Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Σύνθεση Προγραμμάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΖΑΜΑΝΗΣ-ΔΗΜΗΤΡΟΥΛΑΣ ΙΩΑΝΝΗΣ-ΟΡΕΣΤΗΣ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2020

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Σύνθεση Προγραμμάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΖΑΜΑΝΗΣ-ΔΗΜΗΤΡΟΥΛΑΣ ΙΩΑΝΝΗΣ-ΟΡΕΣΤΗΣ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Ιουλίου 2020.

.....	.....	.....
Νικόλαος Σ. Παπασπύρου Καθηγητής Ε.Μ.Π.	Δημήτριος Φωτάκης Αν. Καθηγητής Ε.Μ.Π.	Αριστείδης Παγουρτζής Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2020

.....

Ζαμάνης-Δημητρούλας Ιωάννης-Ορέστης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Ζαμάνης-Δημητρούλας Ιωάννης-Ορέστης, 2020.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Στην επιστήμη των υπολογιστών, η Σύνθεση Προγραμμάτων είναι η διαδικασία της αυτόματης κατασκευής ένος προγράμματος που ικανοποιεί μια δεδομένη προδιαγραφή υψηλού επιπέδου. Με την θεμελίωση της Θεωρίας των Γλωσσών Προγραμματισμού και την ανάπτυξη της Τεχνητής Νοημοσύνης το συγκεκριμένο πρόβλημα διατυπώθηκε επισήμως και έκτοτε αποτέλεσε έναν από τους κεντρικούς στόχους της επιστήμης των υπολογιστών. Ο εκ φύσεως εκθετικά μεγάλος χώρος αναζήτησης που προκύπτει από μια τυπική γλώσσα προγραμμάτισμου, καθώς και η εγγενής δυσκολία της ακριβούς εύρεσης και διατύπωσης των απαραίτητων προδιαγραφών το καθιστούν ένα ιδιαίτερα απαιτητικό προβλήμα. Ο τομέας της Σύνθεσής Προγραμμάτων σε συγκερασμό με άλλους τομείς,όπως η Μαθηματική Λογική, οι Τυπικές Μέθοδοι και η Τεχνητή Νοημοσύνη έχουν αναπτύξει μια πληθώρα μεθόδων για την προσεγγισή του, επιτρέποντας την ενσωμάτωση σε ένα ευρύ φάσμα εφαρμογών. Σήμερα, με τον αριθμό των προγραμματίσιμων συσκευών να αγγίζει τα δισεκατομμύρια και τον όγκο του λογισμικού, των λειτουργικών συστημάτων και των γλωσσών προγραμματισμού να αυξάνεται επίσης ραγδαία, η Σύνθεση Προγραμμάτων ανέρχεται σε ένα ελπιδοφόρο εργαλείο για την αντιμετώπιση αυτής της συνεχώς αυξανόμενης πολυπλοκότητας. Σκόπος αυτής της διπλωματικής εργασίας αποτελεί η διερεύνηση αυτού του εκτενούς πεδίου καθώς και η επισκόπηση των διαφορετικών μεθόδων που έχουν αναπτυχθεί για την προσεγγισή αυτού του προβλήματος.

Λέξεις κλειδιά

Σύνθεση Προγραμμάτων, Τυπικές Μέθοδοι, Θεωρία Γλωσσών Προγραμματισμού, Τεχνητή Νοημοσύνη, Μαθηματική Λογική.

Abstract

In computer science, Program Synthesis is the task of automatically constructing a program that satisfies a particular high-level specification. Alongside with the foundation of the Theory of Programming Languages and the development of Artificial Intelligence the specific problem was stated formally and since then became one of the most central goals in computer science. By its nature the exponentially large search space produced by an arbitrary programming language, together with the inherent difficulty of finding and formulating the necessary specifications make it an incredibly challenging problem. The field of Program Synthesis in cooperation with other fields, such as Mathematical Logic, Formal Methods and Artificial Intelligence have developed a variety of methods in order to tackle it, enabling its integration in a wide range of practical applications. Today, with the number of programmable devices in the world in the order of billions and the volume of software, operating systems and programming languages increasing rapidly, Program Synthesis rises as a promising tool for controlling this continuously incresing complexity. The goal of this diploma thesis is the investigation of this vast domain together with an overview of the various methods developed to approach this problem.

Key words

Program Synthesis, Formal Methods, Theory of Programming Languages, Artificial Intelligence, Mathematical Logic

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή αυτής της διατριβής, κ. Νίκο Παπασπύρου, για τη βοήθεια και την εμπιστοσύνη που μου παρείχε. Επίσης θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου, για όλη την αγάπη και τη στήριξη τους.

Ζαμάνης-Δημητρούλας Ιωάννης-Ορέστης,

Αθήνα, 30η Ιουλίου 2020

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-3-20, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Ιούλιος 2020.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Περιεχόμενα

Περίληψη 5

Abstract 7

Ευχαριστίες 9

Περιεχόμενα 11

Κατάλογος σχημάτων 13

1. Εισαγωγή 15
1.1 Περιγραφή προβλήματος - Ιστορική Αναδρομή 15
1.2 Σκοπός της εργασίας 15
1.3 Δομή της εργασίας 16

2. Θεωρητικό Υπόβαθρο 17
2.1 Γενικά 17
2.2 Ορισμός Σύνθεσης Προγραμμάτων 17
2.3 Άξονες - Απαιτήσεις Σύνθεσης Προγραμμάτων 17
2.3.1 Προδιαγραφές 18
2.3.2 Χώρος Προγραμμάτων 18
2.3.3 Μέθοδος Αναζήτησης 19

3. Παραγωγικές-Κατασκευαστικές Μέθοδοι 21
3.1 Γενικά 21
3.2 Μέθοδος Tableau 21
3.3 Top-Down Type-driven search 21

4. Επαγωγικές Μέθοδοι 23
4.1 Γενικά 23
4.2 Bottop-up search 23
4.2.1 Σύνθεση μέσω ενοποίησης 24
4.3 Συστήματα Περιορισμών - Constraint Based Systems 24
4.3.1 Συμβολική εκτέλεση 25
4.3.2 Επαλήθευση προγραμμάτων - Λογική του Hoare 25

5. Στοχαστικές Μέθοδοι 27
5.1 Γενικά 27
5.2 MCMC - Metropolis-Hastings 27
5.3 Γεννετικός Προγραμματισμός 27

6. Συμπεράσματα 29

Βιβλιογραφία 31

Κατάλογος σχημάτων

3.1 Manna Waldinger Tableau Method 22

4.1 Oracle Loop 24
4.2 CEGIS LOOP 25
4.3 Symbolic execution tree example 26
4.4 Hoare Rules 26

5.1 Genetic Loop Procedure 28

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή προβλήματος - Ιστορική Αναδρομή

Στην επιστήμη των υπολογιστών, η σύνθεση προγράμματων αποτελεί τη διαδικασία της αυτόματης κατασκευής ενός προγράμματος που ικανοποιεί αποδεδειγμένα μια δεδομένη επίσημη προδιαγραφή υψηλού επιπέδου. Σε σύγκριση με τους τυπικούς μεταγλωτιστές που μεταφράζουν από μια γλώσσα υψηλού επιπέδου σε μια χαμηλού επιπέδου γλώσσα μηχανής, η σύνθεση προγραμμάτων πραγματοποιεί κάποιου είδους αναζήτηση στον χώρο των προγραμμάτων που ορίζει η εκάστοτε γλώσσα ώστε να κατασκευάσει ένα πρόγραμμα που θα ανταποκρίνεται στους περιορισμούς που του έχουν επιβληθεί, οι οποίοι μπορεί να έχουν την μορφή παραδειγμάτων εισόδου-εξόδου, λογικών προτάσεων-περιορισμών, δηλώσεων, εντολών φυσικής γλώσσας κ.α. Η συγγραφή προγραμμάτων αποτελεί παραδοσιακά ανθρώπινη δημιουργία άλλα, η ιδέα της αυτόματης κατασκευής εφαρμόσιμων λύσεων (αλγορίθμων) με αποδείξεις, με συνδυασμό των υπολύσεων ενός προβλήματος χρονολογείται από το 1932 στον κλάδο των κατασκευαστικών μαθηματικών. Το 1957 ο Alonzo Church διατύπωσε επίσημα το πρόβλημα της σύνθεσης αναλογικών κυκλωμάτων που ικανοποιούν δοθέντες μαθηματικές απαιτήσεις [Chur63]. Αν και η συγκεκριμένη διατύπωση αφορά τα κυκλώματα θεωρείται από πολλούς ερευνητές η πρώτη διατύπωση του προβλήματος σύνθεσης προγραμμάτων. Στην συνέχεια με την θεμελίωση και ανάπτυξη κλάδων όπως η Θεωρητική Επιστήμη των Υπολογιστών, οι Τυπικές Μέθοδοι και η Τεχνητή Νοημοσύνη το πρόβλημα της σύνθεσης προγραμμάτων άρχισε να προσεγγίζεται και να συμπορεύεται με εκατέρωθεν επιτεύγματα από αυτούς τους διαφορετικούς κλάδους. Ειδικά μετά το 1960 δημιουργήθηκαν δύο πολύ μεγάλα ρεύματα αντιμετώπισης του προβλήματος, με το πρώτο να σχετίζεται με την παραγωγική-κατασκευαστική μέθοδο με την παράλληλη ανάπτυξη των αυτόματων προγραμμάτων για αποδείξεις θεωρημάτων όπου βασική ιδέα αποτέλεσε η εξαγωγή ενός ζητούμενου προγράμματος από την απόδειξη τις ισχύος της αντίστοιχης προδιαγραφης, που είχε σαν αποτέλεσμα την κατασκευή των πρώτων ολοκληρωμένων παραγωγικών συστημάτων[Mann71, Mann80, **?**, Gree69, Gree81] καθώς και των πρώτων συστημάτων αυτόματων μετασχηματισμών[Burs77, Pnue89]. Σύντομα, η εγγενής δυσκολία της ύπαρξης ολοκληρωμένων τυπικών προδιαγραφών οδήγησε στην δημιουργία του δεύτερου μεγάλου ρεύματος μεθόδων που αφορούσε την επαγωγική-αφαιρετική μέθοδο,βασιζόμενη κυρίως σε προδιαγραφές αποτελούμενες από παραδείγματα εισόδου-εξόδου και επίσης την δημιουργία των πρώτων ολοκληρωμένων συστημάτων επαγωγικής σύνθεσης[Smit75, Summ77], καθώς και την άνθιση ένος επίσης μελλόντικα μεγάλου υποκλάδου της σύνθεσης προγραμμάτων, του γεννετικού προγραμματισμού [Koza92]. Παρά τις δυσκολίες που περιλαμβάνει σαν πρόβλημα με την συνεχή ανάπτυξη και την ενσωμάτωση νέων μεθόδων και εργαλείων έχουν επιτευχθεί ήδη πλαίσια που το καθιστούν εφαρμόσιμο σε πολλούς τομείς της Πληροφορικής καθώς και σε άλλους τομείς της ανθρώπινης ζωής.

1.2 Σκοπός της εργασίας

Σκοπός της παρούσας εργασία σας είναι μια ανάλυση του προβλήματος της σύνθεσης προγραμμάτων και η απαρίθμηση των βασικών τρόπων που υπάρχουν στην βιβλιογραφία για την επιτυχή αντιμετώπιση του.

1.3 Δομή της εργασίας

Το πρώτο κεφάλαιο περιλαμβάνει μια γενική-ιστορική εισαγωγή,το δευτέρο μια πρώτη ανάλυση των βασικών συστατικών του προβλήματος της σύνθεσης, το τρίτο αναφέρεται στις παραγωγικές μεθόδους, το τέταρτο στις επαγωγικές, το πέμπτο στις στοχαστικές και τέλος το έκτο περιλαμβάνει κάποια συμπεράσματα.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

2.1 Γενικά

Τα προγράμματα των υπολογιστών είναι μια ιδιαίτερη κλάση μαθηματικού φορμαλισμού που αποτυπώνουν αλγορίθμους, τυπικές συνταγές για την επίλυση κλάσεων προβλημάτων. Τα προγράμματα παρουσιάζουν ιδιαίτερα ενδιαφέροντα χαρακτηριστικά καθώς απαιτούν την αλληλεπίδραση με δεδομένα για να αποκτήσουν νόημα, απαιτούν δηλαδή εισόδους ώστε να παράγουν εξόδους. Η εκφραστική τους δύναμη είναι ανάλογη με την γλώσσα προγραμματισμού στην οποία είναι γραμμένα. Κάθε Turing-complete γλώσσα είναι επαρκής ώστε να περιγράψει όλες τις υπολογίσιμες συναρτήσεις. Θεωρούμε ένα πρόγραμμα υπολογιστή μια πεπερασμένη διακριτή δομή αποτελούμενη από εντολές ικανή να πραγματοποιήσει υπολογισμούς, και ιδανικά εκφραζόμενη για τα πλαίσια της σύνθεσης προγραμμάτων σε ένα αφηρημένο συντακτικό δέντρο (AST). Οι κανόνες που ορίζουν την συντακτική εγκυρότητα ενός προγράμματος εκφράζονται με μια τυπική γραμματική, πάλι ιδανικά στα πλαίσια αυτής της εργασίας μια γραμματική χωρίς συμφραζόμενα. Ο Alan Turing έχει αποδείξει ότι το πρόβλημα της απόφασης για τον τερματισμό ενός προγράμματος είναι μη επιλύσιμο (halting problem), και επομένως προφανώς για ένα πρόγραμμα που δεν τερματίζει δεν έχουμε λόγο να ασχολούμαστε με την αυτόματη κατασκευή του. Άρα στο σύνολο αυτής της εργασίας θα αναφερόμαστε σε προγράμματα που τερματίζουν.

2.2 Ορισμός Σύνθεσης Προγραμμάτων

Ορίζουμε την σύνθεσης προγραμμάτων σαν ένα διατεταγμένο ζεύγος (P,Correct), όπου P είναι μια δεδομένη γλώσσα προγραμματισμού και Correct:P->Boolean είναι ένα κατηγόρημα ορθότητας. Η λύση του (P,Correct) έγγυται στην εύρεση ενός προγράμματος p που ορίζεται στην γλώσσα προγραμματισμού P ώστε να επαληθεύται το κατηγόρημα ορθότητας:

$$p \in P : Correct(p) \tag{2.1}$$

Έτσι η σύνθεση προγραμμάτων ανάγεται σε ένα συνδυαστικό πρόβλημα αναζήτησης στο χώρο των προγραμμάτων που όριζει η εκάστοτε δοσμένη γλώσσα προγραμματισμού, που ικανοποιεί τους περιορισμούς που του επιβάλει το πλαίσιο του (απαιτήσεις προδιαγραφής,κανόνες γλώσσας,σύνταξη κ.α.)

2.3 Άξονες - Απαιτήσεις Σύνθεσης Προγραμμάτων

Οι βασικοί άξονες, όπως προαναφέραμε, που χαρακτηρίζουν την σύνθεση είναι κατά βάση τρεις:

- Η επίσημη προδιαγραφή (σκοπός/πρόθεση του χρήστη/συστήματος)
- Ο χώρος αναζήτησης των προγραμμάτων
- Η μέθοδος εξερεύνησης του χώρου

2.3.1 Προδιαγραφές

Οι δύο πιο κλασσικοί και διαδεδομένοι τρόποι αναπαράστασης των προδιαγραφών είναι οι φόρμουλες μαθηματικής λογικής, συγκεκριμένα ένα κατηγόρημα είσοδου φ(χ) (precondition) και ένα κατηγόρημα εξόδου ψ(χ,z) (postcondition), καθώς και τα παραδείγματα εισόδου-εξόδου.

Έτσι προκύπτει:

$$(\forall x)[\phi(x) \rightarrow (\exists z)\psi(x,z)] \tag{2.2}$$

Συνήθως το ζεύγος precondition και postcondition είναι αρκετό για τον πλήρη προσδιορισμό της απαίτησης, και συναντάται συνήθως στις παραγωγικές μεθόδους (αναλυτικοί μετασχηματισμοί που παρέχουν ορθότητα εκ κατασκευής). Παρόλα αυτά η ύπαρξη και η διατύπωση τους είναι συχνά εξίσου απαιτητική με την επίλυση του προβλήματος.

Από την άλλη τα παραδείγματα είναι πιο φυσική μορφή και συνήθως παρέχεται σαν μια πεπερασμένη λίστα από ζευγάρια εισόδων εξόδων: T = ordered list of (I,O).

Ιδανικά πρέπει να μην περιέχουν διπλότυπα (καθώς οδηγούν σε αμφισημίες):

$$\nexists(in_1, out_1), (in_2, out_2) \in T : in_1 = in_2 \wedge out_1 = out_2 \tag{2.3}$$

Και να είναι συνεπή:

$$\nexists(in, out_1), (in, out_2) \in T : out_1 \neq out_2 \tag{2.4}$$

Αν και φυσικός και αποδοτικός τρόπος προσδιορισμού, καθώς και σημείο εκκίνησης πολλών επαγωγικών και στοχαστικών μεθόδων, σε αυτή την περίπτωση υπάρχει κίνδυνος απροσδιοριστίας καθώς τα παραδείγμα δεν είναι ολοκληρομένη προδιαγραφή αλλά μερική:

$$\nexists(in, out) \in T \tag{2.5}$$

Τέλος σαν προδιαγραφή μπορούν να δοθούν ίχνη εκτέλεσης, φυσική γλώσσα, μερικώς συμπληρωμένα προγράμματα, παρεμφερή προγράμματα κ.α. Με σκοπό να μειωθεί ο κίνδυνος του underspecification, ειδικά σε περίπλοκες περιπτώσεις έχουν προταθεί ιδέες χρήσης πολυτροπικών προδιαγραφών [Sing12].

2.3.2 Χώρος Προγραμμάτων

Ο χώρος των υποψήφιων προγραμμάτων (ή χώρος αναζήτησης) είναι βασική σχεδιαστική επιλογή για όλο το πρόβλημα και ένας από τους βασικούς άξονες δυσκολίας του. Για μια τυχαία επιλεγμένη γλώσσα προγραμματισμού αυξάνεται εκθετικά έως αστρονομικά γρήγορα καθιστώντας την αναπαράσταση και εξερεύνηση του μη-ανιχνεύσιμη. Σημαντική διαφορά σε αυτό τον τομέα παίζει η επιλογή της γλώσσας προγραμματισμού. Κάτα κύριο λόγο στην σύνθεση προγραμμάτων χρησιμοποιούνται ειδικά κατασκευασμένες γλώσσες ειδικού σκοπού (domain specific languages), με γνώμονα την διατήρηση της εκφραστικότητας και αποτελεσματικότητας, αλλά ταυτόχρονα τον περιορισμό του χώρου ώστε η αναζήτηση του να είναι εφικτή. Επίσης είναι διαδεδομένη η χρήση γλωσσών που προσομοιάζουν συναρτησιακές γλώσσες (ώστε να αποφεύγονται οι παρενέργειες).

Καλύτερη δυνατότητα αιτιολόγησης επίσης προσφέρει η χρήση συντακτικά αφηρημένων δέντρων (AST) και ειδικών γραμματικών χωρίς συμφραζόμενα (context-free grammars) όπως:

$$\begin{aligned} var &= a|b|c \\ expr &= expr + expr \\ &|expr * expr \\ bexp &= expr > expr|expr > 0 \end{aligned} \tag{2.6}$$

Η απαριθμητική απεικόνιση του χώρου αναζήτησης αν και χρησιμοποιείται σύχνα οδήγει σε έκηξη του χώρου καταστάσεων ακόμα και για πολύ μικρά εξεταζόμενα προγράμματα. Για αυτό το λόγο τα τελευταία χρόνια υπάρχει μια στροφή στη χρήση πιο συμβολικών μεθόδων απεικόνισης (lattice spaces, version spaces algebras), που επιτρέπουν την αναπαράσταση ολόκληρων κλάσσεων προγραμμάτων και έχουν ήδη πρακτικές εφαρμογές σε διάφορα πεδία (formal methods, model checking).

2.3.3 Μέθοδος Αναζήτησης

Οι τεχνικές εξερεύνησης του χώρου προγραμμάτων διακρίνονται ανάλογα με την μεθοδό τους σε απαριθμητικές (bottom-up,top-down,bidirectional search), που όπως θα δούμε προσφέρουν πρακτικές μεν αλλά δύσκολα επεκτάσιμες λύσεις, συμβολικές (constraint systems,abstract interpretation) που δίνουν το περιθώριο αποδοτικής επιχειρηματολογίας για πιο μεγάλους και περίπλοκους χώρους, καθώς και στοχαστικές (markov models, genetic programming, neural networks).

Κεφάλαιο 3

Παραγωγικές-Κατασκευαστικές Μέθοδοι

3.1 Γενικά

Πρώιμη έρευνα στον κλάδο των Κατασκευαστικών Μαθηματικών σε συνδυασμό με την διατύπωση της ισχύος του ισομορφισμού Curry-Howard κατέδειξε την άρρικτη σχέση των Προγραμμάτων με τις Μαθηματικές Αποδείξεις [Howa80]. Σε αυτή την κατεύθυνση πραγματοποιήθηκε σημαντική δουλειά από πολλούς επιστήμονες ανοίγοντας το δρόμο για την επίτευξη αυτόματων μηχανισμών αποδείξεων θεωρημάτων [Dijk68, Buch69] και στην συνέχεια την εμφάνιση των πρώτων συστημάτων αυτομάτης σύνθεσης προγραμμάτων [Gree69, Gree81, Wald69]. Τα περισσότερα από αυτά τα συστήματα σύνθεσης χρησιμοποίησαν την προσέγγιση της απόδειξης θεωρημάτων σαν βασική ιδέα: Με στόχο να κατασκευασεί ένα πρόγραμμα που ικανοποιεί μια συγκεκριμένη προδιαγραφή ή μια σχέση εισόδου-εξόδου, το σύστημα αποδεικνύει το θεώρημα που προκύπτει από την συγκεκριμένη σχέση και εξάγει το επιθυμητό πρόγραμμα κατευθείαν από την απόδειξη.

3.2 Μέθοδος Tableau

Οι R. Burstall and J. Darlington καθώς και οι Z. Manna and R. Waldinger πρότειναν ξεχωριστά με δύο πρωτοποριακές εργασίες [Burs77, Mann71] την δημιουργία ένος πλαισίου που δοθέντος μιας υψηλού-επιπέδου προδιαγραφής θα εφαρμόζει συνεχόμενους μετασχηματισμούς με στόχο την εύρεση του τελικού προγράμματος. Συγκεκριμένα στο πλαίσιο των Z. Manna and R. Waldinger (που αποτέλεσε και αποτελεί βασικό πυλώνα σε πολλά από τα συστήματα παραγωγικής σύνθεσης) το πρόβλημα της αυτόματης σύνθεσης προγραμμάτων διατυπώνεται ως εξής: Δοθέντος ενός κατηγόρηματος εισόδου φ(χ) (pre-condition) και ενός κατηγορήματος εξόδου ψ(χ,z) (post-condition) υπολόγισε την μερική συνάρτηση z = f(χ), τέτοια ώστε για κάθε διάνυσμα εισόδου χ, που επαληθεύει την προϋπόθεση φ(χ) η f(χ) ορίζεται και το κατηγόρημα εξόδου ψ(χ,z) γίνεται αληθές. Έτσι η κατασκευή του προγράμματος ανάγεται στην απόδειξη του θεωρήματος:

$$(\forall x)[\phi(x) \rightarrow (\exists z)\psi(x, z)] \tag{3.1}$$

Παρακάτω φαίνεται ο συμπληρωμένος πίνακας για την απόδειξη της έκφρασης (πήγη wikipedia):

3.3 Top-Down Type-driven search

Στο γενικότερο πλαίσιο των από πάνω προς τα κάτω μεθόδων αναζήτησης μια ίδιαιτερα χρήσιμη ιδέα είναι η χρήση τύπων για την καθοδήγηση της αναζήτησης και τον περιορισμό του χώρου των υποψήφιων προγραμμάτων. Σε σύγκριση με την απλή χρήση των κανόνων γραμματικής μιας γλώσσας, ένα σύστημα τύπων προσφέρει έναν επιπλέον μηχανισμό για την απόρριψη μη-έγκυρων προγραμμάτων βασιζόμενο στους παρακάτω πυλώνες:

- Δραστικό περίορισμο του χώρου ανάζητησης, καθώς τα μη-έγκυρα υποψήφια προγράμματα(με βάση το σύστημα τύπων) είναι κατά μέσω όρο πολύ περισσότερα από τα ορθά.

Nr	Assertions	Goals	Program	Origin
1	$A = A$			Axiom
2	$A \leq A$			Axiom
3	$A \leq B \vee B \leq A$			Axiom
10		$x \leq M \wedge y \leq M \wedge (x = M \vee y = M)$	M	Specification
11		$(x \leq M \wedge y \leq M \wedge x = M) \vee (x \leq M \wedge y \leq M \wedge y = M)$	M	Distr(10)
12		$x \leq M \wedge y \leq M \wedge x = M$	M	Split(11)
13		$x \leq M \wedge y \leq M \wedge y = M$	M	Split(11)
14		$x \leq x \wedge y \leq x$	x	Resolve(12,1)
15		$y \leq x$	x	Resolve(14,2)
16		$\neg(x \leq y)$	x	Resolve(15,3)
17		$x \leq y \wedge y \leq y$	y	Resolve(13,1)
18		$x \leq y$	y	Resolve(17,2)
19		$true$	x<y ? y : x	Resolve(18,16)

Σχήμα 3.1: Manna Waldinger Tableau Method

- Σπρώξιμο της πληροφορίας στα παρακάτω φύλλα του δένδρου.
- Απόρριψη εκφράσεων και υποπρογραμμάτων νωρίς, χωρίς να χρειαστεί η αποτιμησή τους.

Το βασικό πλεονέκτημα των τύπων είναι η δυνατότητα που δίνουν να εξεταστεί τοπίκα ένα μερικώς κατασκευασμένο πρόγραμμα, και στην περίπτωση που παραβιάζει τους κανόνες που ορίζει το σύστημα τύπων να απορριφθεί μαζί με όλα τις πιθανές του συνέχειες.

Σε αυτή την λογική τα τελευταία χρόνια προτάθηκαν και πιο εξειδικευμένα συστήματα τύπων, ονομαζόμενα refinement types,τύποι εξοπλισμένοι με κατηγορήματα κάποιας αποφασίσιμης λογικής, που επιτρέπουν ακόμα πιο ραγδαίο κλάδεμα του χώρου αναζήτησης, διατηρώντας βέβαια την ανιχνευσιμότητα του συστήματος τύπων.

$$\{\nu : B | P(\nu)\}$$
$$x : T \rightarrow \{\nu : B | P(x, \nu)\}$$

Σε αυτό το πλαίσιο συχνά χρησιμοποιείται η τεχνική του αμφίδρομου ελέγχου τύπων (bidirectional type checking), που προωθεί πληροφορία τόσο από πάνω προς τα κάτω (από τον τύπο μιας συνάρτησης, στους τύπους των υπομερών της) όσο και από κάτω προς τα πάνω (προωθόντας τον τύπων των συστατικών με σκοπό την εύρεση του τύπου της σύνθεσης τους).

Κεφάλαιο 4

Επαγωγικές Μέθοδοι

4.1 Γενικά

Οι παραγωγικές μέθοδοι που εξετάστηκαν προηγουμένως, προϋποθέτουν την ύπαρξη μιας πλήρους τυπικής προδιαγραφής του προβλήματος, που σε πολλές περιπτώσεις η εύρεση και η διατύπωση της είναι εξίσου δύσκολη με την εύρεση του προγράμματος που την επαληθεύει. Το γεγονός αυτό ώθεισαι από νωρίς πολλούς επιστήμονες στην προσπάθεια του προσδιορισμού του προβλήματος μέσω παραδειγμάτων είσοδου-εξόδου και την εξαγωγή γενικών συμπερασμάτων μέσω αυτών των παραδειγμάτων. Κομβικό σημείο αποτέλεσε η εργασία M. Gold [Gold67], στην οποία έθεσε το ερώτημα της μηχανικής αναγνωρισιμότητας μιας Turing γλώσσας. Ενώ απέδειξε ότι δεν υπάρχει αλγόριθμος γενικής-περίπτωσης που δοθεντος ενός αρχικού προγράμματος θα συμπεραίνει επαγωγικά οποιοδήποτε επιθυμητό πρόγραμμα, ταυτόχρονα απέδειξε ότι αν το τελικό πρόγραμμα περιοριστεί στην κλάση των αναδρομικά αριθμήσεων συναρτήσεων που τερματίζουν, τότε υπάρχει αλγόριθμος που αναγνωρίζει το επιθυμητό πρόγραμμα στο όριο, επιτυγχάνοντας το συγκεκριμένα με την μέθοδο της απαρίθμησης. Η ιδέα αυτή παρείχε ένα ίσχυρο άνω όριο πολυπλοκότητας θέτοντας τις βάσεις για την ανάπτυξη νέων επαγωγικών τεχνικών σύνθεσης προγραμμάτων με χρήση παραδειγμάτων εισόδου-εξόδου [Shaw75, Shap82, Bier78, Smit75, Wins70] και γεφύρωσε τον τομέα της σύνθεσης προγραμμάτων με τον τομέα της μηχανικής μάθησης [Mitc97, Wins70] Οι επαγωγικές αυτές μέθοδοι περιλαμβάνουν τόσο τεχνικές ρητής απαρίθμησης, όσο και πιο συμβολικές τεχνικές συστημάτων περιορισμών που παρέχουν την δυνατότητα ελέγχου ένος πολύ μεγαλύτερου χώρου γενικών προγραμμάτων.

4.2 Bottop-up search

Στην πιο απλή της περίπτωση η μέθοδος αναζήτησης από κάτω προς τα πάνω, κατασκευάζει όλα τα δυνατά προγράμματα από μια γραμματική ξεκινώντας από τα τερματικά της γλώσσας. Εύκολα γίνεται αντιληπτό ότι μια τέτοια μέθοδος δεν είναι πρακτική καθώς ο χώρος αναζήτησης αυξάνεται εκθετικά σε κάθε βήμα.

Algorithm 1: Simple Bottom Up Search

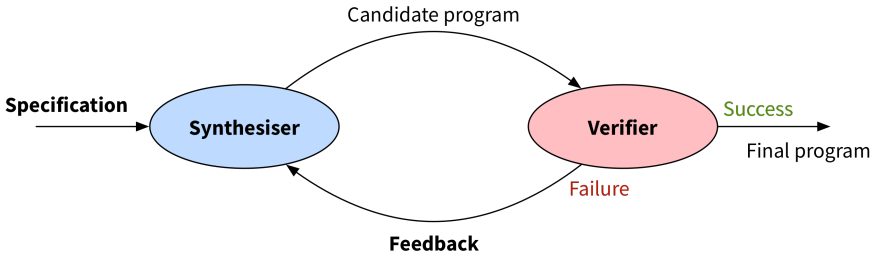
```
Result: Synthesized program p
candidates:= set of all terminals;
while true do
    candidates:= extend(candidates);
    candidates:= remove_equivalents(candidates,inputs);
    foreach p in candidates do
        if isCorrect(p,inputs,outputs) then
            return p;
        end
    end
end
```

4.2.1 Σύνθεση μέσω ενοποίησης

Για την βελτίωση της επεκτασιμότητας της απαριθμητικής μεθόδου, αντί να προσπαθούμε να κατασκεύασουμε ένα πρόγραμμα που θα επαληθεύει ολές τις είσοδου μονομιάς, μπορούμε να χωρίσουμε το πρόβλημα, αναζητώντας προγράμματα που επαληθεύουν ένα μέρος των εισόδων και όταν έχουμε καλύψει όλες τις απαραίτητες περιπτώσεις να συνδυάσουμε αύτα τα υποπορογράμματα, σε ένα ενιαίο πρόγραμμα που εκ κατασκεύης θα λειτουργεί για όλες τις εισόδους-εξόδους. Αυτή η ιδέα αναπτύσσεται ακόμα παραπέρα αν στο χώρο των προγραμμάτων μπορούμε να εισάγουμε μια μορφή ιεραρχίας που θα μας επιτρέπει να ανακαλύπτουμε προγράμματα που καλύπτουν ένα μεγάλο μέρος των περιπτώσεων και στη σύνεχεια τα επεκτείνουμε για να καλύψουμε τις εναπομείναντες οριακές περιπτώσεις.

4.3 Συστήματα Περιορισμών - Constraint Based Systems

Οι απαριθμητικές μέθοδοι αν και απλές και αποδοτικές για κάποιες περιπτώσεις φαίνεται ότι δεν επεκτείνονται αποδοτικά στη γενική περίπτωση. Σε αυτό το σημείο θα μελετήσουμε μια κατήγορια πιο συμβολικών μεθόδων που προσεγγίζουν το πρόβλημα της σύνθεσης σαν ένα παραμετρικό πρόβλημα P(c) που για διάφορες τιμές του c μπόρουμε να αναπαραστήσουμε ένα ευρύ πλήθος πιθανών προγραμμάτων. Έτσι το πρόβλημα της σύνθεσης ανέρχεται στην δημιουργία κατάλληλων περιορισμών σε μια λογική θεωρία που θα ενσαρκώνουν τις απαιτήσεις που καθιστούν οι εκάστοτε δοθείσες προδιαγραφές και στην αποδοτική επιλυσή τους με τη χρήση σύγχρονων εργαλείων επίλυσης (SMT Solvers). Αυτή η μέθοδος μας επιτρέπει να προσεγγίζουμε πολύ μεγαλύτερους και πολύπλοκους χώρους αναζήτησης, αν και όπως θα δούμε με μια σημαντική επιβάρυνση σε υπολογιστικό κόστος. Όπως έγινε φανερό από τις εργασίες των Gold και Shapiro ένα πλαίσιο επαγωγικής εκμάθησης, εισάγει την ιδέα ενός ”μάντη” (oracle), που λειτουργεί σαν black box δεχόμενο ερωτήματα και επαληθεύει ή απορρίπτει την ισχύ της επαγωγικής υπόθεσης παρέχοντας νέα πληροφορία στο σύστημα που ”μαθαίνει”.



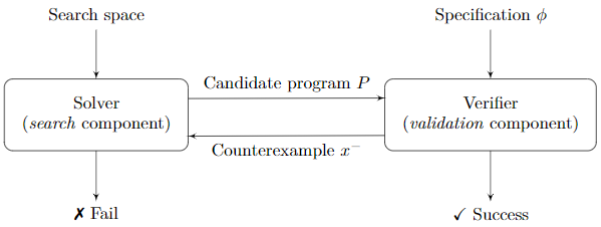
Σχήμα 4.1: Oracle Loop

Αυτή η όμαδα τεχνικών όνομαστηκε Oracle Guided Synthesis και αποτελείται από τα παρακάτω υπομέρη:

- Την συμβολική αναπαράσταση και εκτέλεση της επιθυμητής συμπεριφοράς (specification)
- Την σύνθεση ενός υποψήφιου προγράμματος (synthesizer)
- Την ερώτηση και επαλήθευση του υποψήφιου προγράμματος (verifier)
- Την απόκριση του μάντη (oracle feedback)

Ο synthesizer παράγει ένα πρόγραμμα P και εκτελεί ένα ερώτημα στον verifier (ουσιαστικά ένα Hoare-query) για το αν είναι σωστό το πρόγραμμα που παρήγαγε. Ο verifier συνήθως καλεί έναν smt solver για να αποφανθεί για την αλήθεια της φόρμουλας. Σε περίπτωση που το επαληθεύσει η λύση είναι σωστή. Άλλιως ο synthesizer ψάχνει για ένα νέο πρόγραμμα.

Προχωρώντας αυτή την ιδέα παραπέρα και βασιζομένος σε μια παρόμοια τεχνική από τον τομέα του model checking ο Solar-Lezam στην εργασία που παρουσίασε το σύστημα SKETCH είσηγαγε μια επέκταση του παραπάνω αλγορίθμου, ονομαζόμενη CEGIS (Counter Example Inductive Synthesis), με την οποία τεχνική ο verifier παρέχει τον ερωτόμενο και με ένα αντιπαράδειγμα για την φόρμουλα που μόλις απέρριψε δίνοντας την δυνατότητα στον synthesizer να βελτίωση την οπτική του και να πετύχει γρηγορότερη σύγκλιση.



Σχήμα 4.2: CEGIS LOOP

Σε αυτό το σήμειο θα περιγράψουμε αναλυτικότερα τα βασικά συστατικά που κάνουν εφικτά αυτά τα βήματα. Ειδικότερα:Την μετατροπή των συμπεριφορών σε περιορισμούς, την συμβολική αναπαράσταση και εκτέλεση τους, την διαδικασία επαλήθευσης των προγραμμάτων βασιζόμενη στη λογική του Hoare και τελικά την επίλυση αυτών των περιορισμών με τη χρήση των SMT solvers.

4.3.1 Συμβολική εκτέλεση

Η συμβολική εκτέλεση είναι μια δημοφιλής τεχνική ανάλυσης προγραμμάτων που αναπτύχθηκε μετά το 1970 με σκοπό να βοηθήσει το testing προγραμμάτων. Εξετάζοντας τον κώδικα χρησιμοποιώντας σύμβολα για εισόδους, αντί για συγκεκριμένες τιμές, επιτρέπει την επιχειρημαλογία για ολόκληρες κλάσεις προγραμμάτων. Κάθε στιγμή η συμβολική μηχανή εκτέλεσης διατηρεί μια κατάσταση (stmt, σ, π) όπου:

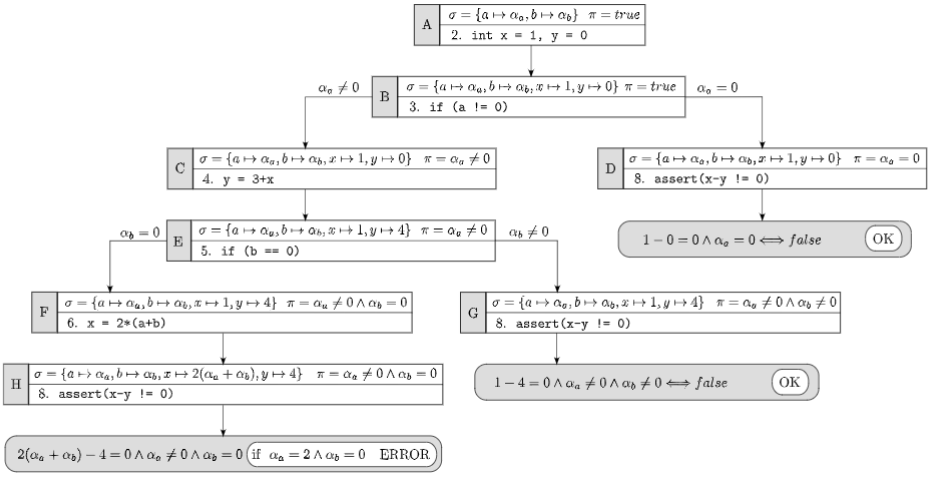
- stmt είναι η επόμενη εντολή που θα εκτελεστεί
- σ είναι ένα σύνολο συμβόλων (symbolic store) που συνδέει τις μεταβλητές του προγράμματος με τις συμβολικές ή τις πραγρατικές τους τιμές
- π δηλώνει το μονοπάτι των περιορισμών (αποτελεί δηλαδή μια φόρμουλα που περιέχει την πληροφοριά των υποθέσεων που αφορούν τις μεταβλητές ανάλογα με τις επιλογές διακλάδωσης που έχει πάρει το πρόγραμμα)

Παράδειγμα συμβολικής μνήμης:

4.3.2 Επαλήθευση προγραμμάτων - Λογική του Hoare

Ακόμα ένα πολύ σημαντικό εργαλείο στην ανάλυση προγραμμάτων αποτελούν οι κανόνες συλλογιστικής της λογικής του Hoare. Που έχουν την μορφή {A} cmd {B} και δηλώνουν ότι ένα πρόγραμμα ξεκινώντας από μια κατάσταση που επαληθεύει ένα κατηγόρημα A, αφού εκτελέσει την εντολή cmd, μεταβαίνει υποχρεωτικά το κάτασταση που επαληθεύει το B.

Οι κανόνες παραγωγής της Λογικής Hoare περιγράφονται συνοπτικά στην παρακάτω εικόνα (από πάνω αρίστρερα προς τα κάτω δεξία) είναι οι: Ανάθεσης,Conditional,Κενός,Σύνθεσης,While Ταυτόχρονα τους συμπληρώνουν οι κανόνες της Συνέπειας καθώς επεκταμένος While κανόνας ώστε να επιτυγχάνεται η ολική ορθότητα (σε αντίθετη περίπτωση δεν εγγυόμαστε τερματισμό).



Σχήμα 4.3: Symbolic execution tree example

$$\frac{}{\vdash \{A[x \to e]\}x := e\ \{A\}} \qquad \frac{\vdash \{A \wedge b\}c_1\ \{B\} \quad \vdash \{A \wedge not\ b\}c_2\ \{B\}}{\vdash \{A\}if\ b\ then\ c_1 else\ c_2\ \{B\}}$$

$$\frac{}{\vdash \{A\}skip\{A\}} \qquad \frac{\vdash \{A\}c_1\ \{C\} \quad \vdash \{C\}c_2\ \{B\}}{\vdash \{A\}c_1; c_2\ \{B\}} \qquad \frac{\vdash \{A \wedge b\}c\ \{A\}}{\vdash \{A\}while\ b\ do\ c\ \{A \wedge not\ b\}}$$

Σχήμα 4.4: Hoare Rules

$$\frac{P_1 \to P_2 \quad , \quad \{P_2\}S\{Q_2\} \quad , \quad Q_2 \to Q_1}{\{P_1\}S\{Q_1\}}$$

(a) Rule of Consequence

$$\frac{< \text{ is a well-founded ordering on the set } D \quad , \quad [P \wedge B \wedge t \in D \wedge t = z]S[P \wedge t \in D \wedge t < z]}{[P \wedge t \in D]\textbf{while } B \textbf{ do } S \textbf{ done}[\neg B \wedge P \wedge t \in D]}$$

(b) While Rule for total-correctness

Κεφάλαιο 5

Στοχαστικές Μέθοδοι

5.1 Γενικά

Όπως είδαμε μέχρι τώρα στόχος της σύνθεσης ήταν να βρούμε ένα πρόγραμμα που θα ικανοποιεί την ζητούμενη προδιαγραφή. Παρόλα αυτά πολλές φορές, αυτό δεν μας εξασφαλίζει αναγκαία την λύση που θέλουμε καθώς ολοκληρωμένες προδιαγραφές που να απορρίπτουν όλα τα μη επιθυματά προγράμματα είναι δύσκολο να βρεθούν. Ειδικά στο πλαίσιο της σύνθεσης με βάση παραδείγματα, είναι φανερό ότι για ένα πεπερασμένο σύνολο παραδειγμάτων μπορούν να υπάρχουν άπειρα προγράμματα που τα επαληθεύουν και αυτό απαιτεί κάποια έξτρα πληροφορία ώστε να διαλέξουμε αυτό που χρειαζόμαστε. Σε αυτή την κατεύθυνση χρήσιμη είναι μια πιθανοκρατική σκοπιά που μας επιτρέπει να αιτιολογούμε για την εώς τώρα γνώση μας με μια σχετική πιθανότητα που θα αλλάζει όσο συλλέγουμε έξτρα γνώση. Οι στοχαστικές μέθοδοι στοχεύουν στην εκμάθηση μιας κατανομής στον χώρο των υποθέσεων και την χρήση αυτής της κατανομής για την περαιτέρω καθοδήγηση του μετώπου αναζήτησης. Οι τρεις αντιπροσωπευτικότερες από αυτές είναι ο αλγόριθμος Metropolis-Hastings, ο γεννετικός-εξελικτικός προγραμματισμός και τα νευρωνικά δίκτυα.

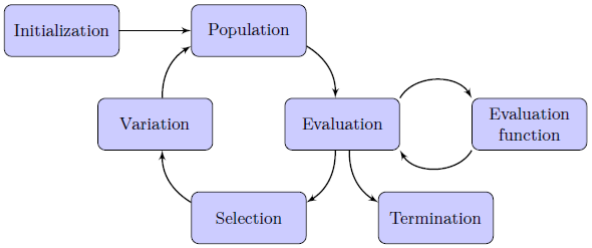
5.2 MCMC - Metropolis-Hastings

Η γενική ιδέα σε αυτή την προσέγγιση είναι να οριστεί μια συνάρτηση κόστους στο χώρο των προγραμμάτων που θα αναθέτει μια τιμή σε κάθε πρόγραμμα. Ιδανικά θα θέλαμε να κατασκευάσουμε έναν πίνακα μεταβάσεων που θα μας δίνει μια στατική κατανόμη με υψηλή τιμή για τα ”καλά-επιθυμητά” προγράμματα και χαμηλή αντίθετα. Έτσι ξεκινώντας από μια τύχαια θέση X0 στον χώρο καταστάσεων και προσωμοιώνοντας την μαρκοβιανή αλυσίδα για n βήματα, το θεμέλιωδες θεώρημα των μαρκοβιανών αλυσίδων θα μας εξασφάλιζε ότι το Xn θα είναι με μεγάλη πιθανότητα ”καλό” πρόγραμμα. Βέβαια η κατασκευή ενός κατάλληλου πίνακα μετάβασης σε έναν τόσο ανομοιόμορφο χώρο και σύχνα ασυνεχή χώρο καταστάσεων είναι δύσκολη και για αυτό επιστρατεύεται ο αλγόριθμος Metropolis-Hastings.[Schk13, Alur13]

5.3 Γεννετικός Προγραμματισμός

Ο γεννετικός προγραμματισμός (genetic programming) είναι μια στοχαστική construct and test προσέγγιση, πιο κοντά στην επαγωγική σύνθεση προγραμμάτων και έχει τις ρίζες του στους εξελικτικούς αλγορίθμους και στην πρωτοποριακή δουλειά των Koza και Holland[Koza92, Holl92]. Προσαρμόζει το πρόβλημα της σύνθεσης σε ένα αλγόριθμο βελτιστοποίησης και βασίζεται στην μεταευρευριστική των εξελικτικών αλγορίθμων, με σκόπο να βελτιώσει επαναληπτικά τα υποψήφια προγράμματα. Οι αλγόριθμοι γεννετικού προγραμματισμού διατηρούν έναν πληθυσμό υποψήφιων προγραμμάτων και εφαρμόζουν μεταλλάξεις και συνδυαστικές πράξεις ώστε να παράγουν ποικιλία προγραμμάτων. Οι μεταλλάξεις πραγματοποιούν τυχαίες αλλαγές, ενώ οι πράξεις μεταξύ τους διασφαλίζουν την επαναχρησιμοποίηση των κατάλληλων υπομερών των προγραμμάτων. Κεντρικό ρόλο έχει η ευριστική συνάρτηση αξιολόγησης (fitness function) που εκφράζει την προοπτική του κάθε υποψήφιου προγράμματος, που σε περίπτωση που μεγιστοποιηθεί έχουμε λύση και τερματισμό, αλλιώς τα ηνία ανα-

λαμβάνει ένας είδικος ”τελεστής επιλογής” που κάνει διαλογή των πιο υποσχόμενων γονιών και τους διαχωρίζει από τον υπόλοιπο πληθυσμό. Έτσι στο επόμενο βήμα της αναζήτησης οι τελεστές αναζήτησης εφαρμόζονται σε αυτό το επιλεγμένο σύνολο καταλήγοντας στην νέα γενιά απογόνων που θα πάρουν μέρος στο επόμενο στάδιο του εξελικτικού βρόχου.



Σχήμα 5.1: Genetic Loop Procedure

Κεφάλαιο 6

Συμπεράσματα

Όπως είδαμε η σύνθεση προγραμμάτων αποτελεί ένα πολύ απαιτητικό πρόβλημα, αλλά και ένα ανοιχτό ερευνητικό πεδίο που εμπλέκει πολλές περιοχές της επιστήμης των υπολογιστών. Αν και διατυπώμενο από τα πρώτα της θεωρητικής πληροφορικής η δυσκολία του το κρατάει ακόμα σε πρώιμο στάδιο, με τις πρώτες του πρακτικές εφαρμογές να διαφαίνονται αυτά τα χρόνια και την γενική του ενσωμάτωση σε πιο περίπλοκα συστήματα να απαιτεί ακόμα χρόνο, έρευνα και ενδυνάμωση των ήδη υπάρχοντων μεθόδων και εργαλείων. Παρόλα αυτά οι δυνατότητες που προσφέρει, καθώς και η άμεση συσχέτιση του με τόσα διαφορετικά πεδία το καθιστούν μια πολύ υποσχόμενη περιόχη που αξίζει να μελετηθεί περαιτέρω. Σαν ενέργες και μελλοντικές κατευθύνσεις στην γενικότερη επίλυση και επεκτασιμότητα του αναφέρουμε τις ίδεες της πολυτροπικότητας των προδιαγραφών, με σκοπό την περαιτέρω γεφύρωση της πρόθεσης του χρήστη με την απόκριση του συστήματος, την βαθύτερη ανάπτυξη συμβολικών, στατιστικών και αφηρημένων μεθόδων που επιτρέπουν την αντιμετώπιση του προβλήματος της έκρηξης του χώρου καταστάσεων, καθώς και ιδέες μεταφοράς και γενίκευσης της γνώσης σε κάθε στάδιο της αναζήτησης που θα βάλει τις βάσεις για την ενσωμάτωση του σε σύγχρονες παράλληλες αρχιτεκτονικές ανοίγοντας τον δρόμο για την ουσιώδη επεκτασιμότητα του.

Βιβλιογραφία

[Alur13] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak and Abhishek Udupa, *Syntax-guided synthesis*, IEEE, 2013.

[Bald18] Roberto Baldoni, Emilio Coppa, Daniele Cono D'elia, Camil Demetrescu and Irene Finocchi, "A survey of symbolic execution techniques", *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–39, 2018.

[Bier78] Alan W Biermann, "The inference of regular LISP programs from examples", *IEEE transactions on Systems, Man, and Cybernetics*, vol. 8, no. 8, pp. 585–600, 1978.

[Buch69] J. Richard Buchi and Lawrence H. Landweber, "Solving Sequential Conditions by Finite-State Strategies", *Transactions of the American Mathematical Society*, vol. 138, pp. 295–311, 1969.

[Buch90] J Richard Buchi and Lawrence H Landweber, "Solving sequential conditions by finite-state strategies", in *The Collected Works of J. Richard Büchi*, pp. 525–541, Springer, 1990.

[Burs77] R. M. Burstall and John Darlington, "A Transformation System for Developing Recursive Programs", *J. ACM*, vol. 24, no. 1, pp. 44–67, January 1977.

[Chur63] Alonzo Church, "Application of recursive arithmetic to the problem of circuit synthesis", 1963.

[Clar00] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu and Helmut Veith, "Counterexample-guided abstraction refinement", in *International Conference on Computer Aided Verification*, pp. 154–169, Springer, 2000.

[Dijk68] Edsger W Dijkstra, "A constructive approach to the problem of program correctness", *BIT numerical mathematics*, vol. 8, no. 3, pp. 174–186, 1968.

[Gold67] E Mark Gold, "Language identification in the limit", *Information and control*, vol. 10, no. 5, pp. 447–474, 1967.

[Gree69] Cordell Green, "Theorem proving by resolution as a basis for question-answering systems", *Machine intelligence*, vol. 4, pp. 183–205, 1969.

[Gree81] Cordell Green, "Application of theorem proving to problem solving", in *Readings in Artificial Intelligence*, pp. 202–222, Elsevier, 1981.

[Holl92] John H Holland, "Genetic algorithms", *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[Howa80] William A Howard, "The formulae-as-types notion of construction", *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, vol. 44, pp. 479–490, 1980.

[Jha10] Susmit Jha, Sumit Gulwani, Sanjit A Seshia and Ashish Tiwari, "Oracle-guided component-based program synthesis", in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, pp. 215–224, IEEE, 2010.

[Koza92] John R Koza and John R Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1, MIT press, 1992.

[Mann71] Zohar Manna and Richard J Waldinger, "Toward automatic program synthesis", *Communications of the ACM*, vol. 14, no. 3, pp. 151–165, 1971.

[Mann80] Zohar Manna and Richard Waldinger, "A deductive approach to program synthesis", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 2, no. 1, pp. 90–121, 1980.

[Mann84] Zohare Manna and Richard Waldinger, "The Logical Basis for Computer Programming: Deductive Reasoning", 1984.

[Mitc97] Tom M Mitchell et al., "Machine learning. 1997", *Burr Ridge, IL: McGraw Hill*, vol. 45, no. 37, pp. 870–877, 1997.

[Pnue89] Amir Pnueli and Roni Rosner, "On the synthesis of a reactive module", in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 179–190, 1989.

[Schk13] Eric Schkufza, Rahul Sharma and Alex Aiken, "Stochastic superoptimization", *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 305–316, 2013.

[Shap81] Ehud Y Shapiro, "An Algorithm that Infers Theories from Facts.", in *IJCAI*, pp. 446–451, Citeseer, 1981.

[Shap82] Ehud Yehuda Shapiro, *Algorithmic Program Debugging*, Ph.D. thesis, USA, 1982. AAI8221751.

[Shaw75] David E Shaw, William R Swartout and C Cordell Green, "Inferring LISP Programs From Examples.", in *IJCAI*, vol. 75, pp. 260–267, 1975.

[Sing12] Rishabh Singh and Armando Solar-Lezama, "SPT: storyboard programming tool", in *International Conference on Computer Aided Verification*, pp. 738–743, Springer, 2012.

[Smit75] David Canfield Smith, "Pygmalion: a creative programming environment", Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1975.

[Sriv10] Saurabh Srivastava, Sumit Gulwani and Jeffrey S Foster, "From program verification to program synthesis", in *Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 313–326, 2010.

[Summ77] Phillip D Summers, "A methodology for LISP program construction from examples", *Journal of the ACM (JACM)*, vol. 24, no. 1, pp. 161–175, 1977.

[Wald69] Richard J Waldinger and Richard CT Lee, "PROW: A step toward automatic program writing", in *Proceedings of the 1st international joint conference on Artificial intelligence*, pp. 241–252, 1969.

[Wins70] Patrick H Winston, "Learning structural descriptions from examples", 1970.