημα στις δυαδικές ακολουθίες ψηφίων, αν αντιστοιχίσουμε σε κάθε μία από αυτές έναν μοναδικό φυσικό αριθμό. Γιά το σκοπό αυτό χρειάζεται να ορίσουμε μιά συνάρτηση που ουσιαστικά μετατρέπει κάθε ακολουθία σε έναν φυσικό. Τη συνάρτηση αυτή θα τη συμβολίζουμε με $[\![\cdot]\!]$ (ο συμβολισμός αυτός θα χρησιμοποιείται πολύ συχνά από εδώ και μπρός, κάθε φορά που θέλουμε να ορίσουμε τη σημασιολογία μιας γλώσσας). Στη συγκεκριμένη περίπτωση, η $[\![\cdot]\!]$ μετατρέπει ακολουθίες δυαδικών ψηφίων σε φυσικούς αριθμούς (δηλαδή $[\![\cdot]\!] : Seq \to Nat$), και ορίζεται ως ακολούθως:

$$\begin{array}{rcl} [\![\mathtt{0}]\!] & = & 0 \\ [\![\mathtt{1}]\!] & = & 1 \\ [\![\mathbf{S}\mathtt{0}]\!] & = & 2 \times [\![\mathbf{S}]\!] \\ [\![\mathbf{S}\mathtt{1}]\!] & = & 2 \times [\![\mathbf{S}]\!] + 1 \end{array}$$

Ο παραπάνω ορισμός προσφέρει μιά μέθοδο γιά τον υπολογισμό του δεκαδικού αριθμού που αντιστοιχεί σε δεδομένο δυαδικό.

Παράδειγμα 1.3 Το νόημα της ακολουθίας 1100 μπορεί να δοθεί χρησιμοποιώντας τον ορισμό της σημασιολογίας των δυαδικών ακολουθιών, ώς εξής:

$$
\begin{aligned}
[\![\texttt{1100}]\!] &= 0 \\
&= 2 \times [\![\texttt{110}]\!] \\
&= 2 \times (2 \times [\![\texttt{11}]\!]) \\
&= 2 \times (2 \times (2 \times [\![\texttt{1}]\!] + 1)) \\
&= 2 \times (2 \times (2 \times 1 + 1)) \\
&= 12
\end{aligned}
$$

■

Παρατηρήστε οτι στον παραπάνω ορισμό υπάρχει μία εξίσωση για κάθε κανόνα του συντακτικού. Δεν είναι εντελώς προφανές οτι οι παραπάνω εξισώσεις ορίζουν μια μαθηματική συνάρτηση: το σύμβολο $[\![\cdot]\!]$ εμφανίζεται και στη δεξιά αλλά και στην αριστερή πλευρά των εξισώσεων, έχουμε δηλαδή έναν αναδρομικό ορισμό της $[\![\cdot]\!]$. Μπορεί να δειχθεί οτι η $[\![\cdot]\!]$ είναι όντως συνάρτηση, και αυτό διαισθητικά οφείλεται στο γεγονός οτι στο δεξιό μέλος των παραπάνω εξισώσεων η $[\![\cdot]\!]$ εφαρμόζεται σε μιά υποέκφραση της έκφρασης στην οποία εφαρμόζεται στο αριστερό (πχ. στην τρίτη εξίσωση, η $[\![\cdot]\!]$ στα αριστερά εφαρμόζεται στην έκφραση $\mathbf{S}\texttt{0}$ ενώ στα δεξιά στην υποέκφραση $\mathbf{S}$).

Ενα άλλο σημαντικό σημείο που πρέπει να τονιστεί αφορά εξισώσεις της μορφής:

$$[\![\texttt{0}]\!] \;=\; 0$$

πού δόθηκε παραπάνω. Δικαιολογημένα μπορεί κανείς να αναρρωτηθεί αν εξισώσεις σαν και αυτή έχουν κάποιο ουσιαστικό νόημα, ή απλώς επαναλαμβάνουν το ίδιο πράγμα.

Η αλήθεια είναι ότι η εξίσωση αυτή δεν είναι κυκλική. Το `0` που εμφανίζεται στα αριστερά είναι ένα σύμβολο της *γλώσσας-αντικείμενο (object language)*. Η γλώσσα-αντικείμενο είναι η γλώσσα την οποία μελετάμε και στην οποία θέλουμε να δώσουμε νόημα. Στη συγκεκριμένη περίπτωση η γλώσσα αυτή είναι το σύνολο των δυαδικών ακολουθιών, αλλά θα μπορούσε κάλλιστα να είναι μιά οποιαδήποτε γλώσσα προγραμματισμού (πχ. C, Pascal).

Αντίθετα, to 0 που εμφανίζεται στα δεξιά είναι ο φυσικός αριθμός μηδέν που γνωρίζουμε από τα μαθηματικά. Οπως έχουμε πει, σκοπός της ενδεικτικής σημασιολογίας είναι να μετατρέπει τα συντακτικά στοιχεία μιας γλώσσας προγραμματισμού σε γνωστά μαθηματικά αντικείμενα. Τα μαθηματικά αποτελούν λοιπόν την λεγόμενη *μεταγλώσσα (metalanguage)* στην οποία και θα μετατρέπουμε τα προγράμματα της γλώσσας αντικείμενο.

1.4 Συνθεσιμότητα

Η σημασιολογική περιγραφή των δυαδικών ακολουθιών που παρουσιάσαμε στην προηγούμενη παράγραφο έχει την σημαντική ιδιότητα ότι:

> "Το νόημα κάθε σύνθετης εκφράσης δίνεται ως συνάρτηση των νοημάτων των υποεκφράσεων που την αποτελούν"

Η ιδιότητα αυτή ονομάζεται *συνθεσιμότητα*.

Παράδειγμα 1.4 Οι εξισώσεις στον ορισμό της σημασιολογίας των δυαδικών ακολουθιών υπακούουν στον κανόνα της συνθεσιμότητας, διότι ανάγουν τον υπολογισμό της σημασιολογίας των εκφράσεων **S**`0` και **S**`1` στην σημασιολογία της απλούστερης έκφρασης **S**. ∎

Ενα παράδειγμα σημασιολογίας που δεν υπακούει στον κανόνα της συνθεσιμότητας είναι το ακόλουθο:

$$[\![\texttt{while } \mathbf{B} \texttt{ do } \mathbf{C}]\!] = [\![\texttt{if } \mathbf{B} \texttt{ then } (\mathbf{C}; \texttt{while } \mathbf{B} \texttt{ do } \mathbf{C})]\!]$$

Παρόλο που η ιδιότητα που εκφράζεται από την παραπάνω εξίσωση είναι κάτι που θα θέλαμε να ισχύει, η εξίσωση δεν υπακούει στον κανόνα της συνθεσιμότητας διότι η έκφραση που περικλείεται από το σύμβολο $[\![\cdot]\!]$ στη δεξιά πλευρά δεν αποτελεί υποέκφραση αυτής που εμφανίζεται στην αριστερή πλευρά. Ακόμη και αν γράφαμε κάποια εξίσωση της μορφής:

$$[\![\texttt{while } \mathbf{B} \texttt{ do } \mathbf{C}]\!] = \cdots [\![\mathbf{B}]\!] \cdots [\![\mathbf{C}]\!] \cdots [\![\texttt{while } \mathbf{B} \texttt{ do } \mathbf{C}]\!] \cdots$$

και πάλι δεν θα ίσχυε ο κανόνας της συνθεσιμότητας. Παρόλ'αυτά όπως θα δούμε στο επόμενο κεφάλαιο, μια εξίσωση όπως η παραπάνω μπορεί να λυθεί και η λύση της να χρησιμοποιηθεί ώστε να δοθεί ένας ορισμός της σημασιολογίας του `while` ο οποίος υπακούει στον κανόνα της συνθεσιμότητας.

1.5 Ασκήσεις

1. Δείξτε οτι κάθε ακολουθία δυαδικών αριθμών αναπαριστά (με βάση τον ορισμό της $[\![\cdot]\!]$) έναν μοναδικό φυσικό αριθμό. (Χρησιμοποιείστε επαγωγή στο μήκος της ακολουθίας).

2. Δείξτε οτι κάθε φυσικός αριθμός αναπαρίσταται από τουλάχιστον μία ακολουθία δυαδικών αριθμών.

3. Με βάση τη γλώσσα των δυαδικών ακολουθιών που ορίστηκε στις προηγούμενες παραγράφους, μπορούμε να ορίσουμε τη γλώσσα των δυαδικών ακολουθιών που δεν έχουν αρχικά μηδενικά, ως εξής:
   $\mathbf{S}' ::= \texttt{0} \mid \texttt{1} \mid \texttt{1}\mathbf{S}$
   Ορίστε τη σημασιολογία της νέας αυτής γλώσσας.

Chapter 2

Μια Απλή Προστακτική Γλώσσα

2.1 Εισαγωγή

Στο κεφάλαιο αυτό θα μελετήσουμε μια γλώσσα προγραμματισμού που αποτελείται από *εκφράσεις (expressions)* και *εντολές (commands)*. Η βασική σημασιολογική έννοια που θα χρησιμοποιήσουμε είναι αυτή της *κατάστασης (state)*, των τιμών δηλαδή που έχουν οι μεταβλητές στη μνήμη του υπολογιστή. Γενικά, η τιμή που έχει μιά έκφραση του προγράμματος εξαρτάται από την παρούσα κατάσταση του υπολογιστή. Η κατάσταση αυτή μπορεί να αλλάξει με την εκτέλεση μιας εντολής (πχ. με την καταχώρηση μιας νέας τιμής σε μιά από τις μεταβλητές του προγράμματος). Η πορεία εκτέλεσης των εντολών ενός προγράμματος καθορίζεται από τις λεγόμενες *δομές ελέγχου* (όπως είναι για παράδειγμα το `if-then-else`, το `for`, το `while`, κλπ). Οι γλώσσες που υποστηρίζουν το παραπάνω στύλ προγραμματισμού ονομάζονται *προστακτικές*.

Η προστακτική γλώσσα που θεωρούμε σε αυτό το κεφάλαιο είναι "απλή" με την έννοια οτι δεν υποστηρίζει `goto`, δομές δεδομένων, δείκτες, και άλλα χαρακτηριστικά που είναι πολύ συνηθισμένα σε μια τυπική προστακτική γλώσσα. Καθώς σκοπός μας είναι η παρουσίαση των κύριων ιδεών της σημασιολογίας των γλωσσών προγραμματισμού, δεν θα εξετάσουμε στις σημειώσεις αυτές τα "προχωρημένα" αυτά χαρακτηριστικά.

2.2 Εκφράσεις και Εντολές

2.2.1 Συντακτικό

Η απλή προστακτική γλώσσα που θα ορίσουμε υποστηρίζει δύο διαφορετικούς τύπους δεδομένων: τιμές αληθείας (boolean) και φυσικούς αριθμούς (natural). Επιπλέον τύποι δεδομένων (όπως int, real, char, κλπ) μπορούν να προστεθούν στη γλώσσα χωρίς ιδιαίτερο πρόβλημα. Τα συντακτικά στοιχεία της γλωσσας μπορούν να χωριστούν σε τρεις κατηγορίες: λογικές εκφράσεις (boolean expressions), αριθμητικές εκφράσεις (numerical expressions) και εντολές (commands).

Οι βασικές αριθμητικές εκφράσεις είναι μια σταθερά (`0`) και ένας τελεστής (`succ`) ο οποίος παράγει τον επόμενο φυσικό αριθμό. Οι βασικές λογικές εκφράσεις είναι η σταθερά `true` όπως και εκφράσεις που μπορούν να παραχθούν με τη χρήση τελεστών (πχ. `not`, `and`, κλπ). Γενικά, στη γλώσσα μπορούν να προστεθούν περισσότερα σύμβολα (πχ. `false`, `1`, `2`, κλπ) και περισσότεροι τελεστές, χωρίς πρόβλημα.

Οι κύριες εντολές που υποστηρίζει η γλώσσα είναι: η "κενή" εντολή (`skip`) η οποία δεν έχει κανένα απολύτως αποτέλεσμα, η σύνθεση εντολών που συμβολίζεται με ";", και ένας μηχανισμός βρόχου. Ο βρόχος `for` **N** `do` **C** εκτελείται υπολογίζοντας την τιμή της αριθμητικής έκφρασης **N** και μετά εκτελώντας το σώμα **C** τόσες φορές όσες είναι η τιμή της αριθμητικής έκφρασης. Ο βρόχος αυτός ονομάζεται και *οριστικός* διότι ο αριθμός των επαναλήψεων καθορίζεται πριν αρχίσει η εκτέλεση του.

Αρχικά, η γλώσσα δεν διαθέτει μηχανισμούς αλλαγής της κατάστασης (δηλαδή της μνήμης). Ο μηχανισμός αυτός, που είναι η γνωστή εντολή ανάθεσης (assignment), θα μελετηθεί στην επόμενη παράγραφο.

Το συντακτικό της γλώσσας μπορεί να δοθεί με τη βοήθεια context-free γραμματικής, στην οποία το σύμβολο **C** αναπαριστά μια εντολή, το σύμβολο **B** μιά λογική έκφραση και το **N** μιά αριθμητική έκφραση.

Οι κανόνες παραγωγής γιά τις εντολές είναι οι ακόλουθοι:

$$\begin{array}{rcl}
\mathbf{C} & ::= & \texttt{skip} \\
 & | & \mathbf{C}_0 ; \mathbf{C}_1 \\
 & | & \texttt{for}\ \mathbf{N}\ \texttt{do}\ \mathbf{C} \\
 & | & \texttt{if}\ \mathbf{B}\ \texttt{then}\ \mathbf{C}_0\ \texttt{else}\ \mathbf{C}_1 \\
 & | & (\mathbf{C})
\end{array}$$

Οι κανόνες παραγωγής γιά τις λογικές εκφράσεις είναι οι ακόλουθοι:

$$\begin{array}{rcl}
\mathbf{B} & ::= & \texttt{true} \\
 & | & \texttt{not}\ \mathbf{B} \\
 & | & \mathbf{B}_0\ \texttt{and}\ \mathbf{B}_1 \\
 & | & \mathbf{N}_0 < \mathbf{N}_1 \\
 & | & \mathbf{N}_0 = \mathbf{N}_1 \\
 & | & \mathbf{B}_0 = \mathbf{B}_1 \\
 & | & \texttt{if}\ \mathbf{B}\ \texttt{then}\ \mathbf{B}_0\ \texttt{else}\ \mathbf{B}_1 \\
 & | & (\mathbf{B})
\end{array}$$

Τέλος, οι κανόνες παραγωγής γιά τις αριθμητικές εκφράσεις είναι:

$$\begin{array}{rcl} \mathbf{N} & ::= & \mathtt{0} \\ & | & \mathtt{succ}\ \mathbf{N} \\ & | & \mathtt{if}\ \mathbf{B}\ \mathtt{then}\ \mathbf{N}_0\ \mathtt{else}\ \mathbf{N}_1 \\ & | & (\mathbf{N}) \end{array}$$

2.2.2 Σημασιολογία

Η βασική έννοια που χρησιμοποιείται για τη σημασιολογία των προστακτικών γλωσσών προγραμματισμού είναι αυτή της κατάστασης (state). Η κατάσταση ουσιαστικά αναπαριστά τη μνήμη του υπολογιστή. Οπως γνωρίζουμε, η τιμή μιας έκφρασης σε μιά προστακτική γλώσσα προγραμματισμού εξαρτάται από τις μεταβλητές που υπάρχουν στην έκφραση αυτή, οι τιμές των οποίων φυλάγονται στη μνήμη του υπολογιστή. Επομένως, η σημασιολογία των προστακτικών γλωσσών πρέπει πάντα να λαμβάνει υπ'όψιν της και να χρησιμοποιεί την έννοια της κατάστασης.

Στην παρούσα φάση της, η γλώσσα που ορίσαμε στις προηγούμενες παραγράφους δεν διαθέτει ούτε μεταβλητές αλλά ούτε και εντολές ανάθεσης. Επειδή όμως η γλώσσα αυτή θα επεκταθεί στις επόμενες παραγράφους ώστε να υποστηρίζει και τα χαρακτηριστικά αυτά, και επειδή ήδη έχουμε εισάγει κάποιες (έστω απλές) εντολές, θα δώσουμε ακολούθως τη σημασιολογία της γλώσσας με βάση την έννοια της κατάστασης.

Εστω S λοιπόν ένα σύνολο καταστάσεων (το οποίο προς το παρόν δεν το καθορίζουμε επακριβώς). Διαισθητικά, **το νόημα μιας έκφρασης της γλώσσας είναι μιά συνάρτηση η οποία γιά κάθε κατάσταση του υπολογιστή δίνει την τιμή της έκφρασης σε σχέση με την κατάσταση αυτή. Το νόημα μιας εντολής είναι μια συνάρτηση η οποία μετασχηματίζει την τρέχουσα κατάσταση του υπολογιστή σε μια νέα κατάσταση.**

Για τον ορισμό της σημασιολογίας της γλώσσας θα χρησιμοποιήσουμε τρεις διαφορετικές συναρτήσεις: την συνάρτηση $\mathcal{C}[\![\cdot]\!]$ η οποία δίνει νόημα στις εντολές της γλώσσας. Την συνάρτηση $\mathcal{B}[\![\cdot]\!]$ η οποία δίνει νόημα στις λογικές εκφράσεις, και την συνάρτηση $\mathcal{N}[\![\cdot]\!]$ η οποία δίνει νόημα στις αριθμητικές εκφράσεις. Οι συναρτήσεις αυτές ορίζονται αναδρομικά, και ο ορισμός της κάθε μίας χρησιμοποιεί τις υπόλοιπες.

Η συνάρτηση $\mathcal{B}[\![\cdot]\!]$ ορίζεται ως εξής:

$$\mathcal{B}[\![\texttt{true}]\!]s = true$$

$$\mathcal{B}[\![\texttt{not } \mathbf{B}]\!]s = \begin{cases} true, & if\ \mathcal{B}[\![\mathbf{B}]\!]s = false \\ false, & if\ \mathcal{B}[\![\mathbf{B}]\!]s = true \end{cases}$$

$$\mathcal{B}[\![\mathbf{B}_0 \texttt{ and } \mathbf{B}_1]\!]s = \begin{cases} true, & if\ \mathcal{B}[\![\mathbf{B}_0]\!]s = true\ and\ \mathcal{B}[\![\mathbf{B}_1]\!]s = true \\ false, & otherwise \end{cases}$$

$$\mathcal{B}[\![\mathbf{N}_0 < \mathbf{N}_1]\!]s = \begin{cases} true, & if\ \mathcal{N}[\![\mathbf{N}_0]\!]s < \mathcal{N}[\![\mathbf{N}_1]\!]s \\ false, & otherwise \end{cases}$$

$$\mathcal{B}[\![\mathbf{N}_0 = \mathbf{N}_1]\!]s = \begin{cases} true, & if\ \mathcal{N}[\![\mathbf{N}_0]\!]s = \mathcal{N}[\![\mathbf{N}_1]\!]s \\ false, & otherwise \end{cases}$$

$$\mathcal{B}[\![\mathbf{B}_0 < \mathbf{B}_1]\!]s = \begin{cases} true, & if\ \mathcal{B}[\![\mathbf{B}_0]\!]s < \mathcal{B}[\![\mathbf{B}_1]\!]s \\ false, & otherwise \end{cases}$$

$$\mathcal{B}[\![\texttt{if } \mathbf{B} \texttt{ then } \mathbf{B}_0 \texttt{ else } \mathbf{B}_1]\!]s = \begin{cases} \mathcal{B}[\![\mathbf{B}_0]\!]s, & if\ \mathcal{B}[\![\mathbf{B}]\!]s = true \\ \mathcal{B}[\![\mathbf{B}_1]\!]s, & otherwise \end{cases}$$

$$\mathcal{B}[\![(\mathbf{B})]\!]s = \mathcal{B}[\![\mathbf{B}]\!]s$$

Η συνάρτηση $\mathcal{N}[\![\cdot]\!]$ ορίζεται ως εξής:

$$\mathcal{N}[\![\texttt{0}]\!]s = 0$$

$$\mathcal{N}[\![\texttt{succ } \mathbf{N}]\!]s = \mathcal{N}[\![\mathbf{N}]\!]s + 1$$

$$\mathcal{N}[\![\texttt{if } \mathbf{B} \texttt{ then } \mathbf{N}_0 \texttt{ else } \mathbf{N}_1]\!]s = \begin{cases} \mathcal{N}[\![\mathbf{N}_0]\!]s, & if\ \mathcal{B}[\![\mathbf{B}]\!]s = true \\ \mathcal{N}[\![\mathbf{N}_1]\!]s, & otherwise \end{cases}$$

$$\mathcal{N}[\![(\mathbf{N})]\!]s = \mathcal{N}[\![\mathbf{N}]\!]s$$

Τέλος, η σημασιολογία των εντολών δίνεται από τη συνάρτηση $\mathcal{N}[\![\cdot]\!]$ η οποία ορίζεται ως εξής:

$$\begin{array}{lcl}
\mathcal{C}[\![\texttt{skip}]\!]s & = & s \\
\mathcal{C}[\![\mathbf{C}_0;\mathbf{C}_1]\!]s & = & \mathcal{C}[\![\mathbf{C}_1]\!](\mathcal{C}[\![\mathbf{C}_0]\!]s) \\
\mathcal{C}[\![\texttt{for}\ \mathbf{N}\ \texttt{do}\ \mathbf{C}]\!]s & = & \mathcal{C}[\![\mathbf{C}]\!]^n(s),\ where\ n = \mathcal{N}[\![N]\!]s \\
\mathcal{C}[\![\texttt{if}\ \mathbf{B}\ \texttt{then}\ \mathbf{C}_0\ \texttt{else}\ \mathbf{C}_1]\!]s & = & \begin{cases} \mathcal{C}[\![\mathbf{C}_0]\!]s, & if\ \mathcal{B}[\![\mathbf{B}]\!]s = true \\ \mathcal{C}[\![\mathbf{C}_1]\!]s, & otherwise \end{cases} \\
\mathcal{C}[\![(\mathbf{C})]\!]s & = & \mathcal{C}[\![\mathbf{C}]\!]s
\end{array}$$

2.2.3 Παρατηρήσεις

Με βάση την σημασιολογία που ορίσαμε παραπάνω, μπορούμε να αποδείξουμε διάφορες ιδιότητες που ισχύουν γιά τα δομικά στοιχεία της γλώσσας. Μια τεχνική που χρησιμοποιείται πολύ συχνά για το σκοπό αυτό, είναι η λεγόμενη *δομική επαγωγή (structural induction)*. Η τεχνική αυτή είναι ουσιαστικά μια εφαρμογή της συνηθισμένης επαγωγής στην συντακτική δομή των εκφράσεων μιας γλώσσας (πιό συγκεκριμένα, στο βάθος του συντακτικού δέντρου των εκφράσεων της γλώσσας). Μια ιδιότητα αποδεικνύεται:

- Για κάθε ένα από τα πρωτόγονα στοιχεία του συντακτικού της γλώσσας.
- Για κάθε σύνθετη έκφραση της γλώσσας, με βάση την υπόθεση οτι οι εκφράσεις που την αποτελούν έχουν την ιδιότητα.

Παράδειγμα 2.1 Με τη χρήση δομικής επαγωγής μπορεί να δειχτεί ότι για τη γλώσσα που έχουμε ορίσει (η οποία δεν περιέχει εντολές ανάθεσης) ισχύει:

$$\mathcal{C}[\![\mathbf{C}]\!]s = s$$

για κάθε εντολή $\mathbf{C}$ και για κάθε κάθε κατάσταση s. Για την επαγωγική βάση αρκεί να εξετάσουμε την περίπτωση $\mathbf{C} = \texttt{skip}$, για την οποία προφανώς το ζητούμενο ισχύει καθώς $\mathcal{C}[\![\texttt{skip}]\!]s = s$. Μένει να δείξουμε οτι το ζητούμενο ισχύει γιά τις πιό σύνθετες εντολές, με βάση την υπόθεση ότι ισχύει γιά όλες τις υποεντολές που τις συνθέτουν. Αυτό απαιτεί να εξετάσουμε τις τέσσερις περιπτώσεις μιας σύνθετης εντολής (δηλαδή τη σύνθεση εντολών, το `for`, το `if` και την παρενθετοποίηση). Θα δείξουμε την περίπτωση της σύνθεσης:

$$\begin{array}{lcll}
\mathcal{C}[\![\mathbf{C}_0;\mathbf{C}_1]\!]s & = & \mathcal{C}[\![\mathbf{C}_1]\!](\mathcal{C}[\![\mathbf{C}_0]\!]s) & (Semantics) \\
 & = & \mathcal{C}[\![\mathbf{C}_1]\!](s) & (Hypothesis) \\
 & = & s & (Hypothesis)
\end{array}$$

Οι υπόλοιπες περιπτώσεις σύνθετων εντολών μπορούν να δειχτούν με ανάλογο τρόπο. ∎

2.3 Εντολές Ανάθεσης

Η πιό σημαντική ίσως έλλειψη της γλώσσας που ορίσαμε στις προηγούμενες παραγράφους είναι η απουσία μεθόδων γιά να προσπελάσει και να αλλάξει κανείς την κατάσταση (δηλαδή τη μνήμη). Γιά το σκοπό αυτό πρέπει αρχικά να οριστεί ένα σύνολο *ονομάτων μεταβλητών (variable-identifiers)*. Το σύνολο αυτό το συμβολίζουμε με Var, και περιέχει στοιχεία όπως $\mathtt{i}$, $\mathtt{j}$, κλπ. Οι μεταβλητές που ανήκουν στο Var χωρίζονται σε δύο υποσύνολα: τις λογικές (boolean) και τις αριθμητικές (natural). Για το σκοπό αυτό υποθέτουμε την ύπαρξη μιας συνάρτησης π, η οποία προσδιορίζει τον τύπο της κάθε μεταβλητής. Για παράδειγμα, αν $\mathtt{i}$ είναι μια λογική μεταβλητή, τότε $\pi(\mathtt{i}) = boolean$, ενώ αν $\mathtt{j}$ είναι μια αριθμητική μεταβλητή, τότε $\pi(\mathtt{j}) = natural$.

Το συντακτικό της γλώσσας μπορεί να επαυξηθεί ώστε να επιτρέπει τη χρήση μεταβλητών αλλά και εντολών ανάθεσης. Πιο συγκεκριμένα, επεκτείνουμε τους κανόνες παραγωγής των εκφράσεων και των εντολών της γλώσσας, ως εξής:

$$
\begin{array}{lcl}
\mathbf{B} & ::= & \ldots \\
 & | & \mathbf{i},\ where\ \mathbf{i} \in Var,\ \pi(\mathbf{i}) = boolean \\
 & & \\
\mathbf{N} & ::= & \ldots \\
 & | & \mathbf{i},\ where\ \mathbf{i} \in Var,\ \pi(\mathbf{i}) = natural \\
 & & \\
\mathbf{C} & ::= & \ldots \\
 & | & \mathbf{i} := \mathbf{B},\ \pi(\mathbf{i}) = boolean \\
 & | & \mathbf{i} := \mathbf{N},\ \pi(\mathbf{i}) = natural
\end{array}
$$

Για να περιγράψουμε τη σημασιολογία των εντολών ανάθεσης καθώς και των μεταβλητών (οι οποίες μπορούν να χρησιμοποιηθούν στις εκφράσεις της γλώσσας), χρειάζεται κατ'αρχήν να ορίσουμε μαθηματικά το σύνολο των καταστάσεων S. Χρειάζεται δηλαδή να ορίσουμε μαθηματικά την έννοια της μνήμης του υπολογιστή. Διαισθητικά, μια κατάσταση $s \in S$ είναι μια συνάρτηση η οποία αντιστοιχίζει σε κάθε μεταβλητή της γλώσσας μια τιμή. Η συνάρτηση αυτή s πρέπει να "σέβεται" τον τύπο της κάθε μεταβλητής. Ετσι η s αντιστοιχίζει σε λογικές μεταβλητές λογικές τιμές, ενώ στις αριθμητικές μεταβλητές εκχωρεί αριθμητικές τιμές. Με βάση τις παρατηρήσεις αυτές, το σύνολο S των καταστάσεων μπορεί να οριστεί ως εξής:

$$
\begin{array}{ll}
S = \{ & s : Var \to Nat \cup \{true, false\} \mid \\
 & s(\mathbf{i}) \in Nat \quad if \quad \pi(\mathbf{i}) = natural \quad and \\
 & s(\mathbf{i}) \in \{true, false\} \quad if \quad \pi(\mathbf{i}) = boolean \quad \}
\end{array}
$$

Η σημασιολογία της γλώσσας μπορεί τώρα να επεκταθεί ώστε να καλύψει και την περίπτωση των μεταβλητών και των εντολών ανάθεσης. Στην δεύτερη από τις παρακάτω εξισώσεις, χρησιμοποιούμε το σύμβολο $\mathbf{E}$ για να αναπαραστήσουμε μια οποιαδήποτε έκφραση της γλώσσας (λογική είτε αριθμητική).

$$\begin{array}{lcl} [\![\mathbf{i}]\!]s & = & s(\mathbf{i}) \\ [\![\mathbf{i} := \mathbf{E}]\!]s & = & (s \mid \mathbf{i} \mapsto [\![\mathbf{E}]\!]s) \end{array}$$

όπου $(s \mid \mathbf{i} \mapsto [\![\mathbf{E}]\!]s)$ είναι μια νέα κατάσταση s' έτσι ώστε $s'(\mathbf{i}) = [\![\mathbf{E}]\!]s$ και για κάθε άλλη μεταβλητή $\mathbf{i}' \neq \mathbf{i}$, $s'(\mathbf{i}') = s(\mathbf{i})$. Διαισθητικά, η δεύτερη εξίσωση λέει ότι η εκτέλεση μιας εντολής ανάθεσης σε μια δεδομένη κατάσταση έχει ως αποτέλεσμα την παραγωγή μιας "νέας" κατάστασης στην οποία έχει αλλάξει μόνο η τιμή της μεταβλητής η οποία σχετίζεται με την ανάθεση. Η νέα τιμή της μεταβλητής αυτής είναι η τιμή του δεξιού μέλους της εντολής ανάθεσης υπολογισμένη στην "παλαιά" κατάσταση.

Με τη χρήση των παραπάνω εξισώσεων μπορεί κανείς να αποδείξει την ισοδυναμία διαφόρων εντολών της γλώσσας. Για παράδειγμα, η εντολή:

$$\mathbf{i} := (\texttt{if } \mathbf{B} \texttt{ then } \mathbf{E}_0 \texttt{ else } \mathbf{E}_1)$$

και η εντολή

$$\texttt{if } \mathbf{B} \texttt{ then } (\mathbf{i} := \mathbf{E}_0) \texttt{ else } (\mathbf{i} := \mathbf{E}_1)$$

μπορούν εύκολα να αποδειχτούν ισοδύναμες με τη χρήση των παραπάνω σημασιολογικών εξισώσεων.

2.4 Αόριστες Επαναλήψεις

Ο βρόχος `for` είναι συχνά ανεπαρκής διότι ο αριθμός των επαναλήψεων καθορίζεται πριν αρχίσει η εκτέλεση των εντολών. Για το λόγο αυτό επεκτείνουμε τη γλώσσα έτσι ώστε να υποστηρίζει *αόριστες επαναλήψεις (indefinite iterations)* με τη μορφή του βρόχου `while`:

$$\begin{array}{rcl} \mathbf{C} & ::= & \ldots \\ & | & \texttt{while } \mathbf{B} \texttt{ do } \mathbf{C} \end{array}$$

Η έκφραση $\mathbf{B}$ υπολογίζεται και αν είναι αληθής εκτελείται η εντολή $\mathbf{C}$. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να γίνει ψευδής η τιμή του $\mathbf{B}$.

Υπάρχει βέβαια το ενδεχόμενο η τιμή του $\mathbf{B}$ να μη γίνει ποτέ ψευδής, και στην περίπτωση αυτή η εκτέλεση του προγράμματος δεν τερματίζεται. Το ενδεχόμενο αυτό δεν υπήρχε στη γλώσσα που είχαμε ορίσει μέχρι τώρα, και μας αναγκάζει να επανεξετάσουμε τι ακριβώς είναι το νόημα μιας εντολής (το οποίο μέχρι τώρα ήταν μια συνάρτηση από το σύνολο καταστάσεων S στο S).

Λαμβάνοντας υπόψιν την περίπτωση του μη-τερματισμού, συμπεραίνουμε ότι το νόημα μιας εντολής θα πρέπει να είναι τώρα μια *μερική συνάρτηση (partial function)* από το S στο S. (Μια μερική συνάρτηση από ένα σύνολο A σε ένα σύνολο B είναι μια συνάρτηση που αντιστοιχίζει στοιχεία του A σε στοιχεία του B, χωρίς τον περιορισμό οτι όλα τα στοιχεία του A αντιστοιχίζονται σε στοιχεία του B). Με βάση τα παραπάνω, η τιμή του $[\![\mathbf{C}]\!]s$ δεν θα είναι ορισμένη αν και μόνο αν η εκτέλεση της εντολής **C** ξεκινώντας από την αρχική κατάσταση s, δεν τερματίζεται. Δεν χρειάζονται αλλαγές στις σημασιολογικές εξισώσεις της γλώσσας που έχουμε δώσει στις προηγούμενες παραγράφους, εκτός του ότι τώρα θα έχουμε σύνθεση μερικών συναρτήσεων στις σημασιολογικές εξισώσεις για το ";" και για το βρόχο `while`.

Αυτό που απομένει είναι να δώσουμε σημασιολογία στο βρόχο `while` η οποία να υπακούει στον κανόνα της συνθεσιμότητας. Εχουμε δεί οτι μια εξίσωση της μορφής

$$\mathcal{C}[\![\texttt{while}\ \mathbf{B}\ \texttt{do}\ \mathbf{C}]\!] = \mathcal{C}[\![\texttt{if}\ \mathbf{B}\ \texttt{then}\ (\mathbf{C}; \texttt{while}\ \mathbf{B}\ \texttt{do}\ \mathbf{C})]\!]$$

δεν επαρκεί, διότι δεν υπακούει στον κανόνα της συνθεσιμότητας. Ακόμη και αν αναλύσουμε το δεξί μέρος χρησιμοποιώντας τη σημασιολογία του `if`, παίρνουμε:

$$\mathcal{C}[\![\texttt{while}\ \mathbf{B}\ \texttt{do}\ \mathbf{C}]\!]s \;\;=\;\; \begin{cases} \mathcal{C}[\![\texttt{while}\ \mathbf{B}\ \texttt{do}\ \mathbf{C}]\!](\mathcal{C}[\![\mathbf{C}]\!]s), & if\ \mathcal{B}[\![\mathbf{B}]\!]s = true \\ s, & if\ \mathcal{B}[\![\mathbf{B}]\!]s = false \end{cases}$$

Και η παραπάνω εξίσωση δεν υπακούει τον κανόνα της συνθεσιμότητας, εκφράζει όμως μια ιδιότητα του `while` η οποία είναι επιθυμητή.

Μπορούμε να απλοποιήσουμε τη μορφή της παραπάνω εξίσωσης αντικαθιστώντας με c το νόημα της εντολής `while`.

$$c(s) \;\;=\;\; \begin{cases} c(\mathcal{C}[\![\mathbf{C}]\!]s), & if\ \mathcal{B}[\![\mathbf{B}]\!]s = true \\ s, & if\ \mathcal{B}[\![\mathbf{B}]\!]s = false \end{cases}$$

Βλέπουμε ότι έχουμε μια εξίσωση με άγνωστο το c, την οποία άμα λύσουμε θα πάρουμε την επιθυμητή σημασιολογία του βρόχου `while`. Η τεχνική που χρησιμοποιούμε για να την επιλύσουμε, μπορεί διαισθητικά να περιγραφεί ως εξής: αρχικά δίνουμε μια "κακή" εκτίμηση c_0, την οποία μετά "βελτιώνουμε" σταδιακά με αναδρομικό τρόπο (παίρνοντας εκτιμήσεις c_1, $c_2 \ldots$). Το όριο της διαδικασίας αυτής θα είναι η επιθυμητή συνάρτηση c. Η διαδικασία αυτή μοιάζει με κάποιες τεχνικές για επίλυση συστημάτων στην αριθμητική ανάλυση, όπου ξεκινάμε με κάποια εκτίμηση για τους αγνώστους των εξισώσεων και συγκλίνουμε σταδιακά στην πραγματική λύση του συστήματος.

Στην περίπτωση μας, ξεκινάμε με την μερική συνάρτηση c_0 η οποία είναι μη-ορισμένη για όλα τα $s \in S$. Με λίγα λόγια, η c_0 έχει για γράφημα της το κενό σύνολο. Για κάθε φυσικό αριθμό i, ορίζουμε:

$$c_{i+1}(s) \;\;=\;\; \begin{cases} c_i(\mathcal{C}[\![\mathbf{C}]\!]s), & if\ \mathcal{B}[\![\mathbf{B}]\!]s = true \\ s, & if\ \mathcal{B}[\![\mathbf{B}]\!]s = false \end{cases}$$

Μπορεί να δειχθεί (με μαθηματική επαγωγή), ότι το γράφημα της c_i είναι υποσύνολο του γραφήματος της c_{i+1}, για κάθε i. Για το λόγο αυτό, το όριο c_∞ της ακολουθίας c_i είναι καλά ορισμένο. Πιό συγκεκριμένα, το c_∞ μπορεί να οριστεί ως εξής: Εστω $s \in S$. Αν υπάρχει

$i \in N$ τέτοιο ώστε το $c_i(s)$ να είναι ορισμένο, τότε $c_\infty(s) = c_i(s)$. Διαφορετικά, το $c_\infty(s)$ είναι μη-ορισμένο.

Μπορούμε τώρα να ορίσουμε την σημασιολογία του βρόχου `while`:

$$\mathcal{C}[\![\texttt{while } \mathbf{B} \texttt{ do } \mathbf{C}]\!] \quad = \quad c_\infty$$

Πρέπει να σημειώσουμε οτι η σημασιολογία που δώσαμε παραπάνω για τον βρόχο `while`, υπακούει στον κανόνα της συνθεσιμότητας, διότι το νόημα του `while` ορίζεται ως συνάρτηση των $\mathcal{B}[\![\mathbf{B}]\!]$ και $\mathcal{C}[\![\mathbf{C}]\!]$ (με πιο πολύπλοκο βέβαια τρόπο από τις περιπτώσεις που είχαμε συναντήσει έως τώρα).

Αυτό που μένει να δειχτεί είναι οτι η σημασιολογία που ορίσαμε για το βρόχο `while` ικανοποιεί τη βασική ιδιότητα που αναφέραμε προηγουμένως, ότι δηλαδή:

Θεώρημα 2.1 *Για όλες τις λογικές εκφράσεις* $\mathbf{B}$ *και για όλες τις εντολές* $\mathbf{C}$*,*

$$\mathcal{C}[\![\texttt{while } \mathbf{B} \texttt{ do } \mathbf{C}]\!] = \mathcal{C}[\![\texttt{if } \mathbf{B} \texttt{ then } (\mathbf{C}; \texttt{while } \mathbf{B} \texttt{ do } \mathbf{C})]\!]$$

Απόδειξη (Μεθοδολογία) Η απόδειξη του παραπάνω θεωρήματος μπορεί να δοθεί χρησιμοποιώντας τον ορισμό του c_∞, και δείχνοντας ότι για κάθε $s \in S$ τα δύο μέλη της παραπάνω εξίσωσης είναι ή και τα δύο μη-ορισμένα ή και τα δύο ορισμένα και ίσα μεταξύ τους. ■

Η μεθοδολογία που ακολουθήσαμε για να ορίσουμε τη σημασιολογία του `while` μπορεί να χρησιμοποιηθεί και σε άλλες διαφορετικές δομές βρόχου (όπως για παράδειγμα είναι ο βρόχος `repeat`).

Η τεχνική για τον ορισμό του c_∞ μπορεί να παρουσιαστεί με ένα παράδειγμα:

Παράδειγμα 2.2 Ας υποθέσουμε οτι προσθέτουμε έναν τελεστή μείωσης κατά ένα (`prev`) στη γλώσσα, και ας θεωρήσουμε την ακόλουθη εντολή:

$$\texttt{while (n} > \texttt{0) do (n} := \texttt{prev n)}$$

Διαισθητικά, η εντολή αυτή πρέπει να είναι ισοδύναμη με την εντολή `n:= 0`. Θα δείξουμε ότι η ισοδυναμία αυτή ισχύει, υπολογίζοντας την τιμή του c_∞. Αρχίζουμε από το $c_0(s)$ το οποίο είναι μη-ορισμένο για όλα τα $s \in S$. Χρησιμοποιώντας τον ορισμό του c_{i+1} ως συνάρτηση του c_i, έχουμε:

$$c_1(s) \quad = \quad \begin{cases} \text{μη ορισμένο}, & if\ \mathcal{N}[\![\mathtt{n}]\!]s > 0 \\ s, & if\ \mathcal{N}[\![\mathtt{n}]\!]s = 0 \end{cases}$$

Μπορούμε τώρα να υπολογίσουμε το c_2:

$$c_2(s) = \begin{cases} c_1([\![\mathtt{n} := \mathtt{prev\ n}]\!]s), & if\ \mathcal{N}[\![\mathtt{n}]\!]s > 0 \\ s, & if\ \mathcal{N}[\![\mathtt{n}]\!]s = 0 \end{cases}$$

$$= \begin{cases} \text{μη ορισμένο}, & if\ (\mathcal{N}[\![\mathtt{n}]\!]s > 0)\ and\ (\mathcal{N}[\![\mathtt{n}]\!]([\![\mathtt{n} := \mathtt{prev\ n}]\!]s) > 0) \\ [\![\mathtt{n} := \mathtt{prev\ n}]\!]s, & if\ (\mathcal{N}[\![\mathtt{n}]\!]s > 0)\ and\ (\mathcal{N}[\![\mathtt{n}]\!]([\![\mathtt{n} := \mathtt{prev\ n}]\!]s) = 0) \\ s, & if\ \mathcal{N}[\![\mathtt{n}]\!]s = 0 \end{cases}$$

$$= \begin{cases} \text{μη ορισμένο}, & if\ (\mathcal{N}[\![\mathtt{n}]\!]s > 0)\ and\ (\mathcal{N}[\![\mathtt{n}]\!]s \geq 2) \\ (s \mid \mathtt{n} \mapsto \mathcal{N}[\![\mathtt{n}]\!]s - 1), & if\ (\mathcal{N}[\![\mathtt{n}]\!]s > 0)\ and\ (\mathcal{N}[\![\mathtt{n}]\!]s = 1) \\ s, & if\ \mathcal{N}[\![\mathtt{n}]\!]s = 0 \end{cases}$$

Κάνοντας τις πράξεις στην παραπάνω παράσταση και απλοποιώντας τις λογικές συνθήκες έχουμε τελικά:

$$c_2(s) = \begin{cases} \text{μη ορισμένο}, & if\ (\mathcal{N}[\![\mathtt{n}]\!]s \geq 2) \\ (s \mid \mathtt{n} \mapsto 0), & if\ (\mathcal{N}[\![\mathtt{n}]\!]s = 1) \\ (s \mid \mathtt{n} \mapsto 0), & if\ (\mathcal{N}[\![\mathtt{n}]\!]s = 0) \end{cases}$$

Γενικώτερα, μπορούμε να δείξουμε ότι:

$$c_i(s) = \begin{cases} \text{μη ορισμένο}, & if\ \mathcal{N}[\![\mathtt{n}]\!]s \geq i \\ (s \mid \mathtt{n} \mapsto 0), & if\ \mathcal{N}[\![\mathtt{n}]\!]s < i \end{cases}$$

Επομένως, όπως το i τείνει στο άπειρο, το c_i τείνει στο c_∞, με $c_\infty(s) = (s \mid \mathtt{n} \mapsto 0)$. Με λίγα λόγια, η συγκεκριμένη εντολή `while` είναι ισοδύναμη με την εντολή ανάθεσης `n:= 0`. Αξίζει να σημειώσουμε ότι στη συγκεκριμένη περίπτωση η c_∞ είναι *ολική (total)* συνάρτηση (είναι δηλαδή ορισμένη για όλα τα $s \in S$).

■

Μια διαισθητική ερμηνεία του c_i είναι ότι αποτελεί μια "υλοποίηση" του βρόχου `while` η οποία δουλεύει σωστά για όλες εκείνες τις περιπτώσεις στις οποίες ο βρόχος τερματίζει σε λιγότερες από i επαναλήψεις, ενώ αυτοκαταστρέφεται σε όλες τις άλλες περιπτώσεις.

2.5 Ασκήσεις

1. Είναι οι εντολές

$$\texttt{for}\ (\texttt{succ}\ \mathbf{N})\ \texttt{do}\ \mathbf{C}$$

   και

$$\mathbf{C}; (\texttt{for}\ \mathbf{N}\ \texttt{do}\ \mathbf{C})$$

   σημασιολογικά ισοδύναμες; Αν ναι, αποδείξτε το. Αν όχι, δώστε αντιπαράδειγμα και εξετάστε τις συνθήκες κάτω από τις οποίες ισχύει η ισοδυναμία.

2. Ας υποθέσουμε ότι θέλουμε να επεκτείνουμε την απλή προστακτική γλώσσα ώστε να υποστηρίζει ταυτόχρονες εντολές ανάθεσης για δύο μεταβλητές. Επιτρέπουμε δηλαδή αναθέσεις της μορφής:

$$\texttt{x}, \texttt{y} := \mathbf{E}_1, \mathbf{E}_2$$

   Υπάρχουν δύο (τουλάχιστον) προσεγγίσεις που μπορεί κανείς να ακολουθήσει για να δώσει νόημα σε τέτοιες εντολές:

   - Οι τιμές των $\mathbf{E}_1$ και $\mathbf{E}_2$ υπολογίζονται και τα αποτελέσματα "τοποθετούνται" στα `x,y`.
   - Η τιμή του $\mathbf{E}_1$ υπολογίζεται και "τοποθετείται" στο `x`, και μετά υπολογίζεται η τιμή του $\mathbf{E}_2$ και "τοποθετείται" στο `y`.

   Εξηγήστε γιατί οι δύο αυτές προσεγγίσεις είναι διαφορετικές και δείξτε οτι η ενδεικτική σημασιολογία μπορεί να χρησιμοποιηθεί για να τις περιγράψουμε με σαφήνεια.

3. Εστω η εντολή

$$\texttt{while}\ (\texttt{i} > \texttt{j})\ \texttt{do}\ (\texttt{i} := \texttt{prev i}; \texttt{j} := \texttt{succ j})$$

   Υπολογίστε αναλυτικά τα c_0, c_1, c_2 και δώστε και το όριο της ακολουθίας των c_i (δηλαδή το c_∞ που είναι και το νόημα του παραπάνω προγράμματος).