
Ανάλυση της Ορθότητας Προγραμμάτων

(HR Κεφάλαιο 4)

Στην ενότητα αυτή θα μελετηθούν τα εξής θέματα:

Η διαδικαστική γλώσσα προγραμματισμού WHILE

Τριάδες Hoare

Μερική και Ολική Ορθότητα Προγραμμάτων

Κανόνες Απόδειξης Μερικής Ορθότητας

Κανόνες Απόδειξης Ολικής Ορθότητας

Ανάλυση της ορθότητας προγραμμάτων

- Αφορά στην απόδειξη ότι ένα πρόγραμμα ικανοποιεί τις προδιαγραφές του.
- Γιατί όχι μοντελοέλεγχος;
 - Ο μοντελοέλεγχος είναι εφαρμόσιμος σε συστήματα με **πολύπλοκη ροή** (συντρέχουσες διεργασίες) και στην **απουσία σύνθετης επεξεργασίας δεδομένων**. Υποθέτει την ύπαρξη μοντέλων με **πεπερασμένο αριθμό καταστάσεων**.
 - Τα σειριακά προγράμματα (συνήθως) εκτελούν **σύνθετη επεξεργασία δεδομένων** και έχουν **απλή ροή** (σειριακή) και **μη-πεπερασμένο αριθμό καταστάσεων**.
- Στην ενότητα αυτή θα μελετήσουμε την τεχνική των Floyd-Hoare για απόδειξη της ορθότητας τέτοιων προγραμμάτων.

Η Τεχνική Floyd-Hoare

- Βασίζεται στην κατασκευή αποδείξεων
 - Δεν εκτελείται εξαντλητικός έλεγχος κάθε κατάστασης του συστήματος (μη πεπερασμένος αριθμός καταστάσεων!). Η απόδειξη κατασκευάζεται με τη χρήση κανόνων (όπως και στον Προτασιακό Λογισμό).
- Είναι ημι-αυτοματοποιημένη:
 - Δεν υπάρχει αλγόριθμος για την τεχνική (όπως στον Μοντελοέλεγχο ή στη Μέθοδο Επίλυσης). Τα βήματα μπορούν να ελεγχθούν από μηχανή και κάποια μπορούν να εκτελεστούν αυτόματα, άλλα όμως απαιτούν ανθρώπινη συμμετοχή.
- Είναι προσανατολισμένη σε στόχους:
 - Εντοπίζουμε μέρη της προδιαγραφής των οποίων ελέγχουμε την ορθότητα.
- Είναι εφαρμόσιμη σε συγκεκριμένο πεδίο εφαρμογών:
 - Σειριακά προγράμματα τα οποία μετατρέπουν τα δεδομένα εισόδου τους σε δεδομένα εξόδου.

Κίνητρα

- Παρά την όποια δυσκολία στη χρήση της, μια μέθοδος ανάλυσης της ορθότητας προγραμμάτων:
 - Εξαναγκάζει τη σαφή διατύπωση των προδιαγραφών του προγράμματος.
 - Συντείνει στην αποδοτικότερη ανάπτυξη και συντήρηση λογισμικού (χρόνος/κόστος). Αντιμετωπίζει τα λάθη από τη φάση της σχεδίασης.
 - Προσφέρει τη δυνατότητα για ασφαλή επαναχρησιμοποίηση κώδικα.
 - Παρέχει εγγυήσεις οι οποίες θεωρούνται απαραίτητες κατά τη δημιουργία συστημάτων κρίσιμης ασφάλειας.

Η Γλώσσα WHILE

Η γλώσσας που θα μελετήσουμε έχει την πιο κάτω σύνταξη.

- Εντολές:

$C ::= x:=E \mid C;C \mid \text{if } B \{C\} \text{ else } \{C\} \mid \text{while } B \{C\}$

- Αριθμητικές Εκφράσεις:

$E ::= n \mid x \mid -E \mid (E+E) \mid (E - E) \mid (E*E)$

όπου το n παίρνει τιμές από τους ακέραιους και το x είναι το όνομα οποιασδήποτε μεταβλητής.

- Λογικές Εκφράσεις:

$B ::= \text{true} \mid \text{false} \mid (!B) \mid (B\&B) \mid (B||B) \mid (E<E)$

Παράδειγμα

- Το πιο κάτω πρόγραμμα υπολογίζει τον n-ιοστό παραγοντικό αριθμό.

```
y = 1;
```

```
z = 0;
```

```
while (z != n) {
```

```
    z := z + 1;
```

```
    y := y * z;
```

```
}
```



$(z < n \ || \ n < z)$

Τριάδες Hoare

- Συμβολισμός:

$$\{\varphi\} C \{\psi\}$$

όπου C το πρόγραμμα υπό μελέτη και φ, ψ συνθήκες που σχετίζονται με τις μεταβλητές που χρησιμοποιεί το C .

- Ερμηνεία:

Κάθε φορά που το πρόγραμμα C εκτελείται σε μια κατάσταση που ικανοποιεί τη συνθήκη φ τότε θα τερματίσει σε μια κατάσταση που ικανοποιεί τη συνθήκη ψ .

Συνθήκες

- Αποτελούν εκφράσεις μια απλής λογικής γλώσσας:
 - Περιέχουν μεταβλητές του προγράμματος, σταθερές, λογικούς τελεστές, κλπ.
 - Όταν προηγούνται μιας εντολής περιγράφουν τους περιορισμούς που πρέπει να ισχύουν κατά την εκτέλεση της εντολής.
προσυνθήκες – preconditions
 - Όταν έπονται μιας εντολής, περιγράφουν τους περιορισμούς που πρέπει να ισχύουν μετά από την εκτέλεση της εντολής.
μετασυνθήκες – postconditions
- Εκφράσεις της μορφής $\{\varphi\} C \{\psi\}$ (τριάδες Hoare) ονομάζονται *προδιαγραφές*.

Παράδειγμα (1)

- Οι προδιαγραφές

$$\{ X = 1 \} X := X + 1 \{ X = 2 \}$$
$$\{ X = 1 \} Y := X \{ Y = 1 \}$$

είναι αληθείς.

- Η προδιαγραφή

$$\{ X = 1 \} Y := X + 1 \{ X = 2 \}$$

είναι ψευδής.

Παράδειγμα (2)

- Η πιο κάτω προδιαγραφή είναι αληθής.

```
{ X = x ∧ Y = y } R := X ; X := Y ;  
Y := R { X = y ∧ Y = x }
```

Οι μεταβλητές x και y που εμφανίζονται στην προ-συνθήκη και όχι στο πρόγραμμα ονομάζονται **βοηθητικές μεταβλητές**. Σκοπός τους είναι να δώσουν αρχικές τιμές στις μεταβλητές.

Μερική και Ολική Ορθότητα

- **Μερική Ορθότητα:** Η προδιαγραφή $\{\varphi\} C \{\psi\}$ είναι αληθής υπό την έννοια της μερικής ορθότητας αν κάθε φορά που το C ξεκινά σε μια κατάσταση που ικανοποιεί την προσυνθήκη φ και τερματίζει τότε η κατάσταση τερματισμού ικανοποιεί τη μετασυνθήκη ψ . Γράφουμε

$$\models_{\text{par}} \{\varphi\} C \{\psi\}$$

- Το πρόγραμμα

```
while true { x := 0 }
```

ικανοποιεί οποιαδήποτε προδιαγραφή μερικής ορθότητας εφόσον δεν τερματίζει.

- **Ολική Ορθότητα:** Η προδιαγραφή $\{\varphi\} C \{\psi\}$ είναι αληθής υπό την έννοια της ολικής ορθότητας αν κάθε φορά που το C ξεκινά σε μια κατάσταση που ικανοποιεί την προσυνθήκη φ τότε τερματίζει και η κατάσταση τερματισμού ικανοποιεί τη μετασυνθήκη ψ . Γράφουμε

$$\models_{\text{tot}} \{\varphi\} C \{\psi\}$$

Τεχνική Floyd-Hoare

- Δημιουργία τυπικών αποδείξεων για προδιαγραφές προγραμμάτων.
- Χρησιμοποιεί
 - Αξιώματα και κανόνες εξαγωγής συμπερασμάτων που παρέχει η τεχνική
 - Θεωρήματα κλασικών μαθηματικών.

Αξίωμα Εντολής Ανάθεσης

$$\{\varphi[E/x]\} \quad x := E \quad \{\varphi\}$$

- $\varphi[E/x]$ είναι το αποτέλεσμα της αντικατάστασης όλων των εμφανίσεων της μεταβλητής x από την έκφραση E στην φ .
- Παράδειγμα:

$$\{y = 2\} \quad x := y \quad \{x = 2\}$$

$$\{x+1 = n+1\} \quad x := x+1 \quad \{x = n+1\}$$

$$\{y = 4\} \quad x := y \quad \{x = 4\}$$

Συζήτηση Αξιώματος

- Προσοχή: Ο κανόνας δεν θα μπορούσε να είναι

$$\{\varphi\} \ x := E \ \{\varphi[E/x]\}$$

αφού αυτό θα έδινε προδιαγραφές όπως: $\{x=6\} \ x := 5 \ \{5=6\}$.

- Το Αξίωμα της Εντολής Ανάθεσης εφαρμόζεται πιο εύκολα στην αντίστροφη πορεία μιας απόδειξης, δηλαδή, γνωρίζοντας την μετασυνθήκη μιας εντολής, παράγουμε την προσυνθήκη της εντολής.
- Εφόσον η αντίστροφη εφαρμογή του κανόνα ισοδυναμεί με την εφαρμογή μιας αντικατάστασης, υλοποίηση του κανόνα για αυτοματοποίηση μιας απόδειξης είναι ξεκάθαρη.

Κανόνας Ενδυνάμωσης της Προσυνθήκης

$$\frac{\varphi \rightarrow \eta \quad \{ \eta \} C \{ \psi \}}{\{ \varphi \} C \{ \psi \}}$$

- **Εξήγηση Κανόνα:** Αν το φ συνεπάγεται το η και η προδιαγραφή $\{ \eta \} C \{ \psi \}$ είναι αληθής τότε και η προδιαγραφή $\{ \varphi \} C \{ \psi \}$ είναι αληθής.

Παράδειγμα

- Έστω

$$\{X+1 = n+1\} \quad X := X+1 \quad \{X=n+1\}$$

Αφού το $(X+1 = n+1)$ συνεπάγεται ότι

$$(X = n)$$

βάσει του κανόνα της Ενδυνάμωσης της Προσυνθήκης συμπεραίνουμε ότι

$$\{X = n\} \quad X := X+1 \quad \{X=n+1\}$$

Κανόνας Αποδυνάμωσης της Μετασυνθήκης

$$\frac{\eta \rightarrow \psi \quad \{\varphi\} C \{\eta\}}{\{\varphi\} C \{\psi\}}$$

- **Εξήγηση Κανόνα:** Αν το η συνεπάγεται το ψ και η προδιαγραφή $\{\varphi\} C \{\eta\}$ είναι αληθής τότε και η προδιαγραφή $\{\varphi\} C \{\psi\}$ είναι αληθής.
- Οι Κανόνες Ενδυνάμωσης και Αποδυνάμωσης ονομάζονται και Κανόνες Συνεπαγωγής.

Παράδειγμα

- Να δείξετε ότι $\{y=5\} \quad x:=y+1 \quad \{x>0\}$

- Απόδειξη*:

Κατ' αρχή, παρατηρούμε ότι από το Αξίωμα Εντολής Ανάθεσης η προδιαγραφή

$$\{y+1=6\} \quad x:=y+1 \quad \{x=6\}$$

είναι αληθής.

Αφού $(y=5) \rightarrow (y+1=6)$, από τον Κανόνα Ενδυνάμωσης Προσυνθήκης προκύπτει ότι:

$$\{y=5\} \quad x:=y+1 \quad \{x=6\}$$

Αφού $(x=6) \rightarrow (x>0)$ με τη χρήση του Κανόνα Αποδυνάμωσης Μετασυνθήκης

$$\{y=5\} \quad x:=y+1 \quad \{x>0\}$$

και η απόδειξη ολοκληρώθηκε.

*Σημείωση: Υπάρχουν και άλλοι τρόποι για να γίνει η απόδειξη.

Κανόνας Ακολουθίας Εντολών

$$\frac{\{\varphi\} C_1 \{η\} \quad \{η\} C_2 \{\psi\}}{\{\varphi\} C_1; C_2 \{\psi\}}$$

- **Εξήγηση:** Για να δείξουμε την ορθότητα της προδιαγραφής $\{\varphi\} C_1; C_2 \{\psi\}$ χρειάζεται να εντοπίσουμε ενδιάμεση συνθήκη η για την οποία να ισχύει ότι $\{\varphi\} C_1 \{η\}$ και $\{η\} C_2 \{\psi\}$.

Παράδειγμα

- Αν γνωρίζουμε ότι

$$\{ X = x \wedge Y = y \} R := X \{ R = x \wedge Y = y \}$$

$$\{ R = x \wedge Y = y \} X := Y \{ R = x \wedge X = y \}$$

$$\{ R = x \wedge X = y \} Y := R \{ Y = x \wedge X = y \}$$

από τον Κανόνα Ακολουθίας Εντολών και τις δύο πρώτες προτάσεις μπορούμε να συμπεράνουμε ότι

$$\{ X = x \wedge Y = y \} R := X ; X := Y \{ R = x \wedge X = y \}$$

και στη συνέχεια ότι

$$\begin{aligned} & \{ X = x \wedge Y = y \} R := X ; \\ & X := Y ; Y := R \{ Y = x \wedge X = y \} \end{aligned}$$

Κανόνας Εντολής if

$$\frac{\{ \varphi \wedge B \} C_1 \{ \psi \} \quad \{ \varphi \wedge \neg B \} C_2 \{ \psi \}}{\{ \varphi \} \text{ if } B \text{ then } C_1 \text{ else } C_2 \{ \psi \}}$$

- **Εξήγηση:** Για να δείξουμε την ορθότητα της προδιαγραφής $\{ \varphi \} \text{ if } B \text{ then } C_1 \text{ else } C_2 \{ \psi \}$ χρειάζεται να δείξουμε ότι από τη προσυνθήκη φ προκύπτει η μετασυνθήκη ψ ανεξάρτητα από τον τρόπο με τον οποίο θα εκτελεστεί η εντολή if, δηλαδή, ανεξάρτητα από την λογική τιμή που θα λάβει η συνθήκη B της εντολής. Έτσι αποδεικνύουμε δύο επιμέρους προδιαγραφές που αντιστοιχούν στις δύο περιπτώσεις B και $\neg B$.

Παράδειγμα

- Με χρήση του κανόνα μπορούμε να δείξουμε ότι η προδιαγραφή

```
{y>1} if (x>0) then (y:= y-1) else  
      (y := y+1) {y>0}
```

είναι αληθής.

- Συγκεκριμένα δείχνουμε ότι:

```
{y>1 ∧ x>0} y:=y-1 {y>0}
```

```
{y>1 ∧ ¬x>0} y:=y+1 {y>0}
```

- Η ορθότητα των δύο προτάσεων προκύπτει μέσω του Αξιώματος της Ανάθεσης και του Κανόνα Ενδυνάμωσης Προσυνθήκης.

Κανόνας Εντολής while

$$\frac{\{\psi \wedge B\} C \{\psi\}}{\{\psi\} \text{ while } B C \{\psi \wedge \neg B\}}$$

- Ο κανόνας βασίζεται στη συνθήκη ψ η οποία ονομάζεται **αμετάβλητη συνθήκη** (invariant) διότι εξακολουθεί να ισχύει μετά από την εφαρμογή του κανόνα.
- **Εξήγηση:** Ο κανόνας λέει ότι αν η ψ είναι μια αμετάβλητη συνθήκη του σώματος μιας εντολής while (όταν ισχύει και η συνθήκη B) τότε η ψ είναι μια αμετάβλητη συνθήκη ολόκληρης της εντολής while.

Παράδειγμα

- Να αποδείξετε ότι:

```
{true}
R:= X;
Q:= 0;
while (Y<= R){
    R:= R-Y;
    Q:= Q+1;
}
{R<Y ∧ X= R + Y*Q}
```

- Ο κώδικας αυτός υπολογίζει το πηλίκο (Q) και το υπόλοιπο (R) της διαίρεσης του X από το Y

Απόδειξη

- Χρησιμοποιώντας τον Κανόνα της Ακολουθίας, θα δείξουμε ότι:
 1. $\{\text{true}\} R:= X;Q:= 0; \{X= R + Y*Q\}$ και στη συνέχεια ότι
 2. $\{X= R + Y*Q\} \text{ while } (Y \leq R)\{R:= R-Y;Q:= Q+1;\} \{R < Y \wedge X= R + Y*Q\}$
- Το πρώτο σκέλος αποδεικνύεται εύκολα μέσω δύο εφαρμογών του Κανόνα της Ανάθεσης, του Κανόνα της Ακολουθίας και του Κανόνα Ενδυνάμωσης της Προσυνθήκης.
- Για το δεύτερο σκέλος έχουμε να δείξουμε ότι:
$$\{R \geq Y \wedge X= R + Y*Q\} R:= R-Y;Q:= Q+1; \{X= R + Y*Q\}$$
- Με βάση τον Κανόνα της Ανάθεσης:
$$\{X= R + Y*(Q+1)\} Q:= Q+1; \{X= R + Y*Q\}$$
- Και επίσης
$$\{X= R-Y + Y*(Q+1)\} R:= R-Y; \{X= R + Y*(Q+1)\}$$

Απόδειξη (συν.)

- ... και επομένως ότι

$$\{X = R + Y * Q\} R := R - Y; \{X = R + Y * (Q + 1)\}$$

- Από τον Κανόνα της Ακολουθίας προκύπτει το ζητούμενο

$$\{X = R + Y * Q\} R := R - Y; Q := Q + 1; \{X = R + Y * Q\}$$

- Με ενδυνάμωση της προσυνθήκης έχουμε ότι:

$$\{X = R + Y * Q \wedge Y \leq R\} R := R - Y; Q := Q + 1; \{X = R + Y * Q\}$$

- Εφαρμόζουμε τον Κανόνα του while παίρνοντας ότι

$$\begin{aligned} & \{X = R + Y * Q\} \\ & \text{while } (Y \leq R) \{ R := R - Y; Q := Q + 1; \\ & \quad \} \\ & \{R < Y \wedge X = R + Y * Q\} \end{aligned}$$

- Και μέσω μιας ακόμα εφαρμογής του Κανόνα της Ακολουθίας το ζητούμενο έπεται.

Πίνακες Απόδειξης

- Αποδείξεις της ορθότητας μιας προδιαγραφής Hoare μπορούν (για ευκολία) να παρουσιαστούν ως πίνακες στους οποίους παρεμβάλουμε προτάσεις ανάμεσα στις γραμμές (εντολές) του προγράμματος. Έτσι, η απόδειξη της προδιαγραφής ενός πρόγραμμα $C_1; C_2; \dots ; C_n$ έχει τη μορφή

$\{\varphi_0\}$

$C_1;$

$\{\varphi_1\}$

Αιτιολόγηση (όνομα κανόνα)

$C_2;$

...

$\{\varphi_{n-1}\}$

Αιτιολόγηση

$C_n;$

$\{\varphi_n\}$

Αιτιολόγηση

- Μια τέτοια απόδειξη ονομάζεται απόδειξη “ταμπλό” (tableaux proof).

Πίνακες Απόδειξης (2)

- Πως προκύπτουν τα φ_i στον πίνακα;
- Δουλεύουμε «προς τα πίσω», ξεκινώντας από τη μετασυνθήκη και προσπαθώντας να εντοπίσουμε κατάλληλες ενδιάμεσες συνθήκες χρησιμοποιώντας τους κανόνες.
- Η διαδικασία υπολογισμού κατάλληλου φ_i στο βήμα

$\{\varphi_i\}$

C_i ;

$\{\varphi_{i+1}\}$

ονομάζεται υπολογισμός της ασθενέστερης προσυνθήκης (weakest precondition): ψάχνουμε να βρούμε τη λογικά πιο ασθενή (χαλαρή) ιδιότητα για την οποία ισχύει η προδιαγραφή. (Η συνθήκη φ είναι ασθενέστερη από τη συνθήκη ψ αν $\varphi \rightarrow \psi$.)

Αποδείξεις ταμπλό – Ανάθεση και Συνεπαγωγή

- Αναθέσεις:
 - Ποια είναι η ασθενέστερη προσυνθήκη για μια εντολή ανάθεσης $x := E$ με μετασυνθήκη ψ ;
 - Η $\psi[E/x]$.
- Συνεπώς το αξίωμα ανάθεσης σε απόδειξη «ταμπλό» έχει ως εξής:

$$\frac{\begin{array}{l} \{\psi[E/x]\} \\ x := E \\ \{\psi\} \end{array}}{\text{Κανόνας Ανάθεσης}}$$

- Συνεπαγωγή:
 - Μπορούμε να γράψουμε μία πρόταση φ ακριβώς κάτω από μία άλλη πρόταση ψ - χωρίς να παρεμβάλλεται κώδικας - αν η φ συνεπάγεται από την ψ

Παράδειγμα 1

- Να δείξετε ότι $\models_{\text{par}} \{\text{true}\} z := x; z := z + y; u := z; \{u = x+y\}$

- Απόδειξη:

$\{\text{true}\}$

$\{x+y = x+y\}$

$z := x;$

$\{z+y = x+y\}$

$z := z + y;$

$\{z = x+y\}$

$u := z;$

$\{u = x+y\}$

Ενδυνάμωση της Προσυνθήκης

Κανόνας Ανάθεσης

Κανόνας Ανάθεσης

Κανόνας Ανάθεσης

Αποδείξεις ταμπλό – Εντολή if

- Ποια είναι η ασθενέστερη προσυνθήκη φ για μια εντολή if B then C_1 else C_2 με μετασυνθήκη ψ ;
- Υπολογίζουμε τη φ ως εξής:
 1. Σπρώχνουμε τη ψ προς τα πάνω μέσω της C_1 (που μπορεί να είναι μια ακολουθία εντολών). Έστω ότι το αποτέλεσμα είναι ο τύπος φ_1 .
 2. Παρόμοια, σπρώχνουμε τη ψ προς τα πάνω μέσω της C_2 . Έστω ότι το αποτέλεσμα είναι ο τύπος φ_2 .
 3. Θέτουμε ως φ την πρόταση $(B \rightarrow \varphi_1) \wedge (\neg B \rightarrow \varphi_2)$.

Παράδειγμα

- Να δείξετε ότι

$$\models_{\text{par}} \{\text{true}\} \text{ Succ}; \{y = x+1\}$$

όπου Succ το πρόγραμμα:

```
a := x+1;
```

```
if (a-1 == 0)
```

```
    y := 1;
```

```
else
```

```
    y := a;
```


Παράδειγμα (συν.)

- **Απόδειξη:**

{true}

{ $x+1-1 = 0 \rightarrow 1 = x+1$ } \wedge ($\neg (x+1-1 = 0) \rightarrow x+1 = x+1$)} **Ενδυνάμωση Προσυνθήκης**

a:= x+1;

{ $a-1 = 0 \rightarrow 1 = x+1$ } \wedge ($\neg (a-1 = 0) \rightarrow a = x+1$)} **Κανόνας Ανάθεσης**

if (a-1 == 0)

 {1= x+1} **Κανόνας Εντολής if**

 y := 1;

 {y = x+1} **Κανόνας Ανάθεσης**

else

 {a = x+1} **Κανόνας Εντολής if**

 y:= a;

 {y = x+1} **Κανόνας Ανάθεσης**

{y = x+1} **Κανόνας Εντολής if**

Αποδείξεις ταμπλό – Εντολή while (1)

- Εντολή while:

$$\frac{\{ \eta \wedge B \} C \{ \eta \}}{\{ \eta \} \text{ while } B \ C \{ \eta \wedge \neg B \}}$$

Σύμφωνα με τον κανόνα, η αμετάβλητη συνθήκη η επιλέγεται έτσι ώστε η ορθότητα της να διατηρείται από τον βρόχο C του `while`. Αυτό σημαίνει ότι αν B είναι αληθής και η είναι αληθής πριν από την εκτέλεση της C και αν το σώμα εντολών C τερματίζει, τότε η είναι επίσης αληθής στο τέλος της εκτέλεσης.

- Η ερμηνεία αυτή είναι συμβατή με την έννοια της μερικής ορθότητας.

Αποδείξεις ταμπλό – Εντολή while (2)

- Ο κανόνας μας επιτρέπει να αποδεικνύουμε προδιαγραφές των οποίων η μετασυνθήκη είναι η σύζευξη της προσυνθήκης με την άρνηση του B, δηλαδή, προδιαγραφές της μορφής:

$$\{ \eta \} \text{ while } B \ C \ \{ \eta \wedge \neg B \}$$

- Πως μπορούμε να χειριστούμε προδιαγραφές της μορφής

$$\{ \varphi \} \text{ while } B \ C \ \{ \psi \}$$

όπου οι ιδιότητες φ και ψ δε σχετίζονται μεταξύ τους;

Αποδείξεις ταμπλό – Εντολή while (3)

- Η απάντηση είναι ότι πρέπει να εντοπίσουμε κατάλληλη αμετάβλητη συνθήκη η τέτοια ώστε
 - $\varphi \rightarrow \eta$
 - $\eta \wedge \neg B \rightarrow \psi$
 - $\{\eta\} \text{ while } B \ C \ \{\eta \wedge \neg B\}$
- Σε τέτοια περίπτωση, οι κανόνες συνεπαγωγής μας οδηγούν στο συμπέρασμα ότι
$$\{\varphi\} \text{ while } B \ C \ \{\psi\}$$
- Το πιο κρίσιμο βήμα της απόδειξης είναι η επιλογή της αμετάβλητης συνθήκης (invariant).

Αποδείξεις ταμπλό – Εντολή while (4)

- **Ορισμός:** Αμετάβλητη συνθήκη σε μία εντολή `while B C` είναι μια πρόταση η τέτοια ώστε

$$\models_{\text{par}} (\eta \wedge B) C (\eta)$$

- Μία χρήσιμη αμετάβλητη συνθήκη συνήθως εκφράζει μία σχέση μεταξύ των μεταβλητών του βρόχου C , μία σχέση που διατηρείται ακόμη και αν οι τιμές των μεταβλητών αλλάξουν.
- Ένας τρόπος για να ανακαλύπτουμε την αμετάβλητη συνθήκη είναι να δημιουργήσουμε ένα ίχνος εκτέλεσης του προγράμματος (trace).

Παράδειγμα

Έστω το πρόγραμμα `fact (x)` με κώδικα

```
y := 1;  
z := 0;  
while (z != x) {  
    z := z + 1;  
    y := y * z;  
}
```

επανάληψη	z	y	z != x
0	0	1	true
1	1	1	true
2	2	2	true
3	3	6	true
4	4	24	true
5	5	120	true
6	6	720	false

Έστω ότι το πρόγραμμα εκτελείται για $x = 6$.

Ο πίνακας επιδεικνύει τις τιμές των μεταβλητών του προγράμματος κατά την εκκίνηση της i -οστής εκτέλεσης του βρόχου.

Παρατηρούμε ότι η αμετάβλητη συνθήκη είναι απλά η $y = z!$

Παράδειγμα (συν.)

Η αμετάβλητη συνθήκη που εντοπίστηκε επίσης διαθέτει τις ιδιότητες που επιθυμούμε:

- είναι αρκετά ασθενής ώστε τελικά να προκύπτει από την προσυνθήκη της εντολής while που είναι $y = 1 \wedge z = 0$
- είναι και αρκετά ισχυρή έτσι ώστε μαζί με την άρνηση της συνθήκης του while να συνεπάγεται τη μετασυνθήκη

$$y = x!$$

Δηλαδή ισχύουν οι συνεπαγωγές

$$(y = 1 \wedge z = 0) \rightarrow (y = z!)$$

και

$$(y = z! \wedge x = z) \rightarrow (y = x!)$$

Εντολές while σε απόδειξη «ταμπλό»:

1. Εντοπίζουμε μια συνθήκη η που υπολογίζουμε να είναι κατάλληλη αμετάβλητη συνθήκη.
2. Προσπαθούμε να αποδείξουμε ότι $\eta \wedge \neg B \rightarrow \psi$ (όπου B είναι η λογική συνθήκη της εντολής while) και ότι $\varphi \rightarrow \eta$. Αν τα καταφέρουμε προχωρούμε στο Βήμα 3, διαφορετικά επιστρέφουμε στο Βήμα 1.
3. Σπρώχνουμε τη συνθήκη η προς τα πάνω μέσα από το σώμα του βρόχου εφαρμόζοντας τους κανόνες που πρέπει ανάλογα με τις εντολές που συναντούμε. Έστω ότι προκύπτει η συνθήκη η' .
4. Αποδεικνύουμε ότι $\eta \wedge B \rightarrow \eta'$, που ουσιαστικά αποδεικνύει ότι το η είναι αμετάβλητη συνθήκη. (Αν αποτύχουμε επιστρέφουμε στο Βήμα 1.)
5. Τέλος, γράφουμε την η πάνω από την εντολή while και αμέσως προηγούμενα τη συνθήκη φ .

Παράδειγμα (συν.)

$\{\text{True}\}$	
$\{1 = 0!\}$	Ενδυνάμωση Προσυνθήκης
$y := 1;$	
$\{y = 0!\}$	Κανόνας Ανάθεσης
$z := 0;$	
$\{y = z!\}$	Κανόνας Ανάθεσης
$\text{while } (z \neq x) \{$	
$\{y = z! \wedge z \neq x\}$	Υπόθεση Αμετάβλητης Συνθήκης & Φρουρός
$\{y * (z + 1) = (z + 1)!\}$	Κανόνας Συνεπαγωγής
$z := z + 1;$	
$\{y * z = z!\}$	Κανόνας Ανάθεσης
$y := y * z;$	
$\{y = z!\}$	Κανόνας Ανάθεσης
$\}$	
$\{y = z! \wedge \neg(z \neq x)\}$	Κανόνας while
$\{y = x!\}$	Κανόνας Συνεπαγωγής

Ολική Ορθότητα

- Το σύστημα αποδείξεων που έχουμε μελετήσει αποδεικνύει τη μερική ορθότητα προγραμμάτων. Δηλαδή, ελέγχει κατά πόσο το πρόγραμμα ικανοποιεί την προδιαγραφή του σε περίπτωση που τερματίζει.
- Δεν ενημερώνει αν το πρόγραμμα τερματίζει ή αν εισέρχεται σε ατέρμονο βρόχο.
- Κατά την ανάλυση ολικής ορθότητας μας ενδιαφέρει να αποδείξουμε ότι και το πρόγραμμα τερματίζει και ικανοποιεί την προδιαγραφή του κατά τον τερματισμό.

Τερματισμός και η εντολή while

- Η μόνη εντολή που μπορεί να προκαλέσει μη τερματισμό στη γλώσσα που μελετούμε είναι η εντολή while.
- Επομένως το Σύστημα Απόδειξης Ολικής Ορθότητας κληρονομεί τους κανόνες/αξιώματα από το Σύστημα Απόδειξης Μερικής Ορθότητας με εξαίρεση τον Κανόνα while.
- Ο αναθεωρημένος Κανόνας while αποτελείται από δύο σκέλη:
 - Την απόδειξη της μερικής ορθότητας, δηλαδή απόδειξη ότι αν ο βρόχος τερματίσει τότε ικανοποιεί την προδιαγραφή του και
 - Την απόδειξη ότι τερματίζει.

Τερματισμός και η εντολή while

- Η απόδειξη τερματισμού προϋποθέτει τον εντοπισμό μιας ακέραιης έκφρασης της οποίας η τιμή μειώνεται κάθε φορά που εκτελείται το σώμα του βρόχου και δεν γίνεται ποτέ αρνητική.
- Μια τέτοια έκφραση ονομάζεται η **μεταβλητή έκφραση** (variant) και η ύπαρξή της αποδεικνύει ότι ο βρόχος τερματίζει.
- Για το πρόγραμμα fact (Διαφάνεια 9-38) μία κατάλληλη μεταβλητή έκφραση είναι η $x - z$:
 - Κατά την εκκίνηση της εκτέλεσης του βρόχου $x - z = x$
 - Με κάθε επανάληψη του βρόχου η τιμή της $x - z$ μειώνεται κατά 1 (αφού το z αυξάνεται κατά 1)
 - Το $x - z$ δεν μπορεί να γίνει ποτέ αρνητικό: όταν γίνει 0 ο βρόχος τερματίζει.

Ο κανόνας Total-while

$$\frac{\{ \eta \wedge B \wedge 0 \leq E = E_0 \} \quad C \quad \{ \eta \wedge 0 \leq E < E_0 \}}{\{ \eta \wedge 0 \leq E \} \quad \text{while } B \quad C \quad \{ \eta \wedge \neg B \}}$$

- Η E είναι η μεταβλητή έκφραση και ως E_0 συμβολίζουμε την αρχική της τιμή.
- **Εξήγηση Κανόνα:** Σύμφωνα με τον κανόνα, αν
 - η συνθήκη η είναι μια αμετάβλητη συνθήκη του βρόχου και επιπλέον
 - η E είναι μια μεταβλητή έκφραση με αρχική μη-αρνητική τιμή E_0 η οποία μειώνεται σε κάθε εκτέλεση του C χωρίς να γίνει μικρότερη από το 0τότε η η είναι μια αμετάβλητη συνθήκη ολόκληρης της εντολής `while` το οποίο τερματίζει.

Παράδειγμα

Θα δείξουμε ότι το πρόγραμμα `fact`

```
y := 1;  
z := 0;  
while (z != x) {  
    z := z + 1;  
    y := y * z;  
}
```

Ικανοποιεί την προδιαγραφή

$\models_{\text{tot}} \{x \geq 0\} \text{ fact } \{y = x!\}$

με αμετάβλητη συνθήκη την $y = z!$

και μεταβλητή έκφραση την $x - z$.

Παράδειγμα (συν.)

$\{x \geq 0\}$	
$\{1 = 0! \wedge 0 \leq x-0\}$	Ενδυνάμωση Προσυνθήκης
$y := 1;$	
$\{y = 0! \wedge 0 \leq x-0\}$	Κανόνας Ανάθεσης
$z := 0;$	
$\{y = z! \wedge 0 \leq x-z\}$	Κανόνας Ανάθεσης
$\text{while } (z \neq x) \{$	
$\{y = z! \wedge z \neq x \wedge 0 \leq x-z = E_0\}$	Υπόθεση Αμ. Συνθήκης & Φρουρός
$\{y * (z + 1) = (z + 1)! \wedge 0 \leq x - (z+1) < E_0\}$	Κανόνας Συνεπαγωγής
$z := z + 1;$	
$\{y * z = z! \wedge 0 \leq x - z < E_0\}$	Κανόνας Ανάθεσης
$y := y * z;$	
$\{y = z! \wedge 0 \leq x - z < E_0\}$	Κανόνας Ανάθεσης
$\}$	
$\{y = z! \wedge \neg(z \neq x)\}$	Κανόνας while
$\{y = x!\}$	Κανόνας Συνεπαγωγής

Τερματισμός εντολών while

- Στη γενική περίπτωση, το πρόβλημα κατά πόσο μια εντολή while τερματίζει είναι **μη αποφασίσιμο**: δεν υπάρχει διαδικασία η οποία με δεδομένο εισόδο μια εντολή while να μπορεί να απαντήσει αν αυτή τερματίζει. Κατ'επέκταση το ίδιο ισχύει και για την εύρεση μεταβλητών εκφράσεων για τέτοιες εντολές.
- Ο πιο κάτω κώδικας επιδεικνύει ένα χαρακτηριστικά «δύσκολο» βρόχο για τον οποίο έχειδειχθεί η αδυναμία εύρεσης μεταβλητής συνθήκης παρόλο που εμπειρικά παρατηρούμε ότι ο βρόχος τερματίζει πάντα.

```
c := x;
while ( c!= 1){
    if (c mod 2 == 0) c := c/2
    else c := 3*c + 1
}
```

- Για $x = 5$ ο βρόχος εκτελεί 6 επαναλήψεις, για $x = 72$ εκτελεί 32 επαναλήψεις για $x = 123456789$ εκτελεί 177 επαναλήψεις...

Σχόλια

- Τα συστήματα κανόνων που μελετήσαμε είναι ορθά: αν μπορούμε να αποδείξουμε την ορθότητα μιας προδιαγραφής μέσω των συστημάτων αυτών τότε οι προδιαγραφές είναι πράγματι ορθές.
- Τα συστήματα αυτά όμως δεν είναι πλήρη: υπάρχουν ορθές προδιαγραφές που δεν αποδεικνύονται βάσει αυτών.
- Επεκτάσεις των συστημάτων έχουν διατυπωθεί για:
 - Συναρτήσεις
 - Παράλληλα προγράμματα
 - Συναρτησιακές Γλώσσες Προγραμματισμού
 - ...

Άσκηση 1

- Να αποφασίσετε κατά πόσο οι πιο κάτω προδιαγραφές είναι ορθές. Σε περίπτωση που μία προδιαγραφή είναι ορθή να δώσετε την σχετική απόδειξη διαφορετικά να μεταβάλετε την προσυνθήκη ή την μετασυνθήκη κατάλληλα έτσι ώστε η προδιαγραφή να ικανοποιείται.

$\{i \neq j\}$ if $i > j$ then $m := i - j$ else $m := j - i$ $\{m > 0\}$

$\{i = 3 * j\}$ if $i > j$ then $m := i - j$ else $m := j - i$ $\{m - 2 * j = 0\}$

$\{x = b\}$ while $x > a$ do $x := x - 1$ $\{b = a\}$

$\{s = 2^i\}$ while $i < n$ do $i := i + 1$; $s := s * 2$ $\{s = 2^i\}$

Άσκηση 2

- Να αποδείξετε την ολική ορθότητα της προδιαγραφής

$$\models_{\text{tot}} \{x > 0\} P \{s = x \cdot (x-1)/2\}$$

όπου ο κώδικας του P δίνεται πιο κάτω.

```
s := 0;
i := 1;
while (i != x) {
    s := s + i;
    i := i + 1;
}
```

Άσκηση 3

- Να αποδείξετε την ολική ορθότητα της προδιαγραφής

$$\models_{\text{tot}} \{0 \leq n\} P \{r = x^n\}$$

όπου ο κώδικας του P δίνεται πιο κάτω.

```
r := 1;
p := x;
e := n;
while e > 0 {
    if (e mod 2) <> 0
        then r := r * p;
    p := p * p;
    e := e div 2;
}
```

Άσκηση 4

- Να αποδείξετε την ολική ορθότητα της προδιαγραφής

$$\models_{\text{tot}} \{0 \leq n\} P \{s = x^n\}$$

όπου ο κώδικας του P δίνεται πιο κάτω.

```
s := 1;
m := x;
e := n;
while (e != 0)
    while (even(e)) do
        m := m*m;
        e := e / 2;
    e := e-1;
    s := s*m
```