ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

Ένα παίγνιο πληθοπορισμού για την τυπική επαλήθευση λογισμικού

A crowdsourcing game for formal software verification

Καραμήτρου Ειρήνη

Επιβλέποντες καθηγητές: Δασκαλοπούλου Ασπασία

Ακρίτας Αλκιβιάδης

2016

ΠΕΡΙΛΗΨΗ

Σε όλα τα στάδια ανάπτυξης συστημάτων υλικού και λογισμικού, είναι απαραίτητος ο έλεγχος της ορθότητας. Η χρήση τυπικών μεθόδων στην επαλήθευση επιτρέπουν την απόδειξη της ορθότητας του συστήματος. Σε συστήματα υλικού, η τυπική επαλήθευση είναι μία εφικτή και, σε κάποιες περιπτώσεις, σχετικά απλή διαδικασία, κυρίως λόγω του πεπερασμένου μεγέθους τους, και γι' αυτό είναι περισσότερο διαδεδομένη. Στο λογισμικό, η διαδικασία αυτή, αν και απαραίτητη, είναι πιο δύσκολη και απαιτεί εξειδικευμένο προσωπικό και χρόνο. Στην παρούσα διπλωματική εργασία, εξετάζεται η χρήση πληθοπορισμού στην τυπική επαλήθευση προγράμματος λογισμικού. Για το σκοπό αυτό, αναπτύχθηκε και παρουσιάζεται εδώ η εφαρμογή "Boats On The River". Με την εφαρμογή αυτή ελέγχεται αν ένα πρόγραμμα σε Java μπορεί να παρουσιάσει σφάλμα, συγκεκριμένα NullPointerException. Η εφαρμογή δέχεται ως είσοδο το πρόγραμμα που πρόκειται να επαληθευθεί και μετατρέπει τον κώδικα σε ένα παιχνίδι. Με το παίξιμο του παιχνιδιού από το κοινό, γίνεται επαλήθευση του προγράμματος. Η έξοδος της εφαρμογής είναι ο κώδικας με την προσθήκη κατάλληλων σχολιασμών που προέκυψαν από το παίξιμο του παιχνιδιού και, αν υπάρχει σφάλμα, το σημείο στον κώδικα, που μπορεί να παρουσιαστεί. Με τον τρόπο αυτό, επιδιώκεται να μειωθεί το κόστος και ο χρόνος που απαιτείται για την τυπική επαλήθευση λογισμικού, αυξάνοντας τον αριθμό των ατόμων που κάνουν τυπική επαλήθευση, καθώς το παίξιμο του παιχνιδιού δεν απαιτεί εξειδικευμένες γνώσεις.

ABSTRACT

At every stage of hardware and software systems development, it is necessary to check the correctness. The use of formal methods in verification enables to obtain proof of system's correctness. In hardware systems, formal verification is a feasible and, in some cases, relatively simple process, mainly because of their finite size, and therefore more widespread. In software, this process, although necessary, it is more difficult and requires specially-trained stuff and time. In this thesis, it is examined the use of crowdsourcing in formal verification of software program. For this purpose, developed and present here the application "Boats On The River". The application checks whether a program in Java is error-free of NullPointerException errors. The application takes as input the program to be verified and converts the code into a game. The crowd, by playing the game, verifies the program. The output of the application is the code with the appropriate annotations generated by playing the game and, if there is an error, the code point, where it can occur. In this way, we aim to reduce the cost and the time required for formal software verification, by increasing the number of people who do formal verification, as well as playing the game does not require any specialized knowledge.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΕΡΙΛΗΨΗ 1

ABSTRACT 2

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ 3

1 ΕΙΣΑΓΩΓΗ 6

1.1) Έλεγχος ορθότητας μέσω δοκιμής (testing) 7

1.1.1) Γενικά 7

1.1.2) Στρατηγική του μαύρου κουτιού (black-box testing)- Στρατηγική του γυάλινου κουτιού (glass-box testing) 7

1.2) Είναι αρκετός ο έλεγχος μέσω δοκιμής; 8

2 ΤΥΠΙΚΗ ΕΠΑΛΗΘΕΥΣΗ 9

2.1) Εισαγωγή 9

2.2) Τυπική επαλήθευση vs έλεγχος με δοκιμή 9

2.3) Επαλήθευση και επικύρωση (V&V) 10

2.4) Τυπική επαλήθευση 10

2.4.1) Έλεγχος μοντέλων 11

2.4.1.1) Μοντελοποίηση 13

2.4.2) Deductive verification 15

2.4.3) Στατική ανάλυση 17

2.5) Επιλογή κατάλληλης μεθόδου 18

3 ΤΥΠΙΚΗ ΕΠΑΛΗΘΕΥΣΗ ΛΟΓΙΣΜΙΚΟΥ 19

3.1) Τυπική επαλήθευση software vs hardware 19

3.2) Τυπική επαλήθευση λογισμικού - Γενικά 20

3.2.1) Αφηρημένη στατική ανάλυση (Abstract Static Analysis) 20

3.2.1.1) Γενικά και ορολογία 22

3.2.1.2) Αφηρημένη ερμηνεία (Abstract Interpretation) 22

3.2.1.3) Αφηρημένα αριθμητικά πεδία (Numerical Abstract Domains) 23

3.2.1.4) Ανάλυση με πεδία που αφορούν τη στοίβα 24

3.2.1.5) Εργαλεία στατικής ανάλυσης 25

3.2.2) Έλεγχος μοντέλων λογισμικού (Software Model Checking) 25

3.2.2.1) Εισαγωγή 25
3.2.2.2) Ρητός και συμβολικός έλεγχος μοντέλων (Explicit and Symbolic Model Checking) 26
3.2.2.3) Αφαίρεση κατηγορήματος (Predicate Abstraction) 27
3.2.2.4) Εργαλεία ελέγχου μοντέλων 29
3.2.3) Έλεγχος φραγμένων μοντέλων (Bounded Model Checking-BMC) 30
3.2.3.1) Υπόβαθρο 30
3.2.3.2) Ξετύλιγμα ολόκληρου του προγράμματος μονομιάς 31
3.2.3.3) Ξετύλιγμα βρόχων ξεχωριστά 31
3.2.3.4) Ολοκληρωμένος BMC για λογισμικό 32
3.2.3.5) Εργαλεία που υλοποιούν τον BMC 33
3.3) Επιλογή κατάλληλης προσέγγισης 34
4 ΜΙΑ ΑΛΛΗ ΠΡΟΣΕΓΓΙΣΗ 35
4.1) Εισαγωγή 35
4.2) Έλεγχος τύπων (type checking) 35
4.2.1) Επαλήθευση μέσω ελέγχου τύπων (verification via type-check) 37
4.2.2) Σχολιασμοί γενικά (annotations) 37
4.2.3) Σχολιασμοί τύπων (type annotations) 38
4.2.4) Pluggable ελεγκτές τύπων (type chechers) - Μεθοδολογία δημιουργίας και χρήσης 39
4.2.5) Πλεονεκτήματα και μειονεκτήματα 40
4.3) Πληθοπορισμός (Crowdsourcing) 41
4.3.1) Χρήση crowdsourcing στην τυπική επαλήθευση (Crowd-Sourced Formal Verification CSFV- Darpa project) 41
4.3.2) Pipe Jam Game 45
5 BOATS ON THE RIVER-ΠΕΡΙΓΡΑΦΗ/ΣΧΕΔΙΑΣΜΟΣ/ΠΡΟΣΟΜΟΙΩΣΗ 48
5.1) Εισαγωγή 48
5.2) Επιλογές μεθόδων, εργαλείων και αντικειμένων επαλήθευσης 49
5.2.1) Έλεγχος τύπων 49
5.2.2) Null-pointer error (NPE) 49
5.2.3) Σχολιασμοί (Annotations) 50
5.2.4) Αντικείμενα επαλήθευσης 50
5.3) Περιγραφή/προσομοίωση σχεδίασης 51
5.3.1) Περιγραφή παιχνιδιού (Boats on the river game) 51
5.3.2) Περιγραφή αντιστοίχησης του κώδικα στο παιχνίδι 53

5.3.3) Περιγραφή μετατροπής του τελικού board σε κώδικα με annotations ή/και αναφορά σφάλματος 57

6 BOATS ON THE RIVER- ΥΛΟΠΟΙΗΣΗ 60

6.1) Εισαγωγή 60

6.2) Περιγραφή αρχείων/κλάσεων 60

i. MyBrooks.java 60

ii. MyBoats.java 61

iii. StringDb.java 62

iv. FileReadWrite.java 62

v. GamePanel.java 63

vi. GameTester.java 65

7 ΣΥΜΠΕΡΑΣΜΑΤΑ-ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ 66

7.1) Συμπεράσματα 66

7.2) Μελλοντικές επεκτάσεις 66

8 ΒΙΒΛΙΟΓΡΑΦΙΑ 68

8.1) Βιβλία 68

8.2) Δημοσιεύσεις 68

8.3) Διαδίκτυο 70

1 ΕΙΣΑΓΩΓΗ

Κατά τη διαδικασία ανάπτυξης ενός συστήματος υλικού ή λογισμικού, ιδιαίτερα σημαντικό ρόλο παίζει ο έλεγχος της ορθής λειτουργίας του συστήματος, δηλαδή η εξέτασή του, ώστε να βρεθούν και να διορθωθούν τυχόν σφάλματα κατά της σχεδίαση και την υλοποίησή του. Τα παραδείγματα από αστοχίες σε συστήματα είναι πολλά, με ποικίλες συνέπειες, αναδεικνύοντας έντονα την ανάγκη απουσίας τους. Ενδεικτικά αναφέρονται τα παρακάτω:

4 Ιουνίου 1996: Ο πύραυλος Ariane 5 εξερράγη 37 δευτερόλεπτα μετά την εκτόξευσή του. Η έκρηξη προκλήθηκε από λάθος στο λογισμικό του SRI (αδρανειακό σύστημα αναφοράς). Η μετατροπή μιας τιμής 64-bit κινητής υποδιαστολής σε έναν 16-bit προσημασμένο ακέραιο προκάλεσε εξαίρεση, καθώς ο αριθμός κινητής υποδιαστολής ήταν μεγαλύτερος από 32767, που είναι ο μέγιστος που μπορεί να αποθηκεύσει ο ακέραιος 16-bit. Το συνολικό κόστος υπολογίζεται σε πάνω από 500.000.000$.

21 Σεπτεμβρίου 1997: Το σκάφος του αμερικανικού πολεμικού ναυτικού USS Yorktown έμεινε ακυβέρνητο για σχεδόν 3 ώρες. Η αιτία ήταν ένα μηδενικό στο σύστημα της βάσης δεδομένων, το οποίο μετά χρησιμοποιήθηκε σε διαίρεση. Έτσι έγινε υπερχείλιση στη βάση δεδομένων και το λογισμικό κατέρρευσε. Το σύστημα δεν έκανε έλεγχο των δεδομένων κατά την εισαγωγή τους στη βάση δεδομένων, ούτε πριν την επεξεργασία τους και το λειτουργικό σύστημα Windows NT 4.0 στο οποίο «έτρεχε», δεν προστάτευε απέναντι σε διαίρεση με το 0.

14 Αυγούστου 2003: 55 εκατομμύρια άνθρωποι στις βορειοανατολικές ΗΠΑ και το Οντάριο του Καναδά μένουν χωρίς ρεύμα. Μια ελέγξιμη διακοπή σε τοπικό επίπεδο επηρέασε τελικά ολόκληρο το δίκτυο, με αποτέλεσμα αρκετές περιοχές να μείνουν χωρίς ρεύμα για μέρες. Δύο τμήματα στο λογισμικό του συστήματος ειδοποίησης στο δωμάτιο ελέγχου ανταγωνίζονταν για τους ίδιους πόρους (race condition), με αποτέλεσμα το σύστημα συναγερμού να «παγώσει» και να σταματήσει να δίνει ειδοποιήσεις. Έτσι, το σύστημα κατέρρευσε χωρίς να γίνει αντιληπτό, χωρίς ηχητικές και οπτικές ειδοποιήσεις, και έτσι δεν υπήρξε έγκαιρη πληροφόρηση για την ανάγκη αναδιανομής στο σύστημα μεταφοράς.

Φαίνεται και από τα παραπάνω, ότι η ορθότητα ενός συστήματος, ιδίως μεγάλης κλίμακας, είναι ιδιαίτερα επιθυμητή και αναγκαία. Βέβαια, σε συστήματα μεγάλης κλίμακας, ο έλεγχος της ορθότητας είναι δύσκολος και απαιτεί χρόνο και εξειδικευμένα άτομα.

1.1) Έλεγχος ορθότητας μέσω δοκιμής (testing)

1.1.1) Γενικά

Για να επαληθεύσουμε τη σωστή λειτουργία ενός συστήματος ο παραδοσιακός τρόπος είναι ο έλεγχος μέσω δοκιμής (testing). Αυτή είναι και η πιο απλή προσέγγιση που όλοι όσοι ασχολούνται με την ανάπτυξη προϊόντων υλικού ή λογισμικού σίγουρα χρησιμοποιούν. Βασίζεται σε εκτεταμένο έλεγχο της συμπεριφοράς του συστήματος απέναντι σε διάφορες εισόδους μέσω δοκιμής. Με αυτόν τον έλεγχο αυξάνεται η ποιότητα και η αξιοπιστία του συστήματος. Τα βασικά βήματα του ελέγχου μέσω δοκιμής είναι τα εξής:

1) Επιλογή των εισόδων απέναντι στις οποίες θα εξεταστεί το σύστημα.

2) Προσδιορισμός του αναμενόμενου αποτελέσματος.

3) Προσομοίωση του συστήματος με τις εισόδους που προσδιορίσαμε και καταγραφή των αποτελεσμάτων/εξόδων.

4) Εξέταση των αποτελεσμάτων/εξόδων της προσομοίωσης και σύγκριση με τα αναμενόμενα.

Είναι αναγκαίο η επιλογή των εισόδων, απέναντι στις οποίες θα εξεταστεί το σύστημα, να είναι εύστοχη, ώστε να καλυφθούν όσο το δυνατόν περισσότερες περιπτώσεις. Η εξέταση όλων των εφικτών περιπτώσεων είναι αδύνατη, λόγω του απαγορευτικά μεγάλου μεγέθους τους, ακόμα και σε σχετικά μικρά συστήματα. Οι βασικές στρατηγικές στο testing, όταν εστιάσουμε στο software, είναι δύο:

- Στρατηγική του μαύρου κουτιού (black-box testing)
- Στρατηγική του γυάλινου κουτιού (glass-box testing)

1.1.2) Στρατηγική του μαύρου κουτιού (black-box testing)- Στρατηγική του γυάλινου κουτιού (glass-box testing)

Με την πρώτη μέθοδο, ο έλεγχος γίνεται χωρίς να παίρνουμε υπόψη μας την εσωτερική υλοποίηση, εξετάζοντας το σύστημα ως ένα μαύρο κουτί. Η επιλογή των εισόδων, επομένως, είναι ανεξάρτητα από την υλοποίηση. Ο έλεγχος γίνεται με την μελέτη των αποτελεσμάτων που αντιστοιχούν σε συγκεκριμένες εισόδους, σε σύγκριση με τα αναμενόμενα. Ο στόχος είναι να διασφαλιστεί ότι εισάγεται κάθε είδους είσοδος και το σύστημα ανταποκρίνεται σε αυτές παράγοντας τις αναμενόμενες εξόδους. Το βασικό πλεονέκτημα αυτής της μεθόδου είναι το ότι δεν έχουμε περιορισμούς που επιβάλλονται από την εσωτερική δομή και λογική. Βέβαια σε πολλές περιπτώσεις ένας πλήρης έλεγχος με αυτή τη μέθοδο είναι αδύνατος, καθώς γίνεται ιδιαίτερα δύσκολο να επιλεγούν αντιπροσωπευτικές περιπτώσεις ελέγχου, ειδικά εφόσον η εσωτερική δομή είναι άγνωστη. Για την επιλογή εύστοχων περιπτώσεων ελέγχου, έχουν

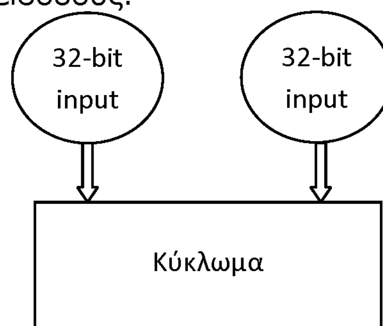
αναπτυχτεί διάφορες προσεγγίσεις. Ενδεικτικά αναφέρονται οι προσεγγίσεις ισοδύναμης διαμέρισης, συνοριακών τιμών, αίτιου-αποτελέσματος.

Με τη στρατηγική του γυάλινου κουτιού, ο έλεγχος γίνεται λαμβάνοντας υπόψη μας την εσωτερική υλοποίηση του συστήματος. Έτσι, η επιλογή των περιπτώσεων ελέγχου γίνεται μετά από μελέτη της δομής και της εσωτερικής υλοποίησης του υπό εξέταση συστήματος, ώστε να ελεγχθεί με διαφορετικούς τρόπους. Εφόσον μελετάται η εσωτερική δομή και υλοποίηση, απαιτείται ο έλεγχος να γίνεται από εξειδικευμένα άτομα.

Συνήθως ακολουθούνται και οι δύο στρατηγικές ή κάποιος συνδυασμός τους, για τον έλεγχο των συστημάτων. Η κύρια διαφορά τους βρίσκεται στο πού εστιάζουν. Η στρατηγική του μαύρου κουτιού εστιάζει στα αποτελέσματα: αν αυτά είναι τα προσδοκώμενα, τότε ο έλεγχος είναι επιτυχής. Η στρατηγική γυάλινου κουτιού από την άλλη, εστιάζει στις λεπτομέρειες. Χρειάζεται να ελέγξει το εσωτερικό του συστήματος και το σύνολο των διαφορετικών περιπτώσεων/«μονοπατιών» ώστε να ολοκληρωθεί ο έλεγχος. Εφαρμόζεται κυρίως σε χαμηλότερα επίπεδα ελέγχου, ενώ αντίθετα, η στρατηγική μαύρου κουτιού εφαρμόζεται κυρίως όταν έχουμε να κάνουμε με πιο ψηλό επίπεδο.

1.2) Είναι αρκετός ο έλεγχος μέσω δοκιμής;

Ο έλεγχος, λοιπόν, είναι απαραίτητος και ενισχύει σημαντικά την αξιοπιστία και την ποιότητα του συστήματος. Όμως, μέσω του ελέγχου (testing), καλύπτουμε κάποιες περιπτώσεις, τις πιο κύριες. Το μέγεθος των συστημάτων κάνει απαγορευτικό τον έλεγχο όλων των περιπτώσεων, οποιαδήποτε και από τις δύο στρατηγικές ακολουθήσουμε. Ας πάρουμε ένα απλό παράδειγμα που αφορά το hardware, όπου θέλουμε να ελέγξουμε την ορθή λειτουργία ενός κυκλώματος, που παίρνει δύο 32-bit εισόδους.



Σχήμα 1.1

Σ' αυτήν την περίπτωση, η δοκιμή όλων των πιθανών συνδυασμών εισόδων είναι πρακτικά αδύνατη, καθώς μιλάμε για $2^{32}*2^{32}=2^{64}$ πιθανών περιπτώσεων.

Επομένως, δεν μπορούμε ποτέ να εγγυηθούμε ότι το σύστημα είναι ορθό στο σύνολό του.

Ο έλεγχος μέσω δοκιμής (testing) μπορεί να βρει την παρουσία λαθών, όχι όμως και την απουσία τους! (Dijktra)

2 ΤΥΠΙΚΗ ΕΠΑΛΗΘΕΥΣΗ

2.1) Εισαγωγή

Είναι απαραίτητη μια διαφορετική προσέγγιση του προβλήματος του ελέγχου, εκτός από τη δοκιμή. Σε αυτήν την κατεύθυνση έχει αναπτυχτεί η προσέγγιση της τυπικής επαλήθευσης (Formal Verification). Κύριο καθήκον της είναι να αποδείξει την ορθότητα (ή μη) του συστήματος και όχι μόνο να εντοπίσει σφάλματα.

Η τυπική επαλήθευση είναι η απόδειξη της ορθότητας ενός συστήματος σύμφωνα με κάποια τυπική προδιαγραφή (formal specification) ή ιδιότητα, με τη χρήση τυπικών μεθόδων των μαθηματικών. Οι τυπικές μέθοδοι (formal methods) αφορούν τεχνικές και εργαλεία που βασίζονται στα μαθηματικά και την τυπική λογική. Είναι «αυστηρές» και χρησιμοποιούνται για τον προσδιορισμό των προδιαγραφών ενός συστήματος, τον σχεδιασμό και την επαλήθευσή του.

Η χρήση τυπικών μεθόδων δεν είναι κάτι καινούριο, χρονολογούνται από τις πρώτες κιόλας ημέρες της επιστήμης των υπολογιστών (Floyd, Hoare, Naur), με αναφορές που πηγαίνουν πίσω ακόμα στους Von Neumann και Turing. Η δύναμη όμως των σημερινών υπολογιστών, κάνει εφικτή την εφαρμογή τους σε μεγάλης κλίμακας βιομηχανικά project.

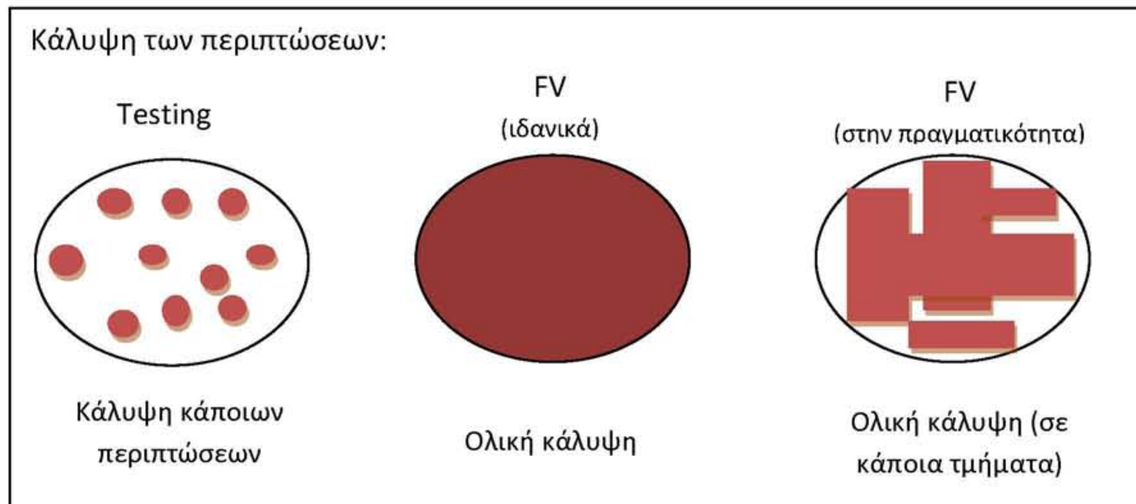
2.2) Τυπική επαλήθευση vs έλεγχος με δοκιμή

Όπως αναφέραμε και παραπάνω, με τον έλεγχο μέσω της δοκιμής δεν μπορούμε να εγγυηθούμε την απουσία σφαλμάτων, σε αντίθεση με τον τυπικό έλεγχο και επαλήθευση, όπου σκοπός μας είναι η απόδειξη της ορθότητας (ή μη). Γίνεται όμως πιο εύκολα, ακόμα και από μη εξειδικευμένα άτομα και με λιγότερο κόστος.

	F.V.	**Testing**
Εύρεση σφαλμάτων	Μέτρια	Καλή
Απόδειξη ορθότητας	Καλή	-
Κόστος	Υψηλό	Χαμηλό

Πίνακας 2.1

Ιδανικά, με αυτή τη μέθοδο καλύπτονται όλες οι περιπτώσεις του συστήματος, σε αντίθεση με τον έλεγχο μέσω δοκιμής (**Σχήμα 2.1**).



Σχήμα 2.1

2.3) Επαλήθευση και επικύρωση (V&V)

Η επαλήθευση (verification) είναι η μία πλευρά του ελέγχου της ποιότητας και αξιοπιστίας του συστήματος. Η άλλη πλευρά (με την οποία δεν θα ασχοληθούμε αναλυτικά) είναι η επικύρωση (validation). Συχνά αναφέρονται μαζί, ως V&V.

<u>Επικύρωση</u>:
⇒ Προσπαθούμε όντως να κατασκευάσουμε το σωστό σύστημα;
<u>Επαλήθευση</u>:
⇒ Κατασκευάσαμε αυτό που προσπαθούσαμε να κατασκευάσουμε;

Δηλαδή, ενώ στην επαλήθευση μας απασχολεί το αν το σύστημα συμφωνεί με τις προδιαγραφές του, στην επικύρωση εξετάζουμε το αν οι προδιαγραφές του συστήματος συμπίπτουν με τις πραγματικές ανάγκες του χρήστη. Και οι δύο διαδικασίες είναι απαραίτητες, ενώ η μία είναι συμπληρωματική της άλλης.

2.4) Τυπική επαλήθευση

Για το υπό έλεγχο σύστημα X, ζητούμενο του τυπικού ελέγχου και επαλήθευσης είναι να αποδείξει ότι:

- Το X κάνει αυτό που είναι σχεδιασμένο να κάνει
- Και <u>τίποτα παραπάνω</u>

Το δεύτερο είναι ίσως και το πιο δύσκολο, καθώς χρειάζεται να απαριθμηθούν όλες οι περιπτώσεις που μπορεί να πάει οτιδήποτε στραβά. Ακόμα, η απόδειξη ότι το X δεν έχει μία ιδιότητα, συνήθως είναι πιο δύσκολη από την απόδειξη ότι έχει μία ιδιότητα.

Η επαλήθευση γίνεται παρέχοντας μία τυπική απόδειξη ενός αφηρημένου μαθηματικού μοντέλου του συστήματος. Χρησιμοποιούμε δηλαδή μία μαθηματική δομή για να μοντελοποιήσουμε το σύστημα και εξάγουμε τις

επιθυμητές ιδιότητες ως θεωρήματα σε σχέση με τη δομή. Ανάλογα με το σύστημα που εξετάζουμε, υπάρχουν και κατάλληλα μαθηματικά μοντέλα που προτιμούμε, για παράδειγμα οι μηχανές πεπερασμένων καταστάσεων (FSMs), ή τα συστήματα μετάβασης (transition systems).

Οι κύριες προσεγγίσεις της τυπικής επαλήθευσης είναι δύο:

- Ο έλεγχος μοντέλων (model checking)
- Επαγωγικές μέθοδοι (deductive verification) ή αλλιώς η απόδειξη θεωρήματος (theorem proving)

Ανάλογα με το τι σύστημα εξετάζουμε, επιλέγουμε συνήθως και την πιο κατάλληλη προσέγγιση. Για παράδειγμα, στον έλεγχο του hardware, χρησιμοποιείται ευρέως ο έλεγχος μοντέλων, ενώ οι επαγωγικές μέθοδοι εφαρμόζονται περισσότερο στον έλεγχο του software ή σε πρωτόκολλα ασφάλειας.

Δίπλα σε αυτές τις προσεγγίσεις, είναι και η στατική ανάλυση, που χρησιμοποιείται πλατιά στην επαλήθευση μεταγλωττιστών (compilers).

2.4.1) Έλεγχος μοντέλων

Η ανάπτυξη της μεθόδου του ελέγχου μοντέλων για την επαλήθευση συστημάτων, προέρχεται από την ανεξάρτητη δουλειά των ζευγαριών: Clarke-Emerson και Queille-Sifakis. Ο όρος επινοήθηκε από τους Clarke-Emerson.

Ο έλεγχος μοντέλων είναι μία αυτοματοποιημένη μέθοδος επαλήθευσης όπου:

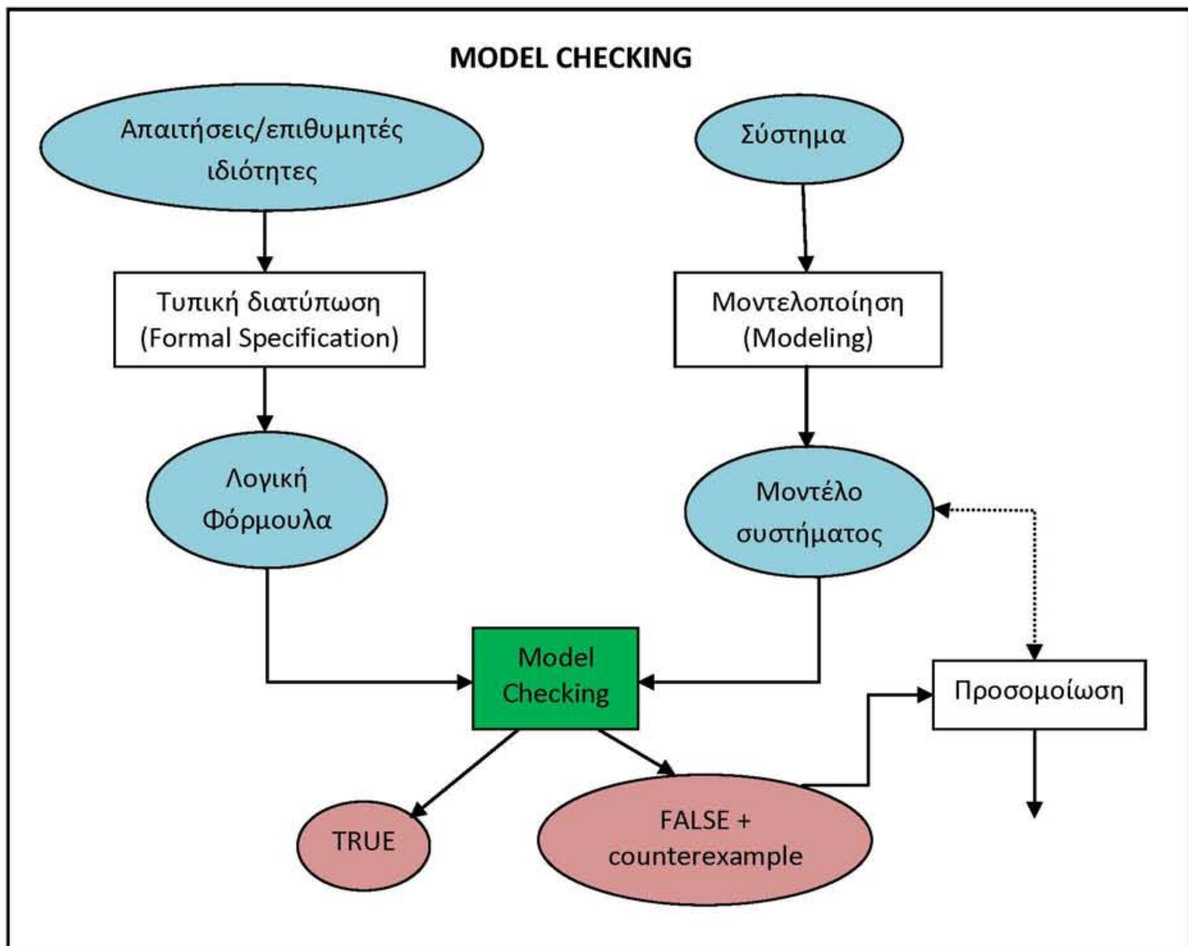
Ο χρήστης παρέχει:

- Ένα μοντέλο του συστήματος (πεπερασμένων καταστάσεων).
- Μία λογική φόρμουλα που περιγράφει τις επιθυμητές ιδιότητες.

Ο αλγόριθμος ελέγχου μοντέλων :

- Ελέγχει συστηματικά αν το μοντέλο ικανοποιεί τη φόρμουλα.
- Αν η ιδιότητα δεν ικανοποιείται, παράγεται ένα αντιπαράδειγμα (counterexample).

Ως σημείο εκκίνησης έχουμε ένα μοντέλο του υπό εξέταση συστήματος. Έτσι, έχουμε ως δεδομένο ότι η επαλήθευση με τεχνικές που βασίζονται σε μοντέλα, είναι μόνο τόσο καλές, όσο το μοντέλο του συστήματος. Οι επιθυμητές ιδιότητες απαιτείται να είναι διατυπωμένες με ακρίβεια και σαφήνεια. Η λογική φόρμουλα στην οποία αποτυπώνονται οι επιθυμητές ιδιότητες, δείχνει *τι* πρέπει να κάνει το σύστημα και *τι δεν* πρέπει. Για συστήματα υλικού και λογισμικού, χρησιμοποιείται συχνά η χρονική λογική (temporal logic) γι' αυτόν τον σκοπό, που μπορεί να δείξει τη συμπεριφορά του συστήματος όχι στατικά, αλλά στον χρόνο. Το μοντέλο του συστήματος δείχνει το *πώς* συμπεριφέρεται το σύστημα. Ο ελεγκτής μοντέλων εξετάζει όλες τις σχετικές καταστάσεις του συστήματος,



Σχήμα 2.2: Model Checking

ώστε να ελέγξει αν ικανοποιούν την επιθυμητή ιδιότητα. Αν βρει μία κατάσταση που παραβιάζει την ιδιότητα, τότε παρέχει ένα αντιπαράδειγμα που δείχνει πώς μπορεί να φτάσει το μοντέλο στην ανεπιθύμητη κατάσταση. Το αντιπαράδειγμα δείχνει, δηλαδή, το μονοπάτι εκτέλεσης που οδηγεί από την αρχική κατάσταση του συστήματος, σε αυτήν που παραβιάζει την υπό εξέταση ιδιότητα. Μέσω της προσομοίωσης, μπορεί να αναπαραχθεί το συγκεκριμένο σενάριο που οδηγεί σε παραβίαση, να βρεθεί το λάθος και να διορθωθεί κατάλληλα (**σχήμα 2.2**).

Ο έλεγχος μοντέλων αναπτύχτηκε για την ανάλυση συστημάτων υλικού και πρωτόκολλων επικοινωνίας. Στη συνέχεια οι αλγόριθμοι ελέγχου μοντέλων προσαρμόστηκαν ώστε να έχουν εφαρμογή και στην ανάλυση του λογισμικού.

Η μεγαλύτερη πρόκληση επαληθεύοντας ένα σύστημα με αυτή τη μέθοδο είναι η αντιμετώπιση του προβλήματος της έκρηξης καταστάσεων (state-space explosion problem). Σε μεγάλα συστήματα, ή σε συστήματα με δεδομένα που μπορούν να πάρουν πολλές διαφορετικές τιμές, ο αριθμός των καταστάσεων του μοντέλου μπορεί να γίνει υπερβολικά μεγάλος. Για την αντιμετώπιση αυτού του προβλήματος, έχουν αναπτυχτεί διάφορες τεχνικές, με σκοπό την μείωση του αριθμού καταστάσεων του μοντέλου. Ενδεικτικά αναφέρουμε: χρήση διαγραμμάτων δυαδικής απόφασης (bdd), χρήση αφαίρεσης, μερική διατεταγμένη μείωση (partial order reduction).

Η διαδικασία του ελέγχου μοντέλων μπορεί να διαχωριστεί σε τρία στάδια:

1) Μοντελοποίηση(πιο αναλυτικά παρακάτω):
   — Μοντελοποίηση του συστήματος που εξετάζεται χρησιμοποιώντας την γλώσσα περιγραφής μοντέλου (model description language) του ελεγκτή μοντέλων που χρησιμοποιείται.
   — Εκτέλεση κάποιων προσομοιώσεων, ως πρώτος έλεγχος για μια γρήγορη αξιολόγηση του μοντέλου
   — Τυπική διατύπωση (formal specification) της ιδιότητας που εξετάζεται χρησιμοποιώντας τη γλώσσα περιγραφής ιδιότητας (property specification language)
2) Τρέξιμο ελεγκτή:
   Σε αυτό το στάδιο τρέχει ο ελεγκτής μοντέλου ώστε να ελέγξει την εγκυρότητα της ιδιότητας στο μοντέλο του συστήματος
3) Ανάλυση:
   Εφόσον η ιδιότητα ικανοποιείται, τότε, αν υπάρχει άλλη ιδιότητα για να ελεγχθεί, εξετάζει πάλι το μοντέλο του συστήματος ως προς τη νέα ιδιότητα.
   Αν δεν ικανοποιείται η ιδιότητα:
   — Γίνεται ανάλυση του αντιπαραδείγματος που παράχθηκε μέσω προσομοίωσης.
   — Βελτίωση του μοντέλου, του συστήματος ή της ιδιότητας.
   — Επανάληψη όλης της διαδικασίας.
   Αν τελειώσει η μνήμη, χρειάζεται να γίνει προσπάθεια για να μειωθεί το μοντέλο και να ξαναδοκιμάσουμε.

2.4.1.1) **Μοντελοποίηση**:

Συνήθης μορφή για την αναπαράσταση των μοντέλων ως σύστημα μετάβασης πεπερασμένων καταστάσεων είναι η δομή Kripke (Kripke structure). Αποτελείται από ένα σύνολο καταστάσεων, ένα σύστημα μετατροπών μεταξύ των καταστάσεων, και μία συνάρτηση η οποία σημειώνει σε κάθε κατάσταση το σύνολο των ιδιοτήτων της που είναι αληθείς. Τα μονοπάτια σε μια δομή Kripke μοντελοποιούν τους υπολογισμούς του συστήματος. Μία δομή Kripke ορίζεται ως εξής:

Έστω AP ένα σύνολο ατομικών προτάσεων. Μία Kripke δομή M πάνω στο AP ορίζεται ως μία διατεταγμένη τετράδα $M=(S, S_0, R, L)$, όπου:

- S: πεπερασμένο σύνολο καταστάσεων
- $S_0 \subseteq S$: ένα σύνολο από αρχικές καταστάσεις
- $R \subseteq S \times S$: μια σχέση μετάβασης. Για κάθε κατάσταση $s \in S$ υπάρχει κατάσταση $s' \in S$, έτσι ώστε $(s, s') \in R$.
- $L : S \rightarrow 2^{AP}$: μια συνάρτηση απόδοσης ετικετών. Αναθέτει σε κάθε κατάσταση ένα σύνολο ατομικών προτάσεων που είναι αληθείς για κάθε κατάσταση.

Ένα μονοπάτι σε μία τέτοια δομή είναι μία πεπερασμένη ή άπειρη ακολουθία καταστάσεων $\pi=s_0s_1s_2\ldots$ έτσι ώστε για κάθε $i\geq0$, $s_i\rightarrow s_{i+1}$. Μία κατάσταση s θεωρείται προσβάσιμη (reachable) στο M, όταν υπάρχει πεπερασμένο μονοπάτι π από μία αρχική κατάσταση s_0 στην s.

Ως προς τη μοντελοποίηση της ιδιότητας που εξετάζεται, απαιτείται ακριβής περιγραφή. Για την περιγραφή των επιθυμητών ιδιοτήτων/απαιτήσεων, χρησιμοποιείται, συνήθως μία γλώσσα περιγραφής ιδιότητας (property specification language). Στα συστήματα που εξετάζονται, αυτόν τον ρόλο παίζει συνήθως, η χρονική λογική (temporal logic*).* Ουσιαστικά, η χρονική λογική είναι μία επέκταση της κλασικής προτασιακής λογικής (propositional logic), με τελεστές που αναφέρονται στη συμπεριφορά του συστήματος στον χρόνο. Μπορεί να περιγράψει ευρύ φάσμα σχετικών ιδιοτήτων συστήματος, όπως:

- Λειτουργική ορθότητα (functional correctness)
  «Το σύστημα κάνει αυτό που θα έπρεπε;»
- Προσεγγισιμότητα (reachability)
  «Υπάρχει δυνατότητα αδιεξόδου;»
- Ασφάλεια (safety)
  «Ποτέ δε θα συμβεί κάτι κακό.»
- Βιωσιμότητα (Liveness)
  «Κάτι καλό τελικά θα συμβεί.»
- Δικαιοσύνη (fairness)
  «Μπορεί, κάτω από κάποιες περιπτώσεις, ένα γεγονός να συμβαίνει άπειρα συχνά;»
- Ιδιότητες πραγματικού χρόνου
  «Το σύστημα ενεργεί στο χρόνο;»

Οι κυριότερες χρονικές λογικές που χρησιμοποιούνται είναι η γραμμική χρονική λογική (Linear Time Logic, LTL) και η λογική υπολογιστικού δέντρου (Computational tree logic, CTL). Στην **LTL**, μπορούν να κωδικοποιηθούν προτάσεις για το μέλλον κάποιου μονοπατιού, ώστε π.χ. μια συνθήκη να είναι τελικά αληθής, ή να είναι αληθής μέχρι ένα άλλο γεγονός να είναι αληθές. Περιγράφει ιδιότητες ως ανεξάρτητες εκτελέσεις. Η σημασιολογία της ορίζεται ως σύνολο εκτελέσεων. Η **CTL** είναι λογική διακλαδιζόμενου χρόνου (branching-time), το μοντέλο της για το χρόνο είναι μια δενδρική δομή, με διαφορετικά μονοπάτια και κάθε ένα από αυτά τα μονοπάτια μπορεί να είναι ένα μονοπάτι που θα πραγματοποιηθεί. Έτσι, μπορεί να περιγράψει ιδιότητες πολλών εκτελέσεων ταυτόχρονα. Η σημασιολογία της ορίζεται με όρους καταστάσεων.

Συμπερασματικά, ο έλεγχος μοντέλων είναι μία γενική προσέγγιση επαλήθευσης που μπορεί να εφαρμοστεί σε ευρύ φάσμα εφαρμογών. Οι ιδιότητες μπορούν να ελεγχθούν ανεξάρτητα, έτσι μπορούμε να εστιάσουμε στις σημαντικές ιδιότητες πρώτα. Είναι πιο κατάλληλος για εφαρμογές πιο απαιτητικές στον έλεγχο και

λιγότερο κατάλληλος για εφαρμογές ποιο απαιτητικές σε δεδομένα, καθώς τα δεδομένα συνήθως κυμαίνονται σε άπειρα πεδία. Επαληθεύει, όμως, το μοντέλο ενός συστήματος και όχι το πραγματικό σύστημα. Όπως είπαμε, αντιμετωπίζει το πρόβλημα της έκρηξης καταστάσεων και η χρήση του απαιτεί ειδική γνώση στην κατασκευή μικρότερων συστημάτων μοντέλων και στην δήλωση των ιδιοτήτων στη φόρμουλα λογικής που χρησιμοποιείται.

Στη συνέχεια παρατίθενται ορισμένα εργαλεία για τον έλεγχο μοντέλων:

Spin, Slam and Static Driver Verifier (Microsoft), CPAchecker, SatAbs, Wolverine, CBMC, LLBMC, Threader, Blast, Copper, ARMC, JPF (NASA), Kratos, SMV, NuSMV, ABC, Uclid, Incisive Formal Analysis (Cadence), RuleBase (IBM), O-in(Mentor Graphics), Magellan (Synopsys), Jasper.

2.4.2) Deductive verification

"Αντί να γίνει αποσφαλμάτωση ενός προγράμματος, θα έπρεπε κάποιος να αποδείξει ότι το πρόγραμμα πληρεί τις προδιαγραφές του και αυτή η απόδειξη να ελεγχθεί από ένα πρόγραμμα υπολογιστή"
John McCarthy, "A Basis for a Mathematical Theory of Computation," 1961

Η δεύτερη προσέγγιση είναι αυτή της επαγωγικής επαλήθευσης. Με τον συγκεκριμένο όρο συνήθως αναφερόμαστε στη χρήση αξιωμάτων και αποδείξεων έτσι ώστε να αποδειχτεί η ορθότητα του συστήματος. Η μέθοδος είναι γνωστή και ως απόδειξη θεωρήματος.

Η ιδέα της αποτύπωσης της ορθότητας ενός συστήματος, ιδιαίτερα όταν μιλάμε για λογισμικό και συγκεκριμένα για ένα πρόγραμμα, σε μαθηματική δήλωση και η απόδειξή της, δεν είναι καινούρια. Είναι τόσο παλιά, όσο και ο προγραμματισμός. Μία από τις πρώτες αποδείξεις προγράμματος πραγματοποιήθηκε από τον Alan Turing το 1949.

Οι αρχικές έρευνες της επαγωγικής επαλήθευσης, εστίαζαν κυρίως στην εγγύηση της ορθότητας κρίσιμων συστημάτων. Λόγω της μεγάλης σπουδαιότητας της σωστής συμπεριφοράς τέτοιων συστημάτων, θεωρούνταν ότι ο σχεδιαστής ή ο ειδικός επαλήθευσης του συστήματος, θα ξόδευε όσο χρόνο χρειαζόταν ώστε να αποδειχτεί η ορθότητα του συστήματος. Αρχικά αυτές οι αποδείξεις γίνονταν όλες με το χέρι. Στη συνέχεια αναπτύχθηκαν εργαλεία λογισμικού για την επιβολή σωστής χρήσης αξιωμάτων και κανόνων απόδειξης.

Σε αυτήν την προσέγγιση, η ορθότητα εκφράζεται μέσω ενός συνόλου μαθηματικών δηλώσεων. Για να επαληθευτεί το σύστημα, ουσιαστικά παρέχεται μία λογική απόδειξη της ιδιότητας που επιθυμούμε. Έτσι, απαραίτητο είναι το σύστημα να έχει, ή να μπορεί να πάρει, τη μορφή αφηρημένου μαθηματικού μοντέλου. Χρησιμοποιεί μια τυπική μέθοδο μαθηματικής συλλογιστικής για το σύστημα, συνήθως με χρήση λογισμικού απόδειξης θεωρημάτων (πολλές φορές

λογικής ανώτερης τάξης). Συνήθως πρόκειται για λογισμικό με μερικούς αυτοματισμούς, που καθοδηγείται όμως από το χρήστη τις περισσότερες φορές, όποτε και εξαρτάται άμεσα από το κατά πόσο έχει κατανοήσει αυτός το σύστημα. Απαιτεί υψηλό επίπεδο γνώσης και εξειδίκευσης.

Οι μέθοδοι που χρησιμοποιούνται σε αυτήν την προσέγγιση είναι κυρίως παραλλαγές της δουλειάς του Hoare και του Dijktra, που επέκτεινε ουσιαστικά τη δουλειά του Hoare. Με τη μέθοδο απόδειξης του Hoare (Hoare-Floyd αλλιώς), αιτιολογείται η ορθότητα ενός συστήματος, συγκεκριμένα ενός προγράμματος, με όρους προ (pre) και μετά (post) συνθηκών.

Η προσέγγιση του Hoare για την απόδειξη της ορθότητας, εισήγαγε την τριπλέτα Hoare (Hoare triple). Πρόκειται για μια φόρμουλα της ακόλουθης μορφής:

$$\{\varphi_{PRE}\}\ P\ \{\varphi_{POST}\}$$

Η παραπάνω φόρμουλα σημαίνει ότι αν η ιδιότητα φ_{PRE} ισχύει πριν το πρόγραμμα P ξεκινήσει, μετά την εκτέλεσή του θα ισχύει η ιδιότητα φ_{POST}. Χρησιμοποιεί αξιώματα, κανόνες και αναφορές ώστε να εξάγει ιδιότητες φ_{PRE} βασισμένες στις και φ_{POST} στο P. Η σύνταξη του P όπως περιγράφεται από τον Hoare, ανταποκρίνεται σε μία απλή γλώσσα με τις συνηθισμένες δομές (αναθέσεις, διακλαδώσεις υπό συνθήκη, επαναλήψεις, διαδοχικές δηλώσεις).

Ο Dijktra επέκτεινε την ιδέα του Hoare με την έννοια του «μετασχηματιστή κατηγορήματος» (predicate transformers). Αντί να ξεκινάει από μια pre-συνθήκη, ξεκινάει από μία post συνθήκη και χρησιμοποιεί τον κώδικα του προγράμματος για τον προσδιορισμό της pre συνθήκης που χρειάζεται να ισχύει ώστε να ισχύσει η post συνθήκη.

Το πόσο μπορεί να αυτοματοποιηθεί η διαδικασία της απόδειξης θεωρήματος, εξαρτάται από το επίπεδο της λογικής που βασίζεται. Όταν βασίζεται σε προτασιακή λογική (propositional logic) μπορεί να αυτοματοποιηθεί πλήρως (SAT-solvers), για λογική πρώτης τάξης (first order logic), αυτοματοποιείται αλλά δεν τερματίζει υποχρεωτικά (SMT-solvers), ενώ για λογική υψηλότερης τάξης (higher order logic) υπάρχει αυτοματισμός, αλλά χρειάζεται αλληλεπίδραση.

Οι αποδείκτες θεωρήματος που αποδεικνύουν ιδιότητες προγραμμάτων, βασίζονται συνήθως σε παραλλαγές της λογικής Hoare. Ωστόσο, η γλώσσα των αποδεικτών θεωρήματος συνήθως είναι μία διάλεκτος της LISP, που βασίζεται στον McCarthy. Οι αποδείκτες θεωρήματος, σε αντίθεση με την επαλήθευση μέσω ελέγχου μοντέλων, δεν χρειάζεται να επισκεφτούν διεξοδικά ολόκληρο τον χώρο καταστάσεων για να επαληθεύσουν τις επιθυμητές ιδιότητες. Μέσω της απόδειξης θεωρήματος μπορεί να αποφανθεί για την ορθότητα ακόμα και για συστήματα άπειρου χώρου καταστάσεων, ή με πολύπλοκους τύπους δεδομένων

και αναδρομές. Αυτό επιτυγχάνεται γιατί οι αποδείκτες θεωρήματος εξετάζουν τους περιορισμούς στις καταστάσεις, και όχι στιγμιότυπα των καταστάσεων. Επομένως, αυτή η προσέγγιση είναι κατάλληλη για επαλήθευση συστημάτων που παίρνουν πολλά δεδομένα, με πολύπλοκες δομές δεδομένων αλλά απλή ροή πληροφορίας.

Κάποια από τα εργαλεία που χρησιμοποιούνται για την απόδειξη θεωρήματος είναι τα εξής: ACL2, Agda, Isabelle, Coq, PVS, Hol, Forte.

Η αξία της ανάπτυξης της επαγωγικής επαλήθευσης είναι μεγάλη. Ωστόσο, πρόκειται για διαδικασία που απαιτεί χρόνο και μπορεί να γίνει μόνο από ειδικούς που έχουν εκπαιδευτεί ειδικά και διαθέτουν εμπειρία. Τυπικά απαιτεί από το χρήστη να κατανοήσει λεπτομερώς γιατί το σύστημα λειτουργεί σωστά, και να μεταδώσει αυτή την πληροφορία προς το σύστημα ελέγχου, είτε με τη μορφή ακολουθίας θεωρημάτων προς απόδειξη είτε με τη μορφή των προδιαγραφών των στοιχείων του συστήματος. Εφαρμόζεται κυρίως σε συστήματα υψηλής ευαισθησίας, όπως τα πρωτόκολλα ασφάλειας. Πλεονέκτημα αυτής της μεθόδου είναι το ότι δεν εξετάζει μόνο συστήματα πεπερασμένων καταστάσεων.

2.4.3) Στατική ανάλυση

Η τεχνική της στατικής ανάλυσης συγκεντρώνει πληροφορίες για την συμπεριφορά του συστήματος από τον πηγαίο κώδικα, χωρίς να τον εκτελεί. Αυτή η περιγραφή είναι πολύ γενική και μπορεί ακόμη και να συμπεριλαμβάνει τις προηγούμενες δύο, κάτι όμως που δε συμβαίνει. Η στατική ανάλυση εκτείνεται από απλούς συντακτικούς ελέγχους, σε επαναληπτικούς υπολογισμούς σταθερού σημείου (fix-point), πάνω σε μία αφαίρεση του συστήματος που εξετάζεται. Εφαρμόζεται κυρίως σε συστήματα λογισμικού, ιδιαίτερα σε μεταγλωττιστές.

Υπάρχουν πολλές διαφορετικές μορφές της στατικής ανάλυσης: αναζήτηση μοτίβου σφαλμάτων (bug-pattern searching), ανάλυση ροής δεδομένων (dataflow analysis), ανάλυση που βασίζεται σε περιορισμούς (constraint-based analysis), ανάλυση τύπων (type analysis), αφηρημένη ερμηνεία (abstract interpretation), συμβολική εκτέλεση (symbolic execution).

Κάποια εργαλεία που χρησιμοποιούν στατική ανάλυση είναι τα εξής: FindBugs, Coverity, Klocwork, CodeSonar (GrammaTech), PolySpace, PREfix/PREfast, AbsInt/Astrée, Clang Static Analyser, Cppcheck, cppclean, Sparse, Splint, Stanse, Space Invader, Predator, SLAyer, jStar, KLEE, Yogi, Speed.

Τέτοια εργαλεία χρησιμοποιούνται ενεργά σε πολλά περιβάλλοντα:

Microsoft – Windows Vista (PREfix/PREfast: περισσότερα από 100k διορθωμένα σφάλματα), Linux kernel (Sparse), Red Hat (Coverity), Airbus flight control (Astrée, AbsInt).

Η στατική ανάλυση δεν χρησιμοποιείται αποκλειστικά για τον έλεγχο ορθότητας, αλλά χρησιμοποιείται και αλλού, όπως στη βελτιστοποίηση, ή στην παραγωγή κώδικα. Μπορεί να χειριστεί μεγάλα συστήματα. Ακόμα, δεν χρειάζεται μοντελοποίηση του περιβάλλοντος και έχουμε υψηλό βαθμό αυτοματοποίησης. Μπορεί να παράγει, όμως, πολλούς λάθος συναγερμούς, δηλαδή προειδοποιήσεις ότι οι προδιαγραφές μπορεί να μην ικανοποιούνται, ενώ καμία πραγματική εκτέλεση του συστήματος δεν τις παραβιάζει. Περισσότερα θα δούμε στην επόμενη ενότητα.

2.5) Επιλογή κατάλληλης μεθόδου

Η χρήση τυπικών μεθόδων για την επαλήθευση συστημάτων, σε συνδυασμό βέβαια και με το testing, μπορούν να βελτιώσουν την ποιότητα και αξιοπιστία του συστήματος. Σε αυτήν την κατεύθυνση έχουν αναπτυχτεί διάφορες μέθοδοι και εργαλεία που τις χρησιμοποιούν. Η επιλογή της πιο κατάλληλης έχει να κάνει με το τι σύστημα εξετάζουμε, την κρισιμότητά του, ποιες ιδιότητες θέλουμε να αποδείξουμε, τους πόρους που διαθέτουμε και έτσι δεν μπορεί να κριθεί, χωρίς να παρθούν υπόψη τα παραπάνω. Μία ελκυστική προσέγγιση, είναι η χρήση κάποιου συνδυασμού μεθόδων και εργαλείων τυπικής επαλήθευσης.

3 ΤΥΠΙΚΗ ΕΠΑΛΗΘΕΥΣΗ ΛΟΓΙΣΜΙΚΟΥ

3.1) Τυπική επαλήθευση software vs hardware

Σε συστήματα υλικού, η επαλήθευση είναι μία εφικτή και σε κάποιες περιπτώσεις σχετικά απλή διαδικασία, κυρίως λόγω του πεπερασμένου μεγέθους τους και έτσι, εφαρμόζεται πλατιά στη βιομηχανία. Εστιάζοντας στην επαλήθευση συστημάτων λογισμικού, τα πράγματα γίνονται πιο σύνθετα.

Αυτό είναι λογικό καθώς:

- Το λογισμικό δεν είναι δυνατό να καθοριστεί σταθερά (undecidable), πχ ένα πρόγραμμα μπορεί να μην τερματίζει.
- Δεν έχει αυστηρά καθορισμένα όρια (unbounded), πχ στοίβα, δυναμική κατανομή μνήμης.
- Οι γλώσσες προγραμματισμού έχουν πολύπλοκη σημασιολογία, που είναι δύσκολο να μοντελοποιηθεί (συναρτήσεις, αναδρομή, δείκτες).
- Στις τεχνικές για το υλικό, το μονοπάτι δεδομένων είναι ενσωματωμένο ουσιαστικά με το μονοπάτι ελέγχου, ενώ στο λογισμικό, αυτά διαφοροποιούνται.
- Είναι δύσκολο να μοντελοποιηθεί η σχέση των δεδομένων σε συμβολικό ελεγκτή μοντέλου (πχ τεράστια BDDs)
- Το λογισμικό έχει λιγότερη «στοιχειοποίηση» (modularity) από το υλικό, έτσι είναι δυσκολότερο να εφαρμοστούν τεχνικές που διασπούν το πρόβλημα σε μικρότερα.

Ο τυπικός έλεγχος και επαλήθευση συστημάτων υλικού είναι περισσότερο διαδεδομένος από συστήματα λογισμικού και για οικονομικούς λόγους. Αυτό έχει να κάνει με τη φύση των σφαλμάτων. Τα σφάλματα στο υλικό συνήθως δεν είναι αποδεκτά από τον χρήστη και κοστίζει πολύ η επιδιόρθωσή τους. Στο λογισμικό, τα σφάλματα είναι περισσότερο ανεκτά από τον χρήστη και η επιδιόρθωσή τους είναι πιο φθηνή. Έτσι, για τις εταιρίες υλικού είναι πιο σημαντικό να επενδυθούν μεγάλες πηγές (χρόνος, χρήματα, εξειδικευμένο προσωπικό) στην επαλήθευση από τις εταιρίες λογισμικού.

Όπως είπαμε όμως στην αρχή, ο έλεγχος μέσω της δοκιμής δεν μπορεί να εγγυηθεί την ποιότητα ούτε των συστημάτων λογισμικού. Ο τυπικός έλεγχος και επαλήθευση, αν και δυσκολότερος, είναι ανάγκη να χρησιμοποιείται και στο λογισμικό. Υπάρχουν κρίσιμα συστήματα λογισμικού (συστήματα εντατικής θεραπείας, οικονομικά, συστήματα ασφαλείας, αντιπυραυλικά κα) που είναι

απαραίτητο να ελέγχονται ολοκληρωμένα. Ακόμα, με την ανάπτυξη παράλληλων προγραμμάτων, ο έλεγχος μέσω της δοκιμής γίνεται πιο δύσκολος.

3.2) Τυπική επαλήθευση λογισμικού - Γενικά

Στην παρούσα εργασία εστιάζουμε στην τυπική επαλήθευση προγραμμάτων λογισμικού.

Στον έλεγχο και την επαλήθευση του λογισμικού, εστιάζουμε στις τυπικές μεθόδους που πληρούν τα παρακάτω κριτήρια:

- Η μέθοδος να παρέχει αυστηρή εγγύηση της ποιότητας. Για αυτόν τον λόγο δεν χρησιμοποιούνται τυχαίες δοκιμές, ή αυτόματη παραγωγή περιπτώσεων ελέγχου. Πρακτικά, η εγγύηση της ποιότητας σπάνια αναφέρεται στην καθολική ορθότητα ολόκληρου του συστήματος, αλλά στην απουσία συγκεκριμένων ελαττωμάτων.
- Η μέθοδος να είναι υψηλά αυτοματοποιημένη και επεκτάσιμη, ώστε να μπορεί να αντιμετωπίσει την τεράστια πολυπλοκότητα των συστημάτων λογισμικού. Δεν αναλύουμε σε αυτό το κεφάλαιο μεθόδους δυναμικής ανάλυσης, ούτε μεθόδους που απαιτούν παρέμβαση από τον χρήστη για την απόδειξη της ορθότητας (όπως οι αποδείκτες θεωρήματος, πχ ACL2, HOL), ούτε εργαλεία που χρειάζονται σχόλια (annotations) από τον προγραμματιστή (παρακάτω).

Θα εξεταστούν μόνο τεχνικές στατικής ανάλυσης που δεν εξαρτώνται από την εκτέλεση του προγράμματος και απαιτούν την ελάχιστη αλληλεπίδραση του χρήστη.

Βασιζόμαστε στο άρθρο "A Survey of Automated Techniques for Formal Software"[1]. Σε αντιστοιχία με αυτό, εξετάζονται και εδώ οι τρεις τεχνικές:

1. Στατική ανάλυση με αφηρημένα πεδία (Abstract Static Analysis)
2. Έλεγχος μοντέλου (Model Checking)
3. Φραγμένος έλεγχος μοντέλου (Bounded Model Checking)

3.2.1) Αφηρημένη στατική ανάλυση (Abstract Static Analysis)

Όπως αναφέρθηκε και στην προηγούμενη ενότητα, η στατική ανάλυση αποτελείται από τεχνικές που υπολογίζουν αυτόματα πληροφορίες για τη συμπεριφορά του προγράμματος, χωρίς να το εκτελούν. Παρά το ότι αυτές οι τεχνικές χρησιμοποιούνται εκτενώς στη βελτιστοποίηση των μεταγλωττιστών,

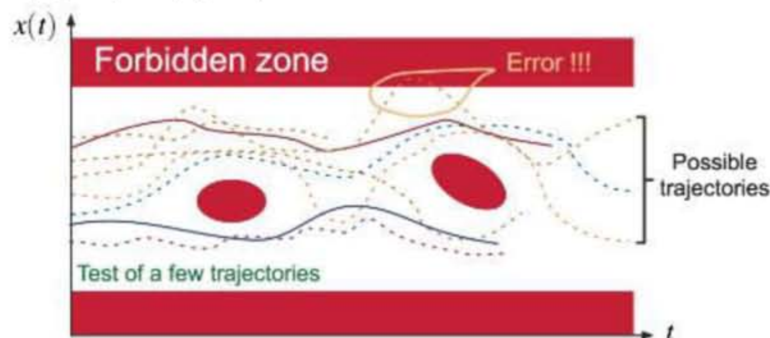
---

[1] D'Silva V., Kroening D., & Weissenbacher G., "A survey of automated techniques for formal software verification", *IEEE Transactions on CAD of Integrated Circuits and Systems, 27*(7), 1165–1178, 2008.

εστιάζουμε στη χρήση τους για την επαλήθευση προγραμμάτων. Καθώς η συμπεριφορά ενός προγράμματος μπορεί να μην είναι δυνατό να καθοριστεί σταθερά (undecidable), η ουσία της στατικής ανάλυσης είναι να υπολογίσει αποτελεσματικά προσεγγιστικές, αλλά ισχυρές εγγυήσεις.

Η στατική ανάλυση αφηρημένης ερμηνείας (abstract interpretation) είναι η μέθοδος η οποία κάνει ισχυρές προσεγγίσεις της σημασιολογίας των προγραμμάτων, βασισμένη σε μονότονες συναρτήσεις σε διατεταγμένα σύνολα. Μπορούμε να τη δούμε ως μερική εκτέλεση ενός προγράμματος που αποκτά πληροφορίες για τη σημασιολογία του (πχ ροή ελέγχου ή δεδομένων) χωρίς να κάνει όλους τους υπολογισμούς.

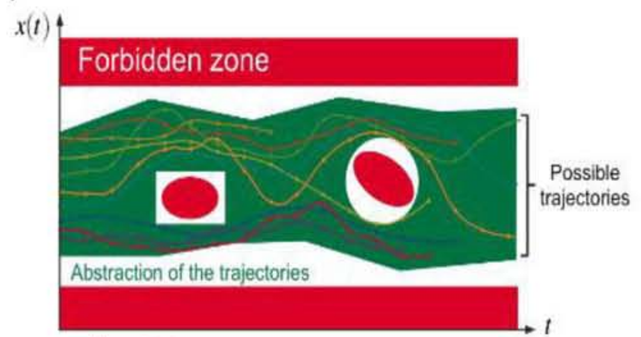
Μπορούμε να φανταστούμε τη συμπεριφορά ενός προγράμματος ως διαδρομές (trajectories) που δείχνουν την εξέλιξη της κατάστασης της x(t), ως συνάρτηση του χρόνου t (διακριτή εφόσον μιλάμε για ελεγκτές σε υπολογιστή). Η κατάσταση της x(t) μπορεί να είναι π.χ. οι τιμές των μεταβλητών του προγράμματος. Οι προδιαγραφές της ασφάλειας του συστήματος, μπορούν να έχουν τη μορφή απαγορευμένων ζωνών (πχ διαίρεση με το 0). Μία τυπική απόδειξη της ορθότητας είναι η απόδειξη ότι από τις προδιαγραφές του προγράμματος συνεπάγεται ότι κανένα ίχνος εκτέλεσης του προγράμματος δεν περνάει από απαγορευμένη ζώνη.



Εικόνα 3.1

Η αφηρημένη στατική ανάλυση, υπολογίζει μία υπέρ-προσέγγιση (over-approximation) των πιθανών διαδρομών. Η αξιοπιστία εξασφαλίζεται από την πλήρη κάλυψη όλων των πιθανών διαδρομών.

Εικόνα 3.2



Εικόνα 3.3

Οι εικόνες 3.1, 3.2, 3.3 είναι από το άρθρο των Cousot, P., Cousot, R., "A gentle introduction to formal verification of computer systems by abstract interpretation".

Η απόδειξη ότι το πρόγραμμα ικανοποιεί τις προδιαγραφές του προκύπτει από το γεγονός ότι όλες οι πιθανές διαδρομές συμπεριλαμβάνονται στην αφαίρεση, ενώ η αφαίρεση δεν τέμνει τις κακές καταστάσεις.

3.2.1.1) Γενικά και ορολογία

Οι τεχνικές στατικής ανάλυσης συνήθως μεταδίδουν (propagate) ένα σύνολο τιμών ενός προγράμματος, μέσω του προγράμματος, μέχρι το σύνολο να κορεστεί (saturate), δηλαδή να μην αλλάζει με την περαιτέρω μετάδοση. Μαθηματικά, τέτοιες αναλύσεις μοντελοποιούνται ως επαναληπτική εφαρμογή μιας μονότονης συνάρτησης. Ο κορεσμός πραγματοποιείται όταν επιτυγχάνεται ένα σταθερό σημείο της συνάρτησης.

Όταν σε έναν γράφο ροής ελέγχου (control flow graph, CGF) αναπαριστούμε τις ακριβείς τιμές των μεταβλητών που μας ενδιαφέρουν κατά την εκτέλεση, τότε έχουμε *συγκεκριμένη ερμηνεία* του προγράμματος (concrete interpretation). Αυτή η προσέγγιση δεν είναι πρακτική, καθώς τα σύνολα τιμών αναπτύσσονται γρήγορα και η μετάδοση δεν κλιμακώνεται. Στην πράξη, μας ενδιαφέρει περισσότερο η αποτελεσματικότητα από την ακρίβεια. Έτσι, χρησιμοποιούμε μία προσέγγιση του συνόλου τιμών ή αγνοούμε συγκεκριμένα στοιχεία της δομής του προγράμματος κατά την μετάδοση. Με βάση τα στοιχεία της δομής που αγνοούμε, γίνεται η ακόλουθη ταξινόμηση της ανάλυσης προγράμματος:

1. Ανάλυση με βάση τη ροή, αν μας ενδιαφέρει η σειρά της εκτέλεσης των δηλώσεων του προγράμματος (flow-sensitive analysis).
2. Ανάλυση με βάση το μονοπάτι, αν θέλουμε να διακρίνουμε μεταξύ των μονοπατιών ενός προγράμματος και να εξετάσουμε μόνο όσα είναι εφικτά (path -sensitive analysis).
3. Ανάλυση με βάση το περιεχόμενο, αν οι κλήσεις των μεθόδων εξετάζονται διαφορετικά, ανάλογα με την τοποθεσία της κλήσης (context-sensitive analysis).
4. Αν η ανάλυση του σώματος των μεθόδων γίνεται στο περιεχόμενο της αντίστοιχης τοποθεσίας της κλήσης (interprocedural).

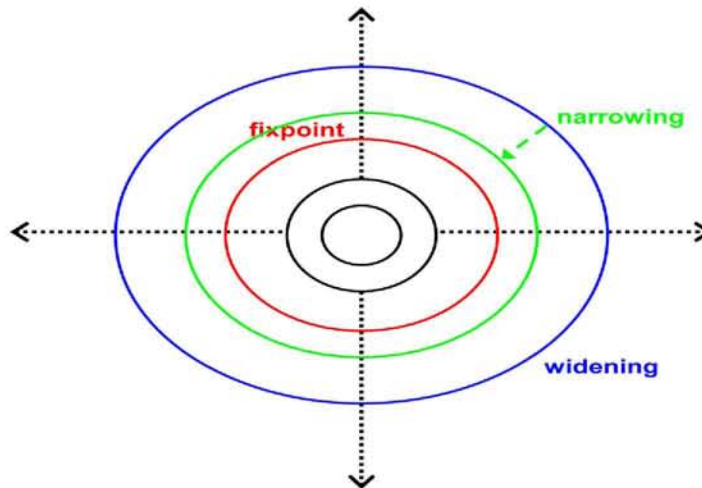
3.2.1.2) Αφηρημένη ερμηνεία (Abstract Interpretation)

Η τεχνική της αφηρημένης ερμηνείας ως πλαίσιο για τη συσχέτιση της αφηρημένης ανάλυσης με την εκτέλεση του προγράμματος, εισήχθη από τους P. Cousot και R. Cousot.

Ένα αφηρημένο πεδίο (abstract domain) είναι μία, κατά προσέγγιση, αναπαράσταση του συνόλου των συγκεκριμένων τιμών. Η αντιστοίχηση των συγκεκριμένων τιμών στις αφηρημένες γίνεται με μία αφηρημένη συνάρτηση (abstraction function). Η αφηρημένη ερμηνεία περιλαμβάνει τον υπολογισμό της

συμπεριφοράς ενός προγράμματος πάνω στο αφηρημένο πεδίο, ώστε να ληφθεί μία προσεγγιστική λύση. Μία αφηρημένη ερμηνεία μπορεί να προέρχεται και από συγκεκριμένη ερμηνεία, όταν για συγκεκριμένες λειτουργίες (πχ ένωση), ορίζουμε αντίστοιχες στο αφηρημένο πεδίο. Εφόσον μεταξύ των αφηρημένων και συγκεκριμένων πεδίων υπάρχουν αυστηροί μαθηματικοί περιορισμοί, τότε τα fix-points που υπολογίζονται στο αφηρημένο πεδίο, είναι εγγυημένα, ισχυρές προσεγγίσεις των συγκεκριμένων fix-points.

Ακόμα και με την αφηρημένη ερμηνεία, μπορεί να χρειάζεται ανέφικτος αριθμός επαναλήψεων ώστε να φτάσουμε σε ένα fix-point. Για να επιταχυνθεί και να εξασφαλιστεί ο τερματισμός, μπορεί να χρησιμοποιηθεί ένας τελεστής διεύρυνσης (widening operator), σε βάρος της ακρίβειας συνήθως. Συμπληρωματικά, μπορεί να χρησιμοποιηθεί ένας τελεστής στένωσης (narrowing operator) για τη βελτίωση της ακρίβειας.



Σχήμα 3.1

3.2.1.3) Αφηρημένα αριθμητικά πεδία (Numerical Abstract Domains)

Μία μεταβλητή x σε ένα σημείο του προγράμματος αντιστοιχεί σε ένα διάστημα $[L_x, H_x]$. Όλες οι τιμές που μπορεί να πάρει η μεταβλητή x χρειάζεται να συμπεριλαμβάνονται στο $[L_x, H_x]$. Η επιλογή των αφηρημένων πεδίων που θα χρησιμοποιηθούν είναι κρίσιμη. Με τα χρόνια, έχουν σχεδιαστεί πολλά αφηρημένα πεδία. Οι ιδιότητες που μπορούν να αποδειχθούν, ποικίλουν ανάλογα με την εκφραστική δύναμη του πεδίου που χρησιμοποιείται. Τα αφηρημένα πεδία διακρίνονται σε:

- Μη σχεσιακά πεδία (nonrelational domains). Γρήγορη και απλή υλοποίηση, αλλά ανακριβής, καθώς δεν καταγράφουν τις σχέσεις μεταβλητών.
  Τέτοια πεδία μπορούν να είναι για παράδειγμα ένα πεδίο Signs με τρεις τιμές: {pos, neg, zero}, ή ένα πεδίο Parities με τιμές: {Even, Odd}.

- Σχεσιακά πεδία (relational domains). Μπορούν να καταγράψουν τις σχέσεις μεταξύ μεταβλητών.
  Παραδείγματα από σχεσιακά αριθμητικά αφηρημένα πεδία είναι τα παρακάτω:
  - Difference Bound Matrices (DBMs). Σύζευξη ανισοτήτων της μορφής $x-y \leq c$ και $\pm x \leq c$. Αρχικά χρησιμοποιήθηκαν στην ανάλυση Petri nets.
  - Οκτάγωνα (Octagons). Πιο εκφραστικό πεδίο με εξισώσεις της μορφής $ax+by \leq c$, όπου a, b$\in \{-1, 0, 1\}$ και c ακέραιος.
  - Οκτάεδρα (Octahedra). Γενίκευση των οκταγώνων για περισσότερες από δύο μεταβλητές.
  - Πολύεδρα (Polyhedra). Σύζευξη ανισοτήτων της μορφής $a_1x_1+\ldots+a_nx_n \leq c$, όπου a_i και c ακέραιοι. Χρησιμοποιούνται πολύ σε ενσωματωμένα συστήματα λογισμικού. Οι απαιτήσεις χρόνου και χώρου για διαδικασίες που χειρίζονται πολύεδρα είναι συνήθως εκθετικές με τον αριθμό των μεταβλητών.

Τα αριθμητικά αφηρημένα πεδία χρησιμοποιούνται για να αποδείξουν ιδιότητες ενσωματωμένων συστημάτων λογισμικού, την ανάλυση της συμπεριφοράς δεικτών, τον χειρισμό string, τον τερματισμό προγράμματος.

3.2.1.4) Ανάλυση με πεδία που αφορούν τη στοίβα

Απαραίτητη για την ανάλυση προγραμμάτων ιδιαίτερα με δείκτες, είναι η αφηρημένη ανάλυση με πεδία που αφορούν τη στοίβα (heap).

- Ανάλυση Alias: Εξετάζει αν δύο μεταβλητές δεικτών έχουν πρόσβαση στην ίδια περιοχή μνήμης.
- Ανάλυση Pointsto: Προσδιορίζει την περιοχή της μνήμης που μπορεί να έχει πρόσβαση μία μεταβλητή. Η ανάλυση, συνήθως, αγνοεί τη ροή ελέγχου (flow-insensitive).
- Ανάλυση σχήματος (Shape analysis): Επαληθεύει ιδιότητες δομών δεδομένων που δημιουργούνται δυναμικά. Ονομάζεται έτσι γιατί οι ιδιότητες σχετίζονται με το «σχήμα» της στοίβας (heap).
- Στατική ανάλυση με κανονικές αφαιρέσεις (Canonical abstractions): Οι κανονικές αφαιρέσεις αποτελούν μία αναπαράσταση, βασισμένη σε γράφο, της στοίβας, μαζί με λογικά κατηγορήματα (logical predicates) που περιγράφουν σχέσεις μεταξύ των κελιών της στοίβας. Μία κανονική αφαίρεση (canonical abstraction) είναι ένας αφηρημένος γράφος με τους κόμβους να αναπαριστούν τα κελιά της στοίβας, τους κόμβους «σύνοψης» (summary nodes) να αναπαριστούν πολλαπλά κελιά στοίβας και κατηγορήματα που μπορούν να πάρουν τρεις τιμές: True, False, Don't Know. Με τη χρήση κατάλληλων κατηγορημάτων, μπορούν να εκφραστούν και να αναλυθούν οι διάφορες σχέσεις.

3.2.1.5) Εργαλεία στατικής ανάλυσης

Για τον εντοπισμό απλών σφαλμάτων, έχουν αναπτυχτεί διάφορα εργαλεία στατικής ανάλυσης προγραμμάτων. Στον εντοπισμό του λάθους που οδήγησε στην έκρηξη του πυραύλου Ariane 5, χρησιμοποιήθηκαν εργαλεία αφηρημένης ερμηνείας.

Το LINT για την C που κυκλοφόρησε το 1979, όπως και αρκετά αργότερα, το FINDBUGS για την Java, είναι εργαλεία με μεγάλη σημασία. Επηρέασαν σημαντικά την ανάπτυξη εργαλείων στατικής ανάλυσης, αν και δεν παρέχουν αυστηρές εγγυήσεις.

Ενδεικτικά αναφέρουμε τα παρακάτω εργαλεία στατικής ανάλυσης: CODESONAR για κώδικα C/C++, K7 για κώδικα Java, PREVENT και EXTEND, από την εταιρία Coverity, Astrée για την επαλήθευση λογισμικού ελέγχου πτήσεων αεροσκαφών, C Global Surveyor (CGS), που αναπτύχθηκε στη NASA για λογισμικό διαστημικών αποστολών.

Τέλος, αρκετά εργαλεία χρειάζονται σχολιασμούς (annotations) όπως τύποι, μετά και προ-υποθέσεις. Έτσι μειώνεται η πληροφορία που έχει να επεξεργαστεί ο στατικός αναλυτής και βελτιώνεται η ακρίβεια, αλλά αυξάνεται το βάρος του προγραμματιστή. Στην επόμενη ενότητα θα αναφερθούμε πιο συγκεκριμένα σε αυτό το ζήτημα.

3.2.2) Έλεγχος μοντέλων λογισμικού (Software Model Checking)

3.2.2.1) Εισαγωγή

Με την ανάλυση που γίνεται χωρίς να παίρνεται υπόψη η ροή ή το περιεχόμενο, μπορεί να αποδειχτεί η απουσία απλών σφαλμάτων. Τα εργαλεία ελέγχου μοντέλων μπορούν να αποδείξουν πιο περίπλοκες ιδιότητες που εκφράζονται σε χρονική ή άλλη λογική, είναι πιο ακριβή και έχουν τη δυνατότητα να παρέχουν αντιπαράδειγμα, αν και αντιμετωπίζουν το πρόβλημα της έκρηξης καταστάσεων.

Όπως είδαμε, ο έλεγχος μοντέλων είναι μία αλγοριθμική μέθοδος που προσδιορίζει αν ένα μοντέλο του συστήματος ικανοποιεί μία προδιαγραφή ορθότητας. Το μοντέλο του προγράμματος αποτελείται από καταστάσεις και μεταβάσεις. Η προδιαγραφή ή η ιδιότητα που εξετάζεται είναι μία λογική φόρμουλα. Η κατάσταση, είναι μία αξιολόγηση του μετρητή προγράμματος, των τιμών όλων των μεταβλητών του προγράμματος και της κατάστασης της στοίβας και του σωρού. Οι μεταβάσεις περιγράφουν το πώς σχετίζεται η μία κατάσταση με την άλλη. Οι αλγόριθμοι ελέγχου μοντέλου, εξετάζουν διεξοδικά τις προσβάσιμες καταστάσεις του μοντέλου. Αν βρεθεί μία κατάσταση που παραβιάζει μία ιδιότητα ορθότητας, παράγεται ένα αντιπαράδειγμα. Τα εργαλεία ελέγχου μοντέλου, επαληθεύουν μερικές προδιαγραφές, συνήθως αυτές της ασφάλειας και βιωσιμότητας. Γίνεται παρουσίαση δύο γενικών προσεγγίσεων για την εξερεύνηση του χώρου καταστάσεων.

3.2.2.2) Ρητός και συμβολικός έλεγχος μοντέλων (Explicit and Symbolic Model Checking)

Βασικό ζήτημα του ελέγχου μοντέλων, όπως είδαμε, είναι η έκρηξη του χώρου καταστάσεων. Ο χώρος καταστάσεων, ιδιαίτερα όταν αφορά ένα πρόγραμμα λογισμικού, είναι εκθετικός ως προς διάφορες παραμέτρους, όπως ο αριθμός των μεταβλητών και το πλάτος των τύπων δεδομένων. Υπολογίζοντας και τις κλήσεις συναρτήσεων, και τη δυναμική κατανομή μνήμης, μπορεί να γίνει άπειρος. Ο συγχρονισμός επιδεινώνει το πρόβλημα. Οι αλγόριθμοι ελέγχου μοντέλων, χρησιμοποιούν οδηγίες στο πρόγραμμα για να δημιουργήσουν σύνολα καταστάσεων που θα αναλυθούν. Αυτές οι καταστάσεις χρειάζεται να αποθηκεύονται, ώστε να διασφαλίζεται ότι τις επισκεπτόμαστε το πολύ μία φορά.

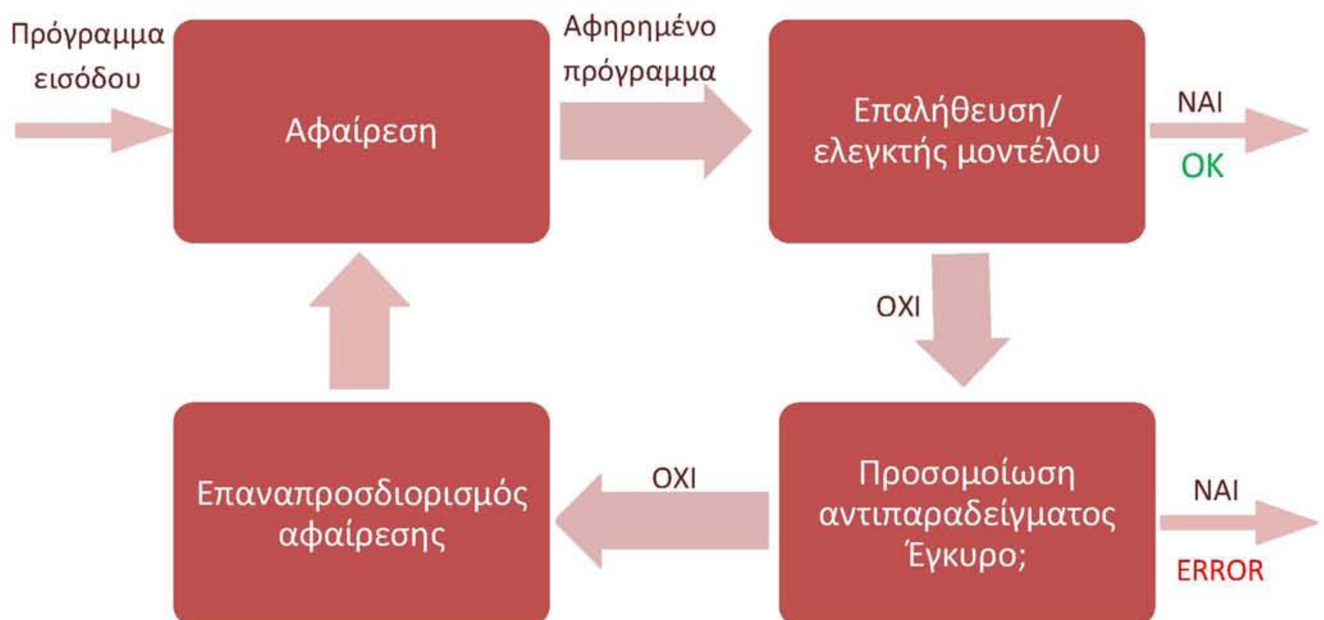
Διακρίνουμε δύο προσεγγίσεις για την αναπαράσταση των καταστάσεων:

- Αλγόριθμοι ελέγχου μοντέλου ρητής-κατάστασης (explicit-state), όπου απευθείας βάζουν δείκτες στις καταστάσεις, και χρησιμοποιούν αλγόριθμους γράφων για την εξερεύνηση του χώρου καταστάσεων. Αφετηρία της εξερεύνησης είναι οι αρχικές καταστάσεις. Η κατασκευή του γράφου γίνεται αναδρομικά, με τη δημιουργία διάδοχων (successors) των αρχικών καταστάσεων. Ο γράφος μπορεί να κατασκευαστεί σε βάθος πρώτα (depth-first), σε πλάτος πρώτα (breadth-first), ή με ευριστικό τρόπο. Μπορούν να εντοπιστούν σφάλματα χωρίς να κατασκευαστεί ολόκληρος ο γράφος, καθώς ελέγχεται απευθείας η παραβίαση ιδιότητας σε καινούριες καταστάσεις. Οι καταστάσεις που εξερευνήθηκαν αποθηκεύονται σε πίνακα κατακερματισμού.
- Αλγόριθμοι συμβολικού ελέγχου μοντέλου (Symbolic model checking), που χρησιμοποιούν ρητές αναπαραστάσεις των συνόλων καταστάσεων. Μπορεί να ξεκινάν από τις αρχικές καταστάσεις, καταστάσεις σφάλματος, ή από την ιδιότητα. Κοινές συμβολικές αναπαραστάσεις είναι τα διαγράμματα δυαδικής απόφασης (BDDs) και η προτασιακή λογική για πεπερασμένα σύνολα και τα πεπερασμένα αυτόματα για άπειρα σύνολα καταστάσεων. Τα BDDs μπορούν να γίνουν πολύ μεγάλα, όπως και τα πεπερασμένα αυτόματα για άπειρα σύνολα καταστάσεων. Οι φόρμουλες προτασιακής λογικής είναι πιο αποδοτικές σε μνήμη, με κόστος στο χρόνο υπολογισμού.

Οι τεχνικές ρητής κατάστασης είναι αποτελεσματικές στον εντοπισμό λαθών και στον χειρισμό συγχρονισμών, ενώ οι συμβολικές τεχνικές αποδεικνύουν αποτελεσματικά την ορθότητα και χειρίζονται καλύτερα το πρόβλημα έκρηξης καταστάσεων (οι συμβολικές τεχνικές επιτρέπουν την επαλήθευση υλικού με πάνω από 10^{20} καταστάσεις, ενώ οι τεχνικές ρητής κατάστασης επαληθεύουν μόνο μερικές χιλιάδες). Μία προσέγγιση για την αντιμετώπιση της έκρηξης του χώρου καταστάσεων είναι η αφαίρεση: αναλύεται μία αφαίρεση του προγράμματος με μικρότερο χώρο καταστάσεων. Η αυτόματη αφαίρεση βασίζεται στην αφηρημένη ερμηνεία, αν και οι αλγόριθμοι διαφέρουν.

3.2.2.3) Αφαίρεση κατηγορήματος (Predicate Abstraction)

Η κυρίαρχη τεχνική αφαίρεσης στον έλεγχο μοντέλου λογισμικού είναι η αφαίρεση κατηγορήματος. Χρησιμοποιεί λογικά κατηγορήματα για την κατασκευή αφηρημένου πεδίου, διαχωρίζοντας τον χώρο καταστάσεων του προγράμματος. Η διαδικασία διαφέρει από την αφηρημένη ερμηνεία, καθώς η αφαίρεση καθορίζεται από το πρόγραμμα και είναι συγκεκριμένη για αυτό. Η πρόκληση σε αυτήν την τεχνική είναι η αναγνώριση κατηγορημάτων, καθώς αυτά είναι που καθορίζουν την ακρίβεια της αφαίρεσης. Η πιο δημοφιλής τεχνική για την αυτόματη επαλήθευση προγράμματος που βασίζεται σε αφαίρεση κατηγορήματος, είναι η τεχνική επαναπροσδιορισμού της αφαίρεσης, καθοδηγούμενοι από το αντιπαράδειγμα, ή αλλιώς η τεχνική CEGAR (Counterexample-Guided Abstraction Refinement). Αν ο έλεγχος μοντέλου της αφαίρεσης δίνει ένα αντιπαράδειγμα που δεν υπάρχει στο συγκεκριμένο πρόγραμμα, τότε χρησιμοποιείται το αφηρημένο αντιπαράδειγμα για να εντοπίσει καινούρια κατηγορήματα και να επιτευχθεί μία αφαίρεση με μεγαλύτερη ακρίβεια.



Σχήμα 3.2: Αλγόριθμος CEGAR

Στο **σχήμα 3.2** φαίνεται ο βασικός βρόχος του αλγόριθμου. Αποτελείται από τα εξής τέσσερα βήματα: κατασκευή αφηρημένου προγράμματος, έλεγχος μοντέλου, ανάλυση του αντιπαραδείγματος, επαναπροσδιορισμός της αφαίρεσης. Μία τέτοια αφαίρεση συνήθως επιτρέπει ψευδείς εκτελέσεις, που δεν ανταποκρίνονται σε πραγματικές εκτελέσεις του αυθεντικού προγράμματος. Σε κάθε επανάληψη το αφηρημένο πρόγραμμα κατασκευάζεται ξανά από το μηδέν.

Τα τέσσερα βασικά βήματα περιγράφονται ως εξής:

1) Αφαίρεση. Για το δοσμένο πρόγραμμα εισόδου, κατασκευάζεται ένα αφηρημένο δυαδικό πρόγραμμα. Το δυαδικό πρόγραμμα περιλαμβάνει δυαδικές μεταβλητές που αναπαριστούν τιμές των κατηγορημάτων αφαίρεσης και αποτυπώνουν τις επιδράσεις των δηλώσεων του αυθεντικού προγράμματος στις τιμές των κατηγορημάτων. Κάθε δήλωση ανάθεσης στο αυθεντικό πρόγραμμα μοντελοποιείται από μία ανάθεση δυαδικής τιμής στις μεταβλητές που αντιστοιχούν σε ατομικά κατηγορήματα στο αφηρημένο πρόγραμμα. Οι συνθήκες διακλάδωσης μοντελοποιούνται από την δυαδική μεταβλητή του αντίστοιχου κατηγορήματος αφαίρεσης.

2)Επαλήθευση. Με τη χρήση ενός ελεγκτή μοντέλου, επαληθεύεται η ορθότητα του αφηρημένου προγράμματος. Αν το αφηρημένο πρόγραμμα είναι ασφαλές, τότε είναι ασφαλές και το αυθεντικό πρόγραμμα και ο βρόχος CEGAR τερματίζει. Αλλιώς, ο ελεγκτής μοντέλου παράγει ένα αντιπαράδειγμα, στη μορφή ίχνους που αναπαριστά την εκτέλεση του προγράμματος από την αρχική κατάσταση στην κατάσταση σφάλματος. Όλοι οι ελεγκτές μοντέλου για δυαδικά προγράμματα είναι συμβολικοί.

3)Προσομοίωση. Σε αυτό το βήμα εξετάζεται το αντιπαράδειγμα για να προσδιοριστεί αν το σφάλμα υπάρχει στο συγκεκριμένο πρόγραμμα, ή είναι ψευδές (spurious). Αυτό γίνεται μέσω της προσομοίωσης του συγκεκριμένου προγράμματος, χρησιμοποιώντας το αφηρημένο αντιπαράδειγμα, ώστε να βρεθεί αν το αντιπαράδειγμα αντιπροσωπεύει την πραγματική συμπεριφορά του προγράμματος. Αν όντως μπορεί να πραγματοποιηθεί το αντιπαράδειγμα, τότε αναφέρεται το σφάλμα και ο βρόχος CEGAR τερματίζει. Αλλιώς, η αφαίρεση χρειάζεται να επαναπροσδιοριστεί ώστε να εξαλειφθεί το μονοπάτι σφάλματος. Με αυτή την προσέγγιση δεν μπορούν να παραχθούν λάθος μηνύματα σφάλματος.

4)Επαναπροσδιορισμός. Τα ανέφικτα αντιπαραδείγματα εξαλείφονται μέσω του επαναπροσδιορισμού της αφαίρεσης. Προσδιορίζονται καινούρια κατηγορήματα με βάση το αντιπαράδειγμα, που χρησιμοποιούνται στην επόμενη επανάληψη και έτσι βελτιώνεται η ακρίβεια της αφαίρεσης, ώστε να μην επιτρέπεται πλέον το αδύνατο αντιπαράδειγμα.

Η αποτελεσματικότητα αυτής της διαδικασίας εξαρτάται από την αποτελεσματικότητα της αφαίρεσης του προγράμματος και των διαδικασιών επαναπροσδιορισμού κατηγορημάτων. Η αφαίρεση του προγράμματος εστιάζει στην κατασκευή των σχέσεων μετάβασης του αφηρημένου προγράμματος, ενώ ο επαναπροσδιορισμός κατηγορήματος εστιάζει στο να ορίσει αποτελεσματικές τεχνικές ώστε να εξαλείψει τα ψευδά αντιπαραδείγματα. Και στα δύο πεδία έρευνας, βασικός παράγοντας είναι το χαμηλό υπολογιστικό κόστος.

Η τεχνική CEGAR αποτελεί τη βάση για μεγάλο αριθμό αυτοματοποιημένων προσεγγίσεων στην επαλήθευση προγραμμάτων.

3.2.2.4) Εργαλεία ελέγχου μοντέλων

Model Checking: το πιο ευρέως διαδεδομένο εργαλείο ελέγχου μοντέλου ρητής-κατάστασης είναι το SPIN. Αρχικά χρησιμοποιήθηκε για την επαλήθευση ιδιοτήτων χρονικής λογικής πρωτόκολλων επικοινωνίας, στη γλώσσα PROMELA. Κάποιοι ελεγκτές μοντέλου λογισμικού, όπως μία πρώτη έκδοση του Java Pathfinder (JPF), μεταφράζει τον κώδικα της Java σε PROMELA και χρησιμοποιεί το SPIN για τον έλεγχο μοντέλου. Επειδή όμως η PROMELA δεν υποστηρίζει σημαντικές λειτουργίες, όπως τη δυναμική κατανομή μνήμης, δεν είναι κατάλληλη για μοντελοποίηση της γλώσσας Java. Οι πιο πρόσφατες εκδόσεις του JPF αναλύουν απευθείας των κώδικα της Java και υποστηρίζουν συμβολικές τεχνικές για σκοπούς ελέγχου του λογισμικού. Άλλα εργαλεία ελέγχου μοντέλου λογισμικού ρητής-κατάστασης είναι τα CMC και ZING.

Predicate Abstraction: το πιο ευρέως διαδεδομένο εργαλείο ελέγχου μοντέλου με αφαίρεση κατηγορήματος είναι το SLAM της Microsoft. Το SLAM περιλαμβάνει το εργαλείο αφαίρεσης κατηγορήματος C2BP, τον ελεγκτή μοντέλου βασισμένου σε BDD, BEBOP για δυαδικά προγράμματα και το εργαλείο προσομοίωσης και επαναπροσδιορισμού, NEWTON. Στη θέση του BEBOP, για τον έλεγχο προδιαγραφών χρονικής λογικής, μπορεί να χρησιμοποιηθεί ο ελεγκτής μοντέλου βασισμένου σε BDD, MOPED.

Το εργαλείο BLAST χρησιμοποιεί την τεχνική lazy abstraction: στο βήμα επαναπροσδιορισμού γίνεται αφαίρεση μόνο των σχετικών κομματιών του αυθεντικού προγράμματος. Η πιο στενή ολοκλήρωση της επαλήθευσης και του σταδίου του επαναπροσδιορισμού, επιτρέπει επιτάχυνση των επαναλήψεων CEGAR.

Τα εργαλεία επαλήθευσης που αναφέρθηκαν, χρησιμοποιούν αποδείκτες θεωρήματος γενικού σκοπού και BDDs για τον έλεγχο μοντέλου. Το εργαλείο SATABS χρησιμοποιεί έναν SAT solver για την κατασκευή των αφαιρέσεων και τη συμβολική προσομοίωση των αντιπαραδειγμάτων.

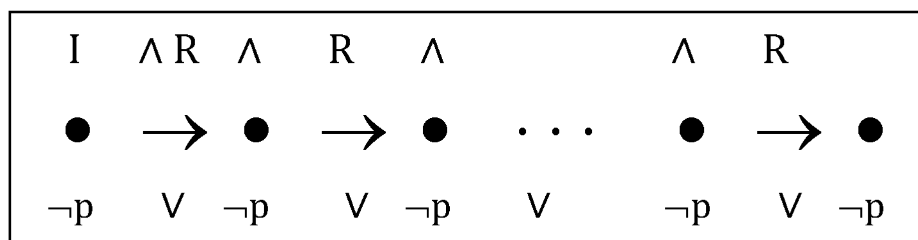
Στην πράξη, τα αντιπαραδείγματα που παρέχονται από τους ελεγκτές μοντέλων έχουν συχνά μεγαλύτερη αξία από μία απόδειξη ορθότητας. Η αφαίρεση κατηγορήματος σε συνδυασμό με την τεχνική CEGAR ελέγχουν κατάλληλα ιδιότητες ασφάλειας σχετικές με τον έλεγχο ροής. Αν και δεν μπορούμε να έχουμε false positives, ο κύκλος αφαίρεση-επαναπροσδιορισμός, μπορεί να μην τερματίζει. Η επιτυχία της προσέγγισης εξαρτάται καθοριστικά από το βήμα του επαναπροσδιορισμού. Η τεχνική της αφαίρεσης κατηγορήματος δεν δουλεύει καλά με πολύπλοκες δομές δεδομένων ή πίνακες, βασισμένους σε σωρό.

3.2.3) Έλεγχος φραγμένων μοντέλων (Bounded Model Checking-BMC)

3.2.3.1) Υπόβαθρο

Ο έλεγχος φραγμένου μοντέλο εισήχθη το 1999 από τον Biere, ως μία συμπληρωματική τεχνική στον μη φραγμένο έλεγχο μοντέλου, βασισμένου σε BDDs. Εφαρμόζεται πλατιά στις τεχνικές τυπικής επαλήθευσης στον κλάδο των ημιαγωγών. Χρησιμοποιούμε τον όρο «φραγμένο» (bounded), γιατί εξερευνώνται μόνο οι καταστάσεις οι οποίες είναι προσβάσιμες με έναν φραγμένο αριθμό βημάτων, k. Στην τεχνική BMC, το μοντέλο που επαληθεύουμε ξετυλίγεται k φορές και γίνεται σύζευξη με την άρνηση της ιδιότητας που εξετάζουμε, ώστε να σχηματιστεί μία προτασιακή φόρμουλα, η οποία εξετάζεται από έναν SAT-solver. Η φόρμουλα μπορεί να ικανοποιηθεί, αν και μόνο αν, υπάρχει μονοπάτι μήκους k όπου δεν ισχύει η ιδιότητα, οπότε και έχει βρεθεί μία ανάθεση που αντιστοιχεί σε ένα αντιπαράδειγμα μήκους k. Η τεχνική είναι αμφισβητήσιμη αν η φόρμουλα δεν μπορεί να ικανοποιηθεί, καθώς μπορεί να υπάρχουν αντιπαραδείγματα με μήκος μεγαλύτερο από k. Μπορεί να αναγνωρίσει όμως πολλά σφάλματα που αλλιώς δεν θα παρατηρούνταν.

Έστω **R**, υποδηλώνει τη σχέση μετάβασης του μοντέλου **M** και περιέχει τα ζευγάρια τρεχουσών και επόμενων καταστάσεων **s,** το **I** υποδηλώνει την αρχική κατάσταση κατηγορήματος και **p** την ιδιότητα που εξετάζουμε. Για να πάρουμε ένα στιγμιότυπο της BMC με k βήματα, αναπαράγουμε την σχέση μετάβασης k φορές. Οι μεταβλητές σε κάθε αναπαραγωγή μετονομάζονται έτσι ώστε η επόμενη κατάσταση του βήματος i να χρησιμοποιείται ως η τρέχουσα κατάσταση του βήματος i+1. Η τρέχουσα κατάσταση κατασκευάζεται από το I, γίνεται σύζευξη των σχέσεων μετάβασης και μία από τις καταστάσεις πρέπει να ικανοποιεί την ¬p. Έτσι, μία ανάθεση που ικανοποιεί τη φόρμουλα, αντιστοιχεί σε ένα μονοπάτι από την αρχική κατάσταση, σε μία κατάσταση που παραβιάζει την p. Το μέγεθος της φόρμουλας είναι γραμμικό στο μέγεθος του μοντέλου και του k.



Σχήμα 3.3

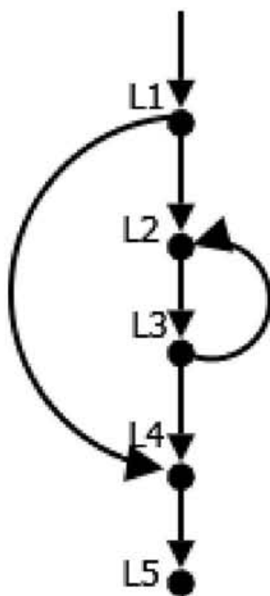
Οπότε έχουμε τη σχέση:

$$\mathrm{BMC}(\mathrm{M},\mathrm{p},\mathrm{k}) = \mathrm{I}(\mathrm{s}_0) \wedge \bigwedge_{i=0}^{k-1} \mathrm{R}(\mathrm{s}_\mathrm{i},\mathrm{s}_{\mathrm{i}+1}) \wedge \bigvee_{\mathrm{i}=0}^{\mathrm{k}} \neg \mathrm{p}(\mathrm{s}_\mathrm{i})$$

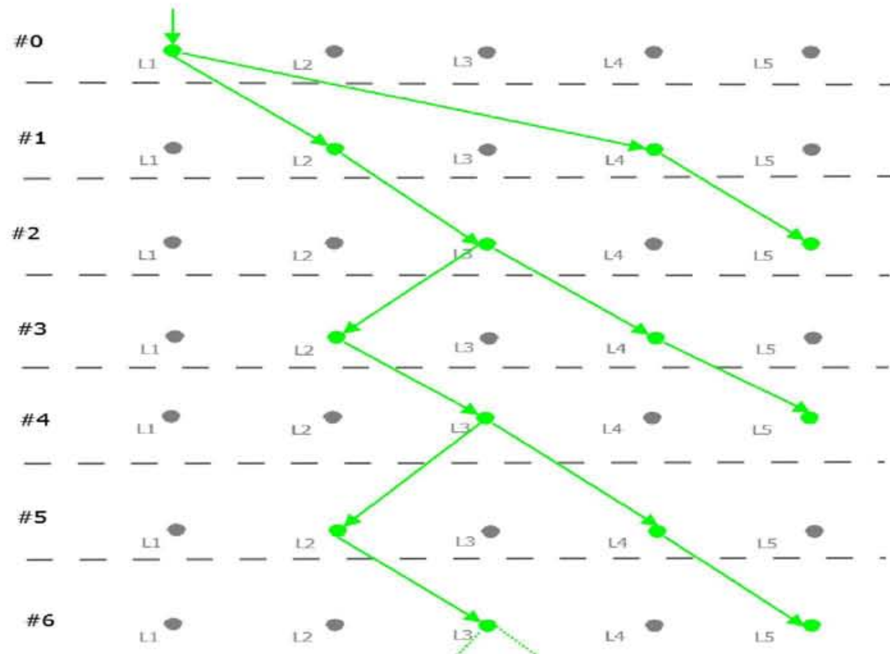
3.2.3.2) Ξετύλιγμα ολόκληρου του προγράμματος μονομιάς

Η BMC εφαρμόζεται και σε λογισμικό επιπέδου συστήματος. Ο πιο άμεσος τρόπος για την εφαρμογή της BMC σε λογισμικό, είναι ο χειρισμός ολόκληρου του προγράμματος ως μία σχέση μετάβασης. Ένα ξετύλιγμα (unwinding) με k βήματα, επιτρέπει την εξερεύνηση όλων των μονοπατιών του προγράμματος με μήκος k ή λιγότερο. Το πρώτο βήμα είναι να αναλυθεί η πιθανή ροή έλεγχου του προγράμματος. Το μέγεθος του βασικού ξετυλίγματος είναι k φορές το μέγεθος του προγράμματος. Καθώς για μεγάλα προγράμματα γίνεται απαγορευτικό, έχουν προταθεί πολλές βελτιστοποιήσεις.

Παράδειγμα: Έστω το μικρό διάγραμμα ροής ελέγχου του **σχήματος 3.4**. Κάθε κόμβος αντιστοιχεί σε ένα βασικό block και οι ακμές στην πιθανή ροή έλεγχου μεταξύ των block. Στο **σχήμα 3.5**, που φαίνεται το ξετύλιγμα της σχέσης μετάβασης, παρατηρούμε ότι το block L1 μπορεί να εκτελεστεί μόνο στο πρώτο βήμα κάθε μονοπατιού. Αντίστοιχα, το L2 μπορεί να εκτελεστεί μόνο στα βήματα 2, 4, 6 κοκ.



Σχήμα 3.4



Σχήμα 3.5

3.2.3.3) Ξετύλιγμα βρόχων ξεχωριστά

Παρατηρώντας το **σχήμα 3.4**, φαίνεται ότι όλα τα μονοπάτια του CFG, επισκέπτονται τα L4 και L5 το πολύ μία φορά. Το ξετύλιγμα, όμως, της σχέσης μετάβασης στο **σχήμα 3.5**, συμπεριλαμβάνει τρία αντίγραφα των L4, L5. Ο πλεονασμός αυτός μπορεί να εξαλειφθεί κατασκευάζοντας μία φόρμουλα που ακολουθεί ένα συγκεκριμένο μονοπάτι εκτέλεσης, και όχι με το χειρισμό ολόκληρου του προγράμματος ως σχέση μετάβασης. Στο ξετύλιγμα βρόχου, αντί να ξετυλιχτεί ολόκληρη η σχέση μετάβασης, κάθε επανάληψη ξετυλίγεται ξεχωριστά. Συντακτικά, αυτό ανταποκρίνεται στην αναπαραγωγή του σώματος

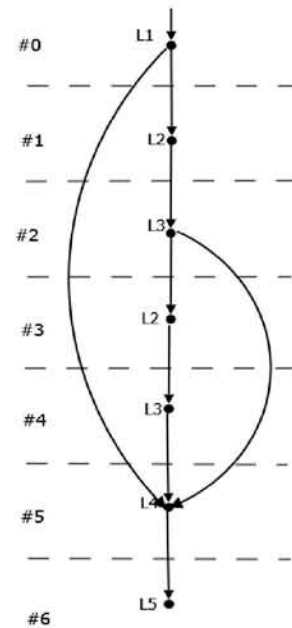
του βρόχου με έναν κατάλληλο φρουρό (ουσιαστικά πρόκειται για μία κατάλληλη if δήλωση, για βρόχους που τερματίζουν πριν την k επανάληψη).

```
while(x){
   BODY;
}
```

→

```
if(x){
   BODY;
   if(x){
      BODY;
   }else{
      assume(false);
   }
}
```

Σχήμα 3.6: Ξετύλιγμα ενός while βρόχου, με βάθος 2.



Σχήμα 3.7

Η επίδρασή του στο CFG φαίνεται στο **σχήμα 3.7**, όπου ο βρόχος μεταξύ των L2 και L3 ξετυλίγεται k=2 φορές.

Αυτό το ξετύλιγμα μπορεί να οδηγήσει σε πιο συμπαγείς φόρμουλες και να χρειαστεί λιγότερους διαχωρισμούς τύπων στην φόρμουλα, αλλά μπορεί να χρειαστεί περισσότερο χρόνο για να φτάσει σε συγκεκριμένες περιοχές.

Το ξετύλιγμα βρόχου διαφέρει από την εξερεύνηση απαρίθμησης που βασίζεται σε μονοπάτι (enumerative path-based exploration). Στο παράδειγμα, το μονοπάτι που ανταποκρίνεται στη διακλάδωση από το L1 στο L4, συγχωνεύεται με το μονοπάτι που ακολουθεί το βρόχο. Έτσι, η φόρμουλα που δημιουργείται είναι γραμμική στο βάθος και στο μέγεθος του προγράμματος, ακόμα και αν υπάρχει εκθετικός αριθμός μονοπατιών στο CFG.

3.2.3.4) Ολοκληρωμένος BMC για λογισμικό

Η μέθοδος BMC, όπως περιγράφηκε, είναι ατελής από τον ορισμό της, καθώς ψάχνει παραβιάσεις της ιδιότητας μόνο μέχρι ένα δοσμένο βάθος. Σφάλματα που βρίσκονται πιο βαθιά, δεν εντοπίζονται. Έτσι, ποτέ δεν μπορεί να επιστρέψει "No Errors". Παρ' όλα αυτά, αν χρησιμοποιηθεί με έναν ελαφρώς διαφορετικό τρόπο, μπορεί να χρησιμοποιηθεί για να αποδείξει ιδιότητες βιωσιμότητας και ασφάλειας. Διαισθητικά, καταλαβαίνουμε ότι αν μπορούσαμε να ψάξουμε αρκετά βαθιά, θα μπορούσαμε να εγγυηθούμε ότι έχουν εξεταστεί όλες οι σχετικές συμπεριφορές. Η πιο βαθιά αναζήτηση θα έβρισκε μόνο καταστάσεις που έχουν ήδη εξερευνηθεί. Το βάθος που μπορεί να το εγγυηθεί αυτό ονομάζεται όριο πληρότητας (completeness threshold). Ο υπολογισμός του κατώτερου ορίου

πληρότητας είναι τόσο δύσκολος, όσο και ο έλεγχος μοντέλου, κατά συνέπεια, στην πράξη γίνονται υπερ-προσεγγίσεις.

Στο λογισμικό, ένας τρόπος για τον καθορισμό του ορίου βάθους για ένα πρόγραμμα, είναι να καθοριστεί σε υψηλό επίπεδο, ο χρόνος εκτέλεσης χειρότερης περίπτωσης (WCET). Ο χρόνος αυτός δίνεται ως ένα όριο στον μέγιστο αριθμό επαναλήψεων των βρόχων και υπολογίζεται, συνήθως, με απλή συντακτική ανάλυση της δομής των βρόχων. Αν αποτύχει η συντακτική ανάλυση, τότε μπορεί να εφαρμοστεί ένας επαναληπτικός αλγόριθμος. Πρώτα μαντεύουμε για το όριο του αριθμού των επαναλήψεων του βρόχου. Στη συνέχεια ο βρόχος ξετυλίγεται μέχρι το συγκεκριμένο όριο, όπως στο σχήμα παραπάνω, αλλά με τη διαφορά ότι αντί για παραδοχή (assumption), χρησιμοποιούμε έναν ισχυρισμό (assertion), που ονομάζεται unwinding assertion. Αν ο assertion παραβιάζεται, τότε υπάρχουν μονοπάτια του προγράμματος που ξεπερνάν το όριο και χρειάζεται να μαντέψουμε και πάλι για το όριο. Αυτή η μέθοδος είναι εφαρμόσιμη αν το πρόγραμμα, ή το σώμα του κεντρικού βρόχου, έχει ένα όριο χρόνου εκτέλεσης.

3.2.3.5) Εργαλεία που υλοποιούν τον BMC

Η τεχνική BMC είναι η καλύτερη για τον εντοπισμό «ρηχών» σφαλμάτων. Παρέχει ένα πλήρες ίχνος του αντιπαραδείγματος, στην περίπτωση που εντοπίζει σφάλμα. Είναι, όμως, πλήρης μόνο για «ρηχά» προγράμματα. Υποστηρίζει το μεγαλύτερο εύρος των κατασκευών των προγραμμάτων, συμπεριλαμβανόμενων των δομών δεδομένων δυναμικά κατανεμημένης μνήμης.

Για την επαλήθευση λογισμικού, χρησιμοποιούνται αρκετές υλοποιήσεις της τεχνικής BMC. Μία από τις πρώτες, για προγράμματα σε C είναι η CBMC που αναπτύχθηκε στην CMU. Υποστηρίζει επίσης C++, SpecC, and SystemC και χρησιμοποιεί την τεχνική ξετυλίγματος βρόχου. Η κύρια εφαρμογή της είναι στον έλεγχο της συνέπειας των μοντέλων κυκλώματος σε επίπεδο συστήματος, σε C ή SystemC, με την υλοποίηση σε Verilog. Η IBM ανέπτυξε μία έκδοση του CBMC για ταυτόχρονα προγράμματα. Το μόνο εργαλείο που υλοποιεί το ξετύλιγμα ολόκληρου του συστήματος μετάβασης, είναι το F-SOFT που αναπτύχθηκε στη NEC Research.

Μία εξειδικευμένη υλοποίηση της BMC, προσαρμοσμένη στις ιδιότητες που ελέγχει, αποτελεί και το εργαλείο SATURN. Ελέγχει δύο διαφορετικές ιδιότητες του κώδικα του Linux kernel: αναφορές σε null-pointers και κλείδωμα των API συμβάσεων. Υλοποιεί το ξετύλιγμα βρόχου και εκτελεί το πολύ δύο ξετυλίγματα του κάθε βρόχου. Σφάλματα που χρειάζονται περισσότερες φορές ξετύλιγμα, δεν εντοπίζονται.

3.3) Επιλογή κατάλληλης προσέγγισης

Σε αυτή την ενότητα έγινε παρουσίαση των τριών κύριων τεχνικών για την αυτόματη επαλήθευση του λογισμικού. Συμπερασματικά, οι τεχνικές στατικής ανάλυσης βασισμένες στην αφηρημένη ερμηνεία, κλιμακώνουν καλά με κόστος την περιορισμένη ακρίβεια, που εκδηλώνεται με, πιθανά, μεγάλο αριθμό ψευδών προειδοποιήσεων. Τα εργαλεία ελέγχου μοντέλου που χρησιμοποιούν αφαίρεση, μπορούν να ελέγξουν πολύπλοκες ιδιότητες ασφάλειας και να δημιουργήσουν αντιπαραδείγματα. Οι ελεγκτές φραγμένου μοντέλου είναι πολύ ισχυροί στον εντοπισμό ρηχών σφαλμάτων, αλλά δεν είναι δε μπορούν να αποδείξουν ακόμα και απλές ιδιότητες, αν το πρόγραμμα έχει βρόχους που είναι πιο βυθείς.

4 ΜΙΑ ΑΛΛΗ ΠΡΟΣΕΓΓΙΣΗ

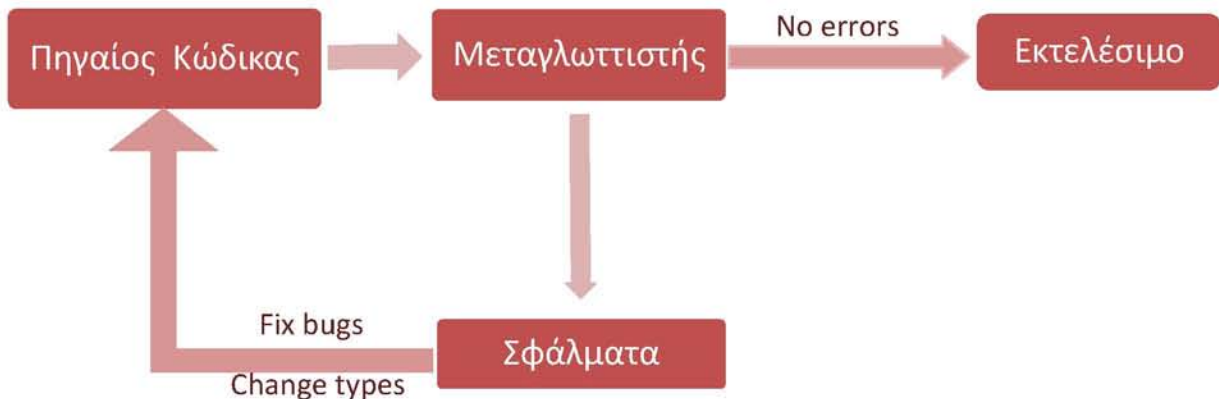
4.1) Εισαγωγή

Στην προηγούμενη ενότητα, στην παρουσίαση της τεχνικής της στατικής ανάλυσης για την επαλήθευση προγραμμάτων, αναφέραμε ότι υπάρχουν διάφορα εργαλεία στατικής ανάλυσης, που χρειάζονται διάφορα annotations, όπως χαρακτηρισμοί τύπων, για να κάνουν την ανάλυση. Δεν επεκταθήκαμε περισσότερο σε αυτή τη μέθοδο επαλήθευσης, γιατί με αυτόν τον τρόπο αυξάνεται αρκετά το βάρος του προγραμματιστή, καθώς χρειάζεται να προσθέτει στο πρόγραμμά του τέτοια annotations. Η προσθήκη, όμως, των annotations, μπορεί να μειώσει την ποσότητα της πληροφορίας που έχει να υπολογίσει ο στατικός αναλυτής και να βελτιώσει την ακρίβεια. Σε αυτή την ενότητα, θα γίνει παρουσίαση αυτής της μεθόδου ελέγχου τύπων για επαλήθευση, με χρήση annotations. Στη συνέχεια, θα δούμε κατά πόσο το βάρος του προγραμματιστή για την προσθήκη των annotations μπορεί να γίνει μικρότερο με χρήση του κοινού (crowdsourcing).

4.2) Έλεγχος τύπων (type checking)

Η πιο πλατιά χρησιμοποιημένη τεχνική στατικής ανάλυσης, και αυτή με την οποία είναι εξοικειωμένοι οι περισσότεροι προγραμματιστές, είναι ο έλεγχος τύπων. Οι τύποι, είναι μέρος όλων σχεδόν των κύριων προγραμματιστικών γλωσσών. Οι κανόνες του ελέγχου τύπων ορίζονται τυπικά από τη γλώσσα προγραμματισμού και επιβάλλονται από τον μεταγλωττιστή. Ως τεχνική στατικής ανάλυσης, χρησιμοποιεί την αφαίρεση. Στα συστήματα τύπων οι προσεγγίσεις (approximations) ονομάζονται τύποι. Μεταξύ άλλων, ελέγχεται ότι τα τελούμενα κάθε τελεστή έχουν τους τύπους που προβλέπονται από τις προδιαγραφές της αρχικής γλώσσας και ότι οι παράμετροι των υποπρογραμμάτων συμφωνούν με τους τύπους που καθορίζονται στις δηλώσεις τους. Για παράδειγμα, η δεικτοδότηση ενός πίνακα από κάποιο πραγματικό αριθμό, στις περισσότερες γλώσσες προγραμματισμού δεν επιτρέπεται.

Οι τύποι περιγράφουν τις τιμές που υπολογίζονται κατά την εκτέλεση του προγράμματος. Τα σφάλματα τύπων αναφέρονται σε εσφαλμένη ή ανεπιθύμητη συμπεριφορά του προγράμματος που προκαλείται από αναντιστοιχία μεταξύ των τύπων δεδομένων του προγράμματος, πχ η χρήση ενός πραγματικού αριθμού ως ακέραιου. Ασφάλεια τύπων (type-safety) ονομάζουμε την απουσία σφαλμάτων τύπων. Ο έλεγχος τύπων είναι η διαδικασία της επαλήθευσης της ασφάλειας τύπου ενός προγράμματος.



Σχήμα 4.1

Στον στατικό έλεγχο τύπων, ο έλεγχος γίνεται κατά τη μεταγλώττιση, ενώ στον δυναμικό, γίνεται κατά την εκτέλεση του προγράμματος. Ανάλογα με το πότε ορίζονται οι τύποι σε κάθε γλώσσα (στη μεταγλώττιση ή στην εκτέλεση), διακρίνονται σε γλώσσες στατικών και δυναμικών τύπων. Επίσης, άλλη μια διάκριση των γλωσσών προγραμματισμού γίνεται ανάλογα με το αν οι τύποι είναι ισχυροί ή αδύναμοι. Οι γλώσσες ισχυρών τύπων (strongly typed) εγγυώνται ότι τα αποδεκτά προγράμματα έχουν ασφάλεια τύπων, ενώ οι γλώσσες αδύναμων τύπων (weakly typed), επιτρέπουν προγράμματα με σφάλματα τύπων.

Τα ορθά συστήματα τύπων διαβεβαιώνουν στατικά ότι το πρόγραμμα έχει ασφάλεια τύπων. Η ορθότητα (soundness) προϋποθέτει χρήση ισχυρών τύπων. Ο στατικός έλεγχος εγγυάται ότι όλες οι εκτελέσεις είναι ασφαλείς. Στον παρακάτω πίνακα φαίνεται η διάκριση μεταξύ κάποιων γλωσσών προγραμματισμού, ανάλογα με τους τύπους που χρησιμοποιούν.

	Ισχυροί τύποι (strong typing)	**Αδύναμοι τύποι (weak typing)**
Στατικοί τύποι (static typing)	Java Pascal ML Modula-3	C C++
Δυναμικοί τύποι (dynamic typing)	Perl Smalltalk	Assembly code

Πίνακας 4.1

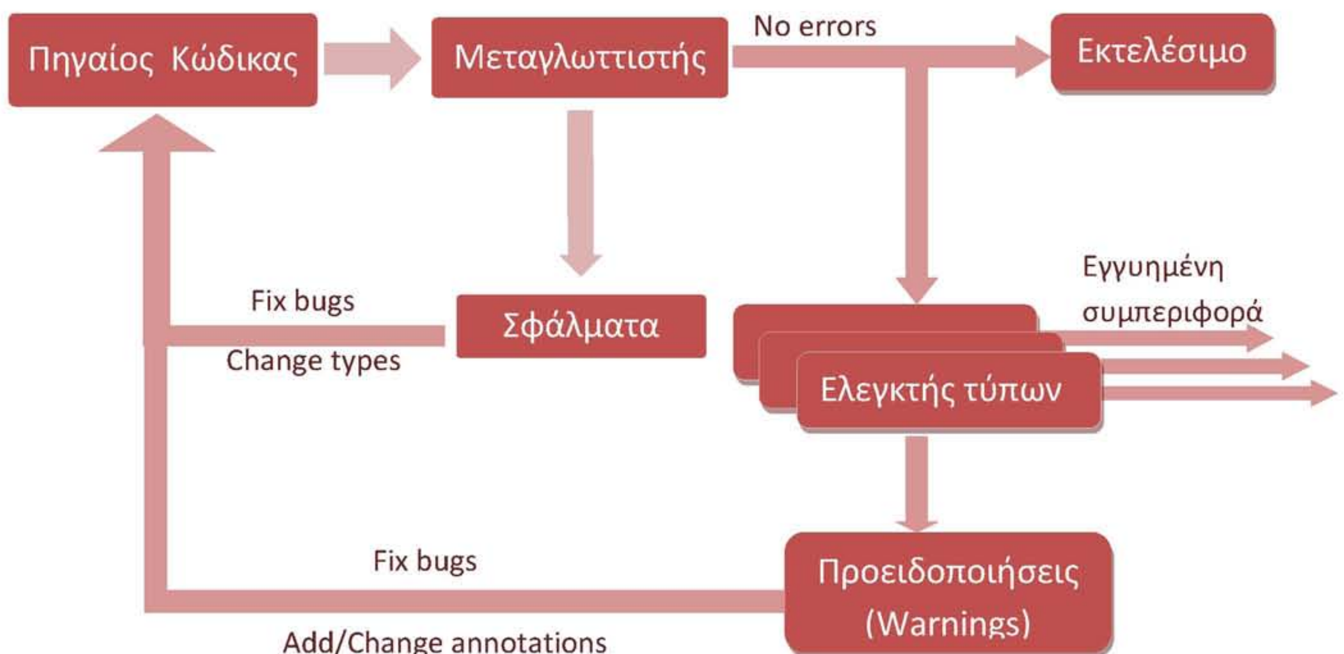
Οι εκφράσεις τύπων (type expressions) περιγράφουν τους πιθανούς τύπους του προγράμματος, πχ `int`, `string`, `array[]`, `Object` κλπ. Στα συστήματα τύπων των γλωσσών, υπάρχουν οι βασικοί τύποι, όπως `int`, `boolean`, `float`. Χρησιμοποιώντας τους βασικούς τύπους, χτίζονται οι κατασκευαστές τύπων (type constructors), όπως για παράδειγμα οι τύποι πινάκων, δομών, ή συναρτήσεων.

Η δουλειά του ελέγχου των τύπων στις γλώσσες στατικού ελέγχου, γίνεται από τον μεταγλωττιστή ενώ τα συστήματα τύπων καθορίζονται από τις προδιαγραφές της κάθε γλώσσας.

4.2.1) Επαλήθευση μέσω ελέγχου τύπων (verification via type-check)

Ο έλεγχος τύπων, ακόμα και στις γλώσσες με ισχυρούς τύπους, δεν μπορεί να αποτρέψει αρκετά σφάλματα, καθώς οι ενσωματωμένοι τύποι των γλωσσών, δεν μπορούν να είναι αρκετά εκφραστικοί ώστε να καλύπτουν όλες τις κατηγορίες σφαλμάτων. Για παράδειγμα, ο ελεγκτής τύπων της Java, εγγυάται ότι ένα πρόγραμμα δεν μπορεί να τερματίσει με μία εξαίρεση μεθόδου που δεν βρέθηκε (`MethodNotFoundException`). Πολλές ιδιότητες ασφάλειας όμως, δεν καλύπτονται, όπως οι εξαιρέσεις null-pointer.

Αν επεκτείναμε όμως, το ενσωματωμένο σύστημα τύπων μιας γλώσσας, θα μπορούσαμε να εκφράσουμε πολύ περισσότερες ιδιότητες και έτσι να εγγυηθούμε την απουσία σφαλμάτων πολύ περισσότερων κατηγοριών. Αυτό μπορεί να γίνει με την προσθήκη σχολιασμών (annotations) στον κώδικα που στη συνέχεια θα ελέγχονται από κάποιο ελεγκτή τύπων.



Σχήμα 4.2

4.2.2) Σχολιασμοί γενικά (annotations)

Οι σχολιασμοί (annotations) είναι μια μορφή μεταδεδομένων, που παρέχουν πληροφορίες για τον κώδικα του προγράμματος, αλλά δεν είναι οι ίδιοι μέρος

του προγράμματος και έτσι, δεν έχουν άμεση επίδραση στον κώδικα που σχολιάζουν.

Τα annotations γενικά, έχουν μια σειρά χρήσεων, όπως:

- Πληροφορίες για τον μεταγλωττιστή. Μπορούν να χρησιμοποιηθούν από τον μεταγλωττιστή για τον εντοπισμό σφαλμάτων ή την αγνόηση προειδοποιήσεων.
- Επεξεργασία κατά τη μεταγλώττιση ή ανάπτυξη. Διάφορα εργαλεία λογισμικού μπορούν να επεξεργαστούν τις πληροφορίες ώστε να δημιουργήσουν κώδικα, αρχεία XML κα.
- Επεξεργασία κατά την εκτέλεση. Κάποια annotations μπορεί να εξεταστούν στο χρόνο της εκτέλεσης.

Στην Java, τα annotations αποτελούν ένα ιδιαίτερο είδος τροποποιητή (modifier). Ένα annotation ξεκινά πάντα με το σύμβολο @. Αυτό δείχνει στον μεταγλωττιστή ότι ακολουθεί annotation. Μπορεί να περιλαμβάνουν στοιχεία και τιμές για αυτά. Πχ:

```
@Author(
   name = "Karamitrou Eirini",
   date = "3/4/2016"
)
class MyClass() { ... }
```

Παράδειγμα 1

και

```
@SuppressWarnings(value = "unchecked")
void myMethod() { ... }
```

Παράδειγμα 2

Ο τύπος των annotations μπορεί να είναι είτε από τους προκαθορισμένους (predefined annotations types) είτε να ορίζονται νέοι. Ο τύπος `@SuppressWarnings` ανήκει στους προκαθορισμένους. Άλλοι τύποι annotations που συμπεριλαμβάνονται στο Java SE API είναι οι: `@Deprecated`, `@Override`, `@Inherited`, `@Retention`. Το annotation `@Author` του παραδείγματος, δεν ανήκει στα προκαθορισμένο, αλλά είναι τροποποιημένο (custom).

Πολλά annotations αντικαθιστούν τα σχόλια στον κώδικα, όπως πχ το `@Author`. Είναι μία μορφή διεπαφών (interfaces). Πριν την Java SE 8, τα annotations εφαρμόζονταν στις δηλώσεις κλάσεων, πεδίων, μεθόδων και άλλων στοιχείων των προγραμμάτων. Στην Java SE 8 μπορούν να εφαρμοστούν σε κάθε χρήση τύπου, οπουδήποτε, δηλαδή, χρησιμοποιείται ένας τύπος, όπως οι εκφράσεις δημιουργίας στιγμιότυπων κλάσεων (`new`), οι μετατροπές ή οι όροι `implements` και `throws`. Αυτή η μορφή των annotations ονομάζεται σχολιασμός τύπων (type annotation).

4.2.3) Σχολιασμοί τύπων (type annotations)

Τα type annotations δημιουργήθηκαν για να υποστηρίξουν βελτιωμένη ανάλυση των προγραμμάτων Java, ώστε ο έλεγχος τύπων να γίνει δυνατότερος. Με τη

μορφή των type annotations μπορούν να εκφραστούν πολλές ιδιότητες ασφάλειας που θέλουμε να ελέγξουμε. Η έκδοση SE 8 της Java δεν παρέχει ένα πλαίσιο ελέγχου τύπων, αλλά επιτρέπει τη χρήση πλαισίου ελέγχου τύπων που υλοποιείται ως αποσπώμενη μονάδα (pluggable module) και χρησιμοποιούνται σε συνδυασμό με τον μεταγλωττιστή Java.

Μπορούν να χρησιμοποιηθούν πολλαπλές μονάδες για έλεγχο τύπων, όπου κάθε μονάδα ελέγχει για ένα διαφορετικό τύπο σφάλματος. Τα pluggable συστήματα τύπων επιτρέπουν πολλαπλά συστήματα τύπων να χρησιμοποιούνται ταυτόχρονα. Με αυτόν τον τρόπο, «χτίζουμε» πάνω από το σύστημα τύπων της Java, προσθέτοντας ειδικούς ελέγχους όταν και όπου χρειάζεται. Οι ελεγκτές σχεδιάζονται ως «προσθήκες» (plug-ins) στον μεταγλωττιστή. Με ορθολογική χρήση των type annotations μαζί με τους pluggable ελεγκτές τύπων, μπορεί να γραφτεί κώδικας που είναι πιο δυνατός και λιγότερο επιρρεπής σε λάθη.

Για παράδειγμα, έστω ότι θέλουμε να εγγυηθούμε για ένα πρόγραμμα ότι μία συγκεκριμένη μεταβλητή δεν θα είναι ποτέ `null`, ώστε να μην έχουμε `NullPointerException`. Η δήλωση της μεταβλητής μπορεί να είναι ως εξής:

```
@NonNull String str;
```

Κατά τη μεταγλώττιση του κώδικα, συμπεριλαμβάνοντας τη NonNull μονάδα, ο μεταγλωττιστής εκτυπώνει μία προειδοποίηση αν εντοπίσει πιθανό πρόβλημα, ώστε να μπορεί να τροποποιηθεί ο κώδικας για την αποφυγή του σφάλματος. Με τη διόρθωση του κώδικα για την απομάκρυνση όλων των προειδοποιήσεων, το συγκεκριμένο σφάλμα δεν θα συμβεί κατά την εκτέλεση του προγράμματος.

Δεν είναι απαραίτητο ο κάθε προγραμματιστής να γράφει τις δικές του μονάδες ελέγχου τύπων καθώς έχουν αναπτυχτεί εργαλεία που περιλαμβάνουν τέτοιες μονάδες. Στην παρούσα εργασία χρησιμοποιείται ο Checker Framework, που δημιουργήθηκε από το Πανεπιστήμιο της Ουάσιγκτον.

4.2.4) Pluggable ελεγκτές τύπων (type chechers) - Μεθοδολογία δημιουργίας και χρήσης

Έστω ότι ο προγραμματιστής παρατηρεί ένα πιθανό πρόβλημα, ή ένα μοτίβο πιθανών προβλημάτων και θέλει να τα εξαλείψει. Για τη βελτίωση του κώδικα μέσω του pluggable ελέγχου τύπων, γενικά ακολουθείται η παρακάτω μεθοδολογία:

1. Ο προγραμματιστής επιλέγει έναν υπάρχοντα ελεγκτή που μπορεί να επαληθεύσει την απουσία του σφάλματος. Αν δεν υπάρχει τέτοιος ελεγκτής, τότε δημιουργεί έναν καινούριο χρησιμοποιώντας τους ορισμούς των annotations που υπάρχουν ήδη.
2. Επιλέγει μέρος του προγράμματος, ή ολόκληρο το πρόγραμμα για έλεγχο. Ο Checker Framework είναι αποτελεσματικός ακόμα και όταν χρησιμοποιείται σε μέρος του προγράμματος. Μπορεί να επιλεγεί το μέρος του προγράμματος

που είναι πιο επιρρεπές σε σφάλματα, το πιο μπερδεμένο, το πιο κρίσιμο, ή αυτό που ταιριάζει καλύτερα στον ελεγκτή.

3. Γράφει και προσθέτει στον κώδικα κατάλληλα annotations, που υποδεικνύουν την κανονική συμπεριφορά του προγράμματος (όχι κάθε τιμή που μπορεί να πάρει, αν υπάρχει σφάλμα, αλλά την επιθυμητή).
4. Τρέχει επαναληπτικά τον κώδικα με τα annotations στον ελεγκτή, μέχρι να μην εμφανίζονται προειδοποιήσεις, οπότε και έχει επαληθευτεί η απουσία λαθών. Για κάθε προειδοποίηση υπάρχουν τρεις πιθανότητες:
   i. Εντοπίζεται και διορθώνεται το σφάλμα.
   ii. Η προειδοποίηση οφείλεται σε ένα annotation που λείπει, οπότε και προστίθεται.
   iii. Η προειδοποίηση είναι ψευδώς θετική (false positive). Αγνοείται, προσθέτοντας ένα annotation `@SuppressWarnings`.
5. Αν έχει παρατηρηθεί μοτίβο ψευδώς θετικών προειδοποιήσεων, τότε επεκτείνεται ο ελεγκτής, ώστε να χειρίζεται αυτόματα αυτές τις περιπτώσεις.

4.2.5) Πλεονεκτήματα και μειονεκτήματα

Η επαλήθευση μέσω ελέγχου τύπων είναι ισχυρή, καθώς παρέχει εγγυήσεις. Ο ελεγκτής μπορεί να εγγυηθεί ότι μία συγκεκριμένη ιδιότητα ισχύει στον κώδικα. Για παράδειγμα, ο Nullness Checker εγγυάται ότι κάθε έκφραση του τύπου `@NonNull` δεν παίρνει ποτέ την τιμή `null`. Ο ελεγκτής παρέχει την εγγύηση εξετάζοντας τα μέρη του προγράμματος και επαληθεύοντας ότι κανένα μέρος του προγράμματος δεν παραβιάζει την ιδιότητα. Η μέθοδος αυτή είναι τυπικά ισοδύναμη με τις υπόλοιπες τεχνολογίες επαλήθευσης, όπως τον έλεγχο μοντέλων. Οι προγραμματιστές είναι ήδη εξοικειωμένοι με τον έλεγχο τύπων, έτσι ταιριάζει στη διαδικασία ανάπτυξης. Ακόμα, ο έλεγχος τύπων μπορεί να πραγματοποιηθεί ανεξάρτητα σε κάθε στοιχείο του συστήματος (modular) και είναι επεκτάσιμος, καθώς μπορούν να δημιουργηθούν νέοι προσαρμοσμένοι ελεγκτές τύπων που επαληθεύουν μία ιδιότητα συγκεκριμένου πεδίου (pluggable).

Αυτή η μέθοδος, όμως, είναι λιγότερο εκφραστική από άλλες μεθόδους επαλήθευσης. Ένα σύστημα τύπων παρέχει μερική επαλήθευση. Είναι σχεδιασμένο ώστε να εντοπίζει συγκεκριμένη κατηγορία σφαλμάτων και, έτσι, μπορεί να παραμένουν, στο πρόγραμμα, σφάλματα άλλης κατηγορίας. Αυτό είναι λογικό, καθώς μία πλήρης τυπική επαλήθευση είναι πρακτικά αδύνατη για πραγματικά συστήματα λογισμικού. Φυσικά, η εγγύηση παρέχεται μόνο για το μέρος του προγράμματος που έχει ελεγχθεί με αυτόν τον τρόπο. Τέλος, και με αυτή τη μέθοδο, μπορεί να έχουμε αρκετά false positives, που όπως είδαμε, χρειάζονται πιο ειδική μεταχείριση.

Το κυριότερο ζήτημα που υπάρχει σε αυτή την τεχνική είναι το ότι δεν είναι πλήρως αυτοματοποιημένη, όπως τονίσαμε από την αρχή. Απαιτείται, δηλαδή, παραπάνω χρόνος από τον προγραμματιστή για την προσθήκη των κατάλληλων

annotations, ή εξειδικευμένα άτομα για αυτό το σκοπό. Στη συνέχεια θα δούμε κατά πόσο μπορούμε να εκμεταλλευτούμε τεχνικές crowdsourcing για τη βελτίωση αυτού του ζητήματος. Ο στόχος είναι η επαλήθευση να κοστίζει λιγότερο και να μπορεί να γίνει σε μικρότερο διάστημα, μειώνοντας την απαραίτητη ικανότητα για επαλήθευση προγραμμάτων και έτσι, αυξάνοντας τον αριθμό των ατόμων που είναι ικανά για την επαλήθευση.

4.3) Πληθοπορισμός (Crowdsourcing)

Οι Estellés-Arolas και González Ladrón-de- Guevara, μετά τη μελέτη πάνω από 40 ορισμών για την έννοια του πληθοπορισμού (crowdsourcing), κατέληξαν στον εξής ορισμό[2]:

«Το crowd sourcing είναι μία μορφή συλλογικής διαδικτυακής δραστηριότητας στην οποία ένα άτομο, ένα ίδρυμα, ένας μη κερδοσκοπικός οργανισμός ή μία εταιρεία προτείνει σε μία ομάδα ατόμων με ποίκιλλες γνώσεις, ετερογένεια και αριθμό, μέσω μίας ανοιχτής πρόσκλησης, να αναλάβουν εθελοντικά μια εργασία. Η ανάληψη της εργασίας, η οποία ποικίλλει σε πολυπλοκότητα και στο βαθμό στον οποίο είναι χωρισμένη και στην οποία το πλήθος πρέπει να συμμετάσχει με προσωπική εργασία, χρήματα, γνώση, εμπειρία, περιλαμβάνει πάντοτε αμοιβαίο όφελος και για τις δύο πλευρές. Οι χρήστες λαμβάνουν την ικανοποίηση κάποιας ανάγκης τους, είτε αυτή είναι οικονομική, είτε κοινωνική αναγνώριση, προσωπική ικανοποίηση, ανάπτυξη ατομικών ικανοτήτων σε κάποιο τομέα, ενώ ο εκκινητής της πρωτοβουλίας (crowdsourcer) αποκτά και χρησιμοποιεί προς όφελός του, αυτά που έχει συνεισφέρει ο χρήστης στο εγχείρημα, τα οποία εξαρτώνται από τη δραστηριότητα που έχει αναλάβει ο χρήστης.»

Το crowdsourcing έχει εφαρμοστεί επιτυχώς σε διάφορα περιεχόμενα, από απλές λειτουργίες στο Amazon, μέχρι την επίλυση περίπλοκων βιομηχανικών προβλημάτων, όπως το InnoCentive. Οι επιχειρήσεις χρησιμοποιούν όλο και περισσότερο αυτήν την προσέγγιση για την πραγματοποίηση συγκεκριμένων λειτουργιών ανάπτυξης λογισμικού. Έτσι, μεγάλα και δύσκολα καθήκοντα, μπορούν να εκτελούνται πολύ γρήγορα από μεγάλα πλήθη που δεν χρειάζονται εκπαίδευση. Το 2011, οι παίχτες του παιχνιδιού Foldit παρήγαγαν ένα ακριβές τρισδιάστατο μοντέλο ενός ενζύμου σε 10 ημέρες μόλις, ένα πρόβλημα που απασχολούσε τους ερευνητές για 15 χρόνια.

4.3.1) Χρήση crowdsourcing στην τυπική επαλήθευση (Crowd-Sourced Formal Verification CSFV- Darpa project)

Εκτός από τα πεδία όπου οι απαιτήσεις για ασφάλεια και προστασία κάνουν το κόστος και τη μακριά, σε χρόνο, ανάπτυξη ανεκτά, στα περισσότερα πεδία ο χρόνος και το κόστος της τυπικής επαλήθευσης είναι το μεγαλύτερο πρόβλημα. Εξετάζοντας καλύτερα την κατάσταση, το κλειδί του προβλήματος είναι το ότι το

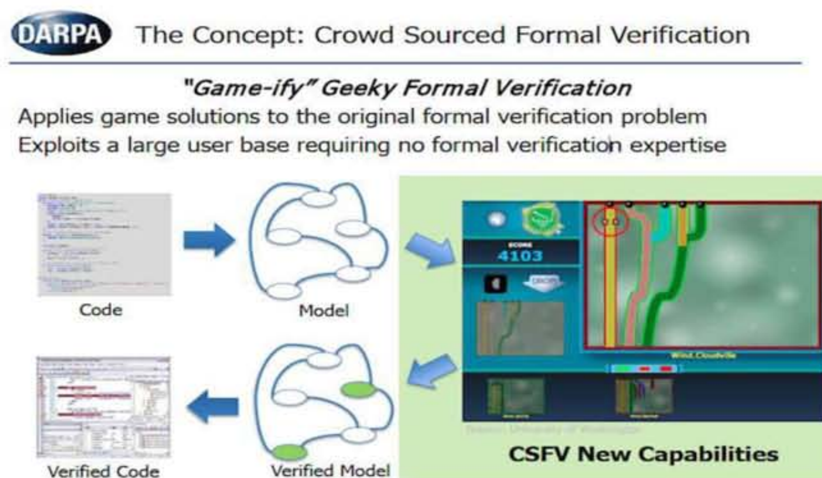
[2] Πηγή: https://el.wikipedia.org/wiki/Crowdsourcing

μέγεθος του δυναμικού που μπορεί να αξιοποιηθεί είναι περιορισμένο, καθώς απαιτούνται συνήθως εξειδικευμένες γνώσεις για τον χειρισμό των εργαλείων τυπικής επαλήθευσης. Το εμπόδιο αυτό μπορεί να ξεπεραστεί, αν επεκταθεί το μέγεθος του δυναμικού που μπορεί να κάνει αποτελεσματικά τυπική επαλήθευση.

Προς τον στόχο της επαλήθευσης με τη βοήθεια του κοινού οδήγησαν δύο συγκεκριμένες περιπτώσεις:

- Η διευθύντρια, από το 2009 μέχρι το 2012, της DARPA (Defense Advanced Research Projects Agency) Dr. Regina Dugan, εξέφρασε ενδιαφέρον στην εφαρμογή του crowdsourcing στην ασφάλεια υπολογιστών.
- Ένα σύνολο συζητήσεων μεταξύ του Michael Ernst και της Jeannette Wing που ξεκίνησαν το 2010, οδήγησαν στην ιδέα της εφαρμογής της «παιχνιδοποίησης» (gamification) στο πεδίο της τυπικής επαλήθευσης. Αν τα προβλήματα τυπικής επαλήθευσης μπορούν να μεταφραστούν σε διασκεδαστικά παιχνίδια, τότε θα μπορούν να απευθυνθούν για επίλυση σε μεγάλο μέγεθος κοινού.

Σε αυτήν την κατεύθυνση, η DARPA ανέπτυξε ένα πρόγραμμα, το Crowd Sourced Formal Verification (CSFV). Το συγκεκριμένο πρόγραμμα στοχεύει στο να εξερευνήσει αν μεγάλος αριθμός μη εξειδικευμένων ανθρώπων, μπορεί να πραγματοποιήσει τυπική επαλήθευση πιο γρήγορα και πιο αποτελεσματικά από την άποψη του κόστους, από ότι με τις συμβατικές μεθόδους. Ο στόχος είναι η να γίνει η επαλήθευση πιο προσιτή, μέσω της δημιουργίας διασκεδαστικών παιχνιδιών που αντανακλούν προβλήματα τυπικής επαλήθευσης. Παίζοντας τα παιχνίδια, ο χρήστης μπορεί να βοηθήσει αποτελεσματικά τα εργαλεία επαλήθευσης λογισμικού να ολοκληρώσουν τη διαδικασία, παράγοντας τις αντίστοιχες αποδείξεις τυπικής επαλήθευσης.



Εικόνα 4.1: CSFV

Πηγή εικόνας: http://www.dtic.mil/dtic/tr/fulltext/u2/a551949.pdf

Τα οφέλη στα οποία προσβλέπει το πρόγραμμα CSFV είναι τα παρακάτω:

- Αυξημένη συχνότητα και σχέση κόστους-αποτελεσματικότητας της τυπικής επαλήθευσης, για περισσότερους τύπους κοινού λογισμικού, έτοιμου να διατεθεί εμπορικά (Commercial Off-the-Shelf - COTS).
- Επέκταση σε μεγάλο βαθμό, του κοινού στη συμμετοχή στην τυπική επαλήθευση.
- Δημιουργία μόνιμης κοινότητας παικτών που ενδιαφέρονται για τη βελτίωση της ασφάλειας λογισμικού.

Εικόνα 4.2: Verigames

Στα πλαίσια του προγράμματος CSFV, τον Δεκέμβριο του 2013, η DARPA ξεκίνησε το διαδικτυακό portal www.verigames.com που πρόσφερε πέντε δωρεάν online παιχνίδια. Αυτά τα παιχνίδια μεταφράζουν τις ενέργειες των παικτών σε annotations στα προγράμματα, που βοηθούν την τυπική επαλήθευση. Το παίξιμο των παιχνιδιών παράγει μαθηματικές αποδείξεις που μπορούν να επαληθεύσουν την απουσία συγκεκριμένων αδυναμιών ή σφαλμάτων λογισμικού σε λογισμικό ανοιχτού κώδικα που είναι γραμμένο σε C ή Java. Μία αρχική ανάλυση δείχνει ότι μη ειδικοί, παίζοντας τα CSFV παιχνίδια, δημιούργησαν εκατοντάδες χιλιάδες annotations.

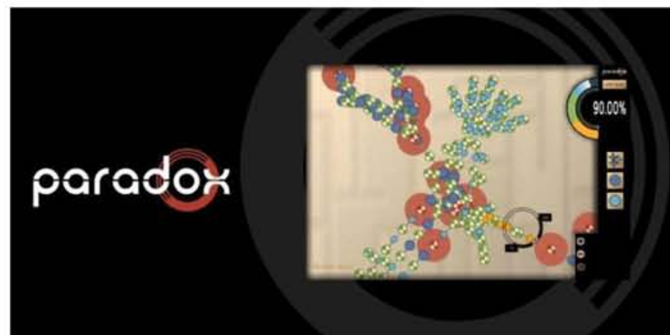
Τα αποτελέσματα ενέπνευσαν την DARPA to 2015, να δρομολογήσει ένα νέο γύρο πέντε νέων παιχνιδιών, με στόχο την επέκταση των επιτυχιών μέχρι σήμερα και την εκμάθηση περισσότερων σε σχέση με την προσέγγιση.

Κάποια από τα παιχνίδια που κάνουν αυτό το σκοπό είναι τα εξής:

— **Flow Jam**: Ανάλυση και ρύθμιση ενός δικτύου καλωδίων, για τη μεγιστοποίηση της ροής του.

Εικόνα 4.2: Στιγμιότυπο από Flow Jam Game

— **Paradox**: Ζητάει από τους παίκτες να βελτιστοποιήσουν τεράστια δίκτυα, χρησιμοποιώντας μια σειρά εργαλείων.



Εικόνα 4.3: Paradox Game

— **Ghost Map Hyperspace**: Ζητάει από τους παίκτες να πολεμήσουν εξωγήινους εισβολείς και να αναχαιτίσουν τις υπερδιαστημικές ρίψεις τους.

Εικόνα 4.4: Ghost Map Hyperspace Game

— **Binary Fission**: Ένα παιχνίδι διάσπασης ατόμων που ζητάει από τους παίκτες να αναμείξουν και να ταιριάξουν κουάρκς, στο όνομα της ασφάλειας του κυβερνοχώρου.

Εικόνα 4.5: Binary Fission Game

Πηγές εικόνων παιχνιδιών: http://games.cs.washington.edu/verigame/flowjam/, http://www.verigames.com

Στο πλαίσια του CSFV, χρηματοδοτούμενοι από την DARPA, η ομάδα προγραμματιστικών γλωσσών και Software Engineering καθώς και το κέντρο επιστήμης παιχνιδιού του πανεπιστημίου της Ουάσιγκτον, ανέπτυξαν το 2012 το παιχνίδι Pipe Jam, που παρουσιάζουν στο "Verification Games: Making Verification Fun"[3]. Σε αυτό βασίζεται όλη η συνέχεια. Αυτό το παιχνίδι αποτελεί τη βάση πάνω στην οποία δημιουργήθηκαν στην πορεία τα παιχνίδια Flow Jam και Paradox που αναφέρονται παραπάνω.

4.3.2) Pipe Jam Game

Το σύστημα δέχεται ως είσοδο ένα πρόγραμμα σε Java και μία ιδιότητα που θέλουμε να επαληθευτεί, εκφρασμένη ως σύστημα τύπων και παράγει ως έξοδο μία απόδειξη ορθότητας ότι το πρόγραμμα ικανοποιεί την ιδιότητα ή μία συγκεκριμένη θέση στον κώδικα όπου το πρόγραμμα παραβιάζει την ιδιότητα, δηλαδή πού μπορεί να μην είναι ασφαλές το πρόγραμμα.

Εικόνα 4.6: Στιγμιότυπο του Pipe Jam Game

Το σύστημα μετατρέπει αυτόματα το πρόγραμμα και την ιδιότητα σε ένα παιχνίδι που μπορεί να παιχτεί από ανθρώπους χωρίς γνώσεις υπολογιστών. Όταν ο παίκτης ολοκληρώνει το παιχνίδι, τότε η τελική διαμόρφωση των στοιχείων του board μπορεί να μεταφραστεί σε απόδειξη ορθότητας του αυθεντικού προγράμματος.

Σε γενικές γραμμές, το παιχνίδι είναι ως εξής:

Ο παίκτης έχει να αντιμετωπίσει ένα σύνολο από πάζλ σχετικά με μπάλες και σωλήνες.

---

[3] W. Dietl, S. Dietzel, M. D. Ernst, N. Mote, B. Walker, S. Cooper, T. Pavlik, and Z. Popović, "Verification games: Making verification fun", in Workshop on Formal Techniques for Java-like Programs (FTfJP), ACM, pp. 42–49, 2012

- Κάθε σωλήνας είναι είτε στενός είτε πλατύς. Ο παίκτης μπορεί να ελέγξει το πλάτος των περισσότερων σωλήνων, ενώ κάποιους άλλους δεν μπορεί να τους τροποποιήσει.
- Κάθε μπάλα είναι είτε μικρή είτε μεγάλη. Μία μικρή μπάλα μπορεί να κυλήσει σε κάθε σωλήνα, ενώ μία μεγάλη μπάλα μπορεί να κυλήσει μόνο σε πλατιούς σωλήνες. Κάποιες μπάλες είναι μόνο μεγάλες.
- Ο στόχος του παίκτη είναι να φροντίσει ώστε οι μπάλες να μην κολλήσουν ποτέ (μεγάλη μπάλα σε στενό σωλήνα).
- Το πάζλ έχει λυθεί όταν ο παίκτης έχει επιλέξει πλάτη σωλήνων που να είναι συμβατά με όλους τους περιορισμούς. Οι περιορισμοί συμπεριλαμβάνουν τα καθορισμένα πλάτη σωλήνων και τις ροές μεταξύ τους.

Ως μέθοδος επαλήθευσης χρησιμοποιείται ο έλεγχος τύπων που περιγράψαμε παραπάνω. Η τελική διαμόρφωση των στοιχείων του board αντιστοιχεί σε ένα σύνολο από annotations τύπων που μπορεί να ελεγχθεί από έναν ελεγκτή τύπων. Ο ανθρώπινος παίκτης δηλαδή κάνει την επαγωγή των τύπων. Ως ελεγκτής τύπων χρησιμοποιείται ο Checker Framework.

Εικόνα 4.7: Στιγμιότυπο του Pipe Jam Game

Το σύστημα αντιστοιχεί τις ιδιότητες ροής τύπων του κώδικα σε ένα δίκτυο σωλήνων. Το πλάτος των σωλήνων, που ελέγχεται από τους παίκτες, απευθείας αντιστοιχεί σε annotations τύπων που μπορούν να ελεγχθούν και έτσι να παραχθεί απόδειξη μερικής ορθότητας.

- Κάθε μπάλα αναπαριστά μία τιμή.
- Κάθε σωλήνας αναπαριστά μία μεταβλητή.
- Η ροή από τον ένα σωλήνα στον άλλον αναπαριστά την ανάθεση μεταξύ των μεταβλητών.

- Το πλάτος των σωλήνων αναπαριστά τους τύπους των μεταβλητών. Γενικά, για ένα σύστημα τύπων με δύο στοιχεία, ο στενός σωλήνας αναπαριστά τον υπο-τύπο, ενώ ο πλατύς τον υπερ-τύπο.
- Οι σωλήνες που δεν μπορούν να τροποποιηθούν αναπαριστούν περιορισμούς του πηγαίου κώδικα του προγράμματος. Για παράδειγμα, σε ένα nullness σύστημα τύπων, η αναφορά x.f, απαιτεί ότι η μεταβλητή x πρέπει να είναι non-null, δηλαδή ο αντίστοιχος σωλήνας να είναι στενός.

Οι περιορισμοί και οι σχέσεις ανάμεσα στα στοιχεία του παιχνιδιού, είναι αναπαραστάσεις των περιορισμών των τύπων του προγράμματος και των σχέσεων ανάμεσα σε στις μεταβλητές του. Παίζοντας το παιχνίδι, επιλέγεται ένας τύπος για κάθε μεταβλητή του προγράμματος. Δηλαδή, μέσω του crowdsourcing μπορεί να επιλυθεί το σημαντικότερο πρόβλημα της τυπικής επαλήθευσης με τη χρήση τύπων, αυτό της προσθήκης των annotations.

Όταν υπάρχει σφάλμα, τότε ο προγραμματιστής εξετάζει μόνο εκείνες τις περιπτώσεις όπου το πλήθος των παικτών δεν μπορούν να λύσουν.

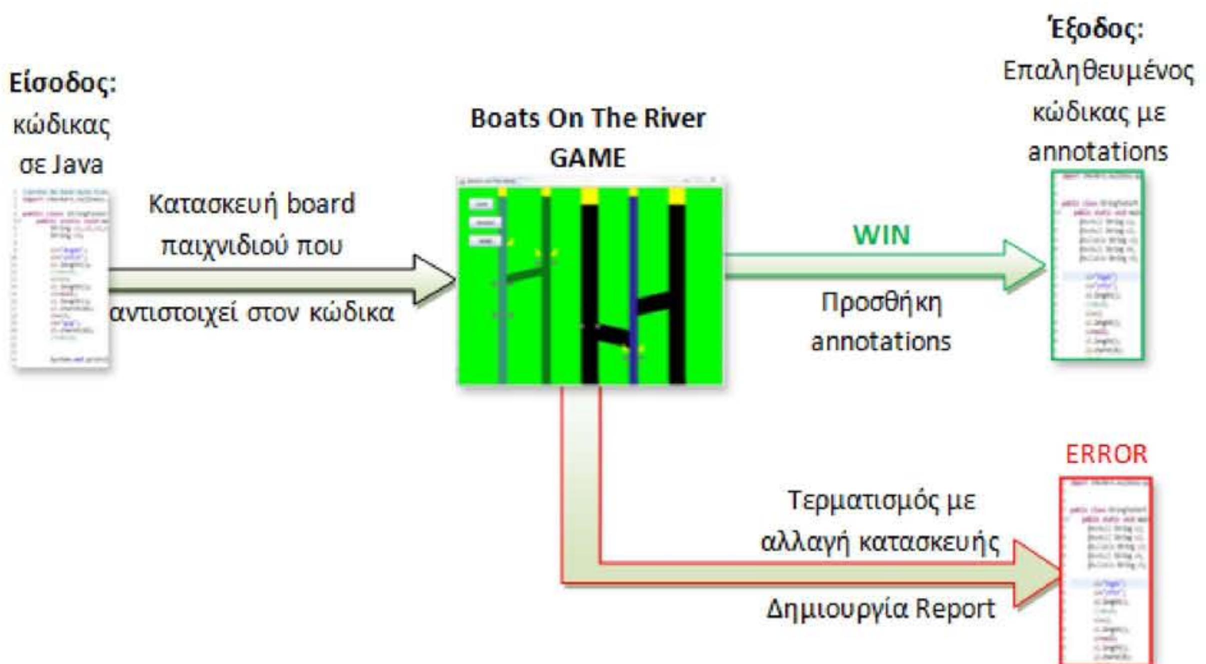
Οι εικόνες προέρχονται από το paper και από το παίξιμο του παιχνιδιού στο:
http://games.cs.washington.edu/verigames-dev/demo/pipes/webgame.php?sample=sample.zip

5 BOATS ON THE RIVER-ΠΕΡΙΓΡΑΦΗ/ΣΧΕΔΙΑΣΜΟΣ/ΠΡΟΣΟΜΟΙΩΣΗ

5.1) Εισαγωγή

Στο συγκεκριμένο κεφάλαιο της παρούσας εργασίας, γίνεται παρουσίαση της εφαρμογής "Boats on the river" που είναι βασισμένη στο Pipe Jam game.

Παίζοντας οι παίκτες το παιχνίδι, χωρίς να χρειάζεται να έχουν ειδικές γνώσεις σε προγραμματισμό ή επαλήθευση, γίνεται τυπική επαλήθευση του σχετικού κώδικα που εξετάζεται.



Σχήμα 5.1: Λειτουργία της Boats on the river

Σε γενικές γραμμές, η λειτουργία της περιγράφεται ως εξής:

I. Αντιστοίχηση του κώδικα εισόδου στο παιχνίδι: Η εφαρμογή παίρνει ως είσοδο ένα πρόγραμμα σε Java και κατασκευάζει το board του παιχνιδιού, ανάλογα με αυτό.
II. Boats on the river game: Παίζοντας οι χρήστες το παιχνίδι, συσχετίζουν τις μεταβλητές του προγράμματος με τύπους.

III. Μετατροπή του τελικού board του παιχνιδιού, σε κώδικα με annotations ή αναφορά σφάλματος:
   - Κερδίζοντας το παιχνίδι, παράγεται ως έξοδος, καινούριο αρχείο που περιλαμβάνει τον κώδικα του προγράμματος, με την προσθήκη κατάλληλων annotations, ώστε να μπορέσει μετά να ελεγχθεί από τον ελεγκτή τύπων.
   - Αν, για να τερματίσει το παιχνίδι, χρειάζεται ο παίκτης να αλλάξει την αρχική κατασκευή του board, σημαίνει ότι στον κώδικα υπάρχει σφάλμα, οπότε, εκτός από το αρχείο με τον κώδικα του προγράμματος, με τα annotations, τυπώνεται μήνυμα με τις γραμμές και με τις μεταβλητές όπου παρουσιάζεται το πρόβλημα, ώστε να ελεγχθεί, στη συνέχεια, από εξειδικευμένους ανθρώπους.

5.2) Επιλογές μεθόδων, εργαλείων και αντικειμένων επαλήθευσης

Για την ανάπτυξη της εφαρμογής, χρειάστηκε να γίνουν συγκεκριμένες επιλογές μεθόδων και εργαλείων καθώς και επιλογές σχετικές με τα αντικείμενα επαλήθευσης, που περιγράφονται, συνοπτικά, παρακάτω:

⇒ Η μέθοδος που χρησιμοποιείται για την τυπική επαλήθευση, όπως και στο Pipe Jam, είναι ο έλεγχος τύπων.
⇒ Η ιδιότητα ασφάλειας η οποία εξετάζεται είναι, συγκεκριμένα, η αποτροπή null-pointer error.
⇒ Η γλώσσα προγραμματισμού στην οποία είναι γραμμένα τα προγράμματα που εξετάζουμε είναι η Java.
⇒ Ως ελεγκτής τύπων χρησιμοποιείται ο Nullness Checker από τον Checker Framework.

5.2.1) Έλεγχος τύπων

Η μέθοδος για την τυπική επαλήθευση είναι ο έλεγχος τύπων (που περιγράφηκε στο τελευταίο κεφάλαιο του προηγούμενου μέρους). Αυτό που θα αναπαρασταθεί με το παιχνίδι, όπως και στο Pipe Jam game, είναι η ροή των τύπων, όχι η ροή των δεδομένων. Έτσι, ενώ τα προγράμματα πολύ συχνά παρουσιάζουν βρόχους στη ροή των δεδομένων τους, στη ροή των τύπων αυτό δεν ισχύει, καθώς εξετάζονται οι σχέσεις και οι ροές μεταξύ τύπων μεταβλητών και όχι συγκεκριμένων τιμών δεδομένων. Η ροή των τύπων έχει να κάνει με τους περιορισμούς των τύπων που προκύπτουν από τη σύνταξη του προγράμματος.

5.2.2) Null-pointer error (NPE)

Στην Java, όταν η ειδική τιμή `null` ανατίθεται στην αναφορά (reference) ενός αντικειμένου, δηλώνει ότι το αντικείμενο «δείχνει» σε άγνωστο κομμάτι δεδομένων. Όταν μία εφαρμογή προσπαθεί να χρησιμοποιήσει ή να προσπελάσει ένα αντικείμενο του οποίου η αναφορά είναι `null` έχουμε την εξαίρεση

NullPointerException. Αυτή την εξαίρεση την συναντάμε στις ακόλουθες περιπτώσεις:

- Κλήση μεθόδου από ένα null αντικείμενο.
- Πρόσβαση ή τροποποίηση πεδίου ενός null αντικειμένου.
- Η προσπάθεια να πάρουμε το μήκος (length) του null, όταν ο τύπος της αναφοράς είναι πίνακας.
- Η πρόσβαση ή τροποποίηση των slots ενός null αντικειμένου, όταν ο τύπος της αναφοράς είναι πίνακας.
- Η προσπάθεια να «πετάξουμε» μία αναφορά null, όταν ο τύπος της αναφοράς είναι υποτύπος της Throwable.
- Η προσπάθεια για συγχρονισμό σε ένα null αντικείμενο.

Η NullPointerException είναι μία εξαίρεση που παρουσιάζεται κατά την εκτέλεση (RuntimeException).

5.2.3) Σχολιασμοί (Annotations)

Τα annotations που χρησιμοποιούνται είναι τα @Nullable και @NonNull.

Το @Nullable είναι ένα annotation τύπου που υποδεικνύει ότι η τιμή της μεταβλητής δεν είναι γνωστό ότι δεν είναι null. Μόνο σε μία τέτοια μεταβλητή μπορεί να ανατεθεί η τιμή null.

Το @NonNull είναι ένα annotation τύπου που υποδεικνύει ότι η έκφραση δεν είναι ποτέ null. Συνήθως δεν χρειάζεται να γράφεται το συγκεκριμένο annotation, καθώς είναι το default. Παρόλα αυτά, στην εφαρμογή γράφεται και αυτό.

5.2.4) Αντικείμενα επαλήθευσης

Μία NullPointerException, όπως είδαμε, μπορεί να παρουσιαστεί από οποιοδήποτε αντικείμενο. Στη συγκεκριμένη εφαρμογή, δεν εξετάζονται όλες οι περιπτώσεις, αλλά ο έλεγχος περιορίζεται στον έλεγχο των string, εξασφαλίζει ότι δεν θα έχουμε null-pointer error από αντικείμενο τύπου string.

Ακόμα, δεν εξετάζονται οι περιπτώσεις κλήσεων μεθόδων, αλλά περιοριζόμαστε σε τοπικά string. Το συγκεκριμένο, ο έλεγχος δηλαδή μόνο τοπικών αντικειμένων, δεν χρειάζεται σχεδόν ποτέ, όπως εξηγεί και το manual του CheckerFramework, αν δεν επεκταθεί σε ολόκληρα java projects. Επιδίωξη, όμως, είναι, μέσω της εφαρμογής, να φανεί η λογική με την οποία μπορεί να γίνει η μετατροπή της τυπικής επαλήθευσης σε παιχνίδι.

Επομένως, αυτό που εξετάζουμε μόνο στο συγκεκριμένο παράδειγμα, είναι την εμφάνιση NullPointerException από προσπάθεια να προσπελαστούν πεδία ενός null τοπικού string.

Η εφαρμογή εξετάζει προγράμματα υποθέτοντας ότι έχουν περάσει από compiler, χωρίς διακλαδώσεις.

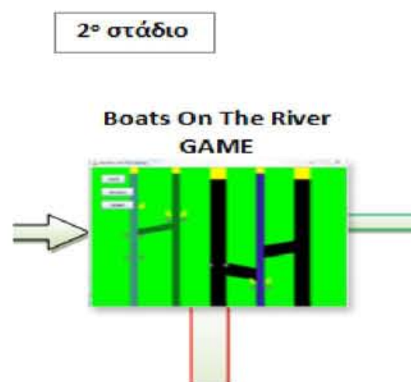
5.3) Περιγραφή/προσομοίωση σχεδίασης

Όπως περιγράφηκε και στην εισαγωγή του κεφαλαίου, η εφαρμογή μπορεί να χωριστεί, σε γενικές γραμμές, σε τρία στάδια: στην αντιστοίχηση του κώδικα εισόδου στο παιχνίδι, στο Boats on the river game και στην μετατροπή του τελικού board του παιχνιδιού, σε κώδικα με annotations ή αναφορά σφάλματος.

Σχήμα 5.2: Λειτουργία Boats on the river με στάδια

Η περιγραφή θα ξεκινήσει με το παιχνίδι (δεύτερο στάδιο), ώστε στη συνέχεια να μπορεί να κατανοηθεί καλύτερα και η αντιστοίχηση του κώδικα σε αυτό καθώς και η παραγωγή του τελικού αρχείου ή και του report.

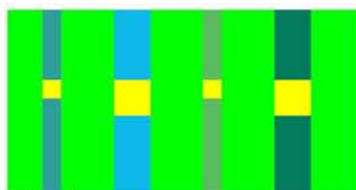
5.3.1) Περιγραφή παιχνιδιού (Boats on the river game)



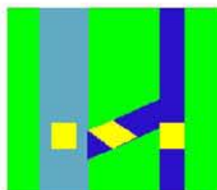
Σχήμα 5.3: 2° στάδιο: Boats on the river game

Το παιχνίδι είναι πολύ απλό. Στο board του παιχνιδιού, υπάρχουν ρυάκια (brooks) στα οποία ταξιδεύουν βάρκες (boats). Σκοπός είναι να μπορέσουν όλες οι βάρκες να φτάσουν στον προορισμό τους (κάτω μέρος του board), ταξιδεύοντας στα ρυάκια, χωρίς να χρειαστεί ο παίκτης να αλλάξει την κατασκευή τους.

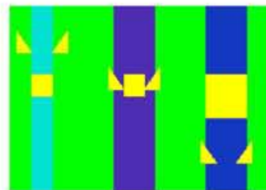
- Αρχικά, σε κάθε ρυάκι αντιστοιχεί και μία βάρκα.
- Τα ρυάκια μπορεί να διασταυρώνονται μεταξύ τους. Σε αυτήν την περίπτωση, το αρχικό ρυάκι διακλαδώνεται και αντιστοιχεί μία βάρκα σε κάθε νέο ρυάκι που δημιουργείται.
- Κάθε ρυάκι μπορεί να είναι είτε μικρό, είτε μεγάλο. Το μέγεθος του αρχικού ρυακιού θα είναι ίδιο με το μέγεθος από όλα τα υπόλοιπα που δημιουργούνται από αυτό.
- Το ίδιο ισχύει και με το χρώμα τους. Όλα τα ρυάκια που δημιουργούνται από ένα συγκεκριμένο, έχουν και το χρώμα του.
- Αντίστοιχα, και η κάθε βάρκα μπορεί να είναι είτε μικρή, είτε μεγάλη.
- Όλες οι βάρκες είναι κίτρινου χρώματος.
- Το αρχικό μέγεθος από τις βάρκες καθορίζεται από το ρυάκι από το οποίο ξεκίνησαν το ταξίδι.
- Για να μπορέσει να ταξιδέψει η βάρκα στο κάθε ρυάκι, χρειάζεται να χωράει σε αυτό. Επομένως, απαιτείται είτε να είναι μικρή η βάρκα, είτε μεγάλο το ρυάκι.
- Στα ρυάκια μπορεί να υπάρχουν αντικείμενα που αλλάζουν το μέγεθος της βάρκας που περνάει από αυτά (resizers). Όταν η βάρκα συναντήσει έναν resizer, το μέγεθός της γίνεται μικρό.
- Στα ρυάκια μπορεί να υπάρχουν βράχοι (rocks). Για να περάσει μία βάρκα από ένα βράχο, πρέπει, στο συγκεκριμένο σημείο, να είναι μικρή.



Εικόνα 5.1: 4 ρυάκια με 4 βάρκες (μικρά και μεγάλα)



Εικόνα 5.2: Διασταύρωση και διακλάδωση



Εικόνα 5.3: Ρυάκια με resizers

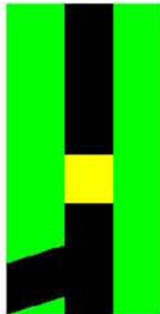
Εικόνα 5.4: Μικρή βάρκα σε μεγάλο ρυάκι, περνάει από βράχο

Για να φτάσουν, δηλαδή, οι βάρκες στον προορισμό τους (σκοπός του παιχνιδιού) χρειάζεται να μην κολλήσουν ποτέ, όταν είναι μεγάλες, είτε σε βράχο, είτε σε μικρότερο ρυάκι. Ο παίκτης μπορεί να αλλάξει μέγεθος στα περισσότερα ρυάκια, κάνοντας κλικ πάνω τους.

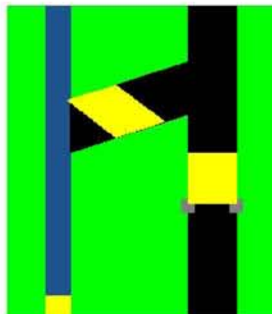
- Υπάρχουν ρυάκια που το μέγεθός τους δεν μπορεί να γίνει μικρό. Αντίστοιχα και το αρχικό μέγεθος των βαρκών που ταξιδεύουν σε αυτό. Αυτά τα ρυάκια έχουν μαύρο χρώμα.
- Σε περίπτωση που μία βάρκα κολλήσει και ο παίκτης δεν μπορεί να την κάνει να φτάσει στον προορισμό της αλλάζοντας το μέγεθος των ρυακιών, τότε μπορεί να σπρώξει τη βάρκα. Σκοπός είναι ο παίκτης να κάνει τα λιγότερα δυνατά σπρωξίματα. Σε μελλοντική δουλειά, για το σκοπό αυτό

θα προστεθεί μείωση στο σκορ που συγκεντρώνει για κάθε σπρώξιμο βάρκας.

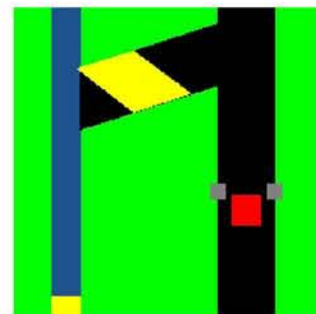
- Σπρώχνοντας τη βάρκα, το μέγεθός της γίνεται μικρό και το χρώμα της κόκκινο. Με αυτόν τον τρόπο, δηλαδή, παρεμβαίνει στην κατασκευή των βαρκών.
- Το παιχνίδι τερματίζει όταν όλες οι βάρκες ταξιδέψουν μέσα στα ρυάκια, δηλαδή όταν φτάσουν στο κάτω μέρος του board, είτε όπως κατασκευάστηκαν, είτε μετά από την παρέμβαση του παίκτη στην κατασκευή τους.



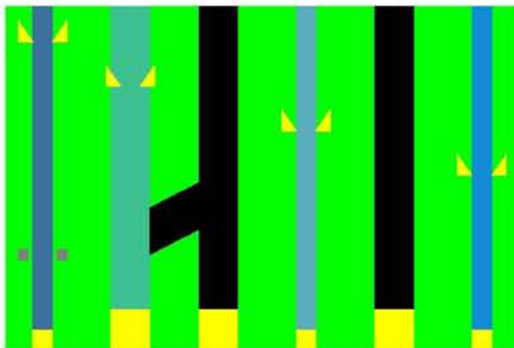
Εικόνα 5.5: Ρυάκι με μέγεθος που δεν τροποποιείται



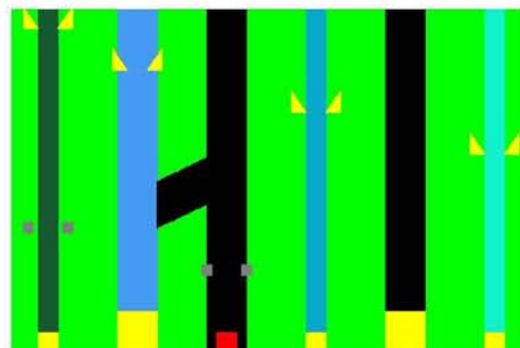
Εικόνα 5.6: Δύο βάρκες κόλλησαν: σε διασταύρωση και σε βράχο



Εικόνα 5.7: Η μία βάρκα σπρώχτηκε



Εικόνα 5.8: Όλες οι βάρκες έφτασαν. Τερματισμός χωρίς παρέμβαση



Εικόνα 5.9: Όλες οι βάρκες έφτασαν. Τερματισμός με παρέμβαση (σπρώξιμο).

5.3.2) Περιγραφή αντιστοίχησης του κώδικα στο παιχνίδι

Σχήμα 5.4: 1ο στάδιο: αντιστοίχηση του κώδικα στο παιχνίδι

Όπως είπαμε, με την εφαρμογή αυτή επιδιώκουμε να ελέγξουμε αν σε ένα πρόγραμμα Java μπορεί να παρουσιαστούν `NullPointerException`, από προσπάθεια να προσπελαστούν πεδία ενός `null` τοπικού string.

Στο πρώτο στάδιο της εφαρμογής, έχουμε ως είσοδο, ένα αρχείο που περιέχει κώδικα σε Java και με βάση αυτόν, κατασκευάζουμε και το board του παιχνιδιού.

Στα προγράμματα που εξετάζονται κάνουμε τις εξής παραδοχές για τα string:

- ✓ Τα String δηλώνονται πάντα με τη μορφή:

```
String myString1, myString2,………;
```

- ✓ Δεν γίνεται ποτέ ανάθεση στην ίδια γραμμή με τη δήλωση.
- ✓ Ο τρόπος για την απευθείας ανάθεση τιμής είναι ο εξής:

```
myString1= "va1ueOfString1";
```

Δηλαδή μόνο μέσω εισαγωγικών (" ").

- ✓ Συσχέτιση μίας μεταβλητής με μία άλλη γίνεται με τον παρακάτω τρόπο:

```
myString1 = myString2;
```

- ✓ Προσπέλαση των πεδίων των string γίνεται όταν αμέσως μετά το όνομα του string ακολουθεί τελεία (.).

```
myString1.1ength();
```

Η αντιστοίχηση του κώδικα με το board του παιχνιδιού γίνεται ως εξής:

- Οι γραμμές του κώδικα αντιστοιχίζονται με το ύψος του board στο παιχνίδι.
- Σε κάθε string του κώδικα αντιστοιχεί και ένα ρυάκι, καθώς και όλες οι διακλαδώσεις που μπορεί να έχει.
- Οι βάρκες που ταξιδεύουν στο ρυάκι, αντιστοιχούν στις τιμές που μπορεί να πάρει το string: null ή not Null.
- Όταν έχουμε συσχέτιση μίας μεταβλητής με μία άλλη (πχ `myStringi=myString2`), τότε έχουμε διασταύρωση μεταξύ του ρυακιού που αναπαριστά το string του οποίου η τιμή ανατίθεται στο άλλο, το οποίο και διακλαδώνεται, και του άλλου. Στο παράδειγμα, το `myString2` διακλαδώνεται και δημιουργείται διασταύρωση μεταξύ των ρυακιών που αναπαριστούν τα δύο string.
- Στα σημεία στο board που αντιστοιχούν σε γραμμές όπου γίνεται απευθείας ανάθεση κάποιας τιμής (not Null) σε string, τοποθετούνται resizers.
- Στα σημεία στο board που αντιστοιχούν σε γραμμές όπου γίνεται προσπέλαση των πεδίων των string, τοποθετούνται βράχοι.
- Τα ρυάκια των οποίων το μέγεθος δεν μπορεί να αλλάξει, αντιστοιχούν σε string τα οποία συσχετίζονται με άλλο string, ή προσπελαύνονται, πριν αρχικοποιηθούν με κάποια τιμή (not Null).
- Αρχικά όλα τα ρυάκια είναι μικρά, εκτός από αυτά των οποίων το μέγεθος δεν αλλάζει, που είναι μεγάλα.

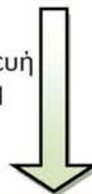
Επομένως, στο πρώτο στάδιο της εφαρμογής, εξετάζεται ο κώδικας εισόδου. Εντοπίζονται τα string που δηλώνονται και ανάλογα με τον τρόπο και τη θέση

που χρησιμοποιούνται, καθώς και με την μεταξύ τους συσχέτιση, κατασκευάζεται το αρχικό board του παιχνιδιού.

Στα παρακάτω παραδείγματα εκτέλεσης, φαίνεται η αντιστοίχιση του board σε δύο διαφορετικούς κώδικες εισόδου.

```java
//ena aplo paradeigma

public class StringTester1 {
    public static void main(String [] args) {
        String s1,s2,s5,s3;
        String s4;

        s1="NonNullvalue1";
        s2="NonNullvalue2";
        s5=null;
        s1=s2;
        s3=null;
        int i=s1.length();
        System.out.println("Strings s1:"+s1+" Length:"+i);
        s4=s3;
        System.out.println("Strings s3:"+s3);
        i=s2.length();
        System.out.println("Strings s2:"+s2+" Length:"+i);
        s2=s5;
        s4="NonNullvalue4";

        System.out.println("Strings s4: "+s4);
        System.out.println("Strings s5: "+s5);

    }
}
```

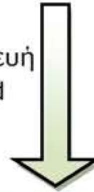
Κατασκευή board

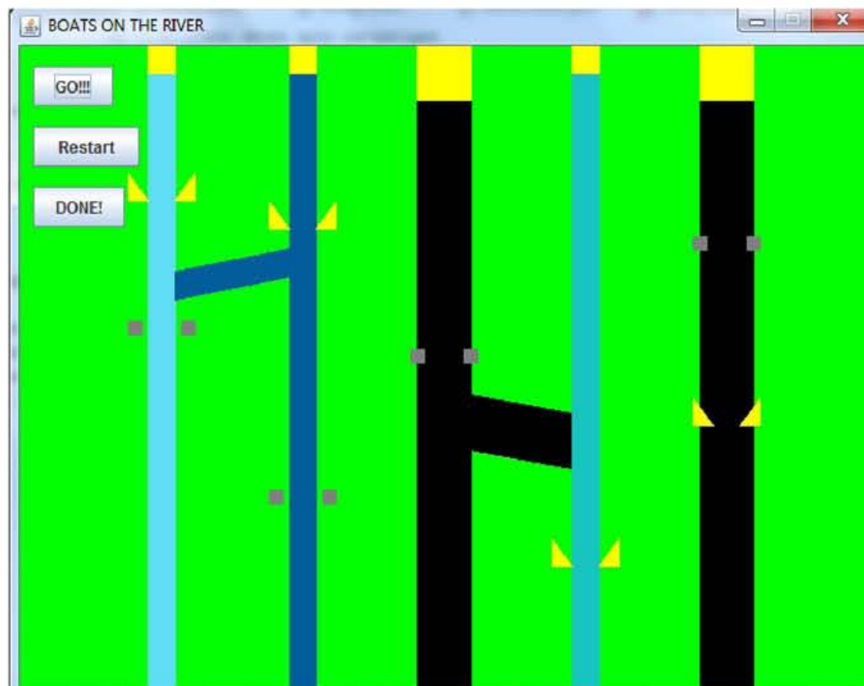


Παράδειγμα 5.1: Κατασκευή board από κώδικα εισόδου. Τα ρυάκια αναπαριστούν με τη σειρά, τα string s1, s2, s5, s3, s4.

```java
//ena akoma aplo paradeigma

public class StringTester2 {
    public static void main(String [] args) {
        String s1,s2,s3,s4;
        String s5;

        s1="NonNullvalue1";
        s2="NonNullvalue2";

        //edo prokyptei sfalma
        char c=s5.charAt(0);
        s1=s2;
        s3=null;
        int i=s1.length();

        //k edo prokyptei sfalma
        int j=s3.indexOf(0);
        System.out.println("Strings s1:"+s1+" Length:"+i);
        s5="NonNullvalue5";
        s4=s3;
        System.out.println("Strings s3:"+s3+" Index Of 0:"+j);
        i=s2.length();
        System.out.println("Strings s2:"+s2+" Length:"+i);
        s4="NonNullvalue4";

        System.out.println("Strings s4: "+s4);
        System.out.println("Strings s5: "+s5+" CharAt 0:"+c);

    }
}
```

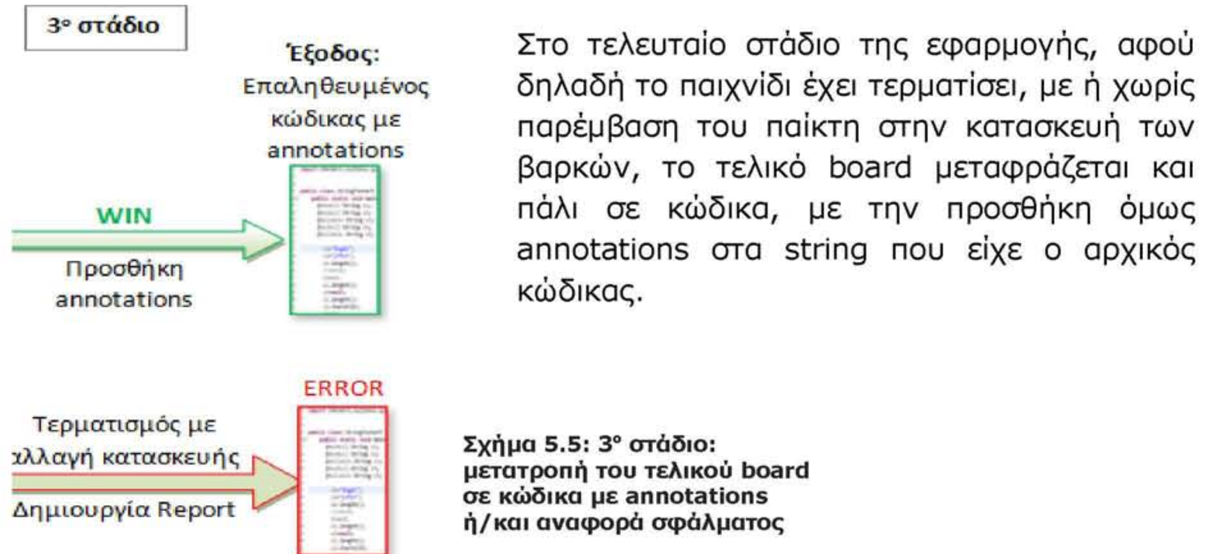
Κατασκευή
board





Παράδειγμα 5.2: Κατασκευή board από κώδικα εισόδου. Τα ρυάκια αναπαριστούν με τη σειρά τα string s1, s2, s3, s4, s5.

5.3.3) Περιγραφή μετατροπής του τελικού board σε κώδικα με annotations ή/και αναφορά σφάλματος

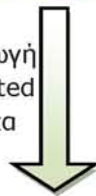


Στο τελευταίο στάδιο της εφαρμογής, αφού δηλαδή το παιχνίδι έχει τερματίσει, με ή χωρίς παρέμβαση του παίκτη στην κατασκευή των βαρκών, το τελικό board μεταφράζεται και πάλι σε κώδικα, με την προσθήκη όμως annotations στα string που είχε ο αρχικός κώδικας.

Σχήμα 5.5: 3ο στάδιο: μετατροπή του τελικού board σε κώδικα με annotations ή/και αναφορά σφάλματος

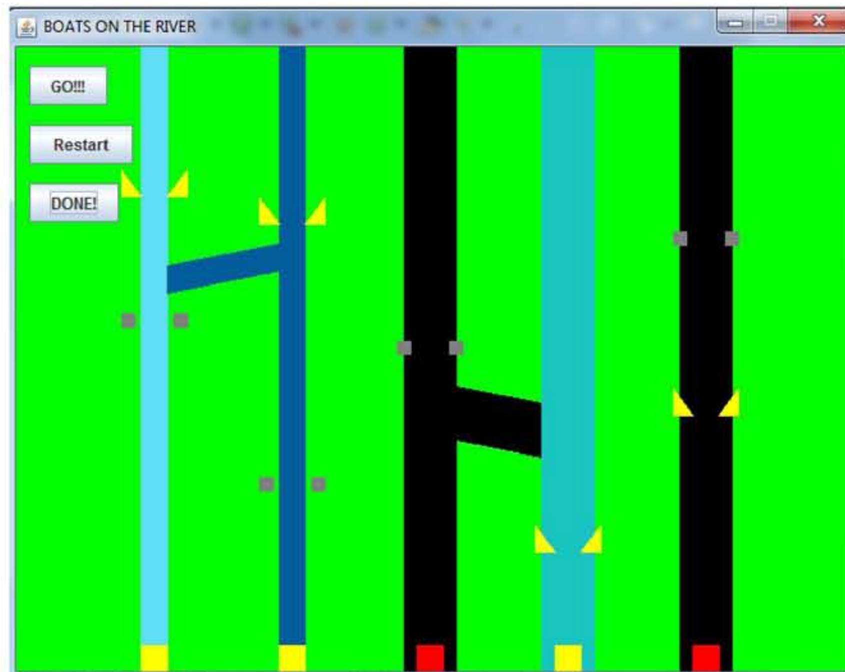
- Εξετάζεται το μέγεθος των ρυακιών που καθόρισε ο χρήστης:
  - Όταν το ρυάκι είναι μεγάλο, τότε σημαίνει ότι μπορεί να πάρει την τιμή `null`, οπότε και χαρακτηρίζεται με το annotation: `@Nullable`.
  - Όταν το ρυάκι είναι μικρό, τότε σημαίνει ότι δεν μπορεί να πάρει την τιμή `null`, οπότε και χαρακτηρίζεται με το annotation: `@NonNull`.
- Δημιουργείται καινούριο αρχείο κώδικα, που είναι ο κώδικας εισόδου μαζί με τα κατάλληλα annotations πριν από τις δηλώσεις των string.
- Όταν στο τελικό board υπάρχουν κόκκινες βάρκες (βάρκες που χρειάστηκε να σπρωχτούν από τον παίκτη), τότε υπάρχει σφάλμα, είτε επειδή έγινε προσπάθεια να προσπελαστεί πεδίο ενός, πιθανά, `null` string (η βάρκα κόλλησε σε βράχο), είτε γιατί ένα string που δεν μπορεί να πάρει την τιμή `null`, παίρνει την τιμή ενός string που μπορεί να την πάρει (διασταύρωση μεγάλου ρυακιού σε μικρό). Σε αυτήν την περίπτωση τυπώνεται μήνυμα με τις γραμμές και με τις μεταβλητές όπου παρουσιάζεται το πρόβλημα, ώστε να ελεγχθεί, στη συνέχεια.

Στη συνέχεια, παρουσιάζονται οι τελικοί κώδικες εξόδου των παραπάνω παραδειγμάτων, μετά την τελική διαμόρφωση των board από τον παίκτη.




```java
import checkers.nullness.quals.*;
//ena aplo paradeigma

public class StringTester1 {
    public static void main(String [] args) {
      @NonNull String s1;
      @Nullable String s2;
      @Nullable String s5;
      @Nullable String s3;
      @Nullable String s4;

        s1="NonNullvalue1";
        s2="NonNullvalue2";
        s5=null;
        s1=s2;
        s3=null;
        int i=s1.length();
        System.out.println("Strings s1:"+s1+" Length:"+i);
        s4=s3;
        System.out.println("Strings s3:"+s3);
        i=s2.length();
        System.out.println("Strings s2:"+s2+" Length:"+i);
        s2=s5;
        s4="NonNullvalue4";

        System.out.println("Strings s4: "+s4);
        System.out.println("Strings s5: "+s5);

    }
}
```

Παράδειγμα 5.1 (συνέχεια): Δεν υπάρχει σφάλμα. Μετατροπή board σε annotated κώδικα



Παραγωγή annotated κώδικα

Report

```
Check string: :s3 in line:18 of initial code
Check string: :s5 in line:12 of initial code
```

```
1  import checkers.nullness.quals.*;
2  //ena akoma aplo paradeigma
3
4  public class StringTester2 {
5      public static void main(String [] args) {
6        @NonNull String s1;
7        @NonNull String s2;
8        @Nullable String s3;
9        @Nullable String s4;
10       @Nullable String s5;
11
12         s1="NonNullvalue1";
13         s2="NonNullvalue2";
14
15         //edo prokyptei sfalma
16         char c=s5.charAt(0);
17         s1=s2;
18         s3=null;
19         int i=s1.length();
20
21         //k edo prokyptei sfalma
22         int j=s3.indexOf(0);
23         System.out.println("Strings s1:"+s1+" Length:"+i);
24         s5="NonNullvalue5";
25         s4=s3;
26         System.out.println("Strings s3:"+s3+" Index Of 0:"+j);
27         i=s2.length();
28         System.out.println("Strings s2:"+s2+" Length:"+i);
29         s4="NonNullvalue4";
30
31         System.out.println("Strings s4: "+s4);
32         System.out.println("Strings s5: "+s5+" CharAt 0:"+c);
33
34     }
35 }
```

Παράδειγμα 5.2 (συνέχεια): Υπάρχει σφάλμα. Μετατροπή board σε annotated κώδικα και report.

6 BOATS ON THE RIVER- ΥΛΟΠΟΙΗΣΗ

6.1) Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει παρουσίαση του τρόπου που υλοποιήθηκε η εφαρμογή Boats On The River.

Η Boats On The River, είναι γραμμένη στη γλώσσα Java, όπως και τα προγράμματα εισόδου που δέχεται. Χρησιμοποιήθηκε ο IDE Eclipse Kepler με υποστήριξη Java SE8. Ακόμα, προστέθηκε το plug-in για τον Checker Framework.

Τα γραφικά που χρησιμοποιούνται στο παιχνίδι είναι πολύ απλά. Χρησιμοποιήθηκαν οι βιβλιοθήκες `Swing` και `AWT (Abstract Window Toolkit)`. Ως πλαίσιο, χρησιμοποιείται ένα αντικείμενο `JFrame`. Για την υλοποίηση του board του παιχνιδιού, χρησιμοποιείται ένα `JPanel`. Χρησιμοποιούνται, ακόμα, στιγμιότυπα του `JButton`, `ActionListener` και `ActionEvent` και υλοποίηση ενός `MouseListener`. Για τα αρχεία εισόδου και εξόδου χρησιμοποιούνται στοιχεία της `java.io`. Για την αναπαράσταση όλων των στοιχείων του παιχνιδιού (brooks, boats, rocks, resizers), χρησιμοποιείται η κλάση `Polygon (java.awt.Polygon)`, ενώ σε αρκετά σημεία γίνεται χρήση `ArrayList (java.util.ArrayList)`. Τέλος, για το «τρέξιμο» του παιχνιδιού, χρησιμοποιείται ένας `Timer (java.util.Timer)` και ένα `TimerTask (java.util.TimerTask)`.

6.2) Περιγραφή αρχείων/κλάσεων

i. MyBrooks.java

Η κλάση `MyBrooks`, υλοποιεί τα ρυάκια. Κάθε φορά που έχουμε διακλάδωση, δημιουργείται στο `GamePanel` και καινούριο αντικείμενο αυτής της κλάσης.

Η `MyBrooks` παίρνει ως παραμέτρους:

- → Το μέγεθος (`Boolean` μεγάλο ή μικρό)
- → Αν μπορεί να τροποποιηθεί ή όχι
- → Αν έχει βράχους
- → Το χρώμα
- → Τις συντεταγμένες από όπου ξεκινάει και τελειώνει το ρυάκι

Σε αυτήν την κλάση υλοποιούνται τα εξής:

- ✓ Ανάλογα με το μέγεθος, καθώς και με το αν διακλαδώνεται ή όχι, υπολογίζονται όλες οι συντεταγμένες που θα έχει το Polygon που θα το αναπαριστά.
  Μέθοδοι: int [] brookxcoords(), int [] brookycoords(), void crossedbrook(int tx3,int ty2,int ty3), void setycoords(int ty1,int ty2).
- ✓ Με βάση τις συντεταγμένες δημιουργείται το Polygon drawbrook.
- ✓ Δημιουργούνται βράχοι, αν υπάρχουν.
  Μέθοδος: void createRock(int yDeref).
- ✓ Δημιουργούνται resizers, αν υπάρχουν.
  Μέθοδος: createBoatResizer(int yDeref).
- ✓ Μπορεί να αλλάζει το μέγεθος του ρυακιού.
  Μέθοδος: void setBig(boolean s).
- ✓ Τέλος, η MyBrooks συμπεριλαμβάνει μεθόδους για την προσπέλαση των πεδίων που συμπεριλαμβάνει.

ii. MyBoats.java

Η κλάση MyBoats υλοποιεί κάθε βάρκα που ταξιδεύει στα ρυάκια. Σε κάθε αντικείμενο της κλάσης MyBrooks που δημιουργείται (σε κάθε ρυάκι), αντιστοιχεί και ένα καινούριο αντικείμενο αυτής της κλάσης.

Η MyBoats παίρνει ως παραμέτρους:

- → Το μέγεθος (Boolean μεγάλο ή μικρό)
- → Αν μπορεί να τροποποιηθεί ή όχι
- → Τις συντεταγμένες από όπου ξεκινάει και τελειώνει το ρυάκι
- → Το χρώμα

Σε αυτήν την κλάση υλοποιούνται τα εξής:

- ✓ Ανάλογα με το μέγεθος και το σημείο που βρίσκεται κάθε στιγμή η βάρκα, υπολογίζονται όλες οι συντεταγμένες που θα έχει το Polygon που θα το αναπαριστά.
  Μέθοδοι: int [] boatxcoords(), int [] boatycoords(), void computelbcross().
- ✓ Με βάση τις συντεταγμένες δημιουργείται το Polygon drawboat.
- ✓ Επιστρέφει τις συντεταγμένες του ρυακιού στο οποίο πρόκειται να κινηθεί.
  Μέθοδοι: int getxofNewBrook(), int getyofNewBrook().
- ✓ Κινείται σε συγκεκριμένο ρυάκι, όσο δεν έχει φτάσει στο τέλος του, στα ίσια και στα πλάγια σημεία του ρυακιού (όταν υπάρχει διασταύρωση).
  Μέθοδος: boolean moveboat(MyBrooks br).
- ✓ Εξετάζεται αν έχει συναντήσει η βάρκα resizer.

Μέθοδος: boolean hasFoundResizer(Polygon Pol).

- ✓ Εξετάζεται αν έχει κολλήσει η βάρκα σε βράχο.
  Μέθοδος: boolean isStuckInRock(Polygon Pol).
- ✓ Η βάρκα μπορεί να σπρωχτεί. Σε αυτήν την περίπτωση το χρώμα της γίνεται κόκκινο και το μέγεθός της μικρό.
  Μέθοδος: void push().
- ✓ Αλλάζει το μέγεθος της βάρκας.
  Μέθοδος: void setBig(boolean size).
- ✓ Τέλος, η MyBoats συμπεριλαμβάνει μεθόδους για την προσπέλαση των πεδίων που συμπεριλαμβάνει (get και set).

iii. StringDb.java

Η κλάση StringDb χρησιμοποιείται για να κρατάει και να αρχικοποιεί τα string που υπάρχουν στον αρχικό κώδικα, ώστε με βάση αυτά να κατασκευαστεί και το board του παιχνιδιού.

Η StringDb παίρνει ως παραμέτρους:

- → Το όνομα του string
- → Τη γραμμή στην οποία συναντάται πρώτη φορά στον κώδικα που εξετάζεται

Κρατάει στοιχεία (boolean) σχετικά με το αν το string είναι null, αν προσπελαύνεται, αν αρχικοποιείται με κάποια τιμή, αν ανατίθεται σε κάποιο άλλο και αν είναι Nullable ή όχι. Σε ArrayList κρατάει τα σημεία του κώδικα όπου εμφανίζεται το string, τα σημεία όπου γίνεται null, τα σημεία προσπέλασης, αρχικοποίησης, τα σημεία όπου η τιμή του ανατίθεται σε κάποιο άλλο και τα string στα οποία ανατίθεται η τιμή του.

iv. FileReadWrite.java

Η κλάση FileReadWrite χρησιμοποιείται για να διαβαστεί το αρχείο εισόδου, να βρεθούν τα string και τα υπόλοιπα στοιχεία που χρειαζόμαστε από το αρχείο, καθώς και για να παραχθεί το αρχείο εξόδου και/ή τα μηνύματα λάθους. Δημιουργεί ένα ArrayList από αντικείμενα της κλάσης StringDb, για να κρατάει όλα τα string του κώδικα, καθώς και οι υπόλοιπες πληροφορίες για αυτά.

Στην FileReadWrite υλοποιούνται τα παρακάτω:

- ✓ Διαβάζεται γραμμή-γραμμή ο κώδικας εισόδου.
  Μέθοδος: void readTheFile().
- ✓ Βρίσκει τα string που δηλώνονται και για κάθε μία δημιουργεί καινούριο αντικείμενο της κλάσης StringDb και το προσθέτει στην σχετική ArrayList που τα κρατάει.

Μέθοδος: void findStrVars(String stringLine, int numberOfLine).

- ✓ Βρίσκει τα σημεία όπου χρησιμοποιούνται τα string.
Μέθοδος: void findUsedVars ().
- ✓ Για κάθε string, για όλες τις φορές που χρησιμοποιείται, εξετάζεται αν: γίνεται προσπέλασή του, αν παίρνει την τιμή null, αν αρχικοποιείται με κάποια τιμή και αν συσχετίζεται με κάποιο άλλο string και με ποιο. Ενημερώνονται αντίστοιχα και τα σχετικά πεδία της StringDb.
Μέθοδος: void checkUsedVars().
- ✓ Δημιουργεί το αρχείο εξόδου γραμμή προς γραμμή. Γράφει τον αρχικό κώδικα και στα σημεία που δηλώνονται τα string, προσθέτει τα annotations @Nullable και @NonNull, με βάση το σχετικό πεδίο του StingDb. Αν έχει βρεθεί κάποιο σφάλμα, τότε τυπώνεται στην κονσόλα η γραμμή και το string όπου παρουσιάζεται το σφάλμα.
Μέθοδος: void WriteAnnotatedCode().

v. GamePanel.java

Σε αυτό το αρχείο συμπεριλαμβάνονται οι κλάσεις RemindTask (extends TimerTask) και GamePanel (extends JPanel implements ActionListener).

Κλάση: RemindTask (extends TimerTask)

Σε αυτήν την κλάση υλοποιείται το τρέξιμο του παιχνιδιού. Πιο συγκεκριμένα:

- ✓ Επιστρέφει το ποιο ρυάκι συμπεριλαμβάνει συγκεκριμένες συντεταγμένες.
Μέθοδος: int [] findBrook(int x,int y).
- ✓ Εκτελεί το τρέξιμο. Για όλες τις βάρκες σε όλα τα ρυάκια, όταν μπορούν να κινηθούν (λόγω μεγέθους, βράχων και θέσης στο ρυάκι), κινούνται μέσω της moveboat(MyBrooks br) της κλάσης myBoats. Αν υπάρχει διασταύρωση, μέσω της findBrook(int x,int y) βρίσκει ποιο είναι το καινούριο ρυάκι στο οποίο μπορεί να κινηθεί. Μέσω των κατάλληλων μεθόδων της myBoats, ελέγχει αν η βάρκα έχει φτάσει σε resizer, όπου κάνει το μέγεθός της μικρό και αν κόλλησε σε διασταύρωση ή βράχο, όπου και περιμένει να σπρωχτεί. Όταν όλες οι βάρκες φτάσουν στον προορισμό τους, ο Timer ακυρώνεται.
Μέθοδος: void run ().

Κλάση: GamePanel (extends JPanel implements ActionListener)

Στην κλάση GamePanel υλοποιούνται οι βασικές λειτουργίες της εφαρμογής.

Δημιουργείται ένα ArrayList (για όλα τα ρυάκια του board), με στοιχεία ArrayList (κάθε στοιχείο της οποίας έχει το αρχικό ρυάκι που αντιστοιχεί σε κάθε string και τα υπόλοιπα στα οποία διακλαδώνεται), με αντικείμενα της

κλάσης MyBrooks. Αντίστοιχα για αντικείμενα της κλάσης MyBoats. (ArrayList<ArrayList<MyBrooks>>hyperBrooks, ArrayList<ArrayList<MyBoats>> hyperBoats).

- ✓ Αρχικά, χρησιμοποιούνται οι μέθοδοι της FileReadWrite, ώστε να διαβαστεί το αρχείο και να κατασκευαστεί το ArrayList με αντικείμενα StringDb, που θα χρησιμοποιηθεί στη συνέχεια, προστίθενται τα κουμπιά GO!!!, Restart και DONE! (αντικείμενα JButton) και καθορίζεται το μέγεθος του Panel.
- ✓ Στη συνέχεια κατασκευάζεται το board.
  - — Για όλα τα string καθορίζεται αν μπορεί να τροποποιηθεί το μέγεθός τους ή όχι. Εξετάζονται τα πεδία του κάθε αντικειμένου StringDb. Αν γίνεται αρχικοποίηση με κάποια τιμή πριν προσπελαστεί ή ανατεθεί σε κάποιο άλλο, τότε μπορεί να τροποποιείται, αλλιώς δεν μπορεί.
  - — Το παραπάνω, όπως και το αν προσπελαύνεται το string ή όχι, χρησιμοποιείται για να δημιουργηθούν τα αντικείμενα της κλάσης MyBrooks. Στην αρχή δημιουργείται μόνο το πρώτο ρυάκι για κάθε string, ανεξάρτητα αν στη συνέχεια διακλαδώνεται. Οι x συντεταγμένες καθορίζονται από το μήκος του panel και τον αριθμό των ρυακιών, ενώ οι y από το μήκος του panel και τον αριθμό των γραμμών του κώδικα εισόδου.
  - — Όταν υπάρχει ανάθεση της τιμής ενός string σε ένα άλλο, τότε έχουμε διακλάδωση και διασταύρωση. Αλλάζουν κατάλληλα οι συντεταγμένες του Polygon που αναπαριστά το αντίστοιχο ρυάκι, με χρήση των μεθόδων της MyBrooks. Δημιουργείται καινούριο αντικείμενο MyBrooks για το καινούριο κομμάτι της διακλάδωσης και συνδέεται με την ArrayList που κρατάει τα αντικείμενα.
  - — Για όλα τα string εξετάζεται αν γίνεται προσπέλαση και αν ναι, προστίθενται βράχοι στο κατάλληλο ρυάκι.
  - — Αντίστοιχα εξετάζεται και για αρχικοποίηση με τιμή, και προστίθενται resizers.
  - — Για κάθε ρυάκι, δημιουργείται και μία βάρκα, ως αντικείμενο της MyBoats. Ανάλογα με το ρυάκι στο οποίο ταξιδεύει, καθορίζονται και το μέγεθός της, οι συντεταγμένες της.

  <u>Μέθοδος</u>: void initbb().
- ✓ Υλοποιούνται τα γραφικά του παιχνιδιού. Ορίζεται χρώμα στο φόντο και σχεδιάζονται όλα τα Polygon που αναπαριστούν τα ρυάκια, τις βάρκες, τους βράχους και τα resizers.
  <u>Μέθοδος</u>: paintComponent(Graphics g).
- ✓ Ορίζεται η απόκριση στα κλικ που γίνονται.

— Όταν γίνεται κλικ πάνω σε βάρκα που έχει κολλήσει, καλείται η μέθοδος push() της MyBoats. Συμπληρώνεται κατάλληλα η αναφορά σφαλμάτων.
— Όταν γίνεται κλικ σε ένα ρυάκι του οποίου το μέγεθος μπορεί να τροποποιηθεί, τότε γίνεται τροποποίηση με τις κατάλληλες μεθόδους της MyBrooks. Αντίστοιχα τροποποιείται και το μέγεθος της βάρκας που αντιστοιχεί σε αυτό.

Ενημερώνονται κατάλληλα τα γραφικά (repaint()).
Μέθοδος: void mousecl(int mx, int my).

✓ Αφού έχει προστεθεί ActionListener στα κουμπιά, υλοποιείται ο χειριστής των γεγονότων.
— Στο πάτημα του κουμπιού GO!!!, αν έστω και μία βάρκα μπορεί να ταξιδέψει σε ένα ρυάκι, τότε δημιουργείται ένα αντικείμενο της κλάσης RemindTask (extends TimerTask).
— Στο πάτημα του κουμπιού Restart, ακυρώνεται ο Timer, καλείται η μέθοδος initbb() για κατασκευή από την αρχή του board και ενημερώνονται κατάλληλα τα γραφικά μέσω της repaint().
— Στο πάτημα του κουμπιού DONE! Γίνεται αντιστοίχηση του μεγέθους των ρυακιών, με το αν μπορούν να κρατήσουν τα αντίστοιχα string την τιμή null και ενημερώνεται το κατάλληλο πεδίο στα StringDb. Καλείται η WriteAnnotatedCode() της κλάσης FileReadWrite.

Μέθοδος: void actionPerformed(ActionEvent e).

vi. GameTester.java

Στο συγκεκριμένο αρχείο συμπεριλαμβάνεται η main. Συμπεριλαμβάνονται οι κλάσεις GameTester και MousepadListener implements MouseListener.

Κλάση: GameTester

Σε αυτήν τη κλάση, δημιουργείται αντικείμενο της κλάσης GamePanel, στο οποίο προσθέτουμε τον MousepadListener. Δημιουργείται το riverGameFrame (JFrame), στο οποίο προστίθεται το αντικείμενο GamePanel.

Η main (static void main(String[] args)) δημιουργεί αντικείμενο της κλάσης GameTester.

Κλάση: MousepadListener (implements MouseListener)

Δημιουργείται ο MouseListener που παίρνει τις συντεταγμένες από το κλικ και τις περνάει στη μέθοδο mousecl(x,y) της GamePanel().

7 ΣΥΜΠΕΡΑΣΜΑΤΑ-ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

7.1) Συμπεράσματα

Ο τυπικός έλεγχος λογισμικού είναι μία διαδικασία που δεν μπορεί να παραβλεφθεί, για κανένα σύστημα λογισμικού. Η εργασία προσανατολίστηκε στο να παρουσιάσει τους τρόπους με τους οποίους μπορεί να πραγματοποιηθεί. Είναι διαδικασία, όμως, που απαιτεί χρόνο και κόστος. Η ανάπτυξη μεθόδων που μπορεί να μειώσουν τις απαιτήσεις σε χρόνο και κόστος είναι απαραίτητη. Γι' αυτό και στα κεφάλαια 4-6 έγινε προσπάθεια να παρουσιαστεί η προσέγγιση της χρήσης crowdsourcing στην τυπική επαλήθευση λογισμικού. Μπορούμε να εξάγουμε ως συμπέρασμα ότι το πρόβλημα της τυπικής επαλήθευσης λογισμικού, είναι δυνατό να αντιστοιχηθεί με ένα παιχνίδι, ώστε παίζοντάς το να γίνεται επαλήθευση.

Όπως είπαμε και παραπάνω στην παρουσίαση, ο έλεγχος τύπων για τοπικές μεταβλητές, συνήθως δεν χρειάζεται. Φάνηκε και στα παραδείγματα, καθώς τα σφάλματα που εντοπίσαμε και η απόδειξη της απουσίας τους ήταν πολύ απλά και θα μπορούσαν να εντοπιστούν και με άλλες μεθόδους, εύκολα. Η επέκταση της εφαρμογής όμως σε ολόκληρα project, μπορεί να συμβάλλει στη λύση του δύσκολου και χρονοβόρου προβλήματος του type inference για μεγάλα συστήματα.

7.2) Μελλοντικές επεκτάσεις

Η εφαρμογή Boats on the river, μέχρι το στάδιο που παρουσιάστηκε, ήταν ένα παράδειγμα για το πώς θα μπορούσε να γίνει ευκολότερη η διαδικασία του software verification, με χρήση crowdsourcing. Ως μελλοντική δουλειά χρειάζεται να αναπτυχτούν τα εξής:

Σχετικά με την αντιστοίχηση των προγραμμάτων στο παιχνίδι:

- Να επεκταθεί η εφαρμογή, ώστε να μην υπάρχουν περιορισμοί στα προγράμματα Java εισόδου που δέχεται (πχ τρόπος δηλώσεων, κλήση μεθόδων, διακλαδώσεις).
- Να μπορεί να γίνει χειρισμός όλων των αντικειμένων που μπορεί να προκαλέσουν `NullPointerException`, και όχι μόνο των string.
- Να αντιστοιχηθούν στο πρόγραμμα και άλλες ιδιότητες ασφάλειας, επιπλέον της αποτροπής Null Pointer Error.

Σχετικά με το ίδιο το παιχνίδι, ως μελλοντική δουλειά μπορούν να υλοποιηθούν αρκετές βελτιώσεις:

- Γραφικά του παιχνιδιού, ώστε να γίνει το παιχνίδι πιο ελκυστικό.
- Υποστήριξη πολλαπλών board και επιπέδων, για την αναπαράσταση μεγαλύτερων κομματιών κώδικα, όπως και κλήσεων. Κάτι τέτοιο μπορεί να ανεβάσει και το ενδιαφέρον των παικτών, καθώς συνιστά μεγαλύτερη πρόκληση.
- Προσθήκη συστήματος σκορ, όπου υψηλότερο σκορ θα σημαίνει λιγότερα σπρωξίματα βαρκών, ώστε να αποφεύγεται από τον παίκτη.
- Τέλος, για να μπορούν να έχουν πρόσβαση οι εν δυνάμει παίκτες στο παιχνίδι, και έτσι να έχει και νόημα η έννοια του crowdsourcing, χρειάζεται να ανέβει στο διαδίκτυο.

8 ΒΙΒΛΙΟΓΡΑΦΙΑ

8.1) Βιβλία:

- Baier C. and Katoen J.-P., "**Principles of Model Checking**", The *MIT Press Cambridge*, *Massachusetts*, *London*, *England*, 2008.
- Clarke E. M., Grumberg O. and Peled D.,"**Model Checking**", The *MIT Press Cambridge*, *Massachusetts*, *London*, *England*, 1999.
- Biere A., "**Handbook of Satisfiability**", *IOS Press*, 2009.

8.2) Δημοσιεύσεις:

- W. Dietl, S. Dietzel, M. D. Ernst, N. Mote, B. Walker, S. Cooper, T. Pavlik, and Z. Popović, "**Verification games: Making verification fun**", in Workshop on Formal Techniques for Java-like Programs (FTfJP), ACM, pp. 42–49, 2012
- Cousot, P., Cousot, R., " **A gentle introduction to formal verification of computer systems by abstract interpretation**", in Logics and Languages for Reliability and Security, J. Esparza, B. Spanfelner, & O. Grumberg (Eds), NATO Science Series III: Computer and Systems Sciences, IOS Press, Volume 25, Pages 1-29, 2010.
- Floyd W., "**Assigning meaning to programs**", in J.T. Schwartz, editor, Proc. Symposium in Applied Mathematics, volume 19, pages 19–32. AMS, 1967.
- Naur P., "**Proofs of algorithms by general snapshots**", BIT, 6:310–316, 1966.
- Hoare A.R., "**An axiomatic basis for computer programming**", Communications of the ACM, 12(10):576–580, Oct. 1969
- Clarke E. M. and Emerson E. A., "**Design and synthesis of synchronization skeletons using branching time temporal logic**", in Logic of Programs, volume 131 of Lecture Notes in Computer Science, pages 52–71, Springer-Verlag, 1981.
- Queille J.-P. and Sifakis J., "**Specification and verification of concurrent systems in CESAR**", in 5th International Symposium on Programming, volume 137 of Lecture Notes in Computer Science, pages 337–351. Springer-Verlag, 1982.

- Ouimet M., "**Formal Software Verification: Model Checking and Theorem Proving**", Technical Report ESL-TIK-00214, Massachusetts Institute of Technology, 2008.

- Filliâtre, J.-C., "**Deductive software verification**", in International Journal on Software Tools for Technology Transfer (STTT) 13(5), 397–403, 2011.

- Ray S., "**Scalable Techniques for Formal Verification**", Springer, 2010.

- D'Silva V., Kroening D., & Weissenbacher G., "**A survey of automated techniques for formal software verification**", IEEE Transactions on CAD of Integrated Circuits and Systems, 27(7), 1165–1178, 2008.

- Cousot P. and Cousot R., **"Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints", i**n Proc. POPL, Los Angeles, California, ACM Press, New York, pages 238—252, 1977.

- Clarke, E., Kroening, D., Sharygina, N., Yorav, K., "**Predicate abstraction of ANSI–C programs using SAT**", Formal Methods in System Design 25, 105–127, 2004.

- Daniel J., Parízek P., "**Predicate Abstraction in Program Verification: Survey and Current Trends**", ICCSW'14, pp. 27–35, 2014.

- Amla N., Du X. , Kuehlmann A., Kurshan R. P. and McMillan K. L., "**An analysis of SAT-based model checking techniques in an industrial environment**" in Correct Hardware Design and Verification Methods (CHARME), pp. 254-268, Springer, 2005.

- Estellés-Arolas E., González-Ladrón-de-Guevara F., "**Towards an Integrated Crowdsourcing Definition**", Journal of Information Science 38 (2): 189–200, 2012

- Stol K. and Fitzgerald B., "**Two's Company, Three's a Crowd: A Case Study of Crowdsourcing Software Development**", in Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 2014.

- LaToza T. D. and van der Hoek A., "**A vision of crowd development**," in Proceedings of the 37th International Conference on Software Engineering, NIER Track, 2015

- Khatib F, Cooper S, Tyka MD, Xu K, Makedon I, Popović Z., Baker D. and Foldit players "**Algorithm discovery by protein folding game players**", PNAS USA, 108 (47) pp. 18949–18953, 2011.

- Dean D. et al. "**Lessons learned in game development for crowdsourced software formal verification**", in 2015 USENIX Summit on Gaming, Games, and Gamification in Security Education 3GSE'15, 2015.

8.3) Διαδίκτυο:

- James Gleick, A bug and a crash, http://www.around.com/ariane.html
- http://www.newsbeast.gr/weekend/arthro/740509/olethria-sfalmata-ton-upologiston/
- Williams L, "Testing Overview and Black-Box Testing Techniques" , 2006 http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf
- Henzinger Tom, EPFL, Model Checking, Theorem Proving, and Abstract Interpretation: The Convergence of Formal Verification Technologies http://www.jaist.ac.jp/~bjorner/ae-is-budapest/talks/Sept19pm2_Henzinger.pdf
- Pelánek Radek, Formal Verification, Model Checking, http://www.fi.muni.cz/~xpelanek/IA158/slides/verification.pdf
- Sharygina Natasha, Formal Verification by Model Checking, https://www.cs.cmu.edu/~aldrich/courses/654-sp05/handouts/model-checking-3.pdf
- Tomáš Vojnar, Formal Analysis and Verification, http://www.fit.vutbr.cz/study/courses/FAV/public/Lectures/fav-lecture-01.pdf
- Rabinovitz Ishai, Directions in Formal Verification of Software, https://www.inf.unibz.it/~artale/FM/IBM.pdf
- Rugina Radu, Introduction to compilers, http://www.cs.cornell.edu/courses/cs412/2003sp/lectures/lec12.pdf
- Oracle, Java Tutorial https://docs.oracle.com/javase/tutorial/java/annotations/type_annotations.html
- The Checker Framework Manual: Custom pluggable types for Java http://types.cs.washington.edu/checker-framework/current/checker-framework-manual.html
- https://en.wikipedia.org/wiki/Crowdsourcing
- DARPA, Crowd Sourced Formal Verification (CSFV) http://www.darpa.mil/program/crowd-sourced-formal-verification
- Verification games: Making verification fun http://homes.cs.washington.edu/~mernst/pubs/verigames-ftfjp2012-abstract.html