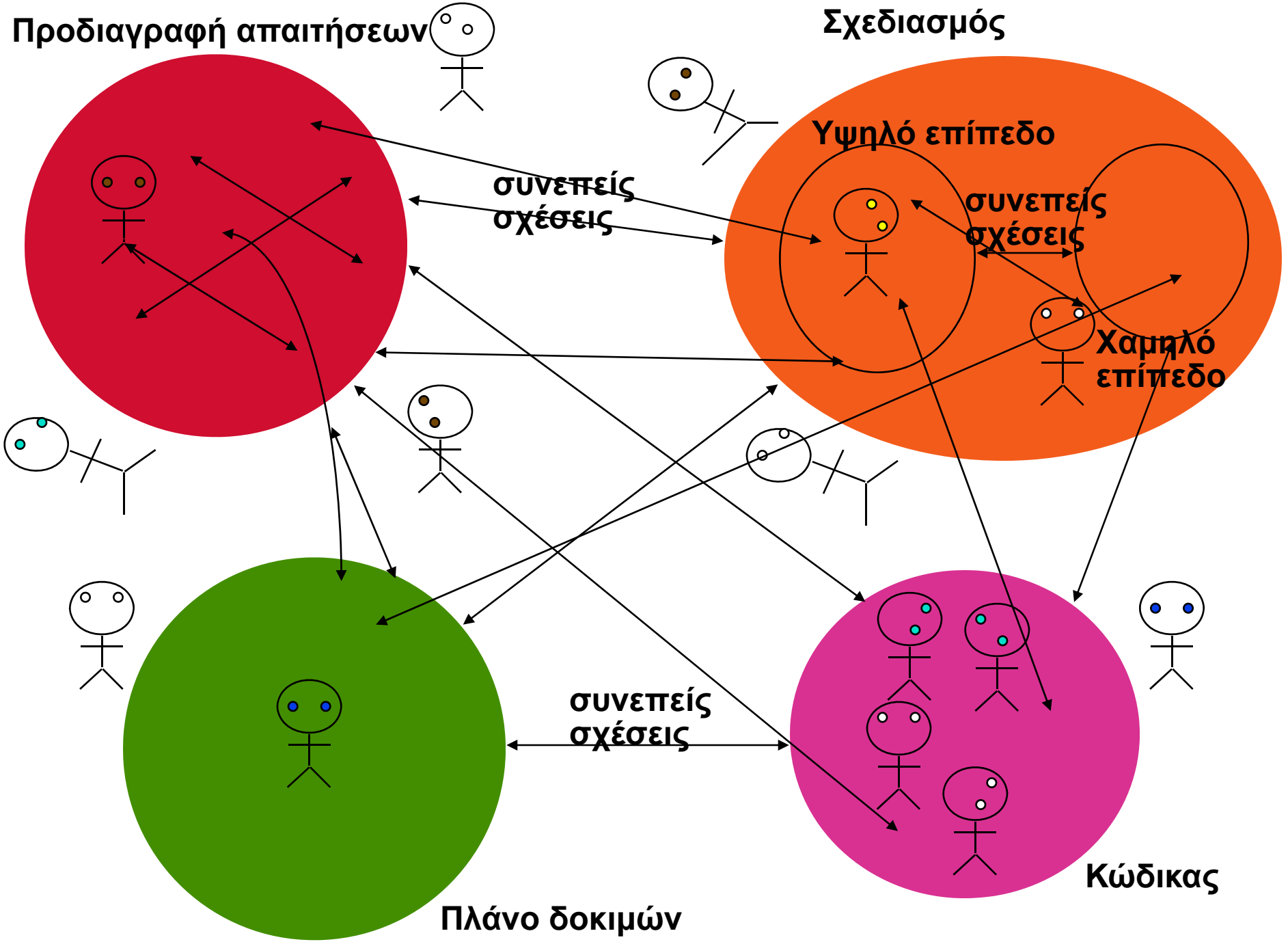
Συγγραφή κώδικα, δοκιμασία, επαλήθευση

Γιάννης Σμαραγδάκης


Προδιαγραφή απαιτήσεων
Σχεδιασμός
Υψηλό επίπεδο
συνεπείς σχέσεις
συνεπείς σχέσεις
Χαμηλό επίπεδο
συνεπείς σχέσεις
Πλάνο δοκιμών
Κώδικας

Συγγραφή κώδικα

- Δεν έχουμε να πούμε πολλά. Θυμηθείτε τι έχουμε πει από την αρχή του εξαμήνου
  - ο κώδικας αντιπροσωπεύει συνήθως γύρω το 15% της προσπάθειας ανάπτυξης λογισμικού
    - αλλά όλα τα άλλα γίνονται για να δουλέψει σωστά αυτό το 15%
  - ο κώδικας πρέπει να ακολουθεί τις προδιαγραφές σχεδίασης
    - τμηματική σχεδίαση, κρύψιμο πληροφορίας

- Κακός/καλός κώδικας γράφεται σε οποιαδήποτε γλώσσα
  - αν και οι μοντέρνες γλώσσες ενθαρύνουν ιδιότητες καλού κώδικα

Η συγγραφή κώδικα είναι στενά συνδεδεμένη με τη δοκιμασία του

- Η ουσία είναι η έλλειψη ελαττωμάτων (faults) που εκφράζονται σαν αστοχίες (failures)

Τεχνικές αξιοπιστίας

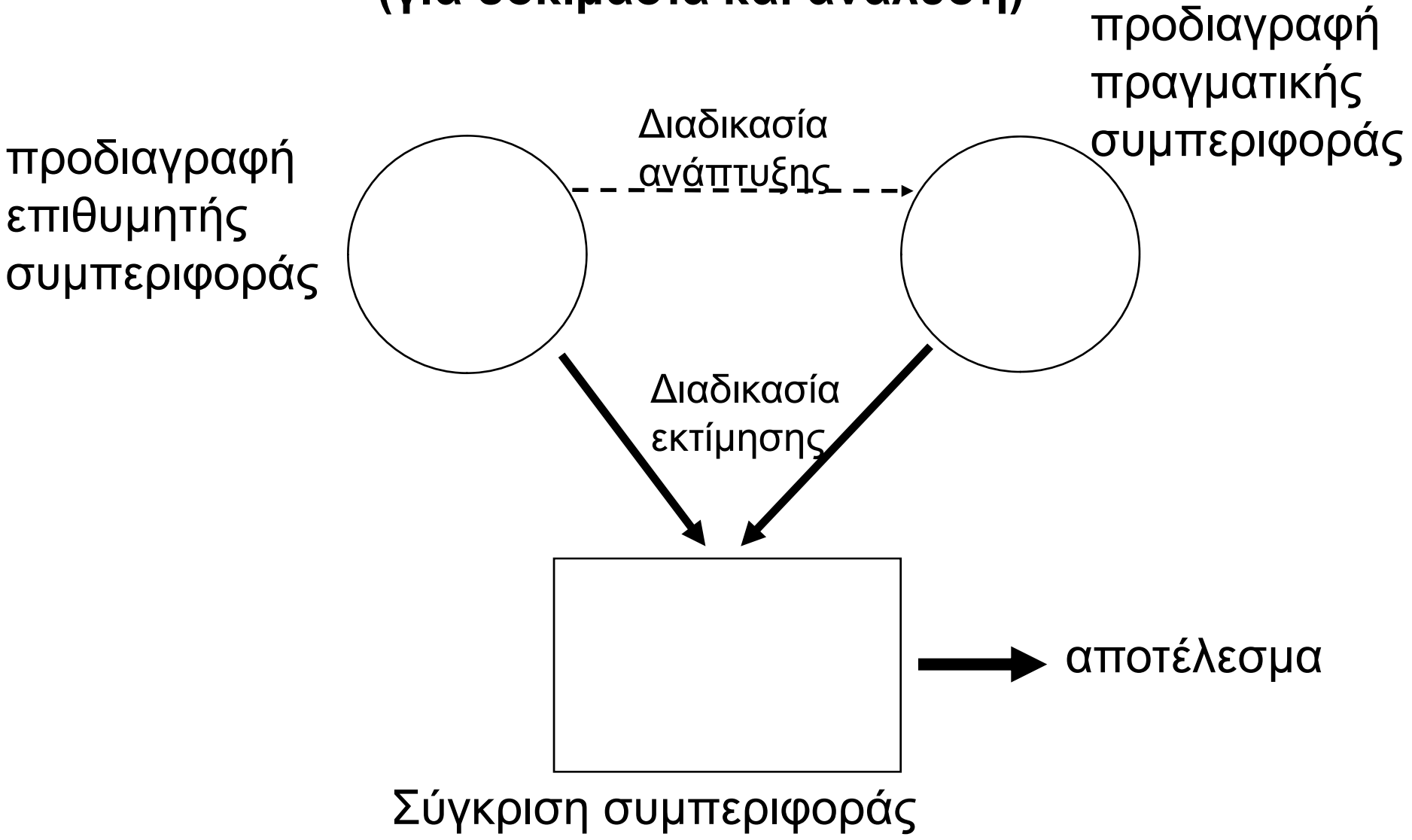
- ***Αποφυγή ελαττωμάτων (fault avoidance)*: τεχνικές ανάπτυξης λογισμικού που μειώνουν τα σφάλματα**
  - **π.χ. καλός σχεδιασμός, ανάπτυξη προτύπων, μαθηματικές προδιαγραφές**
- ***Εξάλειψη ελαττωμάτων (fault elimination)*: τεχνικές ανάλυσης που ανακαλύπτουν ελαττώματα**
  - **δοκιμασία, στατική ανάλυση, επιθεώρηση κώδικα, επαλήθευση**
- ***Πρόβλεψη ελαττωμάτων (fault prediction)*: τεχνικές ανάλυση που προβλέπουν την ύπαρξη σφαλμάτων**
  - **μέτρα ποιότητας, διαδικασίες εκτίμησης**
- ***Ανοχή ελαττωμάτων (fault tolerance)*: τεχνικές εκτέλεσης προγράμματος που ανιχνεύουν και διορθώνουν σφάλματα πριν υπάρξει αστοχία**
  - **κώδικας ανάκαμψης, n-version programming**

Ορισμοί

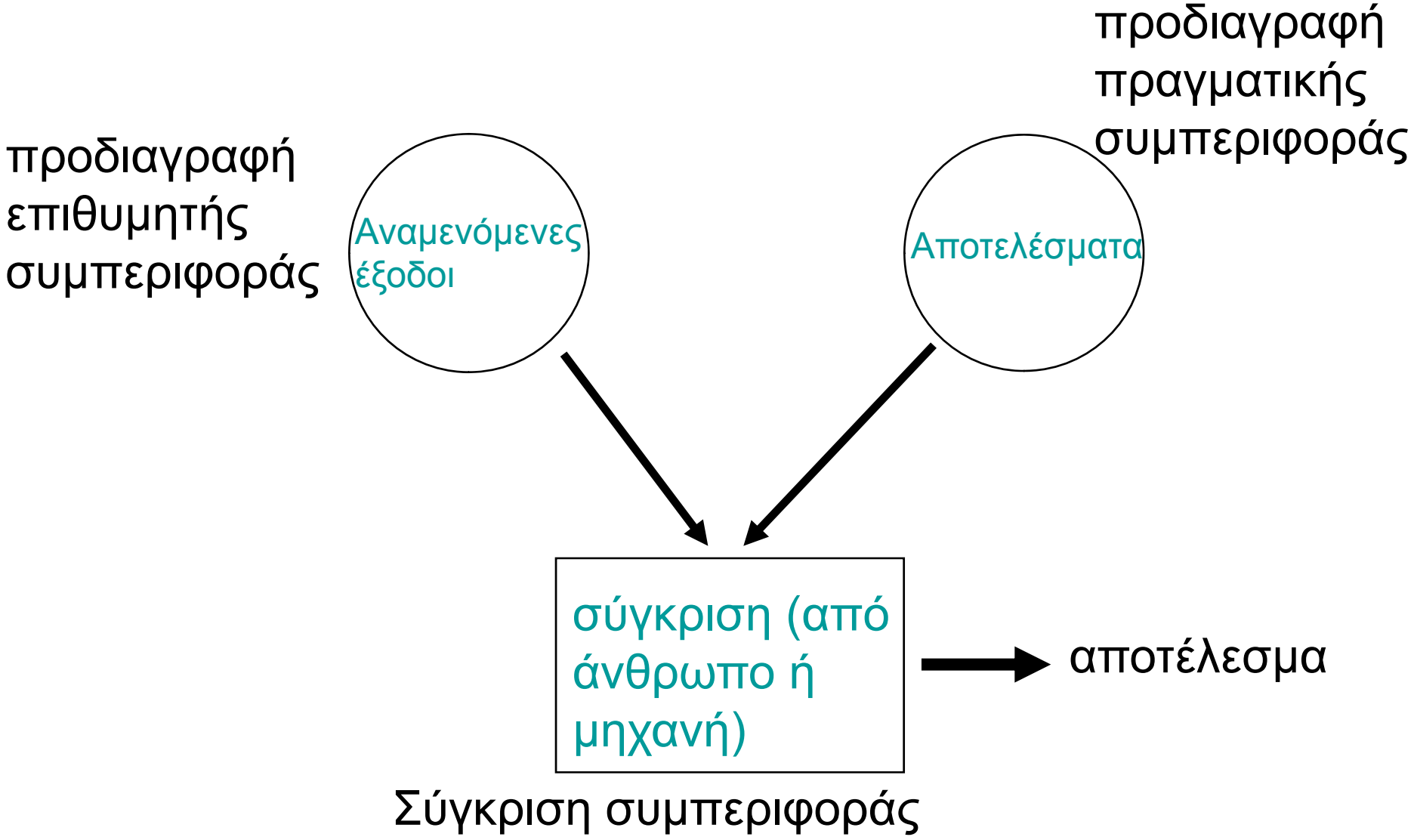
- **Δοκιμασία: συστηματική (;) έρευνα στο χώρο εισόδων του προγράμματος σε αναζήτηση αστοχίας**
- **Debugging: ή ερεύνα για το ελάττωμα που προκάλεσε την αστοχία**
- **Στατική ανάλυση: η εξέταση ενός προγράμματος με σκοπό την εξαγωγή των χαρακτηριστικών συμπεριφοράς του**
- **Δυναμική ανάλυση: η παρατήρηση της εκτέλεσης ενός προγράμματος με σκοπό την εξαγωγή των χαρακτηριστικών συμπεριφοράς του**
- **Επαλήθευση (verification): η χρήση τεχνικών ανάλυσης για αυστηρή απόδειξη της ορθότητας του προγράμματος**

Η βασική προσέγγιση είναι ίδια

(για δοκιμασία και ανάλυση)



Δοκιμασία


προδιαγραφή επιθυμητής συμπεριφοράς
προδιαγραφή πραγματικής συμπεριφοράς
Αναμενόμενες έξοδοι
Αποτελέσματα
σύγκριση (από άνθρωπο ή μηχανή)
αποτέλεσμα
Σύγκριση συμπεριφοράς

Ανάλυση

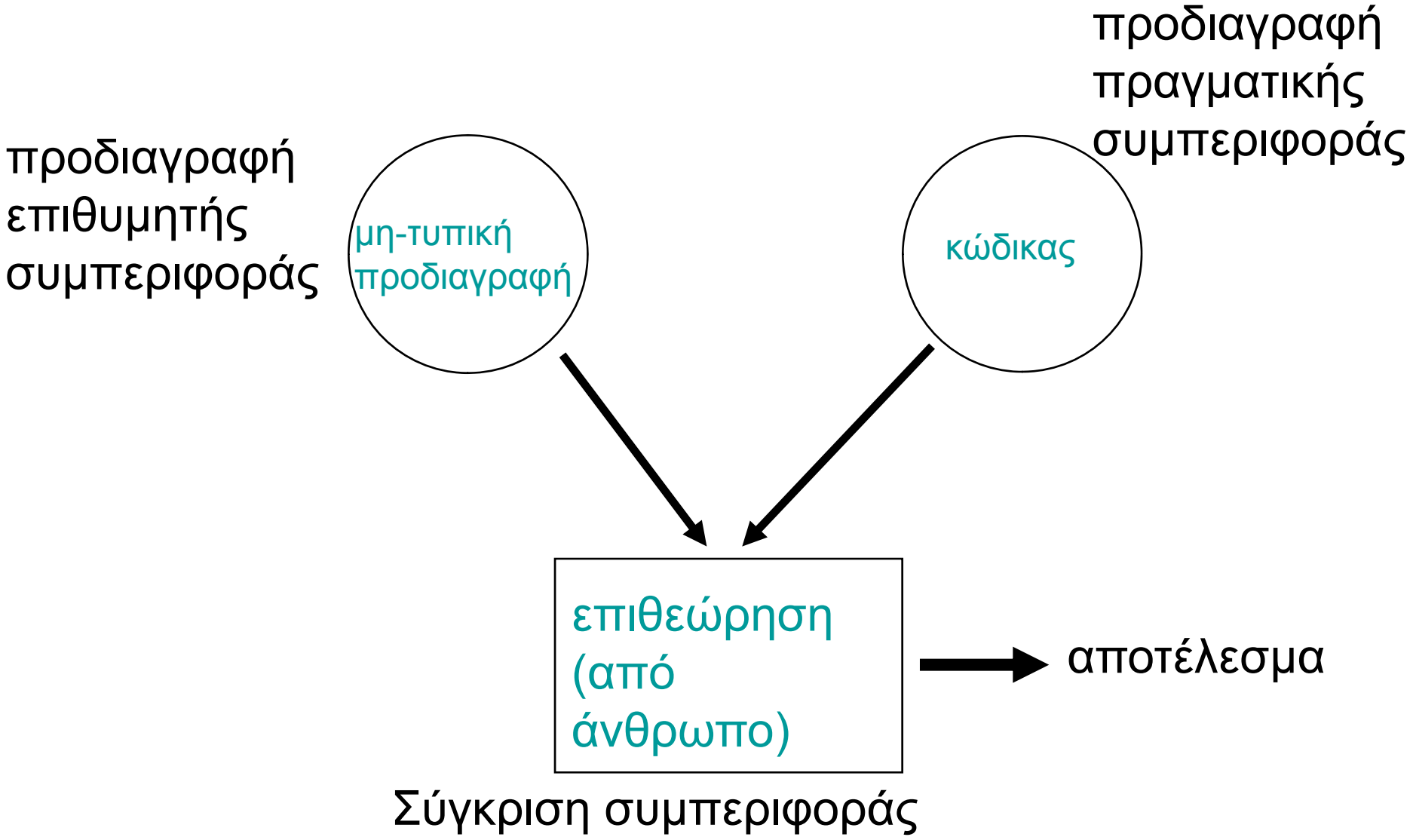
Η δοκιμασία μπορεί να τεκμηριώσει μόνο την ύπαρξη ελαττωμάτων, όχι την απουσία τους

- **Η στατική ανάλυση δεν χρειάζεται εκτέλεση**
  - **μαθηματικό μοντέλο του λογισμικού**
  - **επεξεργασία του με τυπικές μεθόδους**
  - **μπορεί να δείξει την απουσία λαθών**
- **Λύνει τα προβλήματα της δοκιμασίας**
  - **δεν χρειάζεται να κάνουμε δειγματοληψία**
  - **δεν χρειάζεται να ερμηνεύσουμε αποτελέσματα ή να αποφασίσουμε πότε να σταματήσουμε**
- **Περιορισμοί**
  - **τα θεωρήματα που θέλουμε να αποδείξουμε είναι συνήθως πολύ δύσκολα**

Πολλές διαδικασίες ανάλυσης

- Επιθεώρηση
- Συντακτική ανάλυση
- Σημασιολογική ανάλυση
- Ανάλυση ροής
- Επαλήθευση

Επιθεώρηση


προδιαγραφή
επιθυμητής
συμπεριφοράς
μη-τυπική
προδιαγραφή
προδιαγραφή
πραγματικής
συμπεριφοράς
κώδικας
επιθεώρηση
(από
άνθρωπο)
αποτέλεσμα
Σύγκριση συμπεριφοράς

Εύκολοι στατικοί αναλυτές

- Έλεγχος σύνταξης
  - parser
- Σημασιολογικός έλεγχος
  - έλεγχος τύπων, κλπ.

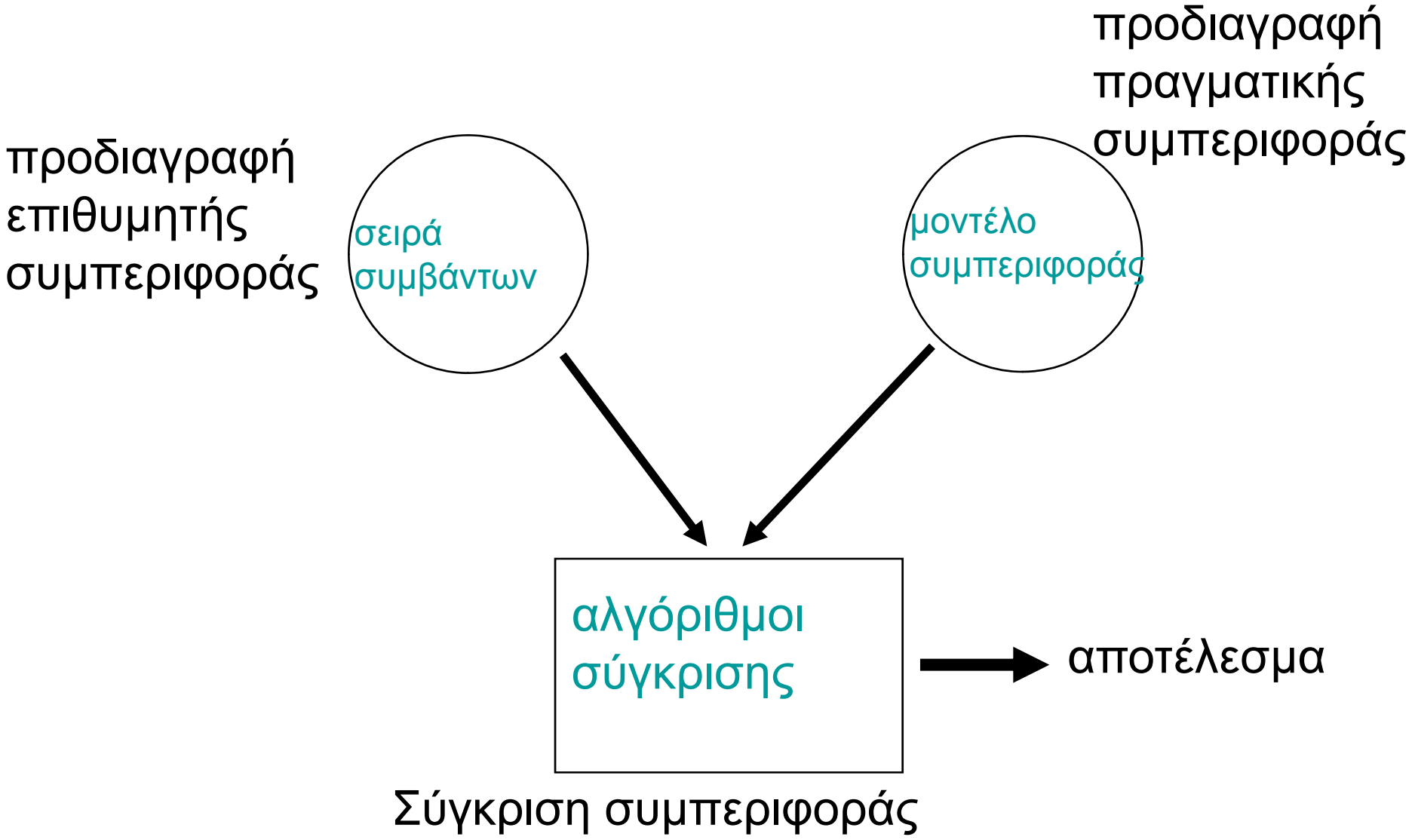
- Παραδείγματα;

Ανάλυση ροής

Έλεγχος μοντέλων (model checking)

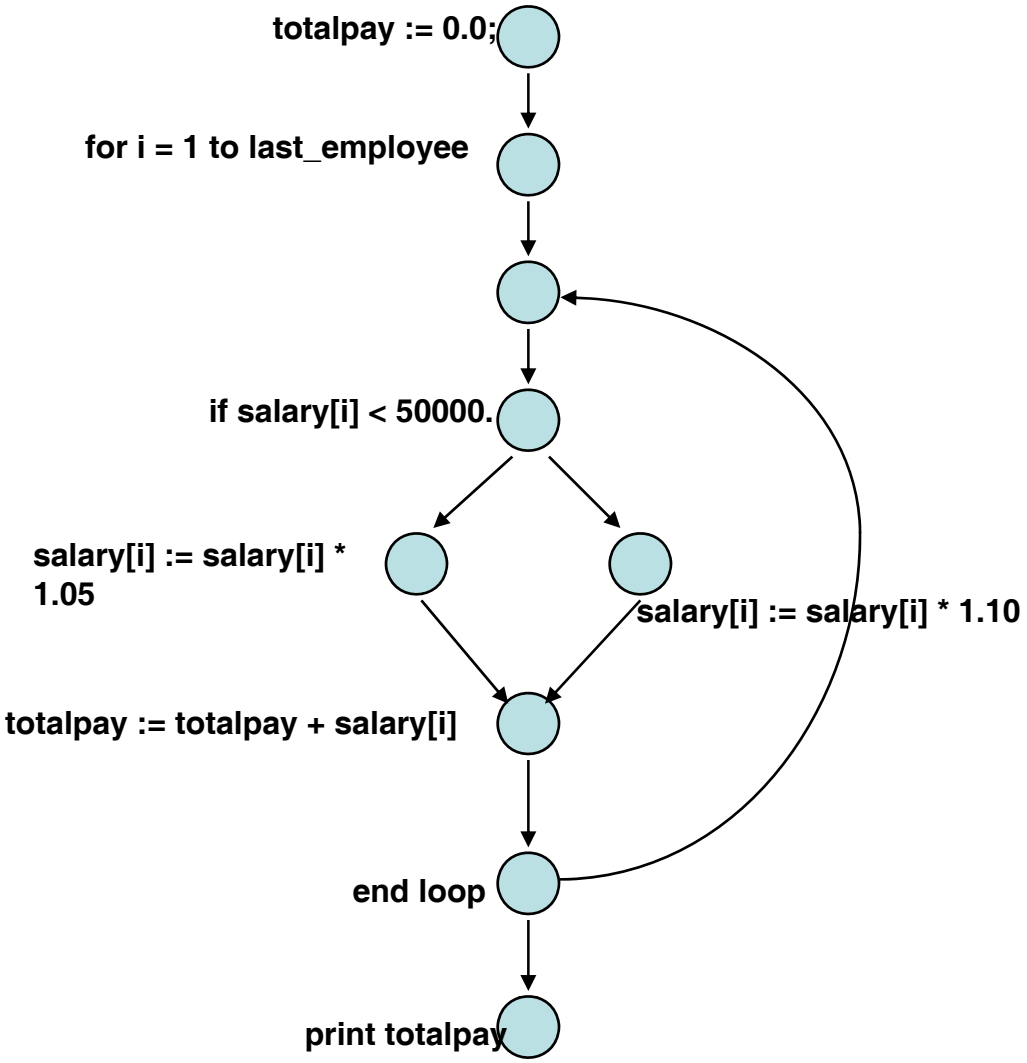
- Προδιαγραφή επιθυμητής συμπεριφοράς: ακολουθία από συμβάντα
- Πραγματική συμπεριφορά: παράγεται από μοντέλο ροής του προγράμματος
  - οι κόμβοι ονοματίζουν ενδιαφέροντα συμβάντα
  - όλες οι δυνατές εκτελέσεις αναπαριστώνται σαν ακολουθίες συμβάντων κατά μήκος ενός μονοπατιού ροής
- Σύγκριση: ανάλογα με την ερώτηση μπορεί να είναι προφανής ή αναλυτική/δύσκολη
- Παραδείγματα:
    - κανένα αρχείο δεν διαβάζεται πριν ανοιχτεί
    - το ασανσέρ δεν κινείται πριν κλείσουν οι πόρτες

Έλεγχος μοντέλων

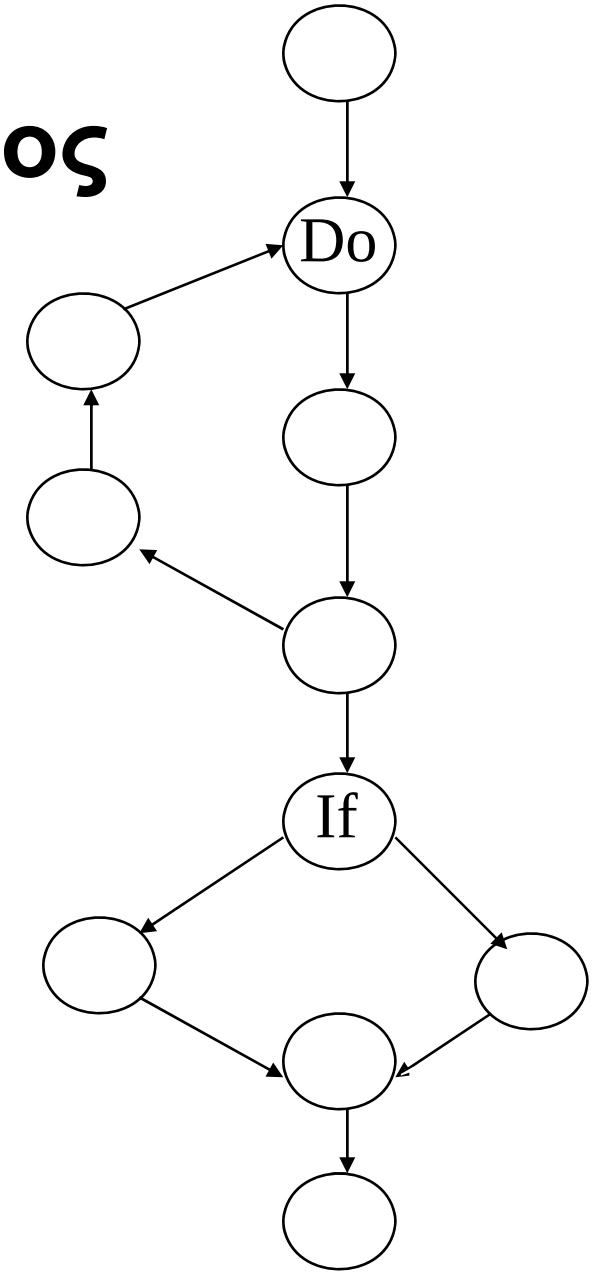


Παράδειγμα

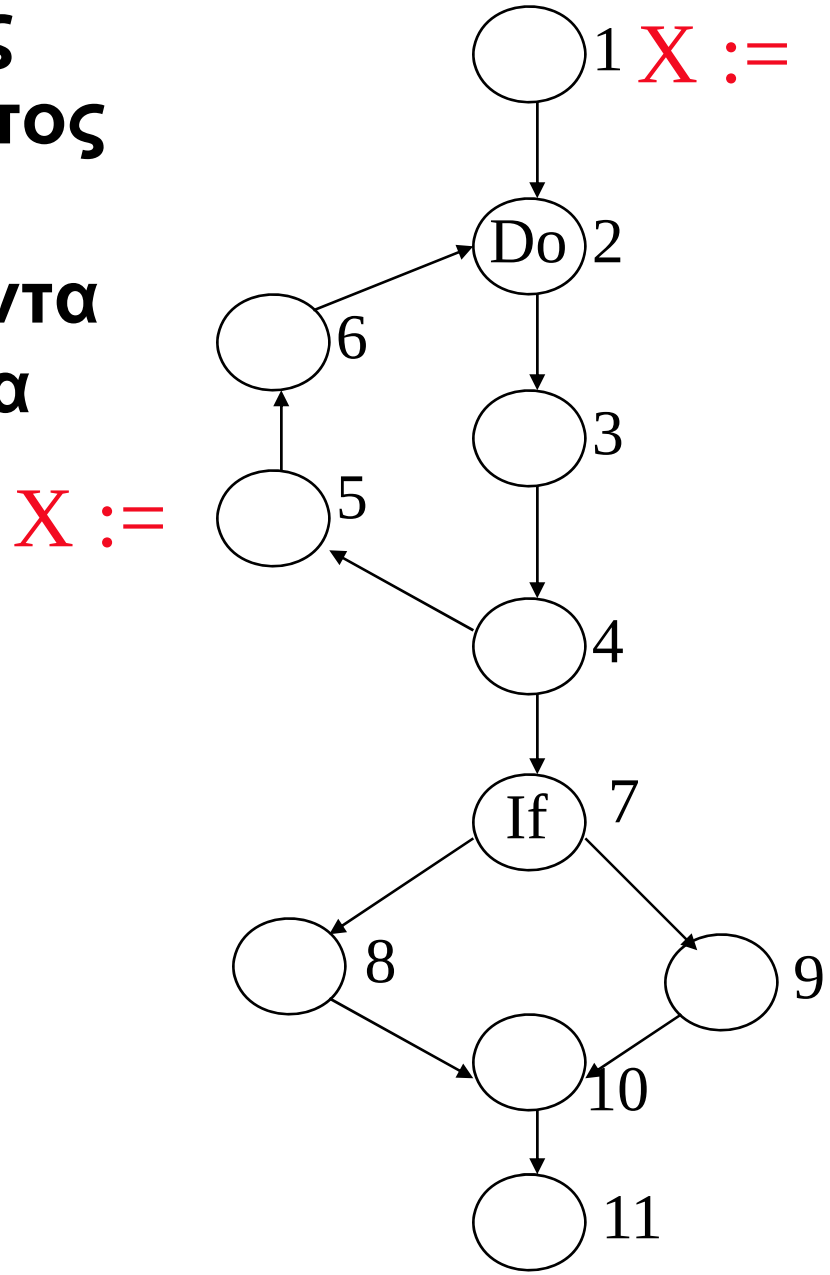
```
totalpay := 0.0;
for i = 1 to last_employee
    if salary[i] < 50000.
        then salary[i] := salary[i] * 1.05;
        else salary[i] := salary[i] * 1.10;
    totalpay := totalpay + salary[i];
    end loop;
print totalpay;
```



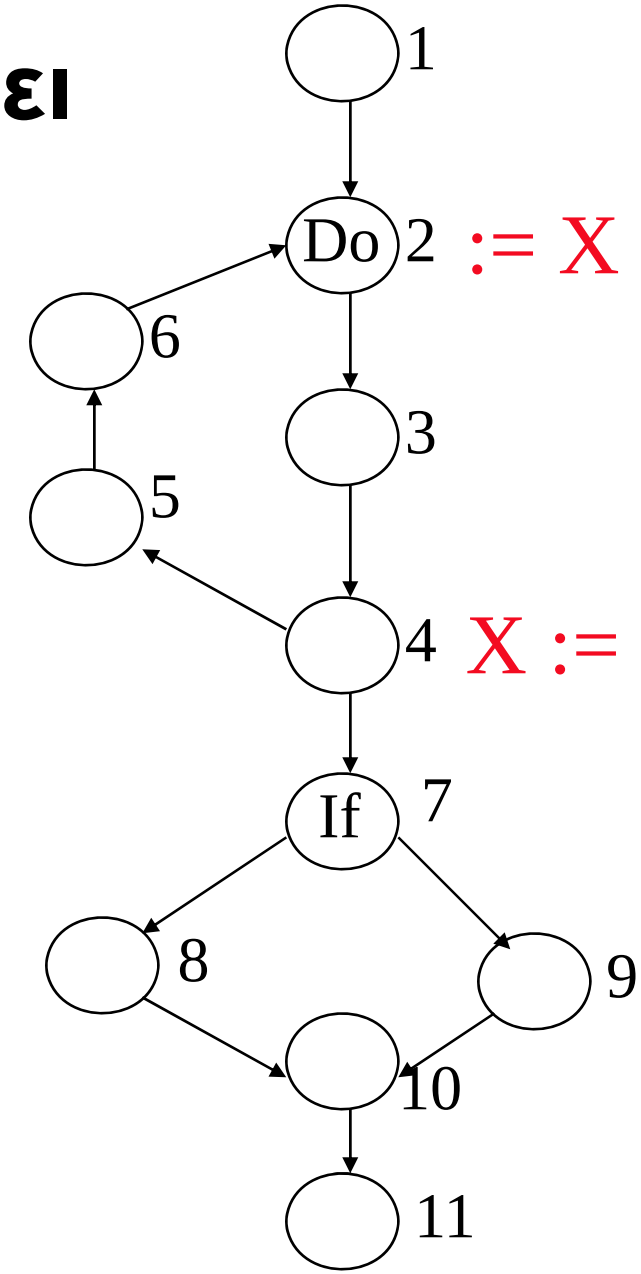
Σκελετός διαγράμματος ροής



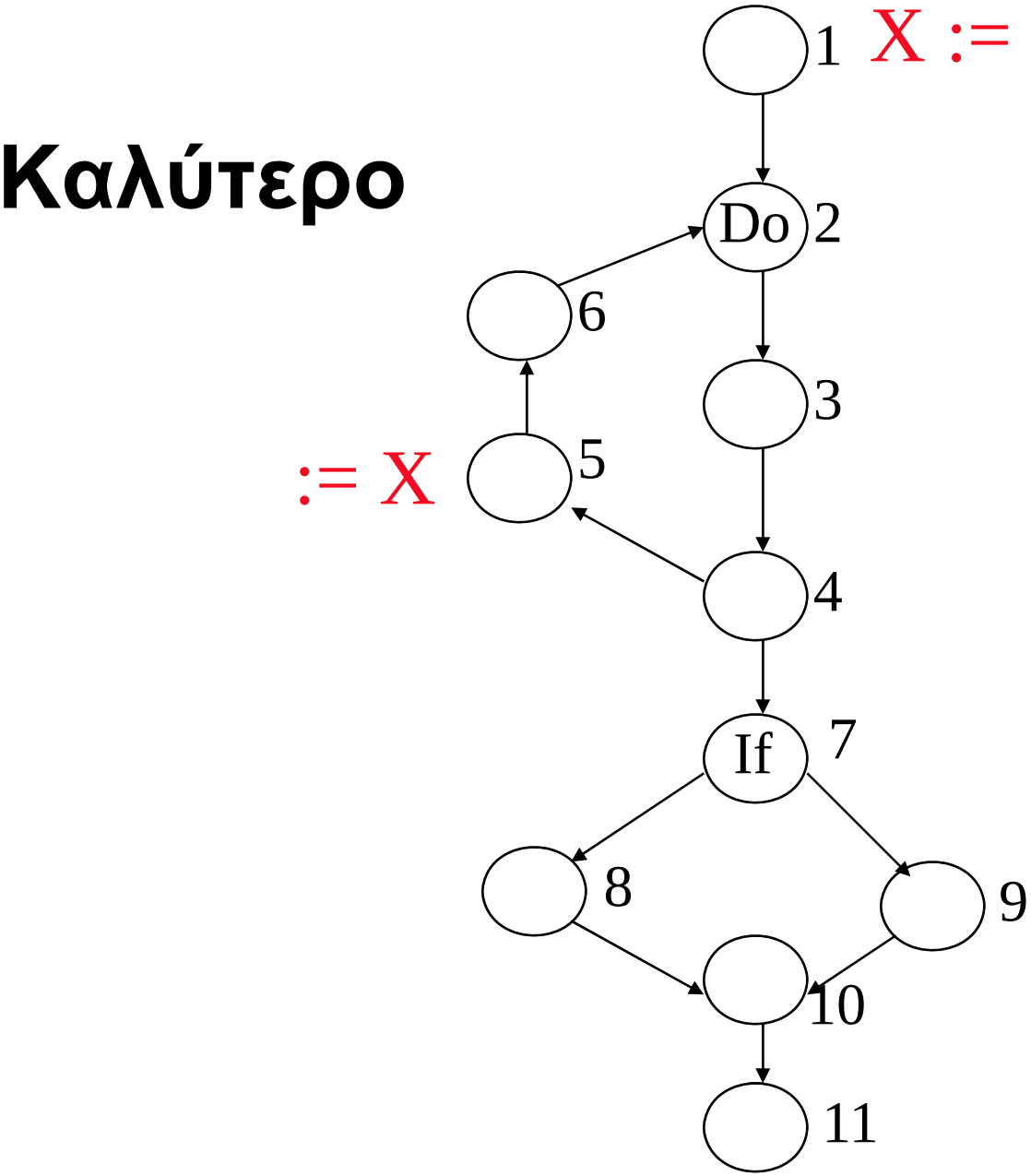
Σκελετός διαγράμματος με ενδιαφέροντα συμβάντα



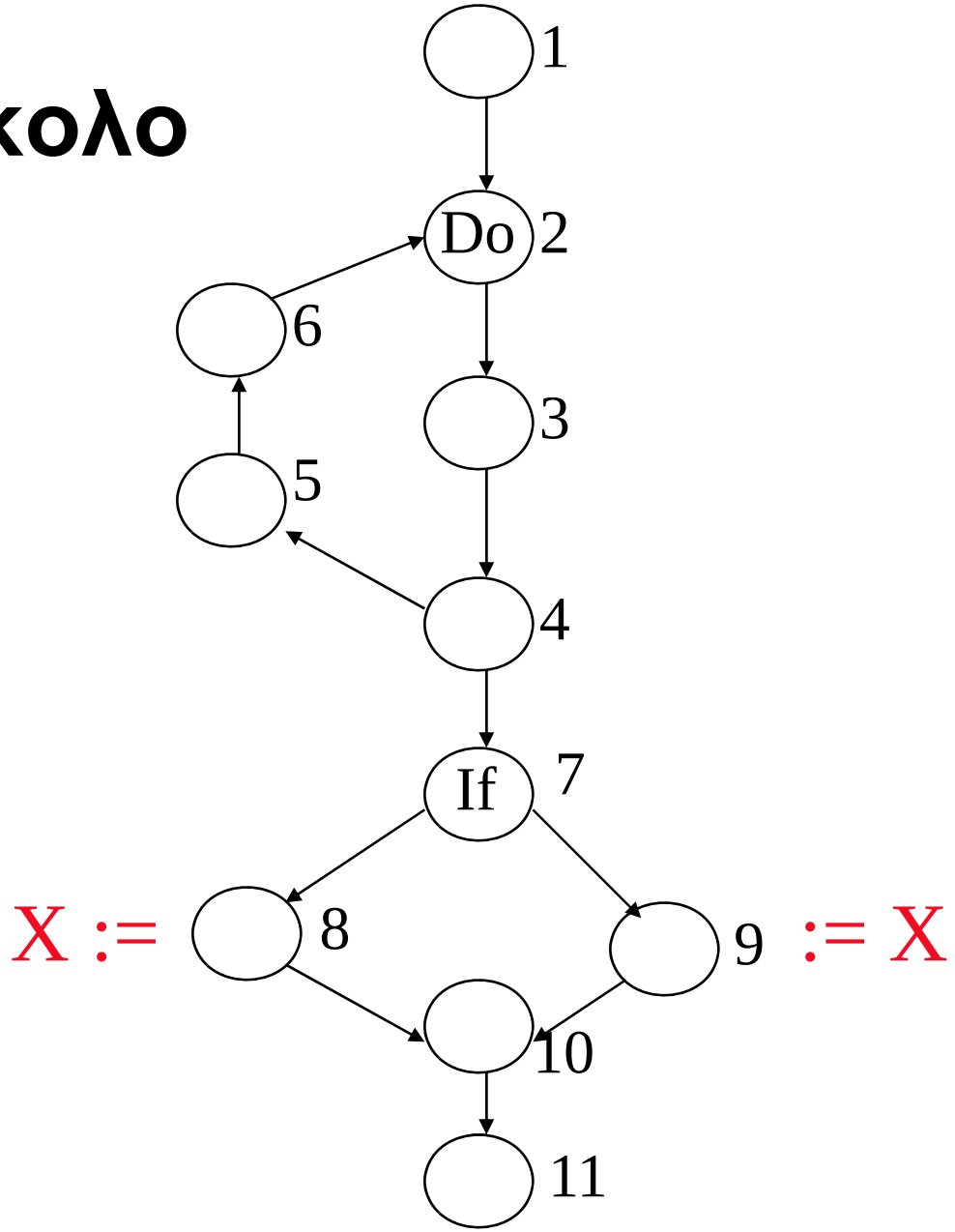
Κάτι δεν πάει καλά



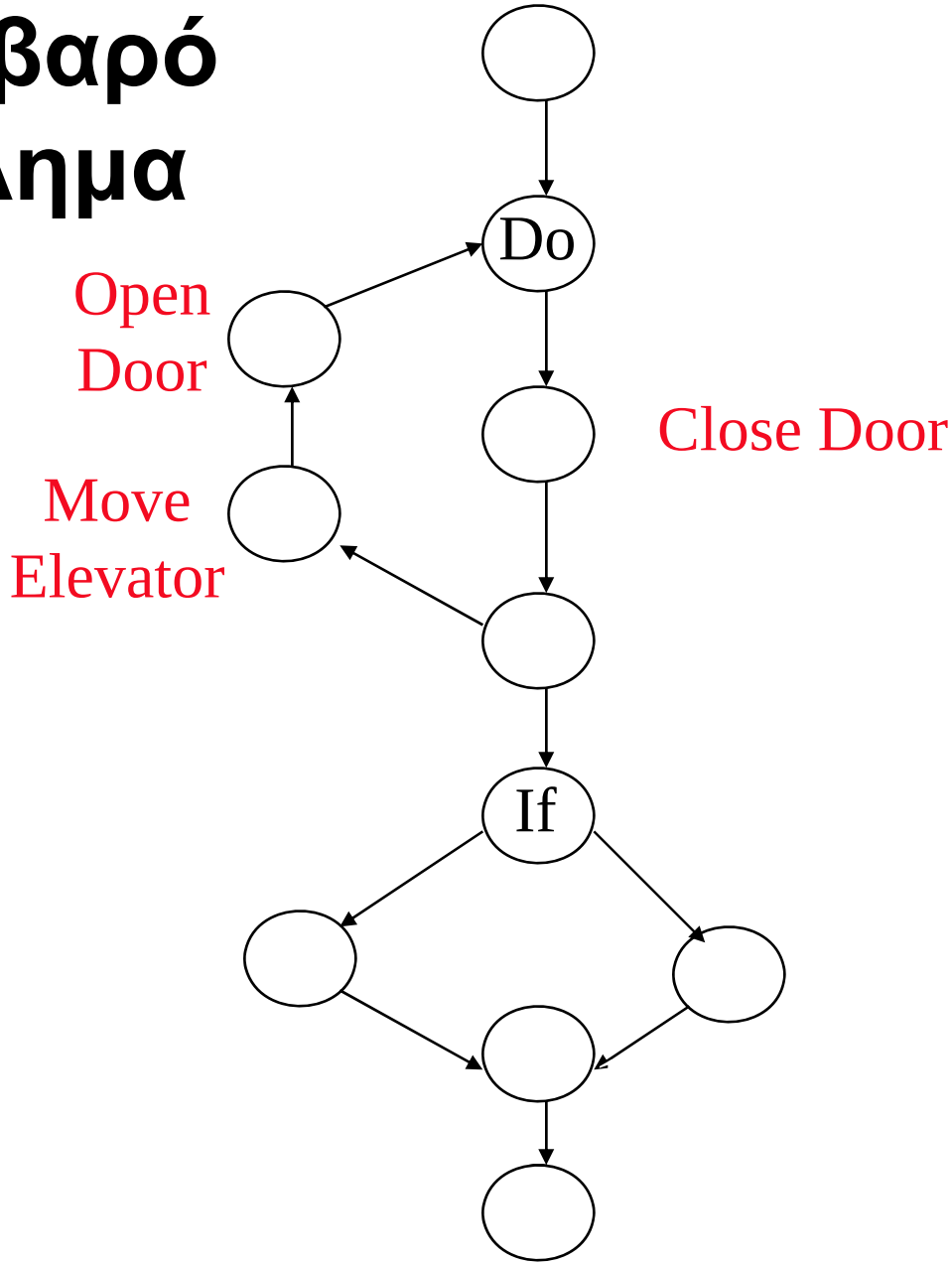
Καλύτερο



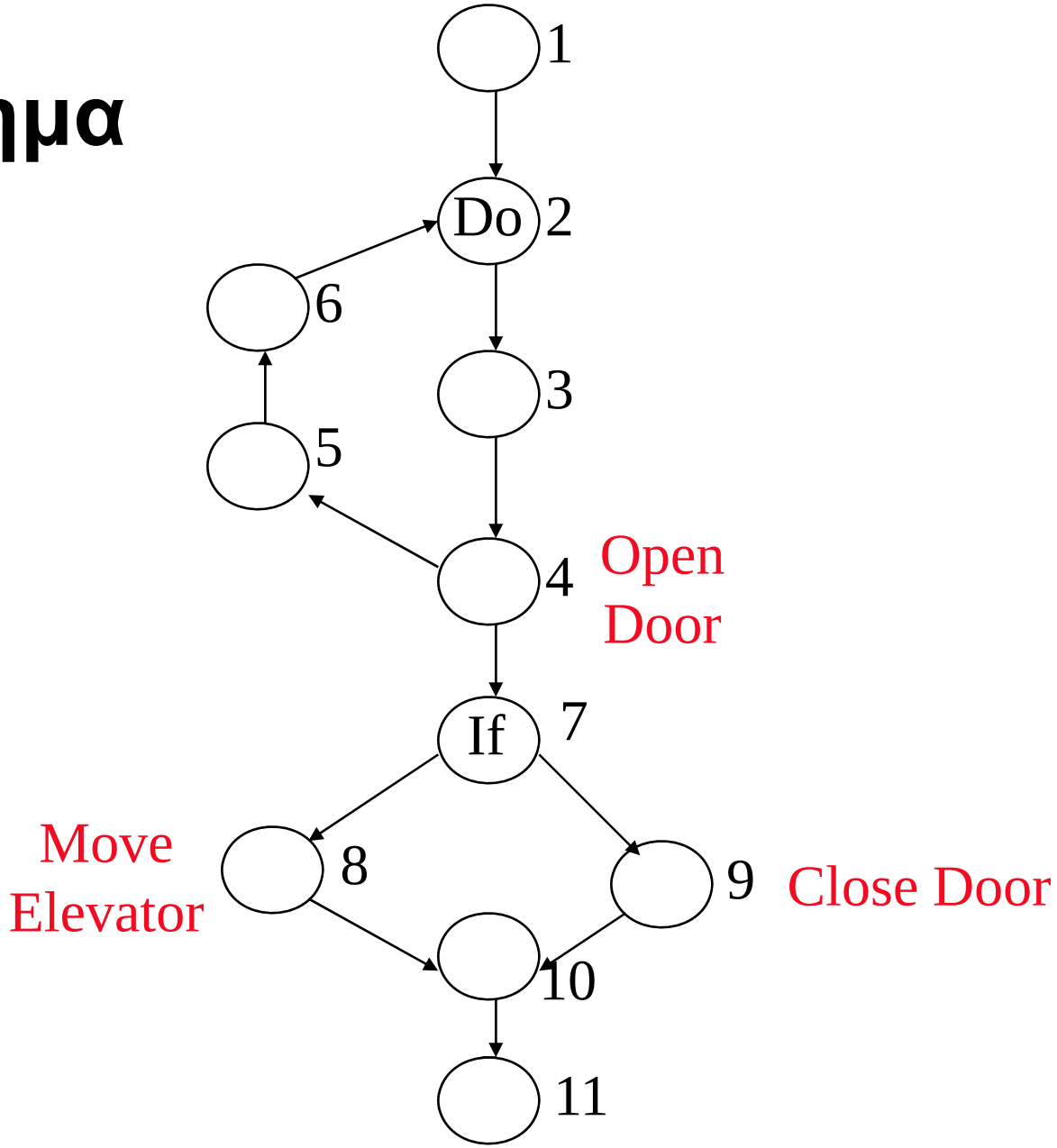
Πιο δύσκολο



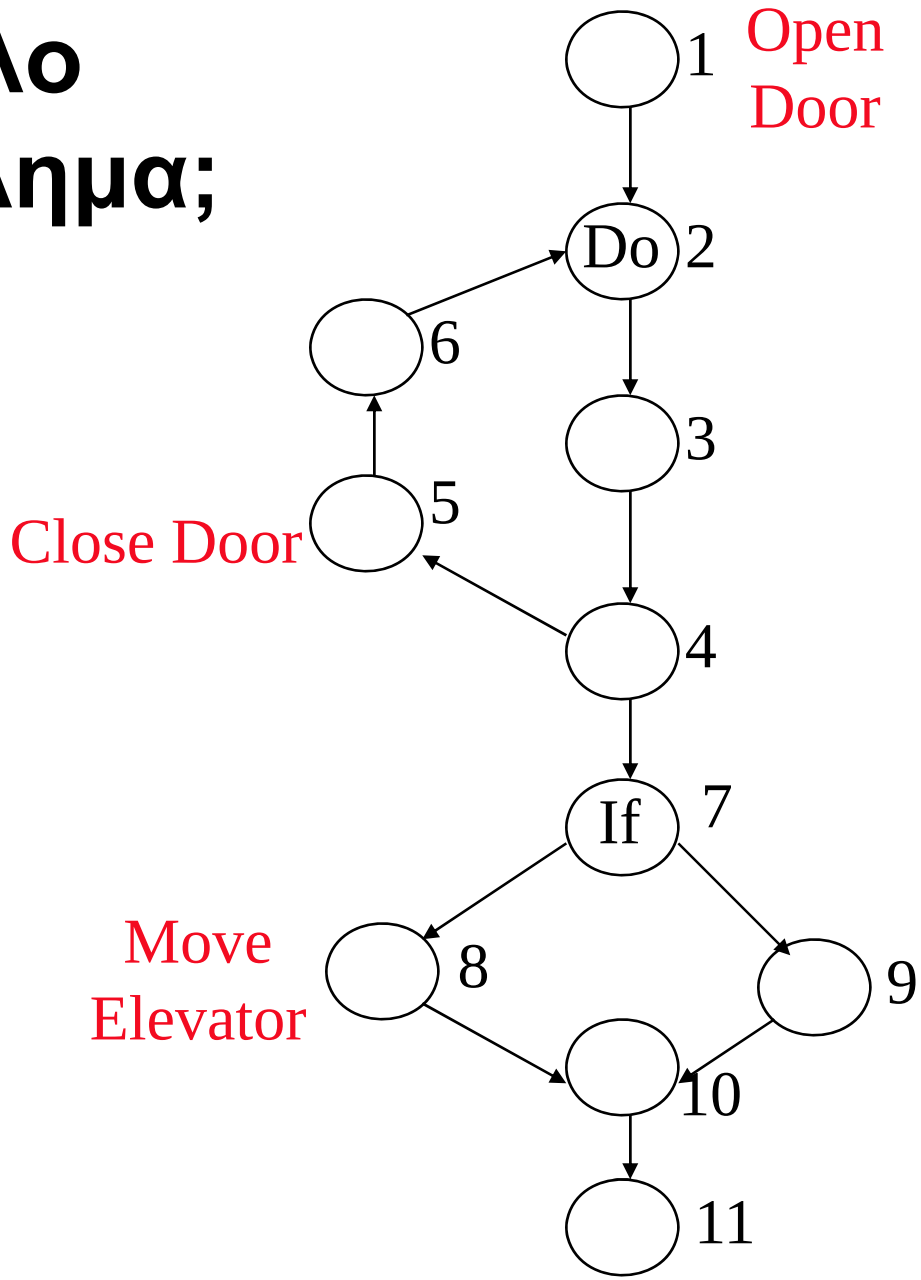
Πιο σοβαρό πρόβλημα



Πρόβλημα



Άλλο πρόβλημα;

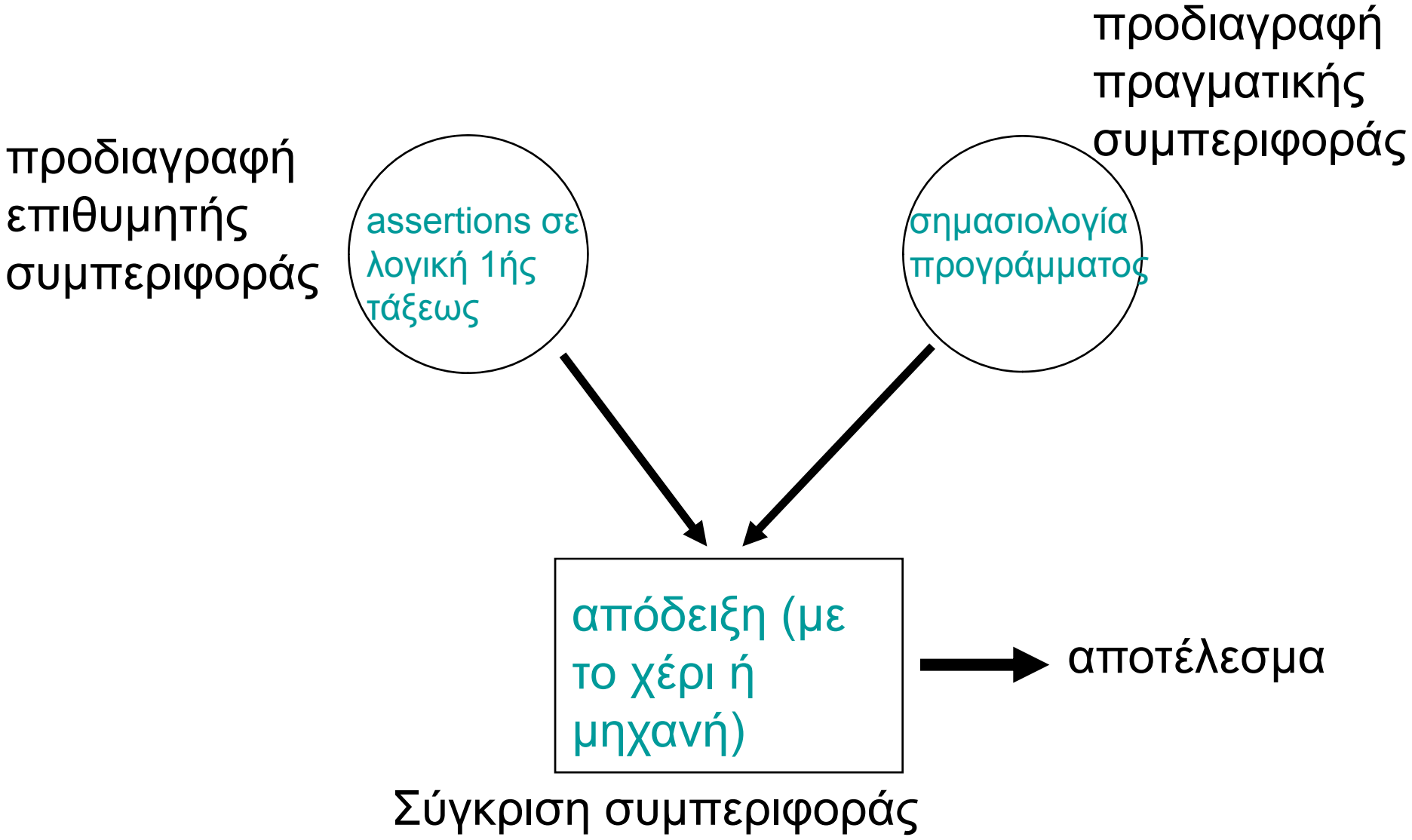


Απόδειξη ορθότητας

Τυπική επαλήθευση

- **Επιθυμητή συμπεριφορά:**
  - **συνήθως σε κάποια μαθηματική λογική**
- **Συμπεριφορά:**
  - **συνήθως συνάγεται από τη σημασιολογία της γλώσσας προγραμματισμού**
- **Σύγκριση:**
  - **απόδειξη θεωρήματος, χρησιμοποιώντας τα αξιώματα της λογικής**
  - **το αποτέλεσμα είναι σίγουρα «απόδειξη» αλλά όχι απαραίτητα «ορθότητας»**

Τυπική επαλήθευση



Η μέθοδος του Floyd (επαγωγικά assertions)

- Δείξε ότι κάθε τμήμα του προγράμματος συμπεριφέρεται σωστά
- Δείξε με επαγωγή ότι όλες οι ακολουθίες εκτέλεσης είναι σωστές
- Δείξε ότι το πρόγραμμα σταματάει

Assertions

- **Assertion: προδιαγραφή μιας συνθήκης που είναι αληθής σε κάποιο σημείο του προγράμματος**

- **Στη μέθοδο του Floyd γράφουμε assertions σε λογική πρώτης τάξης**

- **Τρεις τύποι assertions:**
  - **ο Αρχικό, A_S: στο αρχικό σημείο του προγράμματος**
  - **ο Τελικό, A_F: στο τελικό**
  - **ο ενδιάμεσό, A_i: ("loop invariants") σε διάφορα ενδιάμεσα σημεία του προγράμματος με βάση τον κανόνα:**

Κάθε επανάληψη του loop περνάει από τουλάχιστον ένα ενδιάμεσο assertion

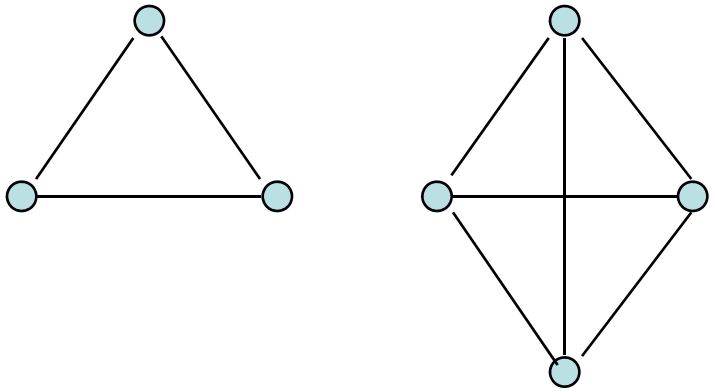
Αποτέλεσμα: Κάθε εκτέλεση του προγράμματος μπορεί να διαιρεθεί σε πεπερασμένο αριθμό τμημάτων κώδικα χωρίς loop που αρχίζει και τελειώνει με assertion

Μαθηματική επαγωγή: π.χ. πόσες ακμές στο C_n

Θεώρημα:

Αν $C_n = (V_n, E_n)$ είναι ένα πλήρες γράφημα με n κόμβους,

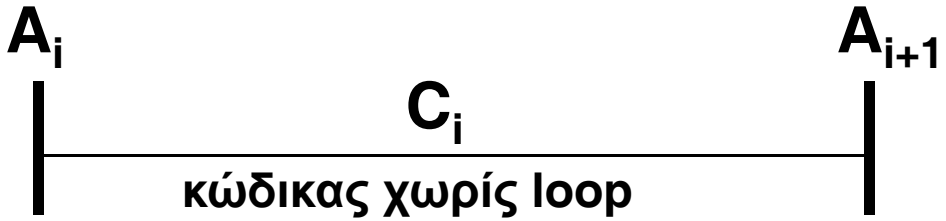
τότε $|E_n| = n * (n-1)/2$



- **Πώς το αποδεικνύουμε;**

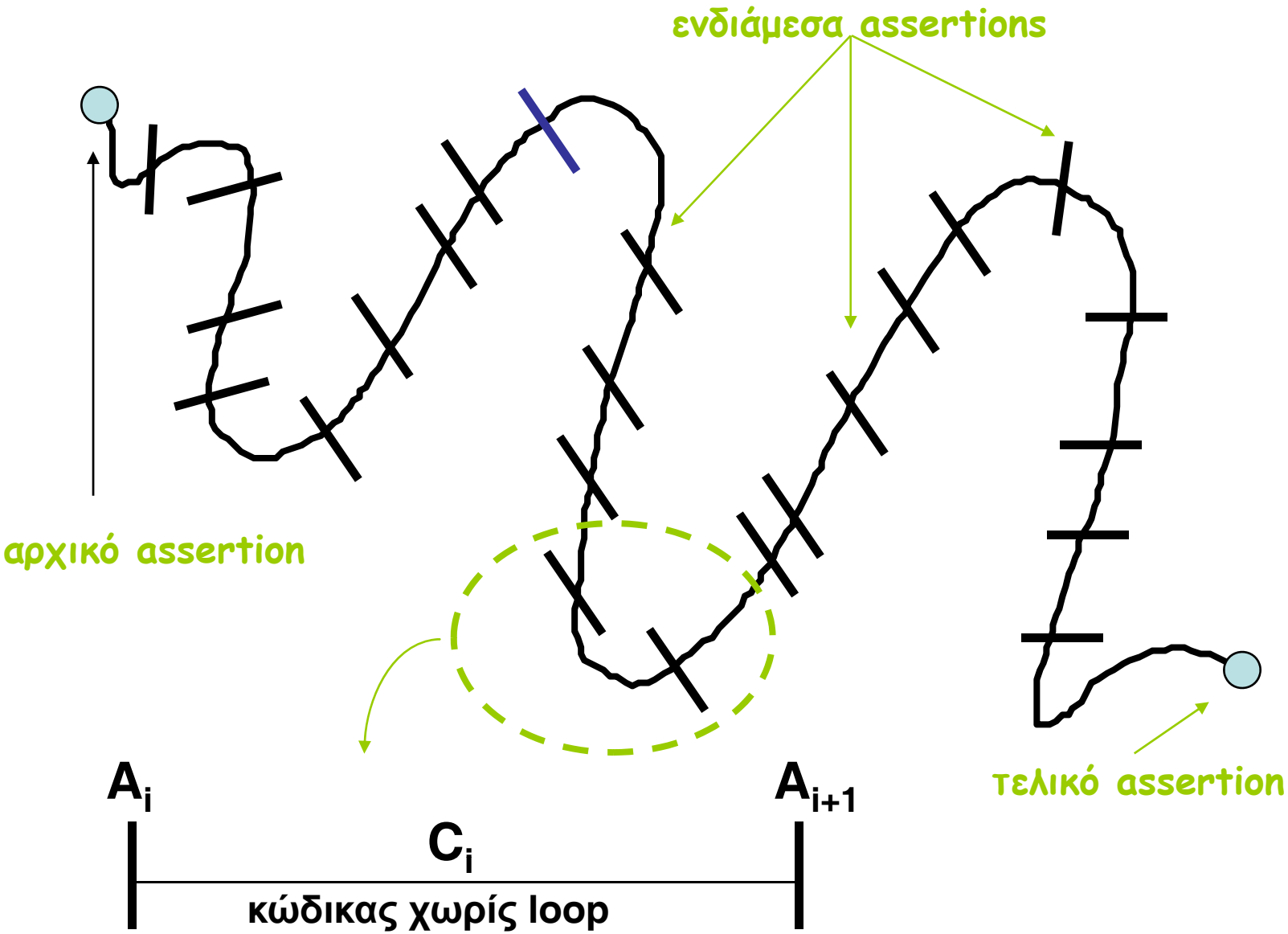
Βασικό συστατικό της μεθόδου του Floyd

- Αποδεικνύουμε ότι σε κάθε εκτέλεση αν το assertion A_i είναι αληθές τότε το A_{i+1} είναι αληθές
- $A_i \Rightarrow (C_i \Rightarrow A_{i+1})$



- Τελική υποχρέωση: αποδεικνύουμε ότι το πρόγραμμα σταματάει

Με σχήμα


ενδιάμεσα assertions
αρχικό assertion
τελικό assertion
A_i
A_{i+1}
C_i
κώδικας χωρίς loop

Loop Invariants

- **Το πρόβλημα είναι ότι οι πιθανές ακολουθίες εντολών σε ένα πρόγραμμα είναι άπειρες**
  - **λόγω loops**
- **Λύση: το loop invariant assertion A_i**
  - **αληθές για κάθε εκτέλεση του loop**
  - **συνδέει με διπλανά assertions**

Το loop invariant πρέπει να αναπαριστά την ουσία της δουλειάς που κάνει το loop

Παράδειγμα ταξινόμησης

```
procedure sort(values, size);
declare values real array[1000], temp real,
                    i, j, size integer;
ASSERT INITIAL
do for i = 1 to size-1
        do for j = i+1 to size
                if values[j] > values[i] then
                        temp := values[i];
                        values[i] := values[j];
                        values[j] := temp;
                ASSERT INNER
        end do;
        ASSERT OUTER
end do;
ASSERT FINAL
end sort;
```

Παράδειγμα ταξινόμησης

```
procedure sort(values, size);
declare values real array[1000], temp real,
                    i, j, size integer;
ASSERT INITIAL
do for i = 1 to size-1
        do for j = i+1 to size
                if values[j] > values[i] then
                        temp := values[i];
                        values[i] := values[j];
                        values[j] := temp;
                ASSERT INNER
        end do;
        ASSERT OUTER
end do;
ASSERT FINAL
end sort;
```

ASSERT OUTER

$$values[i] \geq values[k] \text{ for all } k \text{ such that } i < k \leq size$$

Παράδειγμα ταξινόμησης

```
procedure sort(values, size);
declare values real array[1000], temp real,
                    i, j, size integer;
ASSERT INITIAL
do for i = 1 to size-1
        do for j = i+1 to size
                if values[j] > values[i] then
                        temp := values[i];
                        values[i] := values[j];
                        values[j] := temp;
                ASSERT INNER
        end do;
        ASSERT OUTER
end do;
ASSERT FINAL
end sort;
```

ASSERT INNER

values[i] $\geq$ values[k] for all k such that $i < k \leq j$

Παράδειγμα ταξινόμησης

```
procedure sort(values, size);
declare values real array[1000], temp real,
                    i, j, size integer;
ASSERT INITIAL
do for i = 1 to size-1
        do for j = i+1 to size
                if values[j] > values[i] then
                        temp := values[i];
                        values[i] := values[j];
                        values[j] := temp;
                ASSERT INNER
        end do;
        ASSERT OUTER
end do;
ASSERT FINAL
end sort;
```

ASSERT FINAL

For all i, j, if $i < j \leq$ size, values[i] $\geq$ values [j]

Παράδειγμα ταξινόμησης

```
procedure sort(values, size);
declare values real array[1000], temp real,
                    i, j, size integer;
ASSERT INITIAL
do for i = 1 to size-1
        do for j = i+1 to size
                if values[j] > values[i] then
                        temp := values[i];
                        values[i] := values[j];
                        values[j] := temp;
                ASSERT INNER
        end do;
        ASSERT OUTER
end do;
ASSERT FINAL
end sort;
```

ASSERT INITIAL

size > 0

and

size $\leq$ 1000

Loop Invariants

- **Αν θέλετε να γίνετε σοβαροί προγραμματιστές είναι απαραίτητο να σκέφτεστε τα loop invariants για κάθε loop που γράφετε**
- **Όχι αναγκαία σε αυστηρά μαθηματικά**
- **Αλλά απαραίτητο να καταλαβαίνετε πώς το invariant στηρίζει την όλη λογική του προγράμματός σας**

Παρατηρήσεις

- Οι αποδείξεις είναι μακριές και συχνά σχολαστικές και δύσκολες
  - πολύ συχνά χρειάζονται ιδιότητες μαθηματικών, κλπ.
- Τα assertions δεν είναι εύκολα να παραχθούν σωστά
  - χρειάζονται βαθιά κατανόηση του προγράμματος
- Αυτόματα εργαλεία μπορούν να μας βοηθήσουν

Αυτόματα εργαλεία

- Τα εργαλεία τυπικής επαλήθευσης ή αποδείξεως θεωρημάτων είναι πιο παραγωγικά σε συνεργασία με το χρήστη
  - η δομή της απόδειξης έρχεται από τον άνθρωπο
  - το σύστημα επαληθεύει και κάνει «εύκολα» βήματα
  - ο άνθρωπος επεμβαίνει συχνά και καθοδηγεί

Αποτελέσματα

- Έχει επιτευχθεί η τυπική επαλήθευση πολύ μεγάλων προγραμμάτων
  - σημαντικό για πρωτόκολλα, μεταγλωττιστές, κλπ.
- Περιοχή με εξαιρετικά έντονη δραστηριότητα
- Η έρευνα σε δοκιμασία, ανάλυση, επαλήθευση προγραμμάτων είναι πλέον ένα συνεχές

δοκιμασία -> επαλήθευση model checking -> τυπική επαλήθευση