Ατομική Διπλωματική Εργασία

Αυτόματη παραγωγή περιπτώσεων ελέγχου λογισμικού μέσω συμβολικής εκτέλεσης προγραμμάτων από γενετικούς αλγορίθμους.

Αντώνης Κουρράς

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2008

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αυτόματη παραγωγή περιπτώσεων ελέγχου λογισμικού μέσω συμβολικής εκτέλεσης προγραμμάτων από γενετικούς αλγορίθμους.

Αντώνης Κουρράς

Επιβλέπων Καθηγητής
Ανδρέας Ανδρέου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2008

Ευχαριστίες

Νιώθω χρέος μου να ευχαριστήσω όλους τους ανθρώπους που με διάφορους τρόπους με βοήθησαν στην εκπόνηση αυτής της εργασίας.

Ο επιβλέπων καθηγητής Δρ. Ανδρέας Ανδρέου μου έδωσε την δυνατότητα να ασχοληθώ με το θέμα της αυτόματης δημιουργίας δεδομένων ελέγχου με τη χρήστη συμβολικής εκτέλεσης και των γενετικών αλγορίθμων. Οι κατευθυντήριες οδηγίες που μου έχει δώσει ήταν καθοριστικές στην επιτυχή ολοκλήρωση της εργασίας αυτής. Επίσης τον ευχαριστώ για την κατανόηση και την εμπιστοσύνη που μου επέδειξε καθ' όλη την διάρκεια της υλοποίησης της διπλωματικής μου εργασίας.

Ο Δρ. Αναστάσης Σοφοκλέους, αφιέρωσε πολύτιμο από τον χρόνο του έτσι ώστε να βοηθήσει καθ' όλη την διάρκεια της εκπόνησης της εργασίας. Με τις γνώσεις του και την εμπειρία του στο συγκεκριμένο θέμα με βοήθησε να κατανοήσω διάφορες λειτουργίες της εφαρμογής του, στις οποίες έπρεπε να προσθέσω τον δικό μου κώδικα, αλλά και διάφορες θεωρίες που απαιτούνταν για την κατανόηση του θέματος. Επίσης με βοήθησε πολλές φορές να λύσω προβλήματα που προέκυπταν και λόγω της περιορισμένης μου γνώσης γύρω από το θέμα χρειαζόμουν την πολύτιμη βοήθειά του.

Τέλος, επειδή με αυτή την εργασία, ολοκληρώνονται και οι σπουδές μου ως προπτυχιακός φοιτητής, θα ήθελα να ευχαριστήσω την οικογένεια μου, που με υποστήριξε σε όλες μου τις αποφάσεις με κάθε τρόπο.

Περίληψη

Ο έλεγχος είναι μια διαδικασία η οποία είναι μια αναπόφευκτη διαδικασία για την ανάπτυξη και παραγωγή ενός λογισμικού. Έχουν γίνει αρκετές προσεγγίσεις οι οποίες αναφέρονται στον έλεγχο λογισμικών με την αυτόματη δημιουργία δεδομένων ελέγχου. Με την αύξηση της πολυπλοκότητας του λογισμικού εμφανίστηκε η ανάγκη για εξεύρεση αξιόπιστων και αποδοτικών μεθόδων ελέγχου, έτσι ώστε να είναι επαρκής ο έλεγχος.

Σε αυτή τη διπλωματική εργασία, υλοποιείται μια εφαρμογή η οποία δημιουργεί αυτοματοποιημένα δεδομένα ελέγχου με τη χρήση μιας υβριδικής μεθόδου. Η μέθοδος αυτή χρησιμοποιεί συμβολικό έλεγχο για την εξαγωγή ενός συνόλου περιορισμών το οποίο θα αναφέρεται σε κάθε ένα μονοπάτι του κώδικα. Ακολούθως με τη χρήση ενός γενετικού αλγορίθμου επιλύονται οι πιο πάνω περιορισμοί και εξάγονται τα δεδομένα ελέγχου.

Στην αρχή παρουσιάζεται το θεωρητικό υπόβαθρο καθώς και η ανάλυση του αλγορίθμου που χρησιμοποιήθηκε. Ακολούθως θα παρουσιαστεί η εφαρμογή που έχει υλοποιηθεί και θα παρατεθούν διάφορα πειράματα και αποτελέσματα της. Επίσης θα συγκριθεί το σύστημα αυτό με ένα είδη υπάρχον σύστημα, το JCUTE. Τέλος θα παρουσιαστούν μερικοί περιορισμοί που υπάρχουν καθώς και κάποια σχόλια και εισηγήσεις για μελλοντική εργασία.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Γενικά

Ο έλεγχος των προγραμμάτων στις μέρες μας είναι μια αναπόφευκτη διαδικασία έτσι ώστε να υπάρχει η δυνατότητα μέτρησης τόσο της ποιότητας όσο και της αποδοτικότητας ενός λογισμικού. Είναι ευρέως γνωστό πως πριν την δημοσιοποίηση κάποιου λογισμικού και προώθησης του στην αγορά, περνά από πολλαπλά στάδια ελέγχου. Για τους πιο πάνω λόγους κατανοούμε πως ο έλεγχος λογισμικού είναι ένα αρκετά σημαντικό στάδιο βάση του οποίου κρίνεται (δηλ. αν είναι ορθό ως προς τις αρχικές απαιτήσεις και προδιαγραφές) στο τέλος ένα λογισμικό. Συνεπώς ο έλεγχος λογισμικού αποτελεί αναπόσπαστο κομμάτι τις διαδικασίας παραγωγής λογισμικού.

Η διαδικασία ελέγχου λογισμικού δεν θεωρείται καθόλου εύκολη λειτουργία. Αντιθέτως είναι μια επίπονη και δαπανηρή εργασία μιας και καταλαμβάνει το 25% - 50% του συνολικού κόστους στη διαδικασία παραγωγής ενός λογισμικού. Ακόμη ο χρόνος που χρειάζεται για τη διεκπεραίωση της μπορεί να είναι και μεγαλύτερος από αυτόν του χρόνου κατασκευής του λογισμικού. Πρέπει να επισημανθεί πως ο έλεγχος αποσκοπεί στην εύρεση παρουσίας λαθών και όχι στην υπόδειξη έλλειψης λαθών.

Λόγω της δυσκολίας αυτής της διεργασίας, αρκετοί επιστήμονες ασχολήθηκαν με τεχνικές ελέγχου οι οποίες θα προσφέρουν αυτοματοποιημένα ποιοτικά και αξιόπιστα αποτελέσματα σε σύντομο χρονικά διάστημα, χωρίς την επιβάρυνση του ανθρώπου [1]. Όταν αναφερόμαστε σε ποιοτικά και αξιόπιστα αποτελέσματα εννοούμε την

εύρεση πιθανών λαθών στον πηγαίο κώδικα. Εάν αυτό γίνει εφικτό τότε αυτή η διαδικασία θα πάψει να είναι τόσο δύσκολη και επίπονη, καθώς επίσης θα μειωθεί και το κόστος ανάπτυξης λογισμικού.

Η λέξη κλειδί εδώ είναι το αυτοματοποιημένα. Ένα αυτοματοποιημένο σύστημα ελέγχου λογισμικού είναι ένα σύστημα το οποίο θα παράγει δεδομένα εισόδου βάση του κώδικα του προγράμματος, με τα οποία θα ελέγχει διάφορα κομμάτια (μονοπάτια) του κώδικα. Έτσι με όπλο την υπολογιστική δύναμη των σημερινών υπολογιστών η διαδικασία αυτή θα φθάσει σε υψηλά επίπεδα απόδοσης. Αποτέλεσμα αυτού είναι ο έλεγχος περισσότερων κομματιών του κώδικα σε πολύ λιγότερο χρόνο σε σχέση με τον έλεγχο που μπορεί να κάνει από μόνος του ο άνθρωπος. Συνεπώς η πιθανότητα εύρεσης λαθών θα είναι κατά πολύ μεγαλύτερη. Τα συστήματα αυτά είναι αναγκαία πλέον μετά την δημιουργία πολύπλοκων και πολύ μεγάλων σε έκταση κώδικα λογισμικών.

Λόγω του ότι η εύρεση λαθών σχετίζεται άμεσα με την επιλογή των δεδομένων εισόδου πρέπει να γίνεται «καλή» επιλογή. Μετά από αρκετές μελέτες που έχουν γίνει, δημιουργήθηκαν κάποια κριτήρια κάλυψης ελέγχου βάση τον οποίον κρίνεται η ποιότητα του λογισμικού. Τα κριτήρια αυτά εφαρμόζονται στον κώδικα όταν αυτός αναπαριστάται είτε σαν γράφος ροής δεδομένων είτε σαν γράφος ροής ελέγχου. Βάση αυτών των κριτηρίων η επιλογή των δεδομένων εισόδου κρίνεται ως αρκετά ικανοποιητική.

Η παραγωγή δεδομένων εισόδου (ή δεδομένων ελέγχου) είναι σημαντική για δύο βασικούς λόγους. Αρχικά τα εμπειρικά αποτελέσματα καταδεικνύουν ότι οι έλεγχοι που έχουν επιλεχθεί με βάση τα κριτήρια κάλυψης είναι ικανοί για την ανακάλυψη λαθών και δεύτερον τα κριτήρια επάρκειας κάλυψης είναι αντικειμενικά και ικανά να κρίνουν τη ποιότητα του υπό εξέταση λογισμικού.

1.2 Υποκίνηση Έρευνας

Έχουν γίνει αρκετές προσπάθειες για αυτοματοποίηση της διαδικασίας παραγωγής δεδομένων ελέγχου. Η αυτοματοποίηση είναι ένα σημαντικό βήμα για την ελάττωση του κόστους και του χρόνου, τόσο κατά την δημιουργία λογισμικού αλλά και για την

συντήρηση του. Έχουν δημιουργηθεί και προταθεί αρκετές τεχνικές, με την κάθε μία από αυτές να έχει τα θετικά και τα αρνητικά της.

Σε αυτή την εργασία θα υλοποιηθεί μια από τις πιο πρόσφατες διαδεδομένες τεχνικές, γνωρίζοντας πως παρά τα μερικά προβλήματα που υπάρχουν τα αποτελέσματά της είναι αρκετά καλά. Η τεχνική αυτή είναι ο συμβολικό έλεγχος για την εξαγωγή ενός συνόλου περιορισμών για κάθε μονοπάτι και η επίλυσή τους με τη βοήθεια του γενετικού αλγορίθμου. Ακολούθως στο τέλος θα γνωρίζει ο χρήστης το ποσοστό κάλυψης των κριτηρίων βάση των ακμών και των συνθηκών που έχουν διαπεραστεί.

Στόχος της έρευνας αυτής είναι η υλοποίηση ενός προγράμματος το οποίο θα έχει ως είσοδο ένα πρόγραμμα γραμμένο σε Java και αυτοματοποιημένα, χρησιμοποιώντας την πιο πάνω τεχνική θα παράγει κάποιες εξισώσεις οι οποίες θα αντιστοιχούν σε συγκεκριμένο μονοπάτι του πηγαίου κώδικα και θα τις επιλύει. Μετά την ολοκλήρωση του συστήματος αυτού θα ελεγχθούν προγράμματα τόσο στη δική μου υλοποίηση όσο και στο είδη υλοποιημένο σύστημα JCUTE και ακολούθως θα συγκριθούν τα αποτελέσματά.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα μελετήσουμε τη θεωρία πίσω από την πράξη. Όπως και σε άλλους τομείς έτσι και στην τεχνολογία λογισμικού πρέπει να γίνει μια αρχική μελέτη των θεωριών έτσι ώστε να υπάρξει ένα γερό υπόβαθρο της συγκεκριμένης γνώσης. Με την γνώση αυτή θα είναι το πρώτο βήμα για την δημιουργία ενός αλγορίθμου ο οποίος θα επιφέρει καλά αποτελέσματα.

Στόχος αυτού του κεφαλαίου είναι η παρουσίαση σημαντικών εννοιών τόσο ως προς τον έλεγχο λογισμικού, την παραγωγή περιπτώσεων δοκιμής, το γράφο εξάρτησης καθώς και μία εισαγωγή ως προς τους γενετικούς αλγορίθμους.

2.2 Μέθοδοι Ελέγχου

Πρόσφατη έρευνα [1] η οποία έχει γίνει με θέμα τις μεθόδους ελέγχου κατέληξε στην ύπαρξη τριών τεχνικών.

Οι τεχνικές αυτές είναι:

Έλεγχος Μαύρου κουτιού (Black Box Testing): Ο Black-box έλεγχος ή concrete box είναι μια κατηγορία, όπου οι συνθήκες ελέγχου για το πρόγραμμα, δημιουργούνται

βάση τις λειτουργίες και τις προδιαγραφές του συστήματος. Ο προγραμματιστής το μόνο που χρειάζεται είναι πληροφορίες για τις τιμές εισόδου. Στη συνέχεια γνωρίζοντας αφαιρετικά την λειτουργία του συστήματος, αγνοώντας τον κώδικα (για αυτό λέγεται μαύρο κουτί), παρατηρεί τα αποτελέσματα και τα συγκρίνει με τα επιθυμητά. Η τεχνική αυτή βασίζεται στις προδιαγραφές και τις λειτουργίες του προγράμματος. Λόγο του ότι δεν χρειάζεται γνώση στον κώδικα του λογισμικού, δεν είναι αναγκαίο ο ελεγκτής να είναι ο ίδιος ο προγραμματιστής. Πιο κάτω γίνεται αναφορά στα κύρια πλεονεκτήματα καθώς και μειονεκτήματα της τεχνικής αυτής.

Πλεονεκτήματα:

- Είναι αποτελεσματικότερος σε μεγάλα κομμάτια κώδικα σε σχέση με το white box testing
- Ο ελεγκτής δεν χρειάζεται προγραμματιστικές γνώσεις, αρκεί να γνωρίζει για συγκεκριμένες εισόδους τις επιθυμητές εξόδους.
- Ο προγραμματιστής και ο ελεγκτής είναι ανεξάρτητοι ο ένας απ’ τον άλλο.
- Οι έλεγχοι γίνονται από την πλευρά άποψης του χρήστη.
- Βοηθά στο να ανακαλυφθούν οποιεσδήποτε ασάφειες ή ασυνέπειες στις προδιαγραφές.

Μειονεκτήματα:

- Μόνο ένας μικρός αριθμός πιθανών inputs μπορεί πραγματικά να ελεγχθεί μιας και είναι αδύνατο να ελεγχθούν όλοι οι δυνατοί συνδυασμοί. Αποτέλεσμα αυτού είναι ο μη ικανοποιητικός έλεγχος μιας και δεν ελέγχονται όλα τα πιθανά μονοπάτια.
- Για να δημιουργηθούν οι περιπτώσεις δοκιμής πρέπει να έχουμε σαφή προδιαγραφές (μη διφορούμενες, συγκεκριμένες).
- Μπορεί να υπάρξει περιττή επανάληψη των test inputs, αν ο ελεγκτής δεν ενημερώνεται για τα test cases του προγραμματιστή.
- Δεν μπορεί να κατευθυνθεί προς συγκεκριμένα τμήματα του κώδικα που μπορούν να είναι πολύ σύνθετα και επομένως επιτρέπονται περισσότερα λάθη.
- Οι περισσότερες έρευνες που αφορούν τον έλεγχο έχουν κατευθυνθεί προς την τεχνική του white box testing

Έλεγχος Άσπρου κουτιού (White Box Testing): Η τεχνική αυτή προϋποθέτει πως ο ελεγκτής πρέπει να γνωρίζει τις εσωτερικές λειτουργίες του προγράμματος, δηλαδή να γνωρίζει τη λειτουργία του καθώς και τον πηγαίο κώδικα. Οι δοκιμές ελέγχου σχεδιάζονται βάση της δομής του προγράμματος. Λόγω του ότι τα δεδομένα ελέγχου όπως αναφέρθηκε και πιο πάνω λαμβάνονται βάση του πηγαίου κώδικα, υπάρχει μεγαλύτερη πιθανότητα εντοπισμού λάθη. Κριτήρια κάλυψης της τεχνικής αυτής είναι:

- path coverage
- statement coverage
- branch coverage
- condition coverage
- edge coverage

Οι δοκιμές ελέγχου παράγονται προσεκτικά βάση ενός ή συνδυασμού περισσότερων κριτηρίων κάλυψης. Για μια τιμή εισόδου που δίνεται στο σύστημα ελέγχεται όχι μόνο το αν δημιουργείται το σωστό αποτέλεσμα, αλλά και το πώς προκύπτει. Πιο κάτω γίνεται αναφορά στα κύρια πλεονεκτήματα καθώς και μειονεκτήματα της τεχνικής αυτής.

Πλεονεκτήματα:

- Αυτή η τεχνική αναγνωρίζει με σχετική ευκολία το φαινόμενο της συμπτωματικής ακρίβειας (coincidental correctness). Το φαινόμενο αυτό αναφέρεται στις περιπτώσεις όπου παρόλο το αποτέλεσμα πρόκυπτε ορθό, ο τρόπος που υπολογίζεται να είναι λανθασμένος. Για τα συγκεκριμένα δεδομένα ελέγχου δεν είναι και τόσο σημαντικό αλλά κατά πάσα πιθανότητα για άλλα να μην είναι ορθό το αποτέλεσμα.
- Επίσης με την τεχνική αυτή μπορούμε να ελέγξουμε όλους τους πιθανούς τρόπους λειτουργίας του μιας και ο έλεγχος γίνεται βάση του πηγαίου κώδικα.
- Με την τεχνική αυτή μπορούμε να ανακαλύψουμε περιπτώσεις νεκρού κώδικα. Δηλαδή περιπτώσεις κώδικα που δεν εκτελούνται ποτέ και δεν μπορούν να ανακαλυφθούν με την τεχνική του black box testing.

Μειονεκτήματα:

- Με την τεχνική αυτή είναι πολύ δύσκολο να εντοπίσουμε λάθη λογικής.

- Είναι σχεδόν αδύνατο σε περιπτώσεις μεγάλων συστημάτων να εξεταστούν όλα τα μονοπάτια του κώδικα έτσι ώστε να ανακαλυφθούν όλα τα λάθη που υπάρχουν. Αυτό είναι ένα πρόβλημα το οποίο περιορίζει αρκετά την τεχνική αυτή και γίνονται προσπάθειες επίλυσής του.
- Δεν υπάρχει τρόπος να εντοπισθούν τα μονοπάτια τα οποία έχουν παραληφθεί.

Έλεγχος Γκρίζου Κουτιού (Gray Box Testing)**:** Η τεχνική αυτή είναι ένας συνδυασμός των δύο πιο πάνω τεχνικών. Δηλαδή συνδυάζει τόσο τον έλεγχο βάση της εσωτερικής δομής όσο και τον έλεγχο βάση των προδιαγραφών του λογισμικού. Σκοπός της δημιουργίας της πιο πάνω τεχνικής είναι η όσο το δυνατόν καλύτερη αξιοποίηση των πλεονεκτημάτων των δυο προηγούμενων τεχνικών, με αποτέλεσμα τον καλύτερο έλεγχο.

2.3 Κριτήρια Ελέγχου Λογισμικού – Μετρικές Κάλυψης

Για να γίνει ο έλεγχος ενός λογισμικού πρέπει να επικεντρωθούμε σε κάποια κριτήρια. Τα κριτήρια [1] αυτά αναφέρονται σε διαφορετικούς τομείς του προγράμματος. Στόχος των κριτηρίων αυτών είναι να ανακαλύψει του τομείς του προγράμματος που δεν θα εκτελεστούν από ένα σύνολο περιπτώσεων δοκιμής. Μετά τον εντοπισμό των κομματιών αυτών δημιουργούνται πρόσθετες περιπτώσεις δοκιμής για να αυξηθεί η κάλυψη του κώδικα και εμμέσως η ποιότητα του ελέγχου που γίνεται. Τα κριτήρια ελέγχου μπορούν να χωριστούν στις ακόλουθες ομάδες: Τα κριτήρια ελέγχου μπορούν να χωριστούν στις ακόλουθες ομάδες:

- Structural testing

  Τα κριτήρια ελέγχου καθορίζονται με βάση την κάλυψη συγκεκριμένων ομάδων στοιχείων της δομής του προγράμματος. Για παράδειγμα στο path testing όπου ελέγχεται ένα-ένα μονοπάτι ξεχωριστά. Να σημειωθεί πως ο αλγόριθμος που έχω υλοποιήσει ανήκει σε αυτή τη κατηγορία ελέγχου μιας και η συμβολική εκτέλεση και η παραγωγή δεδομένων ελέγχου αναφέρεται για ένα μονοπάτι κάθε φορά.
- Fault-base testing

  Τα κριτήρια ελέγχου καθορίζονται έτσι ώστε να επικεντρώνονται στην ανίχνευση λαθών στο λογισμικό.

- Error base testing

  Τα κριτήρια ελέγχου επικεντρώνονται σε διάφορες περιπτώσεις δοκιμής οι οποίες επιλέχθηκαν έτσι ώστε να εκτελούν συγκεκριμένες λειτουργίες του προγράμματος για τον έλεγχο συγκεκριμένων λαθών.

Υπάρχει μια μεγάλη ποικιλία των μετρικών κάλυψης. Πιο κάτω θα ακολουθήσει μια περιγραφή μερικών θεμελιωδών μετρικών καθώς και των δυνάμεων και αδυναμιών τους. Οι μετρικές αυτές είναι κριτήρια κάλυψης ροής ελέγχου.

2.3.1 Κάλυψη Εντολής (Statement Coverage)

Σε αυτή τη μετρική μετριέται κάθε εκτελέσιμη εντολή (statement) η οποία πρέπει να εκτελεστεί τουλάχιστον μια φορά. Σκοπός των ελεγκτών είναι η επιλογή κατάλληλων περιπτώσεων δοκιμής βάση των οποίων θα εκτελεστούν όλες οι εντολές του λογισμικού. Το πλεονέκτημα αυτής της μετρικής είναι ότι μπορεί να εφαρμοστεί άμεσα στον κώδικα αντικειμένου (object code) και δεν απαιτεί τον κώδικα πηγής (source code). Το μειονέκτημα της μετρικής αυτής είναι πως σε μερικές δομές ελέγχου δεν ανταποκρίνεται με ορθότητα και τις «παραβλέπει». Παράδειγμα της αδυναμίας αυτής είναι:

```
int* p = NULL;
if (condition)
p = &variable;
*p = 123;
```

Επίσης οι περιπτώσεις δοκιμής καλό είναι να ελέγχουν σε περίπτωση συνθηκών (conditions) και τις δύο περιπτώσεις (μπορεί να υπάρχει σφάλμα σε μία από τις δύο περιπτώσεις). Αυτό όμως δεν μπορεί να εγγυηθεί από αυτή τη μετρική με αποτέλεσμα ο έλεγχος να είναι ελλιπής. Ο λόγος που θεωρείται ελλιπής είναι η άγνοια της μιας από τις δύο περιπτώσεις που ίσως να είναι και η προβληματική. Ακόμη υπάρχει ανεπάρκεια σε βρόγχους λόγω του ότι δεν αναγνωρίζει το κριτήριο αυτό πότε θα τερματίσουν καθώς και στην συγκεκριμένη περίπτωση do-while το οποίο αντιμετωπίζεται ως μια μη εντολή διακλάδωσης. Ακόμη και στις περιπτώσεις των λογικών πράξεων (and &&, or ||) το κριτήριο αυτό δεν ανταποκρίνεται με επάρκεια. Σε γενικές γραμμές, αυτή η μετρική επηρεάζεται περισσότερο από τις υπολογιστικές δηλώσεις απ' ότι από τις αποφάσεις (decisions**).**

2.3.2 Κριτήριο Απόφασης (Decision Coverage)

Σε αυτή τη μετρική δίδεται έμφαση περισσότερο στις εντολές απόφασης. Αντιμετωπίζονται οι σχετικές αδυναμίες που αναφέρθηκαν στην πιο πάνω μετρική (Statement Coverage) γιατί εξετάζονται και οι δύο περιπτώσεις μιας εντολής απόφασης (αληθής και ψευδής). Ένα μειονέκτημα είναι ότι αυτή η μετρική αγνοεί τις συνθήκες που περιέχουν λογικές διακλαδώσεις. Παράδειγμα της αδυναμίας αυτής είναι:

```
if (condition1 && (condition2 || function1()))
        statement1;
else
        statement2;
```

Η μετρική αυτή θα μπορούσε να εξετάσει την πιο πάνω εντολή ελέγχου χωρίς να καλέσει την function1. Ο λόγος είναι πως στην Java εάν είναι αληθής η συνθήκη condition1 τότε θα ελέγξει την συνθήκη condition2 || function1(). Ακολούθως εάν η συνθήκη condition2 είναι αληθής τότε δεν θα καλέσει την συνάρτηση function1(). Άρα ο έλεγχος είναι ανεπαρκής μιας και αυτή η περίπτωση αγνοείται.

2.3.3 Κάλυψη Συνθήκης (Condition Coverage)

Σε αυτή τη μετρική αντιμετωπίζεται το μειονέκτημα της προηγούμενης μετρικής (Decision Coverage) με την ανάλυση των εντολών απόφασης και την εκτέλεση τους ανεξάρτητα από τις υπόλοιπες.

2.3.4 Κάλυψη Πολλαπλής Συνθήκης (Multiple Condition Coverage)

Η διαφορά με το πιο πάνω κριτήριο είναι η ανάλυση των σύνθετων συνθηκών σε κάθε υποσυνθήκη και όχι ως μία σύνθετη συνθήκη. Σε αυτή τη μετρική αναλύονται οι σύνθετες συνθήκες σε πιο απλές και τότε εξάγονται όλοι οι πιθανοί συνδυασμοί. Ακολούθως εκτελούνται όλοι οι συνδυασμοί για να εξαχθούν περιπτώσεις δοκιμής. Ο έλεγχος εφαρμόζεται και για τις δύο περιπτώσεις, αληθής και ψευδής. Για παράδειγμα εάν έχουμε τις παρακάτω συνθήκες,

a && b && (c || (d && e))

((a || b) && (c || d)) && e

τότε η ανάλυση τους βάση της μετρικής αυτής θα είναι :

1. a && b && (c || (d && e))
    - 1. F - - - -
    - 2. T F - - -
    - 3. T T F F -
    - 4. T T F T F
    - 5. T T F T T
    - 6. T T T - -

Οι περιπτώσεις 1,2,3,4,5 αναφέρονται στη ψευδή τιμή της πιο πάνω συνθήκης και η 6 στην αληθή.

2. ((a || b) && (c || d)) && e

- 1. F F - - -
- 2. F T F F -
- 3. F T F T F
- 4. F T F T T
- 5. F T T - F
- 6. F T T - T
- 7. T - F F -
- 8. T - F T F
- 9. T - F T T
- 10. T - T - F
- 11. T - T - T

Οι περιπτώσεις 1,2,3,4,5,6,7,8,9,10,11 αναφέρονται στη ψευδή τιμή της πιο πάνω συνθήκης και η 6 στην αληθή.

Μεγάλο πλεονέκτημα της λύσης αυτής είναι η κάλυψη όλων των απλών συνθηκών με αποτέλεσμα η καλύτερη δυνατή προσέγγιση ως προς τον έλεγχο της σύνθετης συνθήκης.

Να σημειωθεί πως η βελτιστοποίηση των περιορισμών που θα εξαχθούν μετά την ανάλυση κάθε μονοπατιού γίνεται βάση αυτού του κριτηρίου.

2.3.5 Κριτήριο Κάλυψης Μονοπατιών (Path Coverage)

Αυτή η μετρική δίνει έμφαση στην εκτέλεση των εκτελέσιμων μονοπατιών. Ένα μονοπάτι αποτελείται από ένα σύνολο εντολών οι οποίες ξεκινούν από την αρχική εντολή και καταλήγουν σε τελική. Στόχος είναι η εύρεση ενός συνόλου από περιπτώσεις δοκιμής το οποίο θα μπορεί να εκτελεί όλα τα εφικτά μονοπάτια και να αναγνωρίζει τα dead paths. Μειονέκτημα αυτού είναι οι εντολές επανάληψης γιατί είδη μπορούν να εξαχθούν αρκετά μονοπάτια από την πολυπλοκότητα του λογισμικού και επιπλέον αρκετά άλλα βάση του αριθμού των επαναλήψεων. Αυτό όμως δεν είναι επιθυμητό και έτσι οι εντολές επανάληψης αντιμετωπίζονται ως απλές.

2.3.6 Υβριδικά κριτήρια ελέγχου

Τα κριτήρια ελέγχου έχουν τη δυνατότητα να συνδυάζονται έτσι ώστε να αποφέρουν καλύτερα αποτελέσματα. Δύο από τις πιο διαδεδομένες υβριδικές μεθόδους είναι το κριτήριο συνθήκης/απόφασης (condition/decision coverage) και το κριτήριο ακμής/ συνθήκης(edge/condition coverage). Το κριτήριο συνθήκης/απόφασης (condition/decision coverage) έχει το πλεονέκτημα της απλότητας αλλά χωρίς τις ανεπάρκειες των δύο αυτών μετρικών. Το κριτήριο ακμής/ συνθήκης(edge/condition coverage) αναφέρεται στην κάλυψη τόσο των εκτελέσιμων ακμών όσο και στις εκτελέσιμες συνθήκες.

2.4 Τεχνικές Παραγωγής Περιπτώσεων Δοκιμής (Random test data generation)

Υπάρχουν αρκετοί τρόποι όπου μπορούμε να εξάγουμε κατάλληλες περιπτώσεις δοκιμής. Κάθε τεχνική έχει τα θετικά και τα αρνητικά της. Οι τρεις τεχνικές [1] παραγωγής περιπτώσεων δοκιμής είναι οι πιο κάτω :

2.4.1 Τυχαία παραγωγή στοιχείων δοκιμής (Random test data generation)

Είναι η πιο απλή τεχνική ελέγχου. Χρησιμοποιείται για να δημιουργούνται δεδομένα εισόδου για οποιαδήποτε τύπου προγραμμάτων. Όπως καταλαβαίνουμε οι τιμές

ελέγχου λαμβάνονται τυχαία. Το αρνητικό αυτής της μεθόδου είναι ότι δεν είναι και τόσο αποδοτική.

2.4.2 Παραγωγή δεδομένων ελέγχου βάση ενός στόχου (Goal-oriented test data generation)

Είναι πιο δυνατή τεχνική σε σχέση με την προηγούμενη μιας και καθοδηγείται βάση συγκεκριμένου συνόλου από μονοπάτια. Αντί να δημιουργεί δεδομένα εισόδου τέτοια ώστε να διαπερνούν ένα μονοπάτι από την είσοδο προς την έξοδο, δημιουργεί δεδομένα τα οποία διαπερνούν ένα δοθέν απροσδιόριστο μονοπάτι. Με αυτή τη μέθοδο βρίσκουμε ένα σύνολο από δεδομένα εισόδου τα οποία είναι κατάλληλα για όλα τα μονοπάτια τα οποία ανήκουν στο απροσδιόριστο μονοπάτι. Συνεπώς μειώνεται το ρίσκο των μη πρακτικών μονοπατιών και προσδιορίζει την αναζήτηση των δεδομένων εισόδου.

2.4.3 Παραγωγή δεδομένων ελέγχου βάση ενός μονοπατιού (Path-oriented test data generation)

Είναι η δυνατότερη από τις δύο προηγούμενες μιας και αναλύει ένα ειδικό μονοπάτι. Είναι παρόμοια με την πιο πάνω τεχνική (Goal-oriented test data generation) με μόνη διαφορά τη χρήση συγκεκριμένου μονοπατιού. Αυτή η διαφορά δίνει καλύτερες αποδόσεις (καλύπτει περισσότερο το πρόγραμμα). Το μειονέκτημα της λύσης αυτής είναι πως πολύ δύσκολα βρίσκουμε δεδομένα ελέγχου.

Σε αυτή την κατηγορία εντάσσεται η παραγωγή δεδομένων ελέγχου στο σύστημα που έχει υλοποιηθεί. Ο λόγος που προτιμήθηκε η κατηγορία αυτή είναι τα πλεονεκτήματα που προσφέρει η πιο πάνω τεχνική. Η εστίαση σε ένα μόνο μονοπάτι κάθε φορά οδηγεί στην καλύτερη ανάλυση του.

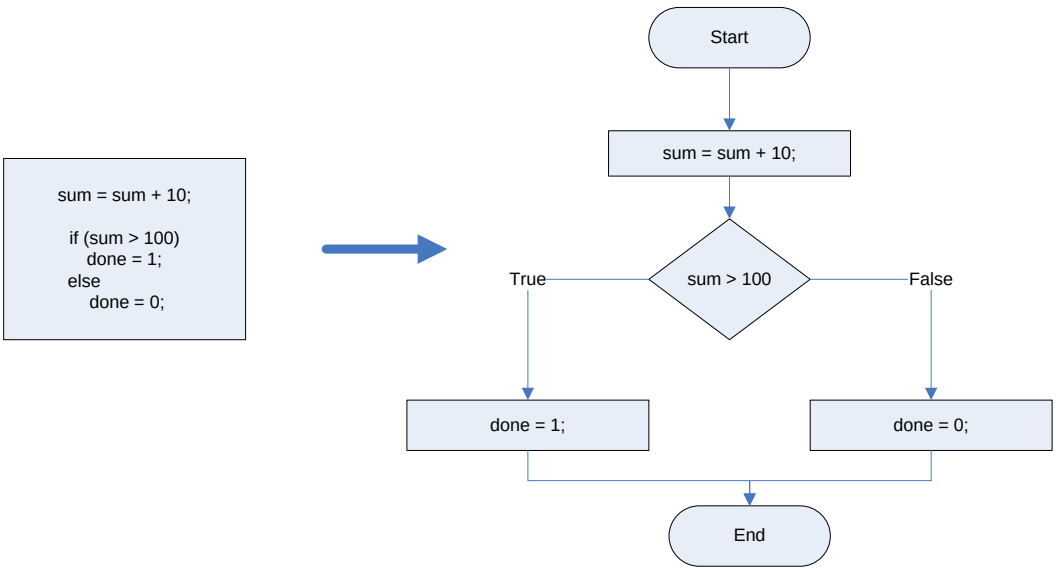
2.5 Αναπαραστάσεις Κώδικα

Υπάρχουν αρκετοί τρόποι για να αναπαραστήσουμε ένα πρόγραμμα. Ο λόγος που χρειάζεται η αναπαράσταση ενός προγράμματος σε σχηματική μορφή είναι η καλύτερη κατανόηση της λειτουργίας του προγράμματος. Η ιδεολογία των γράφων και η

απλότητα της αναπαράστασής τους οδήγησαν στη δημιουργία γράφων έτσι ώστε να αναπαριστάται ένα πρόγραμμα βάση κάποιων παραμέτρων (πχ ροή ελέγχου, ροή δεδομένων κ.α). Παραδείγματα γράφων είναι, ο γράφος ροής ελέγχου (control flow graph), ο γράφος ροής δεδομένων (data flow graph) και ο γράφος εξαρτήσεων (dependence graph). Πιο κάτω θα αναλυθεί ο γράφος ροής ελέγχου ο οποίος χρησιμοποιήθηκε για την αναπαράσταση των προγραμμάτων Java τα οποία χρησιμοποιούνται ως είσοδο στο σύστημα που έχει υλοποιηθεί.

Ο γράφος ροής ελέγχου [2] είναι μία γραφική αναπαράσταση όλων των πιθανών μονοπατιών τα οποία μπορούν να προκύψουν από το αντίστοιχο πρόγραμμα κατά τη διάρκεια της εκτέλεσης τους. Αποτελείται από κόμβους, όπου κάθε κόμβος αντιπροσωπεύει μία εντολή του προγράμματος. Ο γράφος είναι κατευθυνόμενος και από τις κατευθύνσεις των ακμών μπορούμε να παρατηρήσουμε τις μεταπηδήσεις της ροής ελέγχου.

Ο γράφος αυτός δημιουργείται μέσω ιδικών προγραμμάτων (Walkers) τα οποία διαπερνούν τον κώδικα, αναγνωρίζοντας τις διάφορες εντολές. Όπως προαναφέρθηκε, για κάθε εντολή δημιουργείται ένα κόμβος και μια ακμή η οποία τον ενώνει με τον προηγούμενο. Σε περίπτωση εντολής διακλάδωσης τότε οι ακμές που δημιουργούνται είναι δύο.
Στο Σχήμα 2.1 παρατηρούμε ένα παράδειγμα γράφου ροής δεδομένου βάση του κώδικα που παρουσιάζει.



Σχήμα 2.1 –Γράφος Ροής Δεδομένων (CFG)

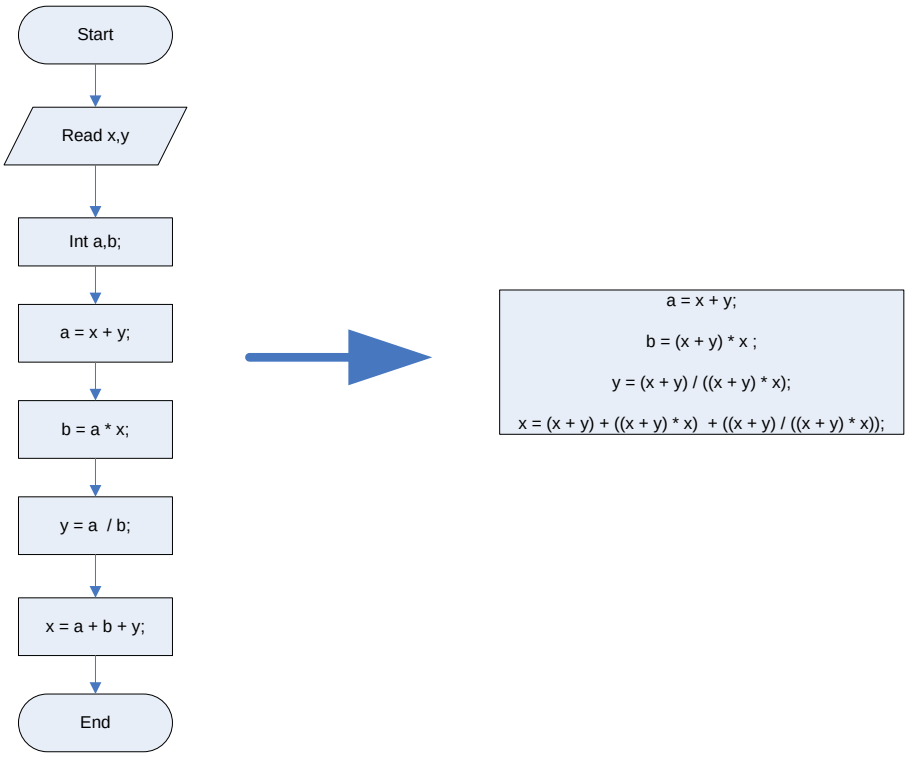
2.6 Συμβολικός Έλεγχος (Symbolic Testing)

Ο συμβολικός έλεγχος ήταν μια θεωρία η οποία ξεκίνησε από τον J. C. King το 1976 [3] και πρόσφατα έχει υλοποιηθεί και στην πράξη. Ο συμβολικός έλεγχος είναι μια μέθοδος ελέγχου λογισμικού η οποία χρησιμοποιεί συμβολικές μεταβλητές και όχι τις μεταβλητές του κώδικα. Ακολούθως βάση της ανάλυσης που θα γίνει θα παραχθούν κατάλληλα δεδομένα ελέγχου για όλα τα μονοπάτια του πηγαίου κώδικα. Ο συμβολικός έλεγχος βοηθά στην απλούστευση των τελικών εξισώσεων που θα δημιουργήσει. Οι εξισώσεις αυτές στη συνέχεια, με την κατάλληλη επεξεργασία καταλήγουν στο να μας δώσουν κατάλληλα δεδομένα εισόδου για το συγκεκριμένο μονοπάτι. Οι μέθοδοι που χρησιμοποιούνται για να εξαχθούν τα δεδομένα ελέγχου είναι οι πιο κάτω:

2.6.1 Συμβολική εκτέλεση (symbolic execution)

Μια στατική τεχνική ανάλυσης (ανάλυση ενός προγράμματος που εκτελείται κατά την διάρκεια εκτέλεσης του προγράμματος) η οποία παράγει μια συμβολική έκφραση (μαθηματική έκφραση) για κάθε μονοπάτι του προγράμματος. Βάση αυτής της συμβολικής έκφρασης γίνεται ο συμβολικός έλεγχος, Η εκτέλεση προχωρά όπως σε μια κανονική εκτέλεση με μόνη διάφορα πως οι τιμές μπορεί να είναι συμβολικοί τύποι πέρα από σύμβολα εισαγωγής, τα οποία αντιπροσωπεύουν όλες τις εισαγωγές (μεταβλητές εισόδου) που ακολουθούν τη δεδομένη πορεία (μονοπάτι). Η συμβολική έκφραση αντιπροσωπεύει τους περιορισμούς ενός μονοπατιού οι οποίοι πρέπει να ισχύουν για να ακολουθηθεί η συγκεκριμένη ροή ελέγχου και να καλυφθεί το μονοπάτι. Οι περιορισμοί αυτοί είναι όλοι διατυπωμένη συμβολικά, δηλαδή όλοι αναφέρονται στις μεταβλητές εισόδου μόνο. Η συμβολική εκτέλεση είναι white-box testing.

Στο Σχήμα 2.2 παρατηρούμε μία συμβολική εκτέλεση. Από το γράφο ροής δεδομένων παρατηρούμε τις τιμές των μεταβλητών μετά τη συμβολική εκτέλεση. Είναι προφανές πως όλες οι τοπικές μεταβλητές (a,b) πλέον αναφέρονται μόνο στις μεταβλητές εισόδου (x,y). Στη συμβολική εκτέλεση κάθε μεταβλητή έχει ως τρέχον τιμή την συμβολική της τιμή ως προς τα δεδομένα εισόδου.



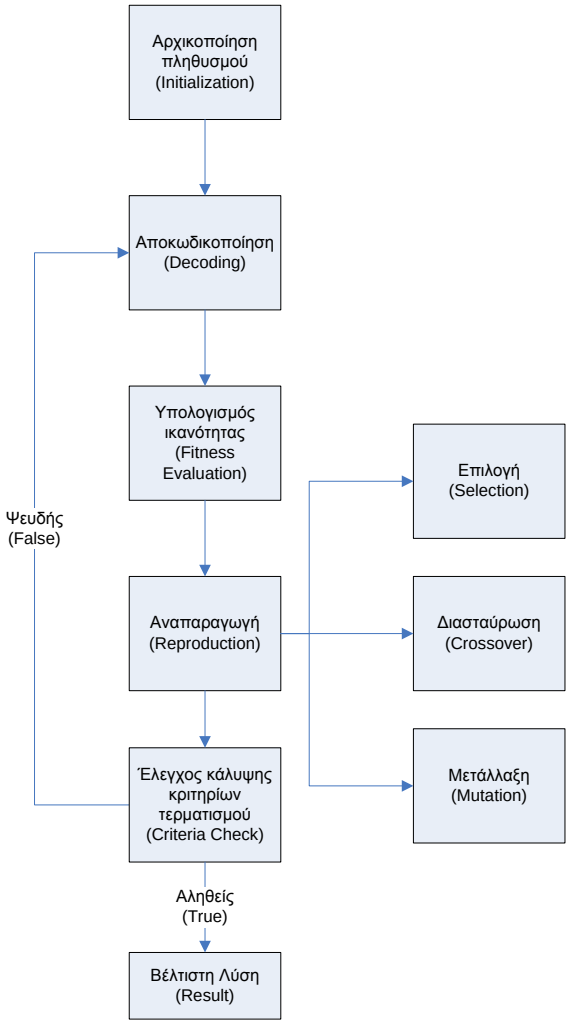
Σχήμα 2.2 – Συμβολική Εκτέλεση

2.6.2 Συμβολική προσομοίωση (symbolic simulation)

Ο συμβολικός προσομοιωτής υπολογίζει τις συμβολικές εκφράσεις για κάθε παραγωγή από την άποψη των μεταβλητών εισαγωγής. Αυτές οι εκφράσεις γενικεύονται από τον συμβολικό προσομοιωτή με αποτέλεσμα να αντιπροσωπεύουν και άλλες πιθανές ακολουθίες μεταβλητών εισόδου δίδοντας τις τιμές των αποτελεσμάτων τους. Αποτέλεσμα αυτού είναι η αύξηση της πιθανότητας εύρεσης λαθών σχεδιασμού (design errors) για ένα μεγάλο φάσμα μεταβλητών εισόδου. Η συμβολική προσομοίωση είναι black-box testing.

2.7 Γενετικός Αλγόριθμος (Genetic Algorithm)

Οι Γενετικοί Αλγόριθμοι (ΓΑ) είναι ένα μοντέλο μηχανισμού μάθησης του οποίου η συμπεριφορά απορρέει από τη μεταφορά μερικών από τους μηχανισμούς της εξέλιξης του φυσικού περιβάλλοντος. Αποτελούν ευπροσάρμοστες μεθόδους επίλυσης διαφόρων προβλημάτων αναζήτησης και βελτιστοποίησης. Μαζί με τον Εξελικτικό Προγραμματισμό, τις Στρατηγικές Εξέλιξης, τα Συστήματα Ταξινόμησης και το Γενετικό Προγραμματισμό αποτελούν μια ξεχωριστή κατηγορία συστημάτων επίλυσης προβλημάτων που είναι ευρύτερα γνωστή ως Εξελικτικοί Αλγόριθμοι.



Σχήμα 2.3 - Γενετικός Αλγόριθμος

Όπως παρατηρούμε και στο Σχήμα 2.3 ο γενετικός αλγόριθμος είναι μια επαναλαμβανόμενη μέθοδος βάση της οποίας αξιοποιούνται οι καλύτερες μέχρι στιγμής λύσεις για την δημιουργία καλύτερης επόμενης γενεάς.

Ο αλγόριθμος δουλεύει ως ακολούθως:
Αρχικοποίηση πληθυσμού (Initialization): Σε αυτό το βήμα επιλέγονται πιθανές λύσεις οι οποίες κωδικοποιούνται στα χρωμοσώματα.

Αποκωδικοποίηση (Decoding): Ακολούθως επιλέγονται όσα χρωμοσώματα όσος και ο αριθμός του πληθυσμού και αποκωδικοποιούνται.

Υπολογισμός ικανότητας (Fitness Evaluation): Για κάθε ένα άτομο του πληθυσμού (χρωμόσωμα) υπολογίζεται η ικανότητα του με το να τρέξει η αντικειμενική συνάρτηση.

Αναπαραγωγή (Reproduction): Το στάδιο της αναπαραγωγής αποτελείται από τρία μέρη:

- Επιλογή (Selection): Βάση της ικανότητας κάθε ατόμου και του αριθμού των ατόμων του πληθυσμού δημιουργείται η ρουλέτα. Κάθε θέση της ρουλέτας αντιπροσωπεύει την πιθανότητα επιλογής ενός ατόμου του πληθυσμού. Ακολούθως επιλέγονται τυχαία ένας αριθμός από άτομα ίσα με τον αριθμό των ατόμων του πληθυσμού. Λόγω του ότι τα πιο ικανά άτομα έχουν περισσότερη πιθανότητα επιλογής, ευελπιστούμε πως η νέα γενεά θα είναι απαρτισμένη με καλύτερα χρωμοσώματα.
- Διασταύρωση (Crossover): Επιλέγονται βάση πιθανοτήτων ζευγάρια των δύο, τα οποία θα διασταυρωθούν. Η διασταύρωση γίνεται σε ένα τυχαίο σημείο.
- Μετάλλαξη (Mutation): Τέλος για κάθε μια θέση όλων των ατόμων υπάρχει μία μικρή πιθανότητα η οποία μπορεί να μεταλλάξει τη θέση εκείνη. Με τη λέξη μετάλλαξη εννοούμε την αλλοίωση του συγκεκριμένου δεδομένου. (εάν στη θέση αυτή έχουμε 0 μετατρέπεται σε 1).

Έλεγχος κάλυψης κριτηρίων τερματισμού (Stop Criteria Check): Ελέγχεται εάν το κριτήριο τερματισμού έχει ικανοποιηθεί. Εάν ναι τότε τερματίζεται ο αλγόριθμος αλλιώς επιστρέφουμε στο βήμα 2.

Βέλτιστη Λύση (Result): Τα δεδομένα που μας παρέχει ο γενετικός είναι τα βέλτιστα δεδομένα για τη συγκεκριμένη περίπτωση.

Μερικά από τα πλεονεκτήματα των γενετικών αλγορίθμων τα οποία οδήγησαν στη χρήσης τους σε αυτή την εφαρμογή είναι:

- Επιλύουν δύσκολα και πολύπλοκα προβλήματα γρήγορα και αξιόπιστα.
- Είναι εξελίξιμοι και επεκτάσιμοι.

- Μπορούν εύκολα να ενσωματωθούν σε είδη υπάρχοντα συστήματα. Η μόνη χρήσιμη πληροφορία είναι αυτή της fitness function.
- Επεξεργάζονται ταυτόχρονα δεδομένα τα οποία ελέγχουν τόσο το χώρο αναζήτησης αλλά και βελτιστοποίησης ταυτόχρονα.

Κεφάλαιο 3

Προηγούμενες Μελέτες

3.1 Εισαγωγή

Εκτός από τη θεωρία καλό είναι να μελετηθούν και προηγούμενες μελέτες έτσι ώστε να αναγνωρισθούν τα προβλήματα που αντιμετώπισαν και να γίνει προσπάθεια επίλυσης. Επιπλέον μια πρώτη επαφή με είδη υλοποιημένα συστήματα οδηγούν στην καλύτερη κατανόηση του προβλήματος που θέλουμε να επιλύσουμε. Αναφορικά με τον έλεγχο λογισμικού καθώς και την αυτοματοποιημένη παραγωγή δεδομένων ελέγχου έγιναν αρκετές προσεγγίσεις. Επίσης αρκετές μελέτες έχουν γίνει και για τη συμβολική εκτέλεση λόγω των αρκετών δυνατοτήτων που παρουσιάζουν. Ο συνδυασμός των δυο πιο πάνω έχει επιφέρει αρκετά καλά αποτελέσματα. Πιο κάτω θα αναφέρω σχετικές μελέτες που έχουν γίνει και θα τις συγκρίνω με την δική μου εργασία.

3.2 Συμβολικός Έλεγχος

Ο συμβολικός έλεγχος είναι μια θεωρία η οποία ξεκίνησε από τον J. C. King το 1976 [3]. Η ανάγκη για συμβολική εκτέλεση παρουσιάστηκε με την αύξηση της πολυπλοκότητας των λογισμικών οδήγησε στην εφαρμογή αυτή της μεθόδου. Η απλοποίηση των μεταβλητών ως προς τις παραμέτρους και η εξαγωγή ενός συνόλου εξισώσεων οι οποίες με την επίλυση τους οδηγούν στον έλεγχο του προβλήματος ήταν κάποια στοιχεία τα οποία οδήγησαν στην ανάπτυξη της μεθόδου αυτής. Ο ορισμός του συμβολικού ελέγχου είναι αφαιρετικός μιας και όταν μιλούμε για υλοποιήσεις υπάρχουν δύο μέθοδοι. Η πρώτη μέθοδος είναι η συμβολική εκτέλεση η οποία

αναφέρεται στο υποκεφάλαιο 3.3 και η δεύτερη στη συμβολική προσομοίωση. Σκοπός της εργασίας αυτής είναι η έμφαση στη μεθοδολογία της συμβολικής εκτέλεσης.

3.3 Δυναμική Δημιουργία Περιπτώσεων Δοκιμής

Το άρθρο [1] επικεντρώνεται σε μια γενική ανάλυση για μεθόδους αυτοματοποιημένης παραγωγής δεδομένων ελέγχου οι οποίες βασίζονται στον πηγαίο κώδικα. Αναλύει το τυπικού συστήματος παραγωγής δεδομένων ελέγχου το οποίο αποτελείται από τον program analyzer, το path selector και το test data generator. Ο αναλυτής προγράμματος (program analyzer) παρέχει όλες τις πληροφορίες που αφορούν το πρόγραμμα (data dependence graph, control flow graphs etc). Με τη σειρά του ο επιλογέας μονοπατιού (path selector) αποφασίζει και καθορίζει ένα σύνολο από μονοπάτια τα οποία θα επεξεργαστούν και βάση αυτών θα δημιουργηθούν τα δεδομένα ελέγχου. Επίσης κάνει αναφορά στο πρόβλημα επιλογής μονοπατιών. Το πρόβλημα αυτό προκύπτει από τον μεγάλο αριθμό μονοπατιών που προκύπτουν από την πολυπλοκότητα των σημερινών λογισμικών με αποτέλεσμα η επιλογή του κατάλληλου μονοπατιού για έλεγχο να είναι κρίσιμη. Η υλοποίηση μου ακολουθεί το τυπικό σύστημα παραγωγής δεδομένων με μόνη διαφορά πως για path selector δεν έχω ένα αυτοματοποιημένο σύστημα επιλογής μονοπατιών αλλά επιτρέπω στον χρήστη να επιλέξει όλα ή μερικά μονοπάτια τα οποία θα χρησιμοποιηθούν για την συμβολική εκτέλεση και την παραγωγή περιπτώσεων δοκιμής για το κάθε ένα. Επίσης οι μόνες πληροφορίες που αξιοποιούνται από τον αναλυτή προγράμματος (program analyzer) είναι ο γράφος ροής δεδομένων. Ένα άλλο πρόβλημα που αναλύεται είναι το πρόβλημα ελέγχου με βάση τις συνθήκες (branches) χωρίς να λαμβάνεται υπόψιν ο υπόλοιπος κώδικας. Αυτό όμως είναι καταστροφικό ιδιαίτερα όταν ενημερώνονται οι μεταβλητές που βρίσκονται τις συνθήκες (branches). Με τη συμβολική εκτέλεση αυτό δεν αποτελεί πλέον πρόβλημα γιατί οι μεταβλητές αντικαθίστανται με την τρέχον τιμή τους βάση των μεταβλητών εισόδου. Το άρθρο αναφέρει και αναλύει τις τεχνικές παραγωγής περιπτώσεων δοκιμής (Random test data generation, Goal-oriented test data generation, Path-oriented test data generation). Εγώ έχω χρησιμοποιήσει το path-oriented test data generation το οποίο έχει τα πιο αποδοτικά αποτελέσματα.

Τα άρθρα [4,5] αναφέρονται σε ένα δυναμικό σύστημα το οποίο παράγει δεδομένα ελέγχου (test data generation framework) βασισμένο στους γενετικούς αλγορίθμους. Αποτελείται από ένα σύστημα ανάλυσης προγράμματος (program analyzer) και από ένα σύστημα παραγωγής δεδομένων ελέγχου (test case generator). Το σύστημα ανάλυσης προγράμματος αναλύει το πρόγραμμα που βρίσκεται υπό έλεγχο και δημιουργεί τον γράφο ροής ελέγχου (control flow graph). Το σύστημα παραγωγής δεδομένων ελέγχου χρησιμοποιεί δύο αλγόριθμους βελτιστοποίησης, τον Batch-Optimistic (BO) και τον Close-Up (CU) και έχει ως κριτήριο κάλυψης το edge/condition. Ο Batch-Optimistic (BO) αλγόριθμος . Ο Close-Up (CU) αλγόριθμος .

Επίσης κάνει αναφορά στο structural testing και συγκεκριμένα στη συμβολική εκτέλεση, την οποία χρησιμοποιώ εγώ στον δικό μου αλγόριθμο. Επισημαίνει διάφορα προβλήματα της δυναμικής παραγωγής δεδομένων ελέγχου (dynamic test data generation), τα οποία μερικά από αυτά έχω αντιμετωπίσει στον δικό μου αλγόριθμο. Τα προβλήματα αυτά είναι η πολυπλοκότητα των κριτηρίων κάλυψης, το fitness landscape problem [1] και η αποδοτικότητα του γενετικού αλγορίθμου.

Για το πρόβλημα της πολυπλοκότητας των κριτηρίων κάλυψης, μειώνω την πολυπλοκότητα γιατί αναλύω το κάθε μονοπάτι ξεχωριστά, βελτιστοποιώντας τα multiple conditions και επιλύοντας τα με τη χρήση του γενετικού αλγορίθμου. Έτσι και ο γενετικός έχει καλύτερα αποτελέσματα μιας και εστιάζεται μόνο σε ένα μονοπάτι. Ένα άλλο πρόβλημα είναι το fitness landscape problem, το οποίο έχω επιλύσει χρησιμοποιώντας ένα τύπο (1/(αριθμός των συνθηκών + τιμή της συνθήκης)) στην fitness function ο οποίος κατατοπίζει τον γενετικό αλγόριθμο στο κατά πόσο κοντά είναι στην επιθυμητή λύση. Η τιμή της συνθήκης επιστρέφει πόσο κοντά η μακριά είναι η προτεινόμενη τιμή του γενετικού αλγορίθμου ως προς την επιθυμητή τιμή (λύση). Τα ίδια προβλήματα αναφέρονται και στο άρθρο [6] το οποίο προτείνει την λύση Chaining Approach.

Το άρθρο [7] επεκτείνει την μελέτη που έχει γίνει στα άρθρα [4,5] με την ενίσχυση του συστήματος από έναν αναλυτή προγράμματος και μια γεννήτρια περίπτωσης δοκιμής, όπου το πρώτο αναλύει τα προγράμματα, δημιουργεί το γράφο ροής καθώς και το γράφο ροής δεδομένων, και αξιολογεί τις περιπτώσεις δοκιμής ως προς το κριτήριο ελέγχου. Η διαφορά με τα δύο προηγούμενα άρθρα είναι η δημιουργία και η αξιοποίηση του γράφου ροής δεδομένων. Ο γενετικός αλγόριθμος παύει πλέον να

λαμβάνει όλες τις πληροφορίες από τον γράφο ροής ελέγχου καθώς παίρνει πληροφορίες και από το γράφο ροής δεδομένων. Πιο λεπτομερώς ο αναλυτής προγράμματος δημιουργεί δυναμικά κατά τη διάρκεια εκτέλεσης του προγράμματος τον γράφο ροής ελέγχου και από αυτόν δημιουργεί το γράφο ροής δεδομένων. Ακολούθως ο γράφος ροής δεδομένων χρησιμοποιείται από τον γενετικό αλγόριθμο ο οποίος βάση των δεδομένων κριτηρίων θα δημιουργήσει τις καλύτερες περιπτώσεις δοκιμής. Η υλοποίηση μου δεν αξιοποιεί τον γράφο εξάρτησης δεδομένων γιατί υλοποιεί path testing όπου είναι ένα unit testing όπου οι απαραίτητες πληροφορίες μπορούν να εξαχθούν από το γράφο ροής ελέγχου. Θα ήταν αχρείαστο και θα πρόσθετε περισσότερη πολυπλοκότητα στο σύστημα, καθώς και περισσότερο χρόνο εκτέλεσης, εάν δημιουργούσα και χρησιμοποιούσα τον γράφο ροής δεδομένων.

Το άρθρο [8] προτείνει την ενσωμάτωση ενός συστήματος παραγωγής γράφου ροής δεδομένων (data flow graph) με ένα υπάρχον σύστημα το οποίο με την βοήθεια των γενετικών αλγορίθμων να δημιουργεί αυτόματα περιπτώσεις δοκιμής βάση του κριτηρίου ροής δεδομένων (data flow coverage). Η διαφορά με τον δικό μου αλγόριθμο είναι το κριτήριο κάλυψης. Η διαδικασία όμως της ανάλυσης του προγράμματος είναι παρόμοια με εκείνη που έχω υλοποιήσει και εγώ.

3.4 Συμβολική Εκτέλεση και δημιουργία περιπτώσεων δοκιμής

Το άρθρο [9] επικεντρώνεται στο path oriented test data generation χρησιμοποιώντας symbolic execution και τεχνικές επίλυσης περιορισμών. Διαχωρίζει τον έλεγχο σε unit testing, integration testing, system testing και user acceptance testing. Οι συγγραφείς χρησιμοποίησαν το unit testing το οποίο είναι white testing. Η προσέγγιση που χρησιμοποιήθηκε στο άρθρο αυτό είναι η χρήση αυτομάτων (extended finite state machines) έτσι ώστε να αποθηκεύουν τις κυριότερες πληροφορίες του πηγαίου κώδικα. Ένα EFSM μπορεί να είναι ντετερμινιστικό ή μη ντετερμινιστικό. Το πρόγραμμα χωρίζεται σε transitions βάση των οποίων αποτελείται το EFSM. Το EFSM χρησιμοποιείται σαν ένα κατευθυνόμενο γράφο μέσω του οποίου μπορούνε γραφικά να δούνε τα διάφορα μονοπάτια. Στο δικό μου αλγόριθμο έχω χρησιμοποιήσει τον IBM γράφο έτσι δεν χρειάστηκε να μεταφέρω σε αυτή τη μορφή τις κυριότερες πληροφορίες του προγράμματος. Επισημαίνεται ότι μετά από μελέτες έχει διαπιστωθεί πως μόνο το 1.4% των μονοπατιών (με συγκεκριμένο μέγεθος) εκτελούνται. Αυτό είναι ένα

πρόβλημα για το path oriented testing το οποίο έχει αναφερθεί και στο άρθρο [1]. Ο αλγόριθμος που χρησιμοποιείται για την εξαγωγή των μονοπατιών είναι ο depth first search (DFS) τον οποίο και εγώ χρησιμοποιώ στη δική μου εφαρμογή. Ένα πρόβλημα το οποίο υπάρχει στον αλγόριθμο αυτό είναι πως κάθε φορά που θα τρέχει πάντα θα βρίσκει «καλά» δεδομένα ελέγχου για όλα τα μονοπάτια με αποτέλεσμα να μην σταματά ποτέ. Αυτό μπορεί να λυθεί με το να οριστούν κάποιοι περιορισμοί. Για παράδειγμα κάθε transition (κόμβος του EFSM) να διαπερνάται μόνο ψ φορές σε κάθε εκτέλεση. Το πρόβλημα αυτό αντιμετωπίζεται στον δικό μου αλγόριθμο με το να σταματά ο γενετικός αλγόριθμος όταν έχει δημιουργήσει δεδομένα ελέγχου τα οποία ικανοποιούν το σύνολο των περιορισμών για το συγκεκριμένο μονοπάτι.

Στο άρθρο [10] γίνεται μία γενική αναφορά στο unit testing και επικεντρώνεται στον συμβολικό έλεγχο. Επισημαίνει τη δυσκολία της συμβολικής εκτέλεσης σε περιπτώσεις βρόγχων και εξηγεί πως για να γίνει εφικτός ο έλεγχος τους πρέπει να υπάρχει ένας σταθερός αριθμός επαναλήψεων. Στην υλοποίηση μου εγώ τις επαναληπτικές συνθήκες τις τεμαχίζω έτσι ώστε να αναγνωρίζω την αρχική τιμή της επαναληπτικής μεταβλητής, τη συνθήκη τερματισμού και την εξίσωση αύξησης της τιμής της επαναληπτικής μεταβλητής. Ακολούθως τα τρία αυτά μέρη ενσωματώνονται στον γράφο ροής ελέγχου και αντιμετωπίζονται ως απλές εντολές. Αξίζει να σημειωθεί πως ο έλεγχος γίνεται και για τις δύο περιπτώσεις της συνθήκης τερματισμού έτσι ώστε να μην υπάρχει ανεπάρκεια στον έλεγχο. Επίσης αναφέρεται στο πρόβλημα της επιλογής δυνατών μονοπατιών, δηλαδή εκτελέσιμων και όχι dead μονοπατιών. Η λύση την οποία ακολούθησα εγώ αναφέρθηκε πιο πάνω. Τέλος αναφέρεται στο σύστημα το οποίο υλοποίησαν καθώς και λεπτομερώς ο αλγόριθμος υλοποίησης.

Το άρθρο [11] αναφέρεται στην ανάλυση της ροής ελέγχου με τη χρήση συμβολικής εκτέλεσης (symbolic execution) και αρίθμηση των μονοπατιών (path enumeration). Η προσέγγιση αυτή, δηλαδή ο συνδυασμός των πιο πάνω τεχνικών, έγινε για την αποφυγή των βρόγχων που δεν εκτελούνται με τη συμβολική εκτέλεση. Η πιο πάνω ανάλυση χωρίζεται σε τρία βήματα, την ανάλυση ελέγχου, τη ανάλυση χαμηλού επιπέδου (low-level) και την αποτίμηση του αποτελέσματος. Αρχικά δημιουργείται ο γράφος ροής ελέγχου ο οποίος αντιπροσωπεύει τη ροή ελέγχου του δοθέντος προγράμματος. Κάθε block του γράφου μπορεί να μην αντιπροσωπεύει μόνο μια εντολή, πράγμα που

αντικαθιστά την μετέπειτα περιδιάβαση του γράφου σε πιο δύσκολη, σε σχέση με το γράφο που δημιουργείται στο δικό μου σύστημα όπου κάθε block περιέχει μία εντολή. Η συμβολική εκτέλεση γίνεται σε κάθε block, ξεκινώντας από το πιο πάνω επίπεδο και καταλήγοντας στα φύλλα. Την ίδια μεθοδολογία ακολουθώ και εγώ γιατί με αυτό τον τρόπο διαφυλάσσεται η ορθότητα του προγράμματος μιας και κάθε μεταβλητή πρέπει να έχει την τρέχον συμβολική τιμή ξεκινώντας από την αρχή του κώδικα μέχρι την τρέχον γραμμή του κώδικα. Στο άρθρο αυτό δεν αναφέρεται ο τρόπος εξαγωγής όλων των πιθανών μονοπατιών αλλά πως επιλέγονται τα εφικτά ως προς την εκτέλεση μονοπάτια. Ο τρόπος που ο δικός μου αλγόριθμος εξάγει τα μονοπάτια είναι με depth first search algorithm (DFS).

3.5 JCUTE

Το JCUTE [12] είναι ένα εργαλείο το οποίο παράγει αυτόματα περιπτώσεις δοκιμής χρησιμοποιώντας συμβολική εκτέλεση. Ακολουθεί unit testing. (unit testing: το πρόγραμμα χωρίζεται σε μονάδες (modules), οι οποίες αποτελούν ένα σύνολο από συναρτήσεις, και ελέγχονται ξεχωριστά). Δημιουργεί δεδομένα ελέγχου αναλύοντας δυναμικά τον κώδικα ελέγχου. Όταν οι είσοδοι στο unit αναφέρονται σε δείκτες, είναι memory graphs (μπορεί να έχουμε εις γνώση τις καλούμενες συναρτήσεις οι οποίες δεν υπάρχουν στον κώδικα αλλά σε συναρτήσεις). Συνδυάζει symbolic και concrete execution για να δημιουργήσει δεδομένα ελέγχου για όλα τα προσβάσιμα μονοπάτια του προγράμματος. Όταν βρίσκει νέο branch τότε δημιουργείτε ένα unit test.

Η κύρια ιδέα της μεθόδου που ακολουθήθηκε είναι η αναπαράσταση των δεδομένων εισόδου για το module χρησιμοποιώντας ένα λογικό χάρτη (logical input map) ο οποίος αναπαριστά όλα τα δεδομένα εισόδου συμπεριλαμβανομένου και ενός memory graph, ως μία συλλογή από συμβολικές μεταβλητές. Ακολούθως χρησιμοποιείται για να δημιουργηθούν οι κατάλληλοι περιορισμοί πάνω σε αυτά τα δεδομένα εισόδου εκτελώντας συμβολικά τον κώδικα που ελέγχεται(της μονάδας).

Στην αρχή τροποποίησαν τον κώδικα που θέλανε να ελέγξουν με το να τοποθετήσουν κλήσεις συναρτήσεων οι οποίες δημιουργούν συμβολικό έλεγχο (με την συνάρτηση execute_symbolic(m,e) αποτιμά την έκφραση e συμβολικά και την αντιστοιχεί στο χώρο μνήμης m ενώ με τη συνάρτηση evaluate_predicate(p,b) αποτιμάται συμβολικά το

p και ενημερώνεται το path_c (το path_c στο τέλος της εκτέλεσης θα έχει όλα τα predicates για το συγκεκριμένο μονοπάτι)). Ακολούθως έτρεξαν τον κώδικα αυτό επαναλαμβανόμενες φορές ως ακολούθως. Χρησιμοποίησαν τον λογικό χάρτη Z για να δημιουργήσουν υπαρκτούς γράφους μνήμης για το πρόγραμμα και δύο συμβολικές καταστάσεις (για τους δείκτες και τις αρχέγονες τιμές). Ο κώδικας τρέχει πραγματικά (concrete) (αρχικά εκτελείται η main για να αρχικοποιηθούν οι μεταβλητές και να δεσμευτεί η ανάλογη μνήμη για να δημιουργηθεί το memory graph) στον πραγματικό λογικό γράφο και συμβολικά στις συμβολικές καταστάσεις (δημιουργώντας περιορισμούς για τις επιθυμητές εισόδους οι οποίες θα οδηγούν στο συγκεκριμένο μονοπάτι). Ακολούθως βάση των περιορισμών που έχουν ανευρεθεί (συνήθως αλλάζει το τελευταίο predicate, παίρνουμε την άρνηση του) γίνεται η δημιουργία νέου λογικού χάρτη Z' και συνεχίζεται το ίδιο από την αρχή. Στόχος να ελεγχθούν όλα τα μονοπάτια. Με τον αλγόριθμο DFS προσπαθούν να αναγνωριστούν και να ελεγχθούν όλα τα μονοπάτια (παρόμοιο με Explicit Path Model-Checking). Ουσιαστικά το JCUTE στην αρχή τοποθετεί τις κλήσεις των συναρτήσεων για να δημιουργήσει συμβολικό έλεγχο και στη συνέχεια τρέχει τον κώδικα κανονικά.

Ο λογικός χάρτης εισόδων, αντιστοιχεί λογικές διευθύνσεις σε τιμές οι οποίες είτε είναι λογικές τιμές είτε είναι αρχέγονες τιμές. Αυτός ο χάρτης αναπαριστά το Input memory graph στην αρχή κάθε εκτέλεσης. Βάση αυτού ενημερώνονται οι 2 καταστάσεις A (arithmetic) και P (pointers).
Ο λόγος που το JCUTE παρουσιάζει λογικές διευθύνσεις είναι ότι υπάρχει πιθανότητα να μετατραπεί η τιμή του ως δυναμικά (ως προς το χρόνο) Επίσης απλουστεύει τις εισόδους. Ακόμη με τις λογικές διευθύνσεις δεν χρειάζεται να υπάρχει memory graph μιας και γνωρίζουμε πως συνδέονται τα κελιά. Με τη χρήση του χάρτη αυτού μπόρεσαν να αποφύγουν το πρόβλημα των δεικτών και των πινάκων.

Το JCUTE χρησιμοποιεί το CIL framework για να μετατρέπει σύνθετες καταστάσεις (χωρίς κλήση συναρτήσεων) σε πιο απλές καταστάσεις με την αλλαγή σύνθετων μεταβλητών σε μια σειρά από προσωρινές μεταβλητές. Συνήθως οι εκφράσεις αυτές προκύπτουν μετά από την συμβολική εκτέλεση.

Διαφορές JCUTE με αυτή την εργασία

Το JCUTE χρησιμοποιεί λογικούς χάρτες για την αναπαράσταση του κώδικα ενώ το σύστημα που υλοποιήθηκε ένα γράφο (IBM). Η αναπαράσταση του κώδικα είναι πιο απλή στη μορφή γράφου και πιο εύκολη στη χρήση παρά στους λογικούς χάρτες όπου υπάρχει περισσότερη πληροφορία.

Σε αντίθεση με το σύστημα που υλοποιήθηκε, το JCUTE χρησιμοποιεί constraint solver αντί γενετικό αλγόριθμο για την επίλυση των εξισώσεων που προκύπτουν από την συμβολική εκτέλεση. Αυτή είναι μια ιδιαιτερότητα του συστήματος που υλοποιήθηκε γιατί χρησιμοποιήσαμε υβριδική μέθοδο συμβολικού ελέγχου και χρήση γενετικών αλγορίθμων για την επίλυση των περιορισμών κάθε μονοπατιού.
Το σύστημα που υλοποιήθηκε χρησιμοποιεί βελτιστοποίηση των σύνθετων συνθηκών με αποτέλεσμα τη δημιουργία δεδομένων ελέγχου για όλες τις πιθανούς συνδυασμούς της σύνθετης συνθήκης (θα αναλυθεί περαιτέρω στο κεφάλαιο 4). Αυτό δεν το υποστηρίζει το JCUTE.

Το JCUTE αναλύει αρκετούς τύπους δεδομένων σε αντίθεση με το σύστημα που υλοποιήθηκε το οποίο υποστηρίζει μόνο τους αρχέγονους.

Κεφάλαιο 4

Παρουσίαση Αλγόριθμου

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει εκτενής αναφορά και ανάλυση στα διάφορα συστήματα που έχουν χρησιμοποιηθεί έτσι ώστε να υλοποιηθεί το σύστημα μου. Στο υποκεφάλαιο 4.2 παρουσιάζεται ο Αναλυτής Προγράμματος (BPAS) ο οποίος αναλύει τον κώδικα και δημιουργεί το γράφο ροής ελέγχου. Ακολούθως στο υποκεφάλαιο 4.3 αναφέρεται η Βιβλιοθήκη Barat η οποία έχει χρησιμοποιηθεί για την ανάλυση του γράφου ροής ελέγχου. Στο υποκεφάλαιο 4.4 παρουσιάζεται ο αλγόριθμος.

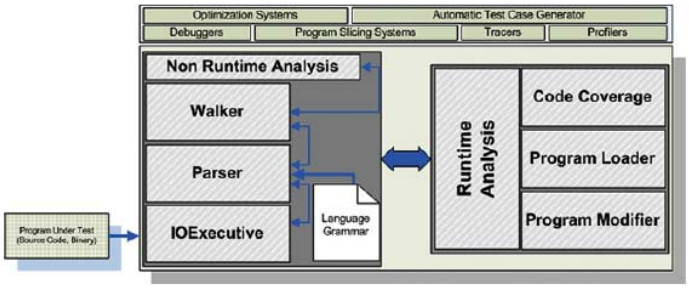
4.2 Αναλυτής Προγράμματος (BPAS)

Το BPAS είναι ένα σύστημα το οποίο αναλύει ένα πρόγραμμα. Το σύστημα αυτό αναφέρθηκε για πρώτη φορά στο έγγραφο [13] όπου χρησιμοποιήθηκε ως επιπλέον σύστημα ανάλυσης προγράμματος. Προς το παρόν είναι υλοποιημένο για να αναλύει προγράμματα γραμμένα σε Java, αλλά με κάποιες τροποποιήσεις μπορεί να χρησιμοποιηθεί και για άλλα προγράμματα τα οποία είναι γραμμένα σε διαφορετικές γλώσσες προγραμματισμού.

Χωρίζεται σε runtime και non-runtime υποσυστήματα τα οποία μπορούν να αναλύσουν μεγάλα και πολύπλοκα προγράμματα. Και τα δύο υποσυστήματα μπορούν να παρέχουν ένα μίγμα πληροφοριών και διαδικασιών για το πρόγραμμα κάτω από τη μελέτη. Κατά συνέπεια, οι εξωτερικές εφαρμογές μπορούν να χρησιμοποιήσουν τη λειτουργία τους για τη λήψη αυτών των πληροφοριών.

Έχει τη δυνατότητα να εκτελέσει στατική ανάλυση ενός προγράμματος, δηλαδή χωρίς να εκτελείτε το πρόγραμμα καθώς και δυναμική εκτέλεση δηλαδή κατά την ώρα που εκτελείται το πρόγραμμα.

Στο Σχήμα 4.1 παρατηρούμε την αρχιτεκτονική του BPAS. Όπως παρατηρούμε και από το σχήμα το BPAS αποτελείται από μερικά επίπεδα IOExecutive, Parser, Walker, Static Analyzer (Non-Runtime Analysis) και Dynamic Analyzer (Runtime Analysis). Το IOExecutive επίπεδο είναι υπεύθυνο για το διάβασμα και το γράψιμο αρχείων Java και και προσδιορίζει τις πληροφορίες για τις μεθόδους, τους τομείς και τις μεταβλητές του προγράμματος χωρίς τη λεπτομερή επεξεργασία ή ανάλυση των δηλώσεών τους. Το Parser επίπεδο παρέχει έναν front-end parser της γλώσσας Java. Το Walker επίπεδο είναι υπεύθυνο για την περιδιάβαση κάθε μέρους του κώδικα καθώς και για την κατασκευή του γράφου ροής ελέγχου (control flow graph). Το Non-Runtime Analysis επίπεδο χρησιμοποιείται για την ανάλυση του προγράμματος και την εξαγωγή της πληροφορίας ότι ισχύουν όλες οι πιθανές λύσεις.



Σχήμα 4.1 – Αρχιτεκτονική BPAS

Η κύριες λειτουργίες του BPAS είναι η εξαγωγή των ουσιαστικών πληροφοριών προγράμματος, η δημιουργία του αντίστοιχου control flow graph [2,14], ο προσδιορισμός κάλυψης κώδικα (code coverage determination) και η δυναμική αξιολόγηση των περιπτώσεων δοκιμής (test cases). Επίσης μπορεί να χρησιμοποιηθεί σε συνδυασμό με προγράμματα βελτιστοποίησης, δημιουργίας περιπτώσεων δοκιμής, τεμαχισμού προγράμματος και άλλα.

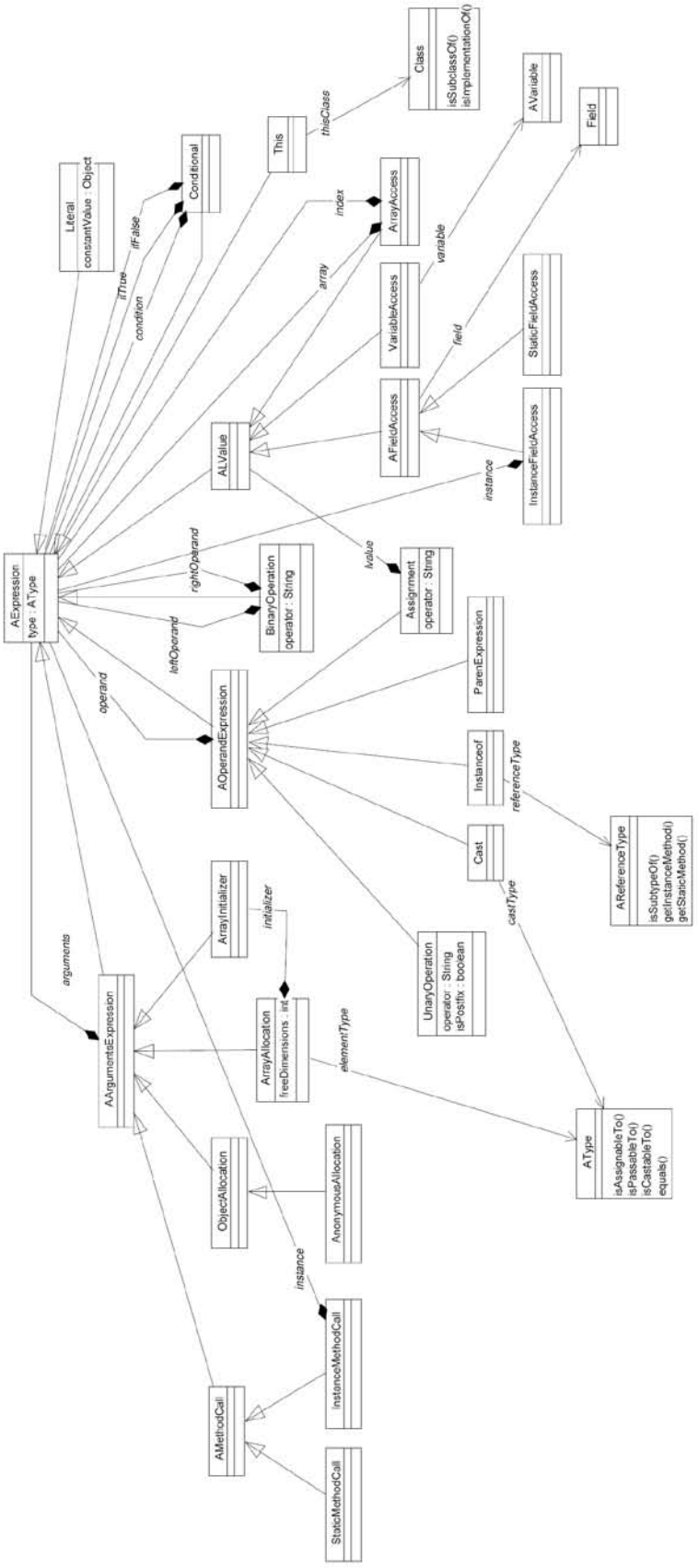
Μπορεί να χρησιμοποιηθεί σε προγράμματα τα οποία κάνουν διόρθωση κώδικα καθώς και έλεγχο. Το BPAS έχει χρησιμοποιηθεί σε αυτόματο σύστημα παραγωγής

περιπτώσεων δοκιμής (Automatic Test Cases Generation System) [4]. Επίσης χρησιμοποιήθηκε για την ανάπτυξη του data flow graph, ο οποίος χτίζεται χρησιμοποιώντας τις πληροφορίες που παρέχονται από το control flow graph του συγκεκριμένου προγράμματος που ελέγχεται [8]. Μία άλλη χρήση ήταν για την δημιουργία ενός συστήματος αυτόματης παραγωγής των βέλτιστων στοιχείων δοκιμής δεδομένου ενός προγράμματος [5]. Το σύστημα αυτό έχει χρησιμοποιηθεί για την αυτόματη δημιουργία γράφου ελέγχου και περιπτώσεις ελέγχου μέσω αλγορίθμων βελτιστοποίησης [15], για τη δημιουργία γράφων εξάρτησης (dependence graph) [16] και για τον αυτοματοποιημένο έλεγχο λογισμικού με βελτιστοποίηση του κριτήριου κάλυψης μονοπατιού [17].

4.3 Βιβλιοθήκη Barat

Η βιβλιοθήκη BARAT [18] υποστηρίζει τη στατική ανάλυση των προγραμμάτων της Java. Χτίζει ένα πλήρες αφηρημένο δέντρο σύνταξης από τον κώδικα του προγράμματος το οποίο είναι υπό έλεγχο. Το δέντρο αυτό εμπλουτίζεται με το όνομα και τις πληροφορίες ανάλυσης της μεταβλητής. Με τη βοήθεια των visitors και των attributes παρέχει τα πρόσθετα χαρακτηριστικά γνωρίσματα. Ουσιαστικά δίνει τη δυνατότητα στους χρήστες να χρησιμοποιούν το δέντρο αυτό και να λαμβάνουν τις αναγκαίες πληροφορίες για κάθε εντολή που βρίσκεται στον ανάλογο κόμβο. Ο χρήστης μπορεί να χειριστεί το όνομα της μεταβλητής, τον τύπο της, την τιμή της καθώς και άλλα χαρακτηριστικά που ίσως χρειαστεί. Το δέντρο μοιάζει αρκετά με ένα δέντρο μεταγλωττιστή όπου όσο πιο μικρό είναι το επίπεδο που βρίσκεσαι τόσο πιο απλοποιημένα και low-level χαρακτηριστικά μπορείς να λάβεις.

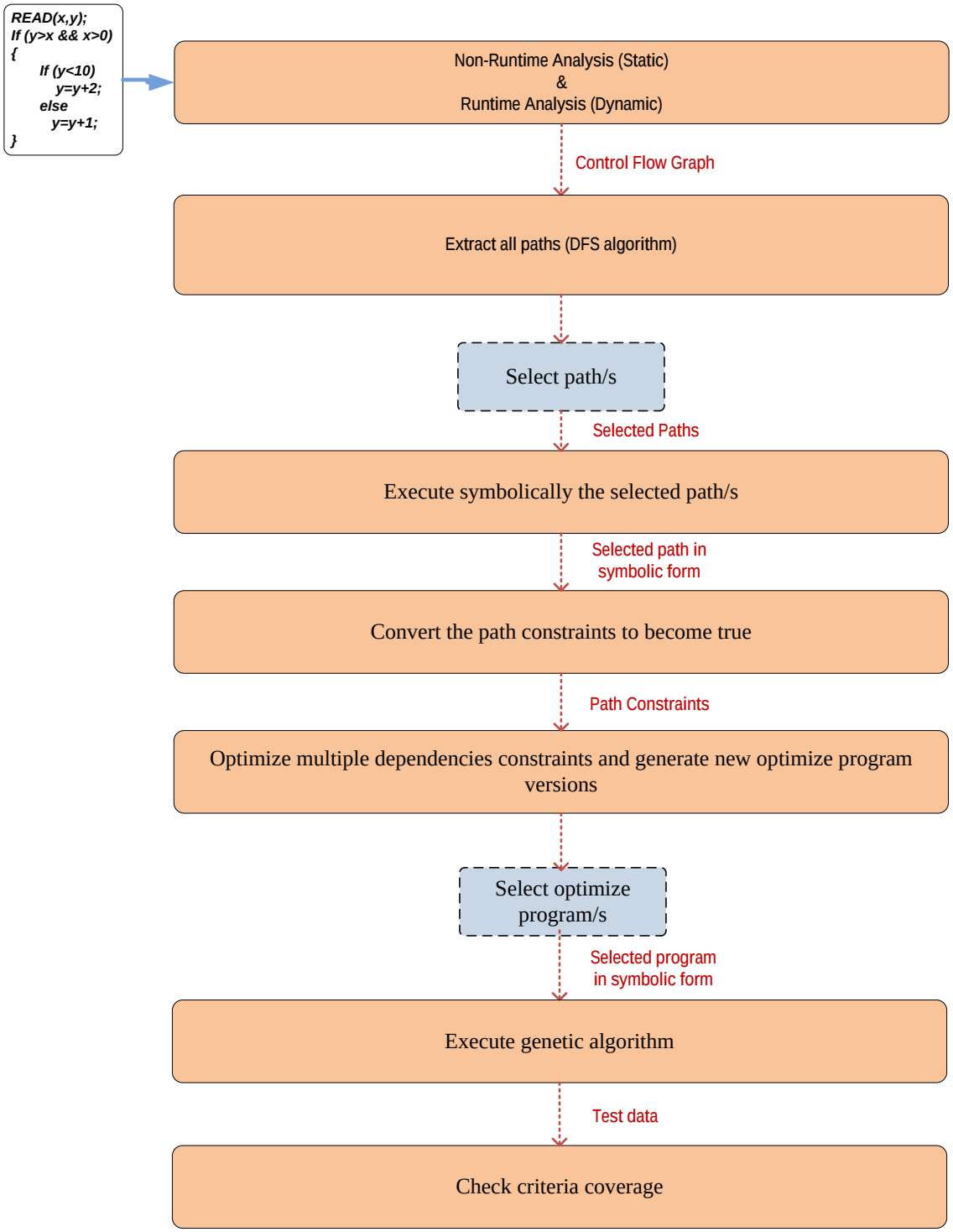
Στο Σχήμα 4.2 παρατηρούμε τον τύπο των εκφράσεων που παρέχει η βιβλιοθήκη BARAT καθώς και τη σχέση μεταξύ τους.



Σχήμα 4.2 Τύποι Εκφράσεων Βιβλιοθήκης Barat

4.4 Ανάλυση αλγόριθμου

Το σχήμα 4.3 παρουσιάζει τον αλγόριθμο που ακολουθήθηκε για την υλοποίηση του συστήματος.



Σχήμα 4.3 – Βήματα Αλγορίθμου

1. Δημιουργία γράφου ελέγχου (create control flow graph)
2. Εξαγωγή όλων των πιθανών μονοπατιών (extract paths)
3. Επιλογή όλων ή συγκεκριμένου μονοπατιού (select path/s)
4. Συμβολική εκτέλεση του/των επιλεγμένου/ων μονοπατιού/ων (execute symbolically the selected path/s)
5. Μετατροπή όλων των path constraints σε true (convert the path constraints to become true)
6. Βελτιστοποίηση των multiple dependencies constraints και δημιουργία βελτιστοποιημένων προγραμμάτων (optimize multiple dependencies constraints and generate new optimize program versions)
7. Επιλογή όλων ή συγκεκριμένου βελτιστοποιημένου προγράμματος (select optimize program/s)
8. Εκτέλεση γενετικού αλγορίθμου πάνω στο/α επιλεγμένο/α βελτιστοποιημένο/α πρόγραμμα/ατα (execute genetic algorithm)
9. Έλεγχος ποσοστού κάλυψης του προγράμματος (check criteria coverage)

Για την καλύτερη κατανόηση κάθε βήματος του αλγορίθμου ακολουθεί μία πιο λεπτομερής ανάλυση.

Δημιουργία γράφου ελέγχου: Με είσοδο ένα πρόγραμμα γραμμένο σε Java, αναλύεται ο κώδικας και δημιουργείται ο γράφος ροής ελέγχου. Η ανάλυση αυτή καθώς και η δημιουργία του γράφου γίνεται εφικτή με την βοήθεια του συστήματος Basic Program Analyzer System (BPAS). Το σύστημα αυτό περιγράφεται λεπτομερώς στην Ενότητα 2 του κεφαλαίου 4. Το κομμάτι αυτό έχει υλοποιηθεί από τον Δρ. Αναστάση Σοφοκλέους [15].

Εξαγωγή όλων των πιθανών μονοπατιών: Βάση του γράφου ροής ελέγχου που έχει δημιουργηθεί από το προηγούμενο βήμα εξάγονται όλα τα πιθανά μονοπάτια από τον αρχικό κόμβο μέχρι το τέλος, $P = \{p_1,\ldots..,p_n\}$όπου n ο αριθμός των παραγόμενων μονοπατιών. Ο αλγόριθμος που χρησιμοποιείται για την εξαγωγή των μονοπατιών είναι ο depth first search (DFS) [19]. Η εξαγωγή των μονοπατιών εξαρτάται από τον πρώτο και τελευταίο κόμβο του αρχικού γράφου.

Επιλογή όλων ή συγκεκριμένου μονοπατιού: Ο χρήστης είτε θα επιλέξει την εκτέλεση όλων των μονοπατιών σειριακά είτε θα επιλέξει ένα συγκεκριμένο. $P_{sel} = \{P_i \mid P_i$: μονοπάτι που επιλέχθηκε$\}$.

Συμβολική εκτέλεση του/των επιλεγμένου/ων μονοπατιού/ων: Συμβολική εκτέλεση για τα επιλεγμένα μονοπάτια, P_{sel}. Αρχικά γίνεται η αντικατάσταση των τοπικών μεταβλητών με τη συμβολική τους τιμή. Η συμβολική τους τιμή είναι η τρέχον τιμή ως προς τις μεταβλητές εισόδου (παραμέτρους). Η εκτέλεση προχωρά κανονικά δηλαδή από πάνω προς τα κάτω με την μετατροπή των τοπικών μεταβλητών βάση της τρέχουσας συμβολικής τους τιμής. Ακολούθως επιλέγονται οι συνθήκες ελέγχου του προγράμματος (path constraints). Η επιλογή των συνθηκών ελέγχου γίνεται με την δημιουργία του γράφου για το συμβολικό πρόγραμμα που δημιουργήθηκε στο προηγούμενο βήμα. Ο συγκεκριμένος γράφος περιέχει μόνο τους περιορισμούς του μονοπατιού λόγω του ότι οι υπόλοιπες πληροφορίες δεν θα είναι χρήσιμες πλέον.

Μετατροπή συγκεκριμένων path constraints σε true: Σε αυτό το βήμα μετατρέπουμε τις ψευδείς συνθήκες σε αληθείς έτσι ώστε να είναι ευκολότερη η ενσωμάτωση του γενετικού αλγορίθμου. Για παράδειγμα

- A && B → ! (A && B) = !A || !B
- A || B → !(A || B) = !A && !B

Οι πιο πάνω είναι απλές περιπτώσεις αλλά είναι η βασική αρχή των μετατροπών πιο σύνθετων συνθηκών.

Βελτιστοποίηση των multiple dependencies constraints και δημιουργία βελτιστοποιημένων προγραμμάτων: Βελτιστοποιούνται οι πολλαπλές συνθήκες ελέγχου (εκείνες που χωρίζονται με τον τελεστή || ή/και &&) και δημιουργούνται νέα βελτιστοποιημένα προγράμματα τα οποία περιλαμβάνουν τις συνθήκες αυτές. Ο τρόπος βελτιστοποίησης εμπνεύστηκε από τον τρόπο που ελέγχει τις συνθήκες η Java. Για παράδειγμα εάν έχουμε A||B||C και θέλουμε να είναι αληθείς καταλήγουμε σε τρεις βέλτιστες περιπτώσεις. Οι περιπτώσεις αυτές είναι:

- να ισχύει το A, (εάν ισχύει το A η Java δεν θα προχωρήσει με τον έλεγχο)

- να μην ισχύει το A και να ισχύει το B (ισχύει το πιο πάνω αλλά στη θέση του A είναι το B)
- να μην ισχύουν τα A και B και να ισχύει το C

Η άλλη περίπτωση είναι αυτή του A && B && C. Αυτή η περίπτωση είναι πιο απλή μιας και πρέπει να ισχύουν και οι τρεις υποσυνθήκες. Αποτέλεσμα της βελτιστοποίησης των multiple dependencies constraints είναι πολλαπλά προγράμματα βάση των πιθανών συνδυασμών που θα προκύψουν.

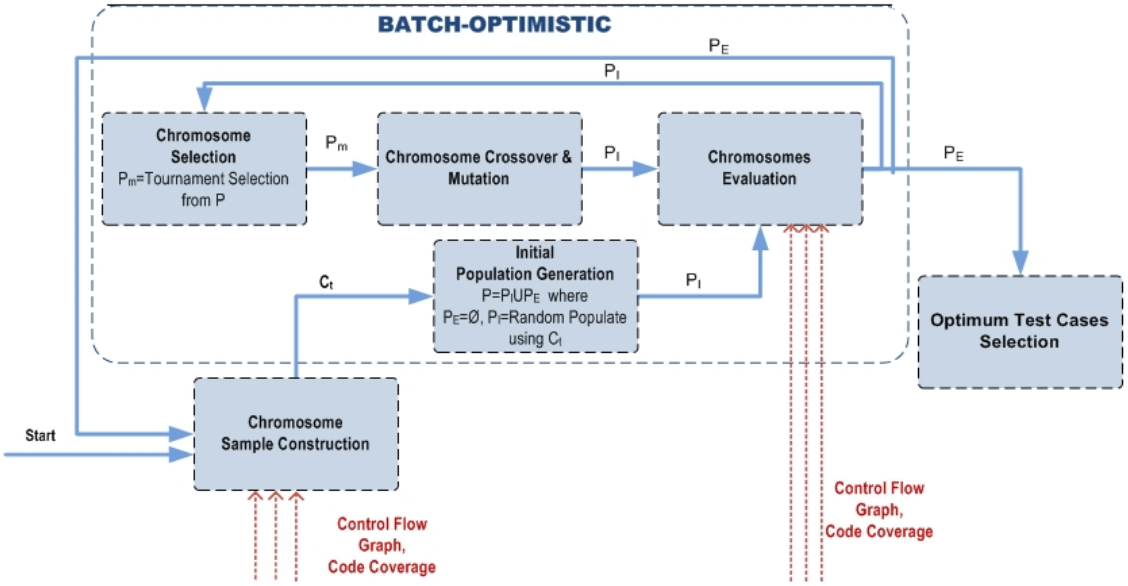
Επιλογή όλων ή συγκεκριμένου βελτιστοποιημένου προγράμματος: Ο χρήστης επιλέγει συγκεκριμένο ή όλα τα βελτιστοποιημένα προγράμματα, πάνω στα οποία θα χρησιμοποιηθεί ο γενικός αλγόριθμος για να εξαχθούν τα δεδομένα ελέγχου. Εάν είχε επιλέξει από πριν να εκτελεστούν όλα τα μονοπάτια τότε δεν θα έχει τη δυνατότητα να κάνει την επιλογή αυτή μιας και θα δημιουργηθούν βέλτιστα προγράμματα για όλα τα μονοπάτια.

Εκτέλεση γενετικού αλγορίθμου πάνω στο/α επιλεγμένο/α βελτιστοποιημένο/α πρόγραμμα/ατα: Για κάθε ένα από τα επιλεγμένα βέλτιστα προγράμματα εκτελείτε ο γενετικός αλγόριθμος ο οποίος παράγει τα δεδομένα ελέγχου για το συγκεκριμένο σύνολο συνθηκών. Εάν ο γενετικός δεν καλύψει όλες τις συνθήκες ελέγχου που περιέχονται στο συγκεκριμένο πρόγραμμα τότε χαρακτηρίζουμε το μονοπάτι αυτό ως νεκρό (dead path). Ο γενετικός αλγόριθμος σταματά όταν όλες οι συνθήκες ικανοποιηθούν. Στο σχήμα 4.4 παρατηρούμε το μοντέλο του γενετικού αλγορίθμου που έχει υλοποιηθεί. Ο γενετικός αλγόριθμος χρησιμοποιεί ως αρχικό πληθυσμό διάφορες τιμές όπου είναι οι πιθανές λύσεις. Κάθε μέλος του πληθυσμού (χρωμόσωμα) αποτελείται από τόσες τιμές όσες και οι παράμετροι του συγκεκριμένου κώδικα. Ακολούθως γίνεται η αξιολόγηση του κάθε μέλους του πληθυσμού βάση της fitness function. Η fitness function που έχει οριστεί είναι:

(1/(αριθμός των συνθηκών + τιμή της συνθήκης))

Ο αριθμός των συνθηκών αντιπροσωπεύει τον αριθμό των περιορισμών που έχουν βρεθεί βάση της συμβολικής εκτέλεσης καθώς και της μετέπειτα βελτιστοποίησης των multiple dependencies constraints. Η τιμή της συνθήκης ορίζεται ως η τιμή βάση της καταλληλότητας του συγκεκριμένου χρωμοσώματος. Για παράδειγμα εάν η συνθήκη

ικανοποιείται τότε η τιμή της συνθήκης θα είναι μηδενική έτσι ώστε να μην μειώνει το αποτέλεσμα από τη fitness.Εάν όμως είναι λανθασμένη και είναι κοντά στην επιθυμητή λύση τότε η τιμή θα είναι 0.01. Τέλος για οποιαδήποτε άλλη τιμή επιστρέφεται η απόλυτη τιμή της διαφοράς του επιθυμητού αποτελέσματος από το τρέχον. Στην περίπτωση συνθήκης ισότητας ή ανισότητας εάν δεν ισχύει τότε η τιμή θα είναι 100 (λόγω του ότι απέχει αρκετά από την επιθυμητή λύση). Στη συνέχεια υπολογίζεται η ρουλέτα βάση της καταλληλότητας του κάθε χρωμοσώματος και ακολούθως η τυχαία επιλογή των μελών όπου βάση κάποιων πιθανοτήτων θα διασταυρωθούν και θα μεταλλαχθούν. Για μία γενεά αθροίζονται όλες οι τιμές καταλληλότητας του κάθε μέλους. Εάν αυτή η τιμή είναι 1 τότε ικανοποιείται το κριτήριο τερματισμού και αποθηκεύουμε το χρωμόσωμα. Με το τέλος των εποχών για ένα συγκεκριμένο μονοπάτι εάν δεν έχει βρεθεί μία λύση τότε το μονοπάτι αυτό θεωρείται «νεκρό». Οι παράμετροι που χρειάζεται ο γενετικός αλγόριθμος όπως ο αριθμός των εξελίξεων, οι τιμές των γενετικών τελεστών και άλλες χρήσιμες πληροφορίες ανακτώνται από ένα αρχείο.



Σχήμα 4.4 – Μοντέλο Γενετικού αλγόριθμου

Πιο κάτω θα παρουσιαστεί ένα παράδειγμα υπολογισμού καταλληλότητας με τη χρήση της fitness function:

Έστω έχουμε ένα σύνολο από περιορισμούς A οι οποίοι απομονώθηκαν από ένα μονοπάτι του κώδικα. Για κάθε ένα περιορισμό ο γενετικός αλγόριθμος θα

προσπαθήσει να τον επιλύσει με κύριο στόχο να βρει τις καταλληλότερες τιμές οι οποίες θα ικανοποιούν όλους τους περιορισμούς.

A = {i > 0, j > 0, k > 0, i + j > k, i = = j}

Z : Τιμή καταλληλότητας

Ο γενετικός αλγόριθμος θα ξεκινήσει με το μέλος Ai. Έστω η τιμή που έχει δημιουργήσει ο γενετικός αλγόριθμος είναι το 1. Τότε η τιμή καταλληλότητας Z θα ισούται με 1/5. Εάν όμως η τιμή ήταν 0 τότε η τιμή καταλληλότητας Z θα ισούται με 1/ 5.01 δηλαδή αρκετά κοντά στην επιθυμητή λύση. Στην περίπτωση που η τιμή είναι το 5 τότε η τιμή καταλληλότητας είναι 1/10.

Έτσι συμπεραίνουμε πως για κάθε ένα περιορισμό το βέλτιστο είναι η τιμή καταλληλότητας να είναι 1/5 έτσι ώστε όταν συναθροιστούν στο τέλος να έχουμε την τιμή 1, δηλαδή τις τιμές των δεδομένων οι οποίες ικανοποιούν το σύνολο Α. Ένα σύνολο δεδομένων ελέγχου το οποίο θα μπορούσε να ικανοποιεί τις συνθήκες του συνόλου Α είναι το {1,1,1 : i,j,k}.

Έλεγχος ποσοστού κάλυψης του προγράμματος : Για όλα τα εφικτά μονοπάτια, δηλαδή τα μονοπάτια στα οποία ο γενετικός βρήκε δεδομένα ελέγχου, ελέγχεται ποιες από τις ακμές και τους περιορισμούς του αρχικού γράφου έχουν καλυφθεί. Συνεπώς εξάγεται και το ποσοστό που έχει καλυφθεί ο γράφος ροής δεδομένων.

4.4.1 Παραδείγματα εκτέλεσης αλγορίθμου

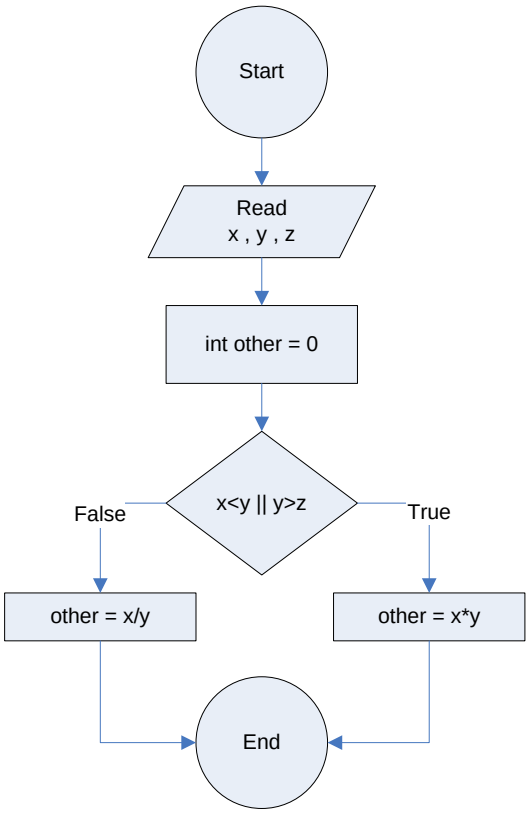
Πιο κάτω θα παρουσιαστούν δύο παραδείγματα προγραμμάτων κώδικα έτσι ώστε να αναλυθεί περαιτέρω η λειτουργία του αλγορίθμου.

Παράδειγμα 1:

```
public class test1 {
  public void methodA(int x, int y, int z)
{
    int other = 0;

    if ((x < y) || (y>z)) {
      other = x / y;
    }
    else
    {
      other = x * y;
    }
  }
}
```

Βάση του πιο πάνω προγράμματος το σύστημα θα δημιουργήσει τον γράφο ροής ελέγχου. Ο γράφος παρουσιάζεται στο σχήμα 4.5.



Σχήμα 4.5 – Γράφος ροής ελέγχου

Ακολούθως θα εξάγει όλα τα πιθανά μονοπάτια. Στην περίπτωση αυτή υπάρχουν δύο μονοπάτια. Για κάθε ένα μονοπάτι θα εκτελεστεί η συμβολική εκτέλεση και θα απομονωθούν οι περιορισμοί του μονοπατιού. Οι περιορισμοί για κάθε μονοπάτι είναι:

1. `(x < y) || (y > z) (true)`
2. `(x >= y) && (y <= z) (false);`

Στη συνέχεια γίνεται η βελτιστοποίηση των περιορισμών του κάθε μονοπατιού. Για το πρώτο μονοπάτι οι βελτιστοποιημένες συνθήκες είναι:

- `(x >= y) && (y > z)`
- `(x < y);`

Για το δεύτερο μονοπάτι οι βελτιστοποιημένες συνθήκες είναι:

- `(x >= y) && (y <= z);`
- `(x < y);`

Ακολούθως θα τρέξει ο γενετικός αλγόριθμος ο οποίος θα δημιουργήσει τα βέλτιστα δεδομένα ελέγχου:

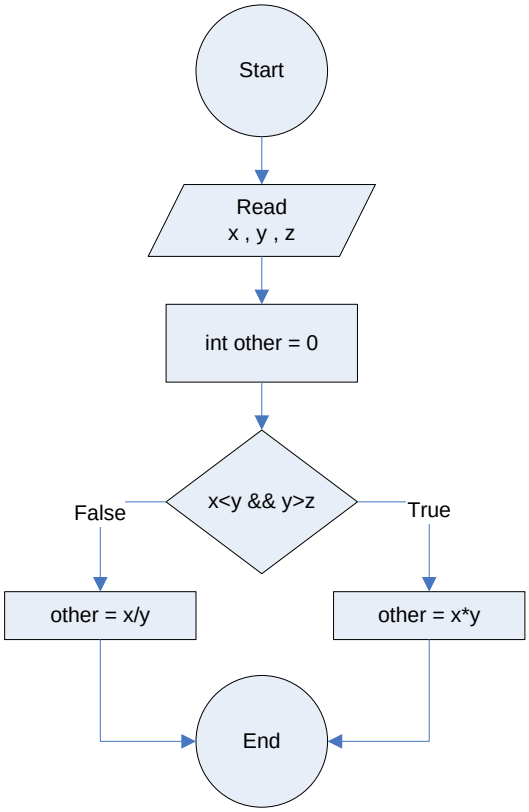
```
x: 4        x: 2        x: 4
y: 3        y: 4        y: 1
z: 0        z: 4        z: 2
```

Παράδειγμα 2:

```
public class test2 {
  public void methodA(int x, int y, int z)
{
    int other = 0;

    if ((x < y) && (y>z)) {
      other = x / y;
    }
    else
    {
      other = x * y;
    }
  }
}
```

Βάση του πιο πάνω προγράμματος το σύστημα θα δημιουργήσει τον γράφο ροής ελέγχου. Ο γράφος παρουσιάζεται στο σχήμα 4.6.



Σχήμα 4.6 – Γράφος ροής ελέγχου

Ακολούθως θα εξάγει όλα τα πιθανά μονοπάτια. Στην περίπτωση αυτή υπάρχουν δύο μονοπάτια. Για κάθε ένα μονοπάτι θα εκτελεστεί η συμβολική εκτέλεση και θα απομονωθούν οι περιορισμοί του μονοπατιού. Οι περιορισμοί για κάθε μονοπάτι είναι:

1. `(x < y) && (y > z) (true)`
2. `(x >= y) || (y <= z) (false);`

Στη συνέχεια γίνεται η βελτιστοποίηση των περιορισμών του κάθε μονοπατιού. Για το πρώτο μονοπάτι οι βελτιστοποιημένες συνθήκες είναι:

- `(x < y) && (y > z)`

Για το δεύτερο μονοπάτι οι βελτιστοποιημένες συνθήκες είναι:

- `(x < y) && (y <= z)`
- `(x >= y)`

Ακολούθως θα τρέξει ο γενετικός αλγόριθμος ο οποίος θα δημιουργήσει τα βέλτιστα δεδομένα ελέγχου:

```
x: 0        x: -1       x: 4
```

```
y: 4        y: 4        y: 4
z: 0        z: 4        z: 2
```

Και στα δύο παραδείγματα το condition coverage είναι 100%. Επίσης και το edge coverage είναι 100%. Από αυτό συμπεραίνουμε πως και το υβριδικό κριτήριο condition/ edge coverage είναι και αυτό 100%. Έτσι ο έλεγχος κρίνεται επαρκής καθώς και τα παραγόμενα δεδομένα επιτυχή.

Κεφάλαιο 5

Πειράματα

5.1 Εισαγωγή

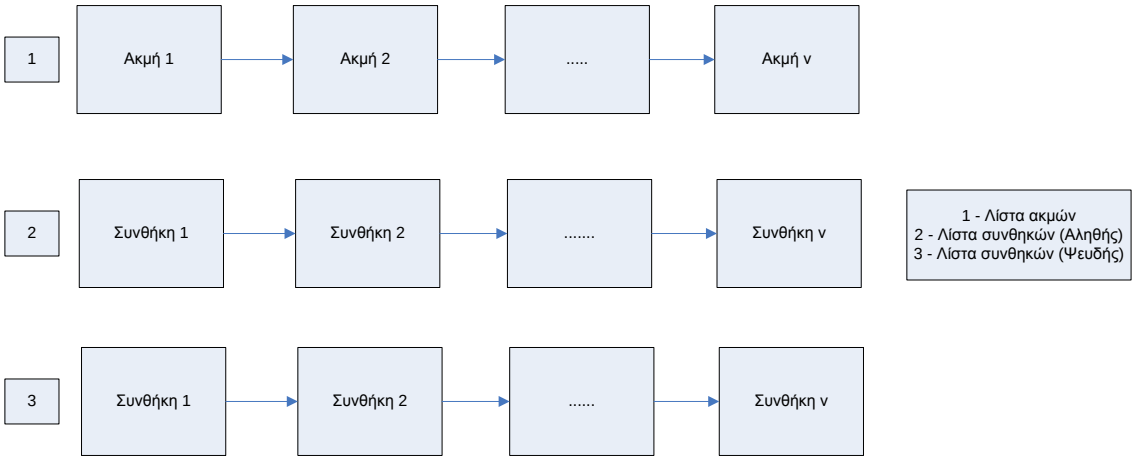
Σε αυτό το κεφάλαιο θα εξηγηθεί η μεθοδολογία που ακολουθήθηκε για την συλλογή αποτελεσμάτων και ελέγχου του συστήματος. Ακολούθως θα παρουσιαστούν διάφορα πειράματα όπου θα έχουν ως είσοδο προγράμματα γραμμένα σε Java, τα οποία θα εκτελεστούν με το σύστημα μας καθώς και με το JCUTE. Τέλος θα γίνει μια σύγκριση των αποτελεσμάτων.

5.2 Μεθοδολογία

Η μεθοδολογία που ακολουθήθηκε είναι η εξής:

Με την εκτέλεση του αλγορίθμου που επεξηγήθηκε στο κεφάλαιο 4, για όλα τα μονοπάτια, αποθηκεύουμε τα δεδομένα ελέγχου που έχουν επιτυχώς επιλύσει τους περιορισμούς κάθε μονοπατιού. Ακολούθως βάση αυτών των τιμών θα γίνεται ο έλεγχος των συνθηκών του αρχικού γράφου που ικανοποιούνται, καθώς και οι ακμές οι οποίες θα διαπεραστούν. Η υλοποίηση της πιο πάνω μεθοδολογίας χρειάζεται τρεις λίστες. Στην πρώτη λίστα αποθηκεύονται οι ακμές τις οποίες διαπερνά το κάθε σύνολο δεδομένων ελέγχου. Εάν πάνω από ένα δεδομένο ελέγχου διαπερνά την ίδια ακμή, η ακμή αυτή θα εμφανίζεται στη λίστα μόνο μία φορά. Ο λόγος είναι πως το ζητούμενο είναι ότι η ακμή αυτή έχει διαπεραστεί τουλάχιστον μία φορά και όχι όσες φορές διαπεράστηκε. Η δεύτερη λίστα αποθηκεύει τις συνθήκες οι οποίες ικανοποιήθηκαν ως προς την αληθή τους πλευρά και η τρίτη λίστα αποθηκεύει τις συνθήκες που ικανοποιήθηκαν ως προς τη ψευδή τους πλευρά. Η εμφάνιση των συνθηκών στις δύο

λίστες είναι μοναδική όπως και στην πρώτη για του ίδιου λόγους οι οποίοι επεξηγήθηκαν και προηγουμένως. Στο σχήμα 5.1 βλέπουμε σχηματικά τις τρεις λίστες.



Σχήμα 5.1 Λίστες Ακμής και Συνθηκών

Με την πιο πάνω μεθοδολογία μπορούμε να υπολογίσουμε το κριτήριο κάλυψης ακμών καθώς και το κριτήριο κάλυψης συνθηκών.

Υπολογισμός κριτηρίου κάλυψης ακμών:

Για να υπολογίσουμε το κριτήριο κάλυψης ακμών πρέπει να γνωρίζουμε τον αριθμό των ακμών που έχουν διαπεράσει τα δεδομένα ελέγχου, καθώς και το συνολικό αριθμός των ακμών του αρχικού γράφου. Ο αριθμός των ακμών που έχουν διαπεραστεί είναι το μέγεθος της λίστας των ακμών και ο αριθμός των ακμών του αρχικού γράφου μπορούμε να τον πάρουμε από το πρώτο βήμα του αλγορίθμου όπου έχει δημιουργηθεί ο αρχικός γράφος. Ο λόγος που ελέγχουμε τα δεδομένα ελέγχου στον αρχικό γράφο είναι για σκοπούς ορθότητας μιας και οι περιορισμοί που αρχικά απομονώνονται περνούν από το στάδιο της συμβολικής εκτέλεσης και της βελτιστοποίησης. Έτσι ο έλεγχος πρέπει να γίνει στον αρχικό γράφο όπου είναι η αναπαράσταση του προγράμματος που είχαμε από τη αρχή ως στόχο να ελέγξουμε. Η εξίσωση για το κριτήριο αυτό είναι:

Ποσοστό κάλυψης = μέγεθος λίστας ακμών / αριθμός ακμών αρχικού γράφου

Υπολογισμός κριτηρίου κάλυψης συνθηκών:

Λόγω του ότι γίνεται κάποια επεξεργασία έτσι ώστε να βελτιστοποιηθούν οι συνθήκες κάθε μονοπατιού, μπορούμε να αναφερθούμε σε κριτήριο κάλυψης πολλαπλών συνθηκών. Για να υπολογίσουμε το κριτήριο αυτό πρέπει να γνωρίζουμε τον αριθμό των συνθηκών οι οποίες έχουν καλυφθεί ως προς την αληθή τους πλευρά καθώς και τον αριθμό των συνθηκών οι οποίες έχουν καλυφθεί από την ψευδή τους πλευρά. Επίσης πρέπει να γνωρίζουμε και τον αριθμό των συνθηκών του αρχικού γράφου. Η μεθοδολογία είναι πανομοιότυπη με την πιο πάνω, όπου δηλαδή ο έλεγχος με την χρήση των δεδομένων ελέγχου γίνεται στον αρχικό γράφο. Ο αριθμός των συνθηκών οι οποίες έχουν καλυφθεί ως προς την αληθή τους πλευρά είναι το μέγεθος της λίστας συνθηκών (αληθής) και ο αριθμός των συνθηκών οι οποίες έχουν καλυφθεί ως προς την ψευδή τους πλευρά είναι το μέγεθος της λίστας συνθηκών (ψευδής). Η εξίσωση για το κριτήριο αυτό είναι:

Ποσοστό κάλυψης = (μέγεθος λίστας συνθηκών (αληθής) + μέγεθος λίστας συνθηκών (ψευδής)) / 2 * αριθμό συνθηκών αρχικού γράφου.

Υπολογισμός κριτηρίου κάλυψης συνθηκών/ακμών

Με τον συνδυασμό των δύο πιο πάνω κριτηρίων μπορούμε να μετρήσουμε το υβριδικό κριτήριο κάλυψης συνθήκης/ακμής. Η εξίσωση για το κριτήριο αυτό είναι:

Ποσοστό κάλυψης = (Ποσοστό κάλυψης κριτηρίου ακμών + Ποσοστό κάλυψης κριτηρίου συνθηκών) / 2

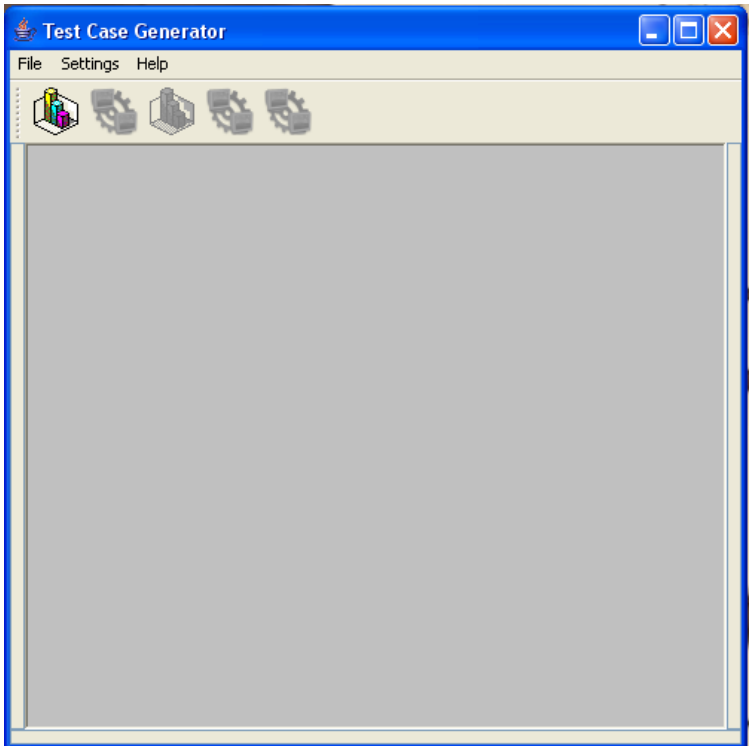
5.4 Περιγραφή Εφαρμογής

Ο χρήστης για να μπορέσει να τρέξει την εφαρμογή θα πρέπει να ορίσει κάποιες παραμέτρους. Μερικοί παράμετροι είναι το μονοπάτι που είναι αποθηκευμένο το πρόγραμμα που θα ελεγχθεί, ο αριθμός των επαναλήψεων του γενετικού αλγορίθμου, η συχνότητα της διασταύρωσης, η συχνότητα της μετάλλαξης, ο αριθμός του πληθυσμού και άλλα. Ο καθορισμός των παραμέτρων αυτών μπορεί να γίνει από ένα αρχείο XML με το όνομα TGSettings.

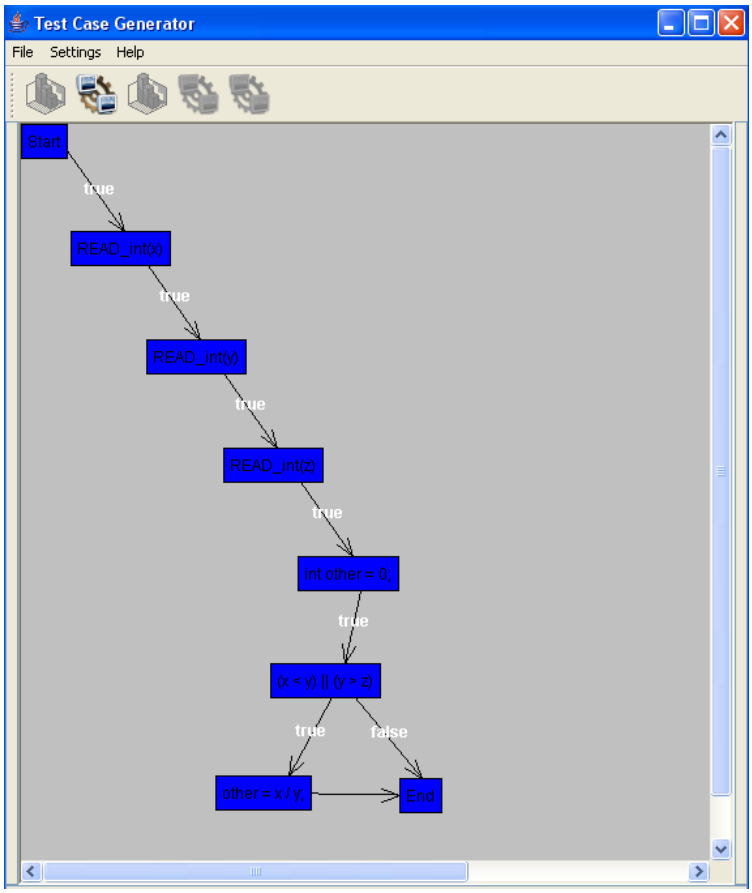
Οι βασικές λειτουργίες της εφαρμογής βρίσκονται στο δεύτερο μενού το οποίο αποτελείται από πέντε κουμπιά. Η εκτέλεση των λειτουργιών πρέπει να γίνεται σειριακά από τα αριστερά στα δεξιά.

Το πρώτο κουμπί αναφέρεται στη δημιουργία του γράφου ροής ελέγχου. Με τη δημιουργία του γράφου, εμφανίζεται σχηματικά στην οθόνη και ενεργοποιείται η επιλογή της δεύτερης λειτουργίας. Το δεύτερο κουμπί αναφέρεται στην επιλογή ενός ή όλων των μονοπατιών και τη συμβολική εκτέλεσή τους. Εάν έχουν επιλεχθεί όλα τα μονοπάτια τότε θα ακολουθήσουν χωρίς την υπόδειξη του χρήστη όλες οι λειτουργίες μέχρι την εμφάνιση των αποτελεσμάτων. Εάν όμως επιλεχθεί μόνο ένα μονοπάτι τότε μετά την επιλογή της δεύτερης επιλογής θα εμφανιστεί στην οθόνη υπό τη μορφή γράφου το σύνολο των περιορισμών που έχουν απομονωθεί μετά την συμβολική εκτέλεση. Το τρίτο κουμπί αναφέρεται στην βελτιστοποίηση του συνόλου των περιορισμών που έχει δημιουργηθεί από την προηγούμενη λειτουργία. Αποτέλεσμα της λειτουργίας αυτής είναι η δημιουργία πιο μικρών προγραμμάτων τα οποία περιέχουν όλους τους συνδυασμούς των βέλτιστων περιορισμών. Το τέταρτο κουμπί αναφέρεται στη λειτουργία όπου ο χρήστης πρέπει να επιλέξει ένα ή όλα τα βέλτιστα προγράμματα που έχουν δημιουργηθεί τα οποία θα τρέξει ο γενετικό αλγόριθμος για να δημιουργήσει τα κατάλληλα δεδομένα ελέγχου. Το πέμπτο και τελευταίο κουμπί αναφέρεται στη λειτουργία των αποτελεσμάτων. Βάση των δεδομένων ελέγχου που έχει δημιουργήσει ο γενετικός αλγόριθμος από τη προηγούμενη λειτουργία και τον αρχικό γράφο μετριέται το ποσοστό κάλυψης του κριτηρίου ακμών, του κριτηρίου συνθηκών καθώς και ο συνδυασμός των δύο.

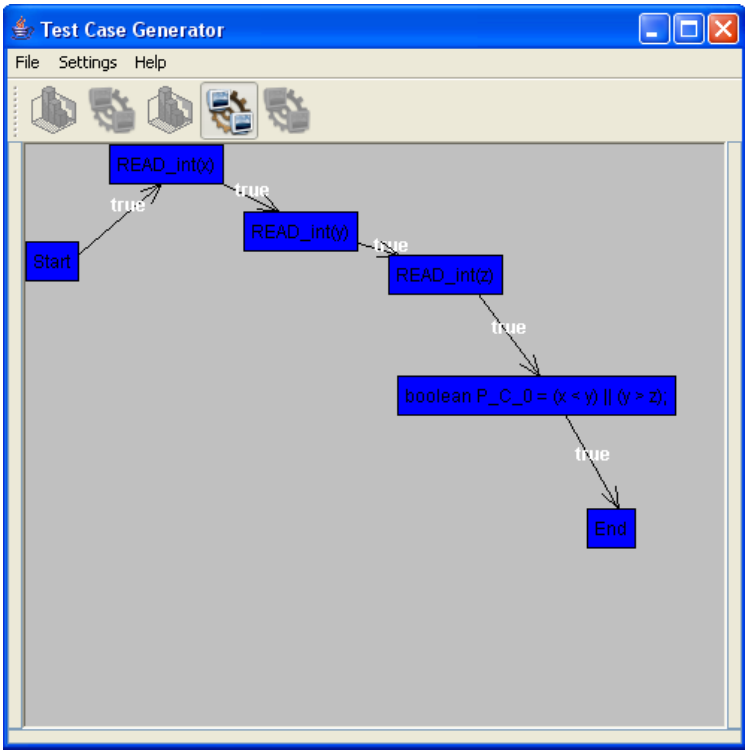
Πιο κάτω ακολουθούν μερικά screenshots για την παρουσίαση της εφαρμογής. Το σχήμα 5.2 αναφέρεται στο κεντρικό μενού της εφαρμογής, το σχήμα 5.3 στην παρουσίαση του γράφου ροής ελέγχου, το σχήμα 5.4 στην παρουσίαση του συνόλου των περιορισμών για το αληθές μονοπάτι υπό τη μορφή γράφου και το σχήμα 5.5 στην παρουσίαση του συνόλου των περιορισμών για το ψευδές μονοπάτι υπό τη μορφή γράφου



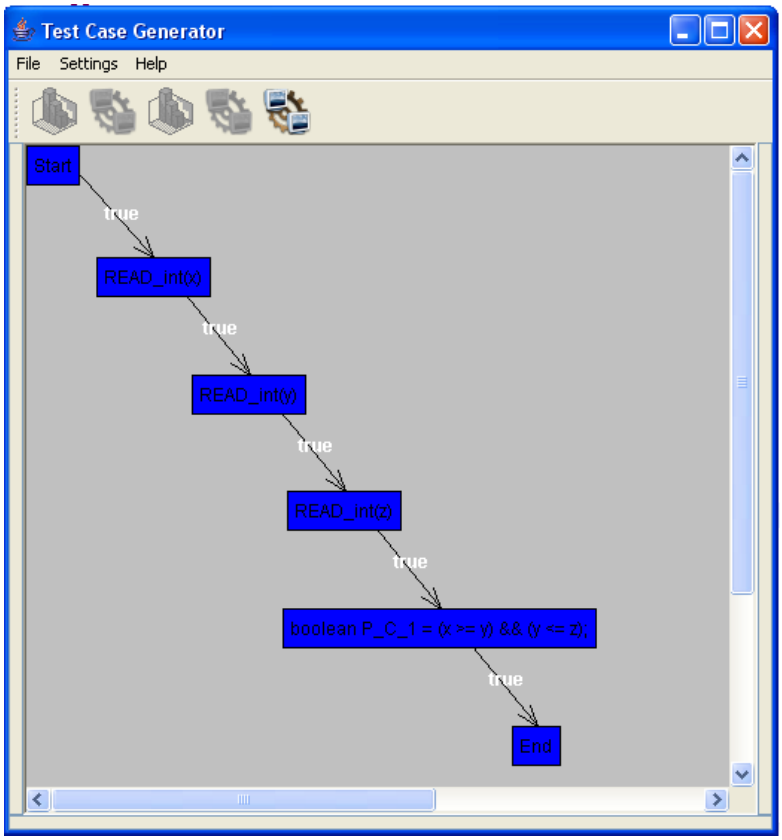
Σχήμα 5.2 Αρχικό μενού εφαρμογής



Σχήμα 5.3 Παρουσίαση γράφου ροής ελέγχου



Σχήμα 5.4 Σύνολο περιορισμών για το αληθές μονοπάτι



Σχήμα 5.5 Σύνολο περιορισμών για το ψευδές μονοπάτι

5.3 Αποτελέσματα και σύγκριση με JCUTE

Σε αυτό το υποκεφάλαιο θα παρουσιαστούν τα αποτελέσματα της εφαρμογής που έχει υλοποιηθεί καθώς και του είδη υλοποιημένου συστήματος JCUTE. Τα δύο αυτά συστήματα προσπαθούν αυτοματοποιημένα να δημιουργήσουν κατάλληλα δεδομένα ελέγχου έτσι ώστε να γίνεται επαρκής έλεγχος. Ο έλεγχος έχει γίνει τόσο για υπαρκτά προγράμματα τα οποία επιλύουν κάποιο πρόβλημα καθώς και για προγράμματα τα οποία έχουν δημιουργηθεί αποκλειστικά για τον έλεγχο της εφαρμογής. Λόγω του ότι δεν έχουν κάποια σημασιολογία σε μερικά μπορεί να μην έχουμε 100% κάλυψη λόγω πιθανών «νεκρών» μονοπατιών που υπάρχουν.

	Triangle Classification			
	#Paths	Edge coverage	Condition coverage	Edge/Condition coverage
JCUTE	4	15.78%	14.7%	15.24%
Our app	57	100%	100%	100%

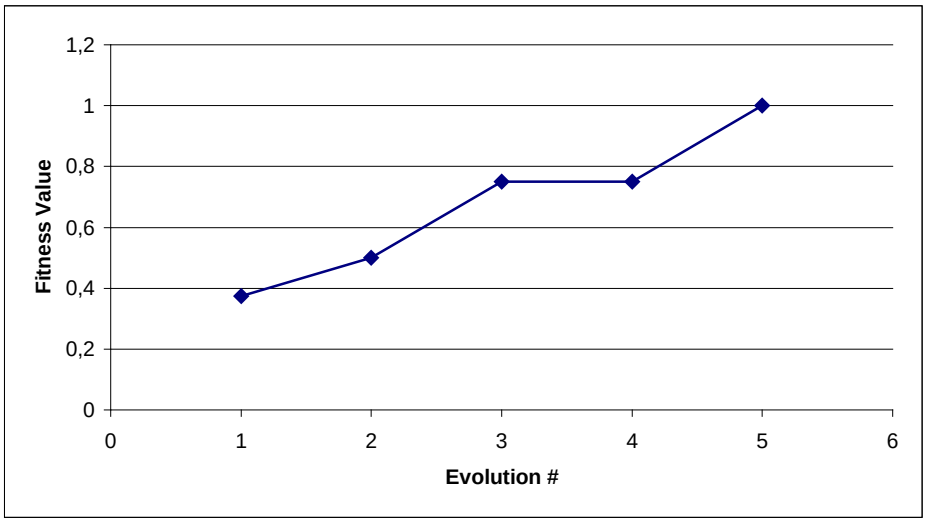
	Find Maximum Number			
	#Paths	Edge coverage	Condition coverage	Edge/Condition coverage
JCUTE	3	85.71%	75%	80.35%
Our app	4	100%	100%	100%

A/A	# predicated in multiple conditions	Complexity	# TC	# Path	Edge coverage	Condition coverage	Edge/ Condition coverage
1	1||	S	3	2	100%	100%	100%
	1||	S	3	2	100%	100%	100%
2	1&&	S	3	2	100%	100%	100%
	1&&	S	3	2	100%	100%	100%
3	2&&	M	6	4	100%	100%	100%
	2&&	M	4	4	100%	100%	100%
4	2||	M	7	4	100%	100%	100%
	2||	M	3	3	81.25%	75%	78.125%
5	2&&, 2||	H	8	4	84.6%	100%	92.3%
	2&&, 2||	H	4	4	100%	100%	100%
6	3(2&&, 1||)	VH	4	8	100%	100%	100%
	3(2&&, 1||)	VH	5	5	100%	100%	100%
7	3(3 &&)	VH	4	8	85%	100%	92.5%
	3(3 &&)	VH	5	5	100%	94.44%	97.2%
8	3(2||, 1&&)	VH	5	8	100%	100%	100%
	3(2||, 1&&)	VH	5	5	100%	100%	100%
9	3(2||, 2&&)	VH	5	8	100%	100%	100%
	3(2||, 2&&)	VH	7	7	100%	100%	100%
10	0	S	1	2	75%	50%	62.5%
	0	S	1	1	75%	50%	62.5%

Οι παράμετροι που έχουν οριστεί για την εκτέλεση των πιο πάνω προγραμμάτων είναι οι εξής:

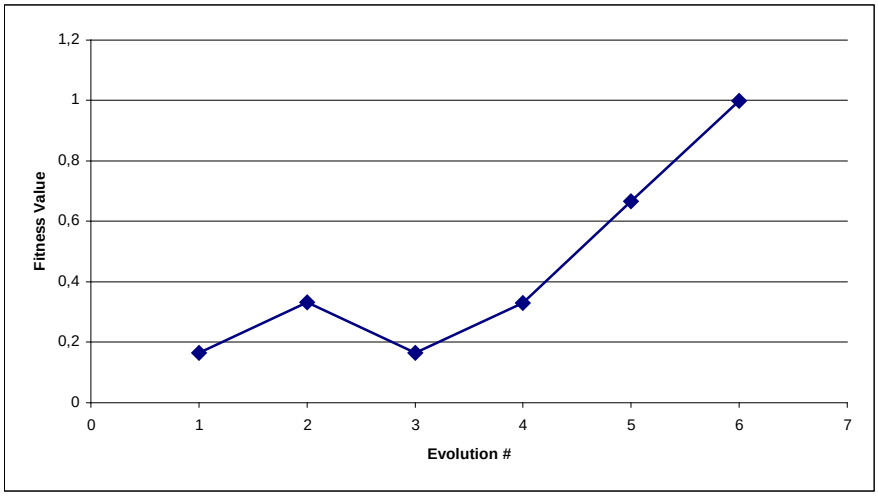
- Αριθμός επαναλήψεων (evolutions): 500
- Συχνότητα διασταύρωσης 100
- Συχνότητα μετάλλαξης 100
- Χρησιμοποιείται Elitism κατά τη διάρκεια εκτέλεσης του γενετικού αλγορίθμου
- Επιλογή κατάλληλων ατόμων με τη μέθοδο της ρουλέτας
- Μέγεθος πληθυσμού: 200
- Ελάχιστος αριθμός πληθυσμού: 50

Στη γραφική παράσταση 6.1 παρουσιάζεται η εξέλιξη του γενετικού αλγορίθμου ως προς το χρόνο βάσει ενός μονοπατιού . Σε αυτή τη γραφική ελέγχθηκε το έκτο μονοπάτι του προγράμματος TriangClassification.java και ποιο συγκεκριμένα το τρίτο βέλτιστο πρόγραμμα που έχει δημιουργήσει η εφαρμογή. Στον άξονα των χ περιγράφεται ο αριθμός της εξέλιξης (evolution number) και στον άξονα των ψ η τρέχον καλύτερη τιμή.



Γραφική Παράσταση 6.1 Έλεγχος με τη χρήση βελτιστοποίησης πολλαπλών συνθηκών

Στη γραφική παράσταση 6.2 παρουσιάζεται η εξέλιξη του γενετικού αλγορίθμου ως προς το χρόνο βάσει ενός μονοπατιού. Σε αυτή τη γραφική ελέγχθηκε το έκτο μονοπάτι του προγράμματος TriangClassification.java το οποίο δεν έχει αναλυθεί, δηλαδή να βελτιστοποιηθεί ως προς τις πολλαπλές συνθήκες. Στον άξονα των χ περιγράφεται ο αριθμός της εξέλιξης (evolution number) και στον άξονα των ψ η τρέχον καλύτερη τιμή.

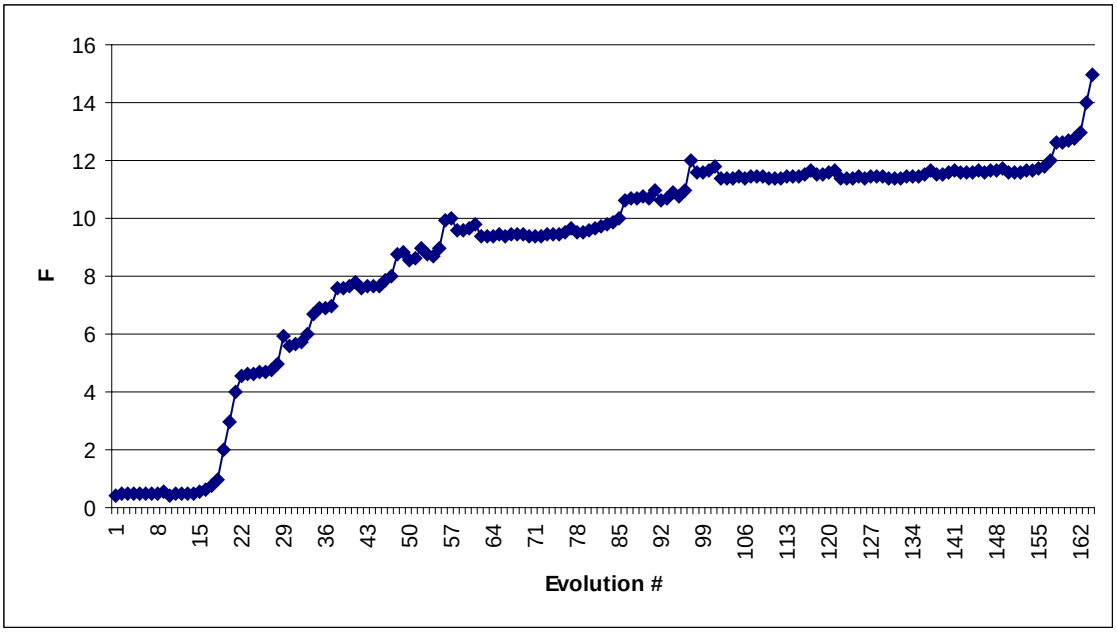


Γραφική Παράσταση 6.2 Έλεγχος χωρίς τη χρήση βελτιστοποίησης πολλαπλών συνθηκών

Παρατηρείται μεταξύ των δύο πιο πάνω γραφικών παραστάσεων πως με τη βελτιστοποίηση των πολλαπλών συνθηκών επιτυγχάνεται γρηγορότερη κάλυψη των περιορισμών του συγκεκριμένου μονοπατιού. Αυτό οφείλεται στο ότι με τη βελτιστοποίηση των πολλαπλών συνθηκών επιτυγχάνεται η απλούστευση τους .

Στη γραφική παράσταση 6.3 παρουσιάζεται η εξέλιξη του γενετικού αλγορίθμου ως προς το χρόνο βάσει ενός προγράμματος, δηλαδή ενός συνόλου από μονοπάτια. Σε αυτή τη γραφική ελέγχθηκε το έκτο μονοπάτι του προγράμματος TriangClassification. Στον άξονα των χ περιγράφεται ο αριθμός της εξέλιξης (evolution number) και στον άξονα των ψ ο αριθμός των δεδομένων ελέγχου τα οποία έχουν επιλύσει ένα μονοπάτι στον οποίο προσθέτετε και η τρέχον της παραγόμενης περίπτωσης δοκιμής η οποία ελέγχει το τρέχον μονοπάτι το οποίο ελέγχεται. Για το πρόγραμμα αυτό έχουν

δημιουργηθεί 15 περιπτώσεις δοκιμής. Για τη δημιουργία των περιπτώσεων δοκιμής χρειάστηκαν 163 εξελίξεις (evolutions). Με αυτά δεδομένα ελέγχου καλύπτονται όλοι οι περιορισμοί των εφικτών μονοπατιών του συγκεκριμένου προγράμματος.



Γραφική Παράσταση 6.3 Έλεγχος ενός συνόλου μονοπατιών

Κεφάλαιο 6

Συμπεράσματα και Μελλοντική Εργασία

6.1 Εισαγωγή

Στο κεφάλαιο αυτό θα αναφερθούν κάποιοι περιορισμοί που υπάρχουν στην εφαρμογή καθώς και τα συμπεράσματα από την εργασία αυτή. Στο τέλος θα υπάρξουν κάποιες ιδέες και παροτρύνσεις έτσι ώστε να βελτιστοποιηθεί τόσο η υπάρχουσα υλοποίηση καθώς και παρόμοιες υλοποιήσεις (μελλοντικές μελέτες).

6.2 Περιορισμοί

Σκοπός της εργασίας αυτής είναι η αυτοματοποιημένη δημιουργία περιπτώσεων δοκιμής με την χρήση μίας υβριδικής μεθόδου η οποία συνδυάζει συμβολική εκτέλεση και γενετικό αλγόριθμο. Είναι γεγονός πως οι δύο αυτές μεθοδολογίες έχουν αρκετά πλεονεκτήματα αλλά και μειονεκτήματα. Βάση αυτών των μειονεκτημάτων δημιουργούνται κάποιοι περιορισμοί οι οποίοι είναι καλό να αναφερθούν.

Η συμβολική εκτέλεση παρά τα όσα θετικά έχουν προαναφερθεί έχει τους δικούς της περιορισμούς. Όταν η πολυπλοκότητα του γράφου είναι αρκετά μεγάλη τότε ο αριθμός των μονοπατιών είναι υπερβολικά μεγάλος με αποτέλεσμα ο έλεγχος να διαρκεί αρκετή ώρα. Επίσης εάν το μονοπάτι είναι αρκετά πολύπλοκο, τότε οδηγούμαστε σε ένα σύνολο πολύπλοκων εξισώσεων οι οποίες είναι δύσκολα να επιλυθούν.

Η χρήση των γενετικών αλγορίθμων για την επίλυση του συνόλου των περιορισμών για κάθε μονοπάτι είναι αρκετά αποδοτική και τις πλείστες φορές επιτυχής. Παρά τα τόσα πλεονεκτήματα που μας προσφέρουν δεν παύουν να έχουν και τις αδυναμίες τους. Καταρχάς λόγω του ότι είναι πιθανοθεωρητικοί, δεν εγγυώνται πως πάντα θα βρίσκουν την βέλτιστη λύση. Υπάρχουν και άλλες αδυναμίες οι οποίες αντιμετωπίστηκαν όπως φαίνεται και στο κεφάλαιο 3.

Η εφαρμογή υποστηρίζει μόνο αρχέγονου τύπους και όχι σύνθετους τύπους δεδομένων. Επίσης δεν υποστηρίζει εντολές επανάληψης.

Παρά τις πιο πάνω αδυναμίες τα θετικά που έχουν οι μεθοδολογίες αυτές υπερτερούν των αδυναμιών τους. Η υβριδική μέθοδος που υλοποιήθηκε ξεπέρασε αρκετές αδυναμίες και πρόβαλε τα θετικά των δύο μεθοδολογιών. Αυτό το συμπεραίνουμε από τα αποτελέσματα του κεφαλαίου 5.

6.3 Συμπεράσματα

Ο έλεγχος των λογισμικών είναι μία αρκετά σημαντική λειτουργία για την δημιουργία ορθών και αξιόπιστων λογισμικών τα οποία θα χρησιμοποιηθούν έτσι ώστε να μας διευκολύνουν τη ζωή. Επίσης λόγο του ότι τα λογισμικά είναι πλέον αρκετά πολύπλοκα πρέπει να δημιουργηθούν κατάλληλες μέθοδοι ελέγχου έτσι ώστε να εκτελούν επαρκή έλεγχο.

Συνοπτικά ο κύριος στόχος αυτής της έρευνας ήταν η δημιουργία ενός συστήματος το οποίο αυτοματοποιημένα θα δημιουργεί δεδομένα ελέγχου. Η μεθοδολογία που έχει ακολουθηθεί είναι ο συμβολικός έλεγχος για την μετατροπή των μονοπατιών σε ένα σύνολο από περιορισμών και η επίλυσή τους με τη χρήση ενός γενετικού αλγορίθμου. Βάση των αποτελεσμάτων που υπάρχουν στο κεφάλαιο 5, διαφαίνεται η αποτελεσματικότητα του πιο πάνω αλγορίθμου ως προς τον έλεγχο ολόκληρου ή αρκετά μεγάλου ποσοστού του κώδικα. Τα δεδομένα ελέγχου που παράγονται μπορούν να δώσουν τη δυνατότητα στο χρήστη να επαληθεύσει την ορθότητα του ελέγχου.

Το σύστημα αυτό παρέχει αρκετά πλεονεκτήματα στο χρήστη:

- Σχηματική απεικόνιση του πηγαίου κώδικα του λογισμικού εισόδου υπό τη μορφή γράφου του κώδικα για την καλύτερη οπτική επαφή του χρήστη καθώς

και για την μετέπειτα επεξεργασία του πηγαίου κώδικα για τη εξαγωγή των δεδομένων ελέγχου

- Βελτιστοποίηση των σύνθετων συνθηκών έτσι ώστε ο έλεγχος να γίνεται για όλες τις πιθανές περιπτώσεις έτσι ώστε να καλύπτει όσο το δυνατό περισσότερο κομμάτι του κώδικα. Επίσης η βελτιστοποίηση γίνεται βάση του ελέγχου που κάνει η Java στις συνθήκες με αποτέλεσμα να περιορίζονται οι περιπτώσεις βελτιστοποίησης.

- Χρήση γενετικού αλγορίθμου για την επίλυση του συνόλου των περιορισμών αποδοτικά και αξιόπιστα.

6.4 Μελλοντική Εργασία

Όπως όλες οι έρευνες έτσι και αυτή έχει περιθώρια βελτιστοποίησης. Μετά από αρκετή τριβή με το θέμα καλό είναι να παρατεθούν μερικά στοιχεία τα οποία θα βοηθήσουν επόμενες μελλοντικές εργασίες τόσο σε αυτή την εφαρμογή όσο και γενικότερα στο θέμα αυτό.

- Το σύστημα θα μπορούσε να υλοποιηθεί έτσι ώστε να μην αναλύει και να ελέγχει μόνο προβλήματα γραμμένα σε Java αλλά και σε άλλες αντικειμενοστρεφείς γλώσσες προγραμματισμού. Αυτό δεν είναι αρκετά δύσκολο μιας και πρέπει να ενσωματωθεί η γραμματική για την νέα γλώσσα Ακολούθως τα επόμενα βήματα θα είναι τα ίδια με αυτά της αναγνώρισης του κώδικα που είναι γραμμένος σε Java.

- Επίσης θα μπορούσε να υποστηρίζει και άλλου τύπους δεδομένων όπως λίστες και πίνακες παρά μόνο τους αρχέγονους τύπους. Για να μπορέσει το σύστημα να ελέγχει μεγαλύτερο εύρος προγραμμάτων θα πρέπει να υλοποιηθεί και η αναγνώριση και επεξεργασία επαναληπτικών εντολών.

- Ακόμη μετά την εξαγωγή όλων των πιθανών μονοπατιών καλό είναι να ελέγχονται από εκεί τα προφανές «νεκρά» μονοπάτια έτσι ώστε να μην προκύπτουν αχρείαστα βελτιστοποιημένα προγράμματα για τα μονοπάτια αυτά

τα οποία δεν θα μπορέσουν να επιλυθούν. Όταν αναφερόμαστε σε προφανές «νεκρά» μονοπάτια είναι τα μονοπάτια όπου μετά από τη συμβολική εκτέλεση θα έχουν συνθήκες όπως 0 != 0.

- Μετά την παραγωγή των δεδομένων ελέγχου καλό είναι να τύχουν κάποιας επεξεργασίας έτσι ώστε να περιοριστεί ο αριθμός τους στο ελάχιστο δυνατό. Στη τρέχουσα υλοποίηση μπορεί να υπάρχουν περισσότερα από ένα δεδομένα ελέγχου για το ίδιο μονοπάτι λόγω των βελτιστοποιήσεων που έχουν γίνει.

- Για καλύτερα αποτελέσματα και λειτουργία του γενετικού αλγορίθμου στην επίλυση των περιορισμών που προκύπτουν από κάθε μονοπάτι θα μπορούσε ο γενετικός αλγόριθμος να λαμβάνει υπόψιν προηγούμενες επιλύσεις έτσι ώστε να είναι σε θέση να δημιουργεί ευκολότερα τα δεδομένα εισόδου σε παρόμοια σύνολα περιορισμών. Για παράδειγμα εάν δύο σύνολα περιορισμών έχουν 5 κανόνες και οι 4 πρώτοι είναι οι ίδιοι, ο γενετικός να το αναγνωρίζει και να προσπαθεί να τροποποιήσει τα δεδομένα βάση του τελευταίου περιορισμού.

- Εκτός από τα δύο κριτήρια το οποία έχουν υλοποιηθεί θα μπορούσαν να ελεγχθούν και άλλα για καλύτερο και ποιοτικότερο έλεγχο του δοθέντος προγράμματος.

Βιβλιογραφία

[1] P. McMinn, 2004, *Search-based Software Test Data Generation: A Survey*, 14 (2), pp 105 - 156

[2] P. Hoschka, C. Huitema INRIA, 2004, *Control flow graph analysis for automatic fast path implementation*, Second IEEE workshop on the architecture and Implementation of high performance communication subsystems

[3] J. C. King, 1976, 1976, *Symbolic execution and program testing*, 19(7), pp 385 – 394

[4] A.A. Sofokleous and A.S. Andreou, *Automatic, Evolutionary Test Data Generation for Dynamic Software Testing*, Journal of Systems and Software, (Elsevier Science: Amsterdam, Netherlands), accepted.

[5] A.A. Sofokleous and A.S. Andreou, 2007, *Batch-Optimistic Test-Cases Generation Using Genetic Algorithms*, Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Patra, Greece, October, pp. 157-164.

[6] Bottaci, L., 2002. *Instrumenting programs with flag variables for test data search by genetic algorithms*. In: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, July , pp.1337–1342.

[7] A.A. Sofokleous and A.S. Andreou, 2008, *Dynamic Search-base test data generation focused on data flow paths*, Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2006), Barcelona, Spain, June, (INSTICC Press: Porto, Portugal), forthcoming.

[8] A.S. Andreou, K.A. Economides and A.A. Sofokleous, 2007, *An automatic software test-data generation scheme based on data flow criteria and genetic algorithms*, Proceedings of the 7th IEEE International Conference on

Computer and Information Technology, Fukushima, Japan, October, (IEEE Computer Society: Los Alamitos, CA, USA), pp 867-872

[9] J. Zhang, C. Xu† and X. Wang, 2004, *Path-Oriented Test Data Generation Using Symbolic Execution and Constraint Solving Techniques*, Software Engineering and Formal Methods, Proceedings of the Second International Conference on

[10] N. Tillmann, W. Schulte, 2006, "Unit Tests Reloaded: Parameterized Unit Testing with Symbolic Execution", IEEE SOFTWARE, March 15, 2006, pp 38-47

[11] D. Kebbal, 2006, *Automatic flow analysis using symbolic execution and path enumeration*, Proceedings of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06),

[12] K. Sen, D. Marinov, G.Agha, 2005, *CUTE: A Concolic Unit Testing Engine for C*, ACM SIGSOFT Software Engineering Notes, 30(5), pp 263.272

[13] A. A. Sofokleous, A. S. Andreou, C. Schizas and G. Ioakim, *Extending and enhancing a basic program analysis system*, Proceedings of the IADIS International Conference, Applied Computing 2006 (AC2006), San Sebastian, Spain, 2006

[14] A.A. Sofokleous, A.S. Andreou and G. Ioakim, 2006 *Creating and manipulating control flow graphs with multilevel grouping and code coverage*, Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS 2006), Paphos, Cyprus, May, (INSTICC Press: Porto, Portugal), pp 259-262.

[15] Διπλωματική Εργασία Αναστάση Σοφοκλέους, *έλεγχος κώδικα λογισμικού – Αυτόματη δημιουργία γράφου ελέγχου ροής και περιπτώσεις ελέγχου μέσω αλγόριθμων βελτιστοποίησης*

[16] Διπλωματική Εργασία Κυριάκου Ιωάννου, Ιούνιος 2007, *Δημιουργία και γραφική απεικόνιση/διαχείριση γράφου εξάρτησης δεδομένων από κώδικα προγραμμάτων γραμμένα σε Java*

[17] Διπλωματική Εργασία Ρεββέκας Χριστοδούλου, Ιούνιος 2006, *Αυτοματοποιημένος έλεγχος λογισμικού με βελτιστοποίηση του κριτηρίου κάλυψης μονοπατιού*

[18] B. Bokowski, A. Spiegel, Freie University at Berlin, 1998. “Barat – A front-end for Java”, Freie University Berlin Technical Report B=98-09, December.

[19] B. Bonet, H. Geffner, 2006, *Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings*