ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΓΛΩΣΣΑΣ ΠΕΡΙΓΡΑΦΗΣ ΥΛΙΚΟΥ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΣΧΕΤΙΚΟΥ ΜΕΤΑΓΛΩΤΤΙΣΤΗ

ΑΓΓΕΛΟΥ ΑΛΕΞΑΝΔΡΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Δημητρίου Γεώργιος
Επίκουρος καθηγητής

Λαμία 3 Νοεμβρίου έτος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΓΛΩΣΣΑΣ ΠΕΡΙΓΡΑΦΗΣ ΥΛΙΚΟΥ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΣΧΕΤΙΚΟΥ ΜΕΤΑΓΛΩΤΤΙΣΤΗ

ΑΓΓΕΛΟΥ ΑΛΕΞΑΝΔΡΟΣ

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

DESIGN OF A HARDWARE DESCRIPTION LANGUAGE AND IMPLEMENTATION OF THE RESPECTIVE COMPILER

ALEXANDROS ANGELOU

FINAL THESIS

ADVISOR

Georgios Dimitriou
Assistant Professor

Lamia, 3rd of November, 2022

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. ✓ Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. ✓ Δέχομαι ότι η αυτολεξεί ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. ✓ Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 15/10/2022

Ο – Η Δηλ.

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Μελετάται ο σχεδιασμός μίας γλώσσας περιγραφής υλικού που υποστηρίζει μοντέρνα χαρακτηριστικά όπως αντικειμενοστρέφεια και συναρτησιακό προγραμματισμό. Το σύστημα τύπων και εκφράσεών της επισημοποιείται με βάση τις αρχές της θεωρίας κατηγοριών και της άλγεβρας λάμδα. Τέλος μελετάται η υλοποίηση ενός μεταγλωττιστή για τη γλώσσα αυτή που παράγει κώδικα σε Verilog, με τη χρήση της γλώσσας προγραμματισμού Rust και των εργαλειών “Lalrpop” και “Logos”.

ABSTRACT

The design of a hardware description language that supports modern features like object orientedness and functional programming is studied. Its type and expression system is formalised on the basis of Category Theory and Lambda Calculus. Finally, the implementation of a compiler for this language that outputs code in Verilog is studied, using the Rust programming language and the tools “Lalrpop” and “Logos”.

Table of Contents

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

Η συγκεκριμένη πτυχιακή εργασία μελετά τον σχεδιασμό μιας νέας, μοντέρνας γλώσσας περιγραφής υλικού, με το όνομα “Iris”, και την υλοποίηση ενός μεταγλωττιστή για την γλώσσα Iris ο οποίος μπορεί να την μεταφράσει σε μία πιο διαδεδομένη γλώσσα περιγραφής υλικού, συγκεκριμένα την Verilog, η οποία αποτελεί πρότυπο της IEEE, για χρήση με όλα τα εργαλεία τα οποία υποστηρίζουν το πρότυπο αυτό. Ο σκοπός του σχεδιασμού είναι η δημιουργία μιας γλώσσας που ακολουθεί πολλά προγραμματιστικά πρότυπα τα οποία ακολουθούνται από όλες τις μοντέρνες γλώσσες προγραμματισμού, με έμπνευση από τέτοιες γλώσσες, καθώς και την μετασκευή των προτύπων αυτών στη νοοτροπία της περιγραφής υλικού αντί για τον προγραμματισμό εφαρμογών, με τρόπο ο οποίος είναι φυσικός και πραγματικά χρήσιμος στο χρήστη για την υλοποίηση προγραμματιστικών μοτίβων που βοηθούν στη διαδικασία της ανάπτυξης του προγράμματος. Τέλος, θα μελετηθεί ο τρόπος κατά τον οποίο ο μεταγλωττιστής υλοποιείται σε γλώσσα rust, του οποίου η υλοποίηση χρήζει μεγάλης αλγοριθμικής και σχεδιαστικής σημασίας, όχι μόνο για την επιτυχή μετάφραση αλλά και για την διασφάλιση της δυνατότητας ανάπτυξης του και προσθήκης χαρακτηριστικών εύκολα επ’ αεί.

ΚΕΦΑΛΑΙΟ 2 Σχεδιασμός

Ο σχεδιασμός της γλώσσας επέρχεται επιχειρηματολογίας με σκοπό την καλύτερη δυνατή εμπειρία χρήστη, δηλαδή την απόλυτη αποτελεσματικότητα όσον αφορά την αναγνωσιμότητα, εκφραστικότητα, λακωνικότητα, ευκολία στην εκμάθηση, ντετερμινιστικότητα, προσβασιμότητα, προβλεψιμότητα, ελαχιστοποίηση του χρόνου ανάπτυξης και οργάνωση του κώδικα. Προκειμένης της διευκόλυνσης των αποφάσεων που αποσκοπούν στην επίτευξη των παραπάνω αποσκοπούμενων χαρακτηριστικών, θεωρήθηκε αποτελεσματική η έμπνευση από ήδη γνωστές, δογματικές και καθιερωμένες στην βιομηχανία της ανάπτυξης εφαρμογών υπολογιστών γλώσσες προγραμματισμού όχι μόνο λόγω του γεγονότος ότι επίσης αποσκοπούν στα προαναφερθέντα αυτά χαρακτηριστικά, αλλά επίσης λόγω του μεγάλου αριθμού προγραμματιστών με εξοικείωση σε αυτές, ώντας καθιερωμένες, ο οποίος συμβάλλει στην επίτευξη συγκεκριμένα της ευκολίας στην εκμάθηση και της προσβασιμότητας. Οι γλώσσες που επιλέχθηκαν κατά κόρον ως παραδείγματα γλωσσών προγραμματισμού προς έμπνευση είναι η Rust, η Kotlin και η JavaScript, καθώς αποτελούν ένα δείγμα των γλωσσών που ικανοποιούν τις προαναφερθείσες προϋποθέσεις το οποίο πηγάζει και από την προσωπική μου αρέσκεια.

(Υποκεφάλαιο 2.1) Προγραμματιστικό πρότυπο

Οι 2 πιο γνωστές γλώσσες μοντελοποίησης υλικού σήμερα (VHDL και Verilog) επιλέγουν ένα προστακτικό πρότυπο προγραμματισμού, το οποίο ριζώνει στην δήλωση υπομερών και συνδεσμολογίας αυτών ως “μεταβλητές,” και την δήλωση συνδεσμολογικών σχέσεων με αυτήν ως εκφράσεις χρησιμοποιώντας το όνομα της ανάλογης μεταβλητής.[9] Η δήλωση του τύπου της κάθε αυτής μεταβλητής καθώς και οι εκφράσεις που σχετίζονται ή περιλαμβάνουν την μεταβλητή αυτή μεταφράζονται ένα-προς-ένα σε ένα ηλεκτρονικό μοντέλο υλικού με την ανάλογη συμπεριφορά που περιγράφεται στο πρόγραμμα που έγραψε ο χρήστης της γλώσσας. Η λογική λοιπόν είναι απλή· η ύπαρξη μίας σύνδεσης ή μιας λογικής υπομονάδας (submodule) ονοματίζεται και δηλώνεται στη μορφή μιας μεταβλητής, των οποίων οι είσοδοι και οι έξοδοι μπορούν να εξηγηθούν με τη χρήση των ανάλογων εκφράσεων. Οι σχέσεις αυτές διαδραματίζονται μέσα στα όρια μιας δήλωσης “μονάδας” (module), η οποία έχει κάποιες εισόδους και εξόδους και κάποιο όνομα. Αυτή η μονάδα μπορεί να χρησιμοποιηθεί αργότερα σαν “τύπος” μεταβλητής, προκειμένου να δηλωθεί μια υπομονάδα με συμπεριφορά ίδια με αυτή που έχει περιγραφεί στην δήλωση της ανάλογης μονάδας.

Η ίδια λογική διέπει και αυτήν την γλώσσα περιγραφής υλικού, αλλά με περεταίρω χαρακτηριστικά τα οποία αποτελούν μέρος του αντικειμενοστραφούς προγραμματισμού και του συναρτησιακού προγραμματισμού. Συγκεκριμένα, εδώ οι μονάδες υλικού δηλώνονται ως συναρτήσεις με εισόδους, και έξοδο την υπομονάδα που περιγράφεται. Η υπομονάδα δεν

συμπεριφέρεται απλά σαν μία σειρά εισόδων και εξόδων, αλλά σαν ένα αντικείμενο το οποίο “περιέχει” τις εξόδους του ως “πεδία,” στα οποία μπορεί να γίνει αναφορά με τη χρήση του ονόματος του αντικειμένου και του πεδίου, όπως γίνεται σε μία αντικειμενοστραφή γλώσσα. Ως αποτέλεσμα, δύναται η αναφορά και η συνδεσμολογία σε ολόκληρες ομάδες συνδέσεων εξόδου, δεδομένου ότι βρίσκονται στο ίδιο αντικείμενο, καθώς επίσης η μεταφορά αναφορών ομάδων εισόδων ανά την ιεραρχία του προγράμματος, δεδομένου πάλι ότι αποτελούν παραμέτρους κατά τις οποίες δημιουργήθηκε το εν λόγω αντικείμενο. Με άλλα λόγια, το “αντικείμενο” είναι ένας τρόπος ομαδοποίησης εισόδων, εξόδων μιας υπομονάδας, της οποίας η συμπεριφορά και η μορφή καθορίζεται από τον τύπο της, δηλαδή της δήλωσης μονάδας της. Περεταίρω, δύναται και η χρήση αντικειμενοστραφών τεχνικών όπως ο πολυμορφισμός με τη χρήση των “διεπαφών” (interfaces), οι οποίες είναι αφηρημένες, ατελής δηλώσεις υποτιθέμενων μονάδων. Οι μονάδες αυτές παράγονται πάντα σε σχέση με κάποια άλλη μονάδα, με βάση των οποίων μπορεί να ολοκληρωθεί η υλοποίηση των αφηρημένων μερών, δοσμένου ότι ο προγραμματιστής έχει δηλώσει τον τρόπο κατά τον οποίο η υλοποίηση ολοκληρώνεται για αυτό το ζευγάρι διεπαφής-μονάδας. Το προϊόν είναι η δυνατότητα δημιουργίας αντικειμένων με την ίδια διεπαφή, αλλά διαφορετική συμπεριφορά κατά τον τύπο του αντικειμένου από το οποίο δημιουργήθηκε το νέο αυτό αντικείμενο. Η δυνατότητα διαφοροποίησης της συμπεριφοράς του αντικειμένου με μία πρότυπη διεπαφή λοιπόν μπορεί να χαρακτηριστεί ως πολυμορφισμός.

(Υποκεφάλαιο 2.2) Σημασιολογία και Δομές

(Ενότητα 2.2.1) Μονάδες (Modules)

Μία μονάδα αντιστοιχεί σε ένα μοντέλο υλικού με συγκεκριμένες εισόδους και εξόδους, και μία συμπεριφορά μεταξύ τους. Αποτελεί τη βασική μονάδα ομαδοποίησης κώδικα στη γλώσσα, και μπορεί να χρησιμοποιηθεί είτε ως υπομονάδα μιας άλλης μονάδας ή ως αντιστοιχία ολόκληρου του περιγραφόμενου κυκλώματος, ρόλο τον οποίο μπορεί να πάρει μονάχα μία μονάδα σε όλο το πρόγραμμα. Μία μονάδα μπορεί να έχει διάφορα χαρακτηριστικά, τα οποία αντιπροσωπεύονται από τα πεδία τα οποία περιέχει. Περεταίρω χαρακτηριστικά εξαρτώνται από τον τύπο της μονάδας, η οποία μπορεί να είναι μια απλή κλασική επώνυμη μονάδα, μία αφηρημένη μονάδα, γνωστή και ως “διεπαφή” (interface), ή μια ανώνυμη μονάδα. Τα μοναδικά χαρακτηριστικά του καθενός θα συζητηθούν παρακάτω.

Λειτουργικά, οι μονάδες χρησιμοποιούνται ως πατρόν για τη δημιουργία αντικειμένων με τα πεδία τα οποία δηλώνονται στη κάθε μονάδα. Τα αντικείμενα αυτά, μοντελοποιημένα όπως δηλώθηκαν από τον τύπο τους, αντιστοιχούν πρακτικά σε μία υλοποίηση του μοντέλου υλικού κατά τον οποίο τυποποιήθηκαν, και της οποίας μπορεί ο χρήστης να εκφράσει τις συνδέσεις μεταξύ των εισόδων, εξόδων, και του περιβάλλοντος του αντικειμένου στο οποίο είναι, αρκεί να αναφέρονται στην εν λόγω δήλωση της μονάδας. Έτσι, ο χρήστης μπορεί να χρησιμοποιήσει την συμπεριφορά ενθυλακωμένη μέσα στο μοντέλο που χρησιμοποιήθηκε ως τύπος του αντικειμένου, δίνοντας του τη δυνατότητα να

δώσει σημασιολογικές συσχετίσεις στη συμπεριφορά του αντικειμένου χωρίς να χρειάζεται να ξέρει την περιεχόμενη λογική απολύτως, απλουστεύοντας την πολυπλοκότητα ανάπτυξης στη γλώσσα.

Μια συνηθισμένη δήλωση μονάδων μοιάζει κάπως έτσι:

```
module FullAdder (a: wire, b: wire, c_in: wire) {
    let z = a ^ b
    public let sum = z ^ c_in
    public let c = (a & b) | (z & c_in)
}
```

Η δήλωση αυτή θα αποτελέσει παράδειγμα και παρακάτω. Στην δήλωση αυτή, δύο μέρη είναι προφανή: η επικεφαλίδα και το κυρίως σώμα. Η επικεφαλίδα αποτελείται από τρία μέρη, την λέξη κλειδί “module” που δηλώνει την έναρξη δήλωσης μιας μονάδας, το όνομα της μονάδας το οποίο μπορεί να χρησιμοποιηθεί ως όνομα τύπου, και τα πεδία εισόδου μέσα σε παρένθεση. Τα δύο τελευταία μέρη, το όνομα της μονάδας και τα πεδία εισόδου, δηλώνουν τα χαρακτηριστικά που απαιτούνται για την αρχικοποίηση ενός αντικειμένου με αυτόν τον τύπο. Το δεύτερο μέρος της όλης δήλωσης είναι το κυρίως σώμα της μονάδας, το οποίο αποτελείται από πεδία και μονάδες-μεθόδους, οι οποίες είναι διαθέσιμες μέσα στην μονάδα αλλά και προαιρετικά από το αρχικοποιημένο αντικείμενο. Η σύνταξη των πεδίων εξηγείται παρακάτω.

Με βάση την παραπάνω δήλωση, ο χρήστης λοιπόν μπορεί πλέων να χρησιμοποιήσει τον τύπο που δημιούργησε με τον παρακάτω τρόπο:

```
let adder = FullAdder (
    a = myWire,
    b = 1’b1,
    c_in = 0’b1,
)
let result = adder.sum
let carry = adder.c
```

Το παραπάνω παράδειγμα αναδεικνύει ένα τρόπο χρήσης της παραπάνω δήλωσης μονάδας μέσα σε μία άλλη μονάδα. Ξεκινά με την αρχικοποίηση ενός αντικειμένου με όνομα “adder” τύπο “FullAdder”, η οποία μέσα στις παρενθέσεις περιέχει τις τιμές που παίρνουν τα πεδία εισόδου. Μετά την αρχικοποίηση, το αντικείμενο χρησιμοποιείται με την σύνταξη

παραπομπής προκειμένου να αποθηκευτούν οι δύο έξοδοι “sum” και “c” του αντικειμένου στα ανάλογα πεδία.

(Ενότητα 2.2.2) Πεδία

Τα πεδία είναι υπομέρη της μονάδας τα οποία δηλώνουν τη συμπεριφορά του αντικειμένου. Δηλώνουν τους μορφισμούς που μπορούν να εφαρμοστούν στο αντικείμενο, οι οποίοι μπορεί να είναι είσοδοι/έξοδοι του, ή μέθοδοι. Όλα τα πεδία χαρακτηρίζονται από ένα τύπο, κατά τον οποίο δύνανται να κρατούν αντικείμενα του ανάλογου τύπου στο αντικείμενο παραγόμενο από την μονάδα στην οποία ανήκει το πεδίο αυτό. Ανάλογα με το που και πως δηλώνονται και τους τροποποιητές πρόσβασης που τους δίνονται, καθορίζεται η κατεύθυνση των δεδομένων που διαχειρίζονται.

- Είσοδοι: δηλώνονται στην εκφραση δήλωσης εισόδων της μονάδας, και μεταφράζονται σε σήματα εισόδων. Σε περίπτωση εισόδου με τύπο υπομονάδας, μεταφράζονται στο σύνολο σημάτων εισόδων της υπομονάδας, αναδρομικά. Δηλώνονται στο μέρος δηλώσεων εισόδων της μονάδας.
- Ιδιωτικά πεδία: τα πεδία αυτά χρησιμοποιούνται μονάχα εντός της μονάδας και δεν εκτίθενται εκτός της μονάδας. Με άλλα λόγια, δεν είναι ούτε σήματα εισόδου, ούτε εξόδου. Δηλώνονται στο κύριο σώμα της μονάδας.
- Δημόσια πεδία: τα πεδία αυτά μεταφράζονται σε σήματα εξόδου και είναι διαθέσιμα για αναφορά σε αυτά χρησιμοποιώντας την ανάλογη σύνταξη στο αντικείμενο με τύπο τη μονάδα στην οποία ανήκει το πεδίο αυτό. Δηλώνονται επίσης στο κύριο σώμα της μονάδας, και διαφοροποιούνται από τα ιδιωτικά πεδία μέσω της λέξης-κλειδιού “public”.

Επώνυμα πεδία

Τα επώνυμα πεδία είναι ο κύριος τρόπος κατά τον οποίο μπορεί κάποιος να γεμίσει μια μονάδα. Πέρα από τον τύπο τους, έχουν και ένα όνομα, κατά το οποίο μπορεί ο χρήστης να αναφερθεί σε αυτά μέσα σε εκφράσεις. Ένα συνηθισμένο επώνυμο πεδίο έχει διαφορετική μορφή ανάλογα με την κατεύθυνση δεδομένων που διαχειρίζεται. Το κύριο μέρος του πεδίου, μοιάζει κάπως έτσι:

```
myField: SomeType = (a & b) | (z & c_in)
```

Αποτελείται από τρία μέρη:

- Το όνομα του πεδίου,
- Τον τύπο του πεδίου, ο οποίος είναι προαιρετικός αν δίνεται η εκφραση,
- Την εκφραση, η οποία είναι προαιρετική αν δίνεται ο τύπος.

Σε περίπτωση που δίνεται η εκφραση αλλά όχι ο τύπος, ο τύπος του πεδίου γίνεται ο ίδιος με τον τύπο της έκφρασης. Σε περίπτωση που δίνεται ο τύπος αλλά όχι η έκφραση, το πεδίο δεν παίρνει αμέσως τιμή. Περεταίρω χαρακτηριστικά μπορεί να απαιτούνται ανάλογα με την κατεύθυνση δεδομένων του πεδίου, συγκεκριμένα:

- Τα πεδία εισόδου αποτελούνται μονάχα από το κύριο μέρος του πεδίου, αρκεί να δηλώνονται στο σωστό μέρος
- Τα ιδιωτικά πεδία δηλώνονται με την λέξη κλειδί “let” στην αρχή της δήλωσής τους.

```
let myField: SomeType = (a & b) | (z & c_in)
```

- Τα δημόσια πεδία δηλώνονται με την ακολουθία λέξεων κλειδιών “public”, “let” στην αρχή της δήλωσής τους.

```
public let myField: SomeType = (a & b) | (z & c_in)
```

Ευρετηριασμένα πεδία

Τα ευρετηριασμένα πεδία δηλώνονται με αριθμούς αντί για ονόματα, και με τους αριθμούς αυτούς μπορεί ο χρήστης να αναφερθεί σε αυτά. Σε αντίθεση με τα επώνυμα πεδία, όλα τα ευρετηριασμένα πεδία σε μία μονάδα πρέπει να έχουν τον ίδιο τύπο, και είναι πάντα δημόσια. Τα ευρετηριασμένα πεδία δηλώνουν το τρόπο κατά τον οποίο το παραγόμενο αντικείμενο από την μονάδα αυτή μπορεί να εκφραστεί σαν πίνακας/παράταξη.

Υπάρχουν δύο τρόποι δήλωσης ευρετηριασμένων πεδίων, ο ένας είναι στατικός, και ανεξάρτητος από τον αριθμό στον οποίο αντιστοιχεί, και ο άλλος είναι ο δυναμικός, με μία μεταβλητή η οποία παίρνει την τιμή του αριθμού πεδίου και μπορεί να χρησιμοποιηθεί μέσα στην εκφραση του πεδίου

```
let [0..63] = 0’b64          // 64 bytes expression

let [r: 64..254] = !r[0]      // 1 byte expression, iterated
```

(Ενότητα 2.2.3) Ανώνυμες μονάδες και μέθοδοι

Οι ανώνυμες μονάδες είναι μονάδες χωρίς όνομα, που μπορούν να χρησιμοποιηθούν σαν εκφράσεις. Οι εκφράσεις αυτές δεν μεταφράζονται σε υλικό μέχρι να κληθούν προς αρχικοποίηση. Συμπεριφέρονται δηλαδή όπως ακριβώς με τις στατικές μονάδες· αποτελούν μονάχα πατρόν για νέα αντικείμενα, τα οποία μεταφράζονται σε υλικό. Η μόνες διαφορές τους από τις στατικές μονάδες είναι ότι από μόνες τους δεν έχουν όνομα, και ότι έχουν πρόσβαση σε όλα τα πεδία των μονάδων που τους περιβάλλουν.

Όταν μία ανώνυμη μονάδα δίνεται ως τιμή σε ένα πεδίο, το πεδίο αυτό ονομάζεται μέθοδος. Οι μέθοδοι είναι ένας πολύ καλός τρόπος ενθυλάκωσης συμπεριφοράς και υλοποίησης δηλωτικού προγραμματισμού. Ακόμη κι αν υλοποιούνται με χρήση ανώνυμων μονάδων, οι μέθοδοι είναι επώνυμες, κατά το όνομα που δίνεται στο πεδίο.

```
module HalfAdder (a: wire, b: wire) {
    public let sum = z ^ b
    public let c = a & b
```

```
    // method
    public let Carry = module (c_in: wire) {
        let z = a ^ b
        public let sum = z ^ c_in
        public let c = (a & b) | (z & c_in)
    }
}
    // ...
    let hAdder = HalfAdder (a = a, b = b)
    let fullAdder = hAdder.Carry(c_in = myCarry)
```

(Ενότητα 2.2.4) Διεπαφές

Οι διεπαφές είναι ο κύριος τρόπος κατά τον οποίο υλοποιείται ο πολυμορφισμός. Είναι πρακτικά αφηρημένες μονάδες με μέρη της συμπεριφοράς τους τα οποία δεν είναι δηλωμένα. Αυτά συμβολίζονται με πεδία τα οποία έχουν δηλωμένο τύπο, αλλά δεν έχουν αρχικοποιηθεί. Οι αφηρημένες αυτές μονάδες μπορούν να ολοκληρωθούν πάντα συναρτήσει ενός άλλου τύπου, όπου ο χρήστης καλείται να ολοκληρώσει τα αφηρημένα πεδία στην δήλωση της διεπαφής. Ο χρήστης μπορεί να υλοποιήσει τη διεπαφή όσες φορές θέλει, αρκεί κάθε φορά να υλοποιείται για διαφορετικούς τύπους. Αυτό λοιπόν καθιστά τις διεπαφές πολυμορφικούς τύπους, καθώς ποτέ δεν είναι γνωστή η απόλυτη συμπεριφορά ενός τύπου διεπαφής χωρίς να είναι γνωστό το από ποιο αντικείμενο προήλθε. Ο τρόπος σύνταξης μιας διεπαφής μελετάται με βάση το ακόλουθο παράδειγμα

```
interface IBinary {
    // abstract fields
    public let a: wire
    public let b: wire
    public let or = a | b
    public let and = a & b
    public let xor = a ^ b
}
// ...
implement IBinary for wire {
```

```
    public let a = this
    public let b = this
}
// ...
let a = 0'b1
let b: IBinary = a
let c = b.and  // 0'b1
```

Στο παραπάνω παράδειγμα δηλώνεται μία διεπαφή με το όνομα “IBinary”, και αργότερα υλοποιείται για τον τύπο “wire”. Η λέξη κλειδί “this” είναι αναφορά το αντικείμενο για το οποίο υλοποιείται η διεπαφή. Φαίνεται, επίσης, πώς μπορεί ένα αντικείμενο τύπου “wire” να μετατραπεί σε IBinary, προκειμένου να χρησιμοποιηθούν τα χαρακτηριστικά του αντικειμένου της διεπαφής, όπως το πεδίο “and”. Η υλοποίηση μπορεί να γίνει και για άλλους τύπους, και θα μπορούσε να αλλάξει την συμπεριφορά του τελικού αντικειμένου τελείως.

ΚΕΦΑΛΑΙΟ 3 Σύστημα Τύπων και Σημασιολογική Ανάλυση

Για τους σκοπούς της εξήγησης της σημασιολογίας της γλώσσας, θα χρησιμοποιηθεί το μοντέλο της θεωρίας κατηγοριών προκειμένου να μοντελοποιηθούν όλες οι διάφορες εκφράσεις. Σύμφωνα με αυτήν, οι κατηγορίες είναι ένα σύνολο μορφισμών πάνω σε αντικείμενα, τους οποίους διέπουν συγκεκριμένους κανόνες[1]. Οι μορφισμοί είναι οι έννοιες που ενώνουν σημασιολογικά τα αντικείμενα. Για τους σκοπούς της μελέτης του συστήματος τύπων, τα σύνολα είναι οι τύποι δεδομένων που μπορεί να χρησιμοποιήσει ένας προγραμματιστής, και οι μορφισμοί είναι όλοι οι διάφοροι τρόποι που μπορεί ο προγραμματιστής να χρησιμοποιήσει προκειμένου να εξάγει ένα διαφορετικό αντικείμενο από το αντικείμενο-πηγή. Η φύση του μορφισμού αυτού εξαρτάται από το μηχανισμό που αντιπροσωπεύει αυτόν τον μορφισμό στη σύνταξη της γλώσσας, ο οποίος μηχανισμός θα σχηματίζει την δική του ξεχωριστή κατηγορία. Σε όλες αυτές τις κατηγορίες, η σύνθεση και η προσεταιριστικότητα εννοείται λόγω του τρόπου που λειτουργεί το σύστημα εκφράσεων.

(Υποκεφάλαιο 3.1) Κατηγορία της Αναφοράς σε Πεδία

Τα πεδία είναι ο απλούστερη κατηγορία που υπάρχει σε αυτή τη γλώσσα. Τελείται με τη χρήση της σύνταξης παραπομπής στο πεδίο. Τα πεδία δηλώνουν σύνθεση, οπότε, η παραπομπή σε αυτό είναι απλά η αναφορά σε μέρος του πρωτότυπου αντικειμένου. Ο

τύπος του προκύπτοντος αντικειμένου εξαρτάται από τον τύπο κατά τον οποίο δηλώθηκε το πεδίο στην δήλωση της μονάδας στην οποία ανήκει. Άρα πρόκειται για μία σχέση που ακολουθεί την σχέση f: X→Y δεδομένου ότι f είναι το πεδίο, X είναι ο τύπος του πρωτότυπου αντικειμένου, και Y είναι ο τύπος του πεδίου. Με βάση την παραπάνω σχέση, είναι προφανές ότι η παραπομπή σε πεδίο μπορεί να είναι σημασιολογικά λανθασμένη μόνο σε περίπτωση αναφοράς σε ανύπαρκτο πεδίο (∄f).

(Υποκεφάλαιο 3.2) Πολυκατηγορία των Μονάδων

Μια πολυκατηγορία (αγγ. multicategory) είναι μια κατηγορία η οποία επιτρέπει μορφισμούς με περισσότερες από μία είσοδο και μια μονάχα[1]. Με άλλα λόγια, επιτρέπει "μορφισμούς πολλαπλής αριότητας". Οι μορφισμοί αυτοί είναι συμβολίζονται με τη σχέση $m:\mathbf{X}\rightarrow Y$, όπου Y είναι το όνομα του συνόλου τιμών στην οποία οδηγεί ο μορφισμός, και $\mathbf{X}$ = (X0, X1, X2, ...) το πεδίο ορισμού πολλαπλής αριότητας του μορφισμού. Με βάση τη παραπάνω σχέση, ο μόνος τρόπος κατά τον οποίο δύναται να είναι λανθασμένος σημασιολογικά ένας πολυκατηρικός μορφισμός είναι εάν το διάνυσμα εισόδου, έστω $\mathbf{I}$, είναι κακώς ορισμένο, λόγω είτε λάθος αριθμού εισόδων ($n_{\mathbf{I}}\neq n_{\mathbf{X}}$) ή λόγω λάθος τύπου εισόδων ($I_n=X_n$ $n<n_{\mathbf{X}}$).

Ο κύριος τρόπος κατά τον οποίο συμβολίζεται ένας μορφισμός πολυκατηργορίας είναι μέσω των μονάδων, η οποία δύναται να παίρνει πλήθος εισόδων. Οι υπόλοιποι τρόποι αποτελούν δευτερογενείς τρόπους που βασίζονται στην κατηγορία των μονάδας. Κατά την αρχικοποίηση, ο χρηστης επικαλείται να δώσει τιμές για όλα τα πεδία εισόδου μιας μονάδας, ανά όνομα, με βάση τα οποία θα εξαρτάται η συμπεριφορά του αντικειμένου που δημιουργείται.

(Ενότητα 3.2.1) Μέθοδοι

Οι μέθοδοι είναι μονάδες που έχουν πρόσβαση στα πεδία της μονάδας στην οποία βρίσκονται. Λειτουργούν ακριβώς όπως οι μονάδες, απλά έχουν σαν περεταίρω είσοδο τα πεδία της περιβάλλουσας μονάδας. Γι' αυτόν τον λόγο δεν μπορούν να αρχικοποιηθούν μεμονωμένα, αλλά μονάχα με σύνταξη παραπομπής. Εξαιρουμένης λοιπόν αυτής της ιδιοτροπίας, οι μέθοδοι λειτουργούν ως τύποι ακριβώς με τον ίδιο τρόπο που λειτουργούν και οι κανονικές μονάδες.

(Ενότητα 3.2.2) Τελεστές εκφράσεων

Οι τελεστές εκφράσεων είναι σύμβολα που χρησιμοποιούνται σε απλές εκφράσεις, ενιαίου ή δυαδικού τύπου, για να εκφράσουν απλές πράξεις όπως αυτές των λογικών πυλών (and/or/not/etc.). Οι τελεστές ενιαίου τύπου χρησιμοποιούνται για μορφισμούς απλών

κατηγοριών, και οι τελεστές δυαδικού τύπου για μορφισμούς δικατηγοριών. Το αποτέλεσμά τους εξαρτάται τόσο από τον τύπο του τελεστή όσο και από τους τύπους δεδομένων των εισόδων της έκφρασης. Και αυτές όμως οι πολύ απλές εκφράσεις υλοποιούνται με τη βοήθεια μονάδων, καθώς ακόμη και η απλούστερη ενιαία ή δυαδική έκφραση μεταξύ πρωτόγονων τύπων απαιτεί υλικό για να υλοποιηθεί, πχ. a | b μπορεί να μεταφραστεί σε μια πύλη OR, η οποία από μόνη της μπορεί να θεωρηθεί ως μια μονάδα υλικού. Με βάση την λογική αυτή, στη γλώσσα περιέχονται ανάλογες διεπαφές για υπερφόρτωση των τελεστών, για κάθε ενιαίο ή δυαδικό τελεστή. Οι διεπαφες αυτές δημιουργούνται όταν για ένα ζευγάρι τύπων χρησιμοποιηθεί ο ανάλογος τελεστής, της οποίας εκφρασης το αποτέλεσμα είναι το αντικείμενο της διεπαφής. Σε περίπτωση που για το ζευγάρι τύπων αυτό δεν υπάρχει υλοποίηση της διεπαφής τελεστή, η εκφραση θεωρείται λανθασμένη.

(Ενότητα 3.2.3) Ανώνυμες μονάδες και τύποι Λάμδα

Ένα ακόμη χαρακτηριστικό στο σύστημα τύπων είναι οι τύποι λάμδα. Οι τύποι αυτοί δεν χαρακτηρίζουν δεδομένα, άλλα τις ίδιες τις μονάδες. Ο σκοπός είναι η δημιουργία μονάδων ανώτερης τάξης[2], δηλαδή μονάδες που μπορούν να πάρουν ως εισόδους άλλες μονάδες, ή/και να έχουν σαν εξόδους μονάδες, γνωστές και ως μέθοδοι. Οι τύποι λάμδα έχουν συγκεκριμένη σύνταξη στη γλώσσα η οποία μοιάζει κάπως έτσι:

```
let higherOrder: (a: A, b: B): T
```

Το παραπάνω πεδίο είναι ανώτερης τάξης καθώς ο τύπος του είναι τύπος λάμδα, δηλαδή μπορεί να περιέχει μία μονάδα, και να χρησιμοποιηθεί επίσης όπως ακριβώς μια μονάδα. Η ανάλογη έκφραση άλγεβρας λάμδα θα ήταν λA.λB.T, της οποίας οι είσοδοι είναι τύπου A και τύπου B και επιστρέφεται ένα αντικείμενο τύπου T. Ο Τύπος T θα πρέπει να είναι πάντα πολυμορφικός, καθώς οι απλοί μοναδικοί τύποι μπορούν να προέλθουν μονάχα από μια μονάδα. Ένα παράδειγμα χρήσης λάμδα τύπων θα ήταν το παρακάτω:

```
trait T { }

module MyModule (a: wire, b: wire) { }

implement T for MyModule { }

let HigherOrder: (a: wire, b: wire): T = MyModule

let obj: T = HigherOrder (a = 1, b = 0)
```

Σε πολλές περιπτώσεις δεν υπάρχει ανάγκη να δημιουργηθούν ολόκληρες κανονικές μονάδες και να υλοποιηθούν διεπαφές μονάχα για να χρησιμοποιηθούν μία φορά σε ένα πεδίο ανώτερης τάξης. Γι' αυτόν τον λόγο, υπάρχει η σύνταξη των ανώνυμων μονάδων, την οποία μπορεί να χρησιμοποιήσει ο προγραμματιστής προκειμενου να δημιουργήσει ένα πεδίο ανώτερης τάξης χωρίς να δημιουργήσει μία νέα στατική μονάδα εξ' ολοκλήρου.

Επίσης, οι ανώνυμες μονάδες έχουν πρόσβαση σε όλα τα πεδία εξωτερικά αυτών. Γι' αυτό πολλές φορές μπορούν να κάνουν τον κώδικα πολύ πιο συνοπτικό και διαβάσιμο. Για παράδειγμα, το παραπάνω παράδειγμα χρήσης τύπων λάμδα μπορεί επίσης να υλοποιηθεί έτσι:

```
trait T { }
let HigherOrder = module (a: wire, b: wire): T { }
let obj: T = HigherOrder (a = 1, b = 0)
```

(Υποκεφάλαιο 3.3) Κατηγορία των Διεπαφών

Η κατηγορία των διεπαφών ορίζει μία σημασιολογικά κρίσιμο ρόλο στη γλώσσα καθώς χρησιμοποιείται για να αντιπροσωπεύσει πολυμορφικές σχέσεις. Το καλούπιασμα (αγγ. Casting) είναι ο μορφισμός της κατηγορίας, και ορίζεται ως ο τρόπος κατά τον οποίο δύναται η μετατροπή του τύπου του αντικειμένου σε έναν άλλον. Σε αντίθεση με την παραπομπή σε πεδίο, το καλούπιασμα δεν αναφέρεται απαραίτητα σε μέρος της σύνθεσης του αντικειμένου, αλλά σε πιθανότατα ένα νέο, τελείως διαφορετικό αντικείμενο, το οποίο εξαρτάται στο πρωτότυπο διαδικαστικά και όχι συνθεσιακά. Οι μορφισμοί, ακολουθούν την σχέση $m: X \to Y$, με m τη διαδικασία μετατροπής του αντικειμένου, X τον τύπο του πρότυπου αντικειμένου και Y τον πολυμορφικό τύπο του προκύπτοντος αντικειμένου. Η διαδικασία μετατροπής m διέπεται από την ύπαρξη υλοποίησης της της διεπαφής Y για τον τύπο X, και είναι μοναδική. Ο μόνοι τρόποι που μπορεί ένα καλούπιασμα να καταστεί σημασιολογικά λανθασμένο είναι εάν ο τύπος-στόχος δεν υπάρχει ($\nexists Y$) ή εάν είναι αδύνατη η μετατροπή ενός αντικειμένου τύπου X σε τύπο Y ($\nexists m$).

(Ενότητα 3.3.1) Πολυμορφισμός

Ο πολυμορφισμός είναι η δυνατότητα της αλλαγής της συμπεριφοράς ενός τύπου με βάση τον μορφισμό από τον οποίο δημιουργήθηκε. Είναι μία έννοια ζωτικής σημασίας στον αντικειμενοστραφή προγραμματισμό[3], και χρησιμοποιείται κυρίως για επαναχρησιμοποίηση κώδικα μέσω της ολοκλήρωσης ή του εμπλουτισμού της συμπεριφοράς του σε συγκεκριμένα σημεία. Οι πολυμορφικοί τύποι δεν είναι ακριβώς τύποι, αλλά πρακτικά συναρτήσεις τύπων, με πεδίο ορισμού το συνολο των τύπων που έχουν μορφισμό "υλοποίησης" προς τον τύπο αυτό, και μία σχέση ένα-προς-ένα με τον τύπο που προκύπτει την οποία δηλώνει ο προγραμματιστής, που ακολουθεί την ακόλουθη σχέση: $m: X \to Y(X)$. Η σχέση αυτή σχηματίζει συνάρτηση γιατί ως προς την συμπεριφορά, $X \neq Z \ \forall \ Y(X) \neq Y(Z)$, και μάλιστα είναι ένα-προς-ένα γιατί $Y(X) \neq Y(Z) \ \forall \ X \neq Y$. Παρ' όλα αυτά, το σύστημα τύπων κατά τη σημασιολογική ανάλυση δεν διακρίνει διαφορά τύπου μεταξύ δύο διαφορετικών

υλοποιήσεων ενός πολυμορφικού τύπου, καθώς απαιτείται από όλες τις υλοποιήσεις η χρήση της ίδιας ακριβώς διεπαφής. Καθώς λοιπόν η διεπαφή είναι η ίδια και τα πεδία είναι αμετάβλητα, μπορούν να εκφραστεί κώδικας με βάση αυτής της διεπαφής, χωρίς τη γνώση της ακριβής συμπεριφοράς που μπορεί να έχει αυτή η διεπαφή. Ο συγκεκριμένος τύπος πολυμορφισμού λύνεται στο επίπεδο της μετάφρασης, και διαφοροποιεί τις μονάδες που τους χρησιμοποιούν σε εκδοχές για κάθε μία από τις πολυμορφικές κατηγορίες που χρησιμοποιήθηκαν σε αυτήν. Αυτή η διαφοροποίηση τις καθιστά και αυτές πολυμορφικές, όμως εδώ δεν χρειάζεται να υλοποιηθεί η νέα συμπεριφορά ρητά, αλλά εννοείται από την αφηρημένη δήλωση η οποία περιέχει τον πολυμορφικό τύπο.

(Ενότητα 3.3.2) Αφαίρεση Συμπεριφοράς και Διεπαφές

Ο τρόπος κατά τον οποίο ο πολυμορφισμός υλοποιείται στην γλώσσα αυτή είναι μέσω των διεπαφών οι οποίες δηλώνουν τις απαιτήσεις της διεπαφής όλων των τύπων στο σύνολο τιμών του συγκεκριμένου πολυμορφικού τύπου. Ο χρήστης τώρα μπορεί να χρησιμοποιήσει το όνομα της διεπαφής αυτής ως τύπο οπουδήποτε, αλλά δεν μπορεί να τον αρχικοποιήσει όπως θα αρχικοποιούσε μία μονάδα.

```
interface IExample {
    // abstract fields
    public let a: wire
    public let not = !a
}
// ...
module UsesIExample (ex: IExample) {
    public out = ex.a
}
```

∃ IExample: Iexample(T) ∈ Program, T ∈ C

∃ UsesIExample: UsesIExample(IExample(T)) ∈ C∀ T ∈ {T ∈ C | ∃ Iexample(T)}

(Ενότητα 3.3.3) Υλοποίηση Διεπαφών

Προκειμένου να γεμίσει ένα πεδίο με πολυμορφικό τύπο, θα πρέπει να δηλώσει μία μονάδα η οποία θα αντιπροσωπεύει την τιμή της συνάρτησης του πολυμορφικού τύπου για έναν άλλον τύπο. Αυτή η δήλωση λέγεται “υλοποίηση”. Η υλοποίηση απαιτεί από τον προγραμματιστή να υλοποιήσει όλα τα πεδία χωρίς υλοποίηση που δηλώνονται στη δήλωση της διεπαφής. Κάθε υλοποίηση που δεν έχει υλοποιήσει όλα αυτά τα πεδία θεωρείται λανθασμένη.

```
implement IExample for wire {  // module “m”
     public let a = this
}
```

IExample(wire) = m

(Ενότητα 3.3.4) Καλούπιασμα

Αφού δηλωθεί η υλοποίηση από τον τύπο X στον πολυμορφικό τύπο Y, δύναται η δημιουργία ενός αντικειμένου τύπου Y από ένα αντικείμενο τύπου X μέσω καλουπιάσματος.

```
let uie = UsesIExample(ex = 1’b1)
let not = uie.out            // 1’b0
```

uie ∈ UsesIExample(IExample(wire))

ΚΕΦΑΛΑΙΟ 4 Χρήση της γλώσσας

Όλα αυτά τα χαρακτηριστικά που προσφέρει η γλώσσα στον προγραμματιστή προσφέρουν αξία στην εμπειρία προγραμματισμού μονάχα εν ύπαρξη ορίων και κανόνων, καθώς επίσης και ήδη υπάρχοντων εργαλείων τα οποία μπορούν να εμπλουτίσουν την εμπειρία του χρήστη. Μελετούνται λοιπόν μερικοί άτυποι κανόνες και προσχήματα τα οποία μπορούν να μεγιστοποιήσουν την ποιότητα της εμπειρίας του προγραμματιστή αν παρθούν υπ’ όψιν, καθώς επίσης μερικές υπογραφές κώδικα που μπορούν να υλοποιηθούν σε μία πρότυπη βιβλιοθήκη.

(Υποκεφάλαιο 4.1) Συμβατική Μορφοποίηση Κώδικα

Υπάρχουν μερικές άτυπες συμβάσεις οι οποίες είναι χαρακτηριστικές στη γλώσσα. Αυτές οι συμβάσεις κάνουν την ανάγνωση κώδικα ευκολότερη σε προγραμματιστές που πιθανότατα δεν έχουν ξαναδιαβάσει το συγκεκριμένο κομμάτι κώδικα, όχι μόνο ελαχιστοποιώντας την πιθανή ενόχλησή τους από τη μορφοποίηση του κώδικα, αλλά επίσης επιτρέποντας σε αυτήν να προσφέρει χρήσιμες πληροφορίες κατά την ανάγνωση. Οι κανόνες αυτοί δεν είναι απαραίτητοι να ακολουθηθούν κατά γράμμα, και μπορεί κάποιος χρήστης ή ομάδα χρηστών να αποφασίσει να χρησιμοποιήσει ένα διαφορετικό στυλ της αρεσκείας του, αλλά με γνώση και υπευθυνότητα ότι όσο αποφεύγει τους παρακάτω κανόνες μπορεί να προκαλέσει σύγχυση σε αναγνώστες που διαβάζουν τον εν λόγω κώδικα για πρώτη φορά.

(Ενότητα 4.1.1) Ονομασία Αναγνωριστικών

Η ονομασία των δομών στη γλώσσα διαφοροποιείται κατά το στυλ της προκειμένου να ξεκαθαριστεί ο τύπος της δομής (μονάδα/πεδίο), καθώς επίσης και τον τρόπο χρήσης της δομής αυτής. Συγκεκριμένα:

- Όλες τις μονάδες, διεπαφές, και πεδία ανώτερης τάξης ονοματίζονται κατά το στυλ "PascalCase": καθε λέξη ξεκινάει με κεφαλαίο

```
module MyModule { }

trait T { }

let HigherOrder = module (): T { }
```

- Όλα τα πεδία που χρησιμοποιούνται ως σταθερές για τον κώδικα ονοματίζονται κατά το στυλ "CAPITAL_SNAKE_CASE": όλες οι λέξεις είναι σε κεφαλαία, και χωρίζονται με " τον χαρακτήρα "_"

```
let SIZE_OF_SOMETHING = 5
```

- Όλα τα υπόλοιπα πεδία ονοματίζονται κατά "camelCase": κάθε λέξη ξεκινάει με κεφαλαίο εκτός της πρώτης.

```
let simpleField = a | b
```

(Ενότητα 4.1.2) Θέση και απόσταση

Κάθε ανοιχτό μπλοκ κώδικα σε άγκιστρα, παρενθέσεις ή αγκύλες μορφοποιείται κατά τους ακόλουθους κανόνες:

1. Προηγείται από τουλάχιστον ένα κενό από τον κώδικα πριν του μπλοκ. Τα άγκιστρα επίσης ακολουθούνται από ένα κενό από τον κώδικα μετά του μπλοκ.
2. Σε περίπτωση που ο κώδικας χωράει σε μία γραμμή, ο ενδότερος κώδικας έχει απόσταση ένα κενό από τα άγκιστρα/παρενθέσεις/αγκύλες
3. Τα άγκιστρα/παρενθέσεις/αγκύλες είναι σε ξεχωριστές γραμμές από τον κώδικα που περιέχουν εάν το μπλοκ χρειάζεται να πάρει παραπάνω από μια γραμμή, και ο ενδότερος κώδικας προηγείται από έναν χώρο τεσσάρων κενών.
4. Σε περίπτωση που πολλά πολύγραμμα μπλοκ περιέχονται το ένα μέσα στο άλλο αναδρομικά, ο χώρος κατά τον οποίο προηγείται ο ενδότερος κώδικας στοιβάζεται με τον χώρο του αμέσως εξωτερικού μπλοκ. Πχ, για ένα μπλοκ μέσα σε άλλα δύο, ο χώρος που απαιτείται είναι $4 \cdot 3 = 12$ κενά.

(Υποκεφάλαιο 4.2) Μοτίβα

Τα μοτίβα είναι τρόποι κατά τους οποίους μορφοποιείται η βάση του κώδικα προκειμένου να προσφέρουν ευκολότερη κατανόηση, να απλοποιήσουν τον αρχιτεκτονική της εφαρμογής και να μειώσουν τα λογικά λάθη μέσω μιας οικείας αρχιτεκτονικής η οποία αποσφαλματίζεται ευκολότερα. Τα μοτίβα που αναφέρονται εδώ προτείνονται προς χρήση με τη γλώσσα.

(Ενότητα 4.2.1) Ενθυλάκωση

Η ενθυλάκωση ακολουθεί την “αρχή της απόκρυψης δεδομένων”[4] ευαίσθητου χαρακτήρα, και της διάθεσής τους μέσω διόδων που διασφαλίζουν την ακεραιότητα τους. Αυτό είναι δυνατό σε αυτή τη γλώσσα μέσω της δημιουργίας υλοποίησης διεπαφών και μεθόδων για κάθε πιθανή θεμιτή και έγκυρη χρήση ενός πεδίου, οι οποίες απαρτίζουν τον μοναδικό τρόπο πρόσβασης σε αυτά τα δεδομένα. Μέσω λοιπόν του περιορισμού της πρόσβασης στα δεδομένα δίνεται η δυνατότητα διασφάλισης της εγκυρότητάς τους.

(Ενότητα 4.2.2) Ενδολειτουργικές, Μονοειδής Μονάδες (αγγ. Monads)

Οι κατηγορίες οι οποίες έχουν μονάχα ένα ταυτοτικό αντικείμενο και διάφορες πράξεις με αυτό λέγονται μονοειδής, και οι κατηγορίες των οποίων όλοι οι μορφισμοί καταλήγουν στο ίδιο αντικείμενο λέγονται ενδολειτουργικές.[1] Μία κατηγορία μπορεί να είναι και ενδολειτουργική και μονοειδής. Οι κατηγορίες αυτές μπορούν να χρησιμοποιηθούν για να “καλύψουν” ένα αντικείμενο προκειμένου να προσφέρουν ενθυλακωμένη λογική στα περιεχόμενά τους. Η ίδια λογική μπορεί να χρησιμοποιηθεί στον ίδιο τον τύπο της μονάδας αυτής. Αυτή η πολλαπλή κάλυψη μπορεί να χρησιμοποιηθεί προκειμένου να εξηγήσει δομές όπως πίνακες, μεταξύ άλλων. Η υλοποίηση των monad μονάδων είναι δυνατή μέσω των διεπαφών.

(Ενότητα 4.2.3) Δηλωτικός Προγραμματισμός

Ο δηλωτικός προγραμματισμός είναι ένα μοτίβο κατά το οποίο ο κώδικας της λογικής είναι κρυμμένος και χωρισμένος σε μικρά απλά μέρη, τα οποία μετά μπορούν να χρησιμοποιηθούν ως αφαιρέσεις και να συνδυαστούν προκειμένου να περιγράψουν περίπλοκη συμπεριφορά. Η λογική πίσω από αυτό το μοτίβο είναι ότι εάν είναι γνωστή η ακεραιότητα και ορθότητα ενός μπλοκ λογικής δεν χρειάζεται απαραίτητα να εξεταστεί κατά τη διαδικασία της αποσφαλμάτωσης. Ο κώδικας έτσι χωρίζεται σε μικρά μέρη που αποσφαλματώνονται εύκολα, και αυτά συνδυάζονται σε εκφράσεις συνδυασμού χρησιμοποιώντας μονάχα τα εργαλεία που προσφέρονται από τα μέρη αυτά. Το μοτίβο αυτό γίνεται ευκολότερο με χρήση αφαιρετικών μηχανσιμών όπως των διεπαφών, ή και μοτίβων όπως της ενθυλάκωσης ή των ενδολειτουργικών μονοειδών μονάδων. Για παράδειγμα, δωσμένου ότι έχουμε μονάδες για έναν φάκελο καταχωρητών, μία μνήμη τυχαίας πρόσβασης, μία λογική μονάδα και ένα κύκλωμα αποκωδικοποίησης εντολών, μπορούμε να τα συνδυάσουμε κάπως έτσι:

```
module CPU {
     public let decoder = Decoder (
          readFrom = memory
```

```
    )
    public let memory = Memory (
        SIZE = 2048,
        input = decoder.MemoryInput(alu, regs)
    )
    public let regs = Registers (
        operands = decoder,
        pcInput = decoder.pc
    )
    public let alu = ALU (
        operands = regs.outputs
        operation = decoder.aluOp
    )
}
```

Είναι ευδιάκριτο πως ο παραπάνω κώδικας δεν έχει ξεκάθαρη λογική, αλλά χρησιμοποιεί ξεχωριστά μέρη τα οποία έχουν το καθένα τη δική τους λογική, καθώς επίσης λογική μεταξύ τους, η οποία μπορεί να αφαιρείται με χρήση διεπαφών.

ΚΕΦΑΛΑΙΟ 5 Υλοποίηση του Μεταγλωττιστή

Ο μεταγλωττιστής της γλώσσας έχει ως σκοπό συγκεκριμένα την μετάφραση του κώδικα όχι σε κάποια δυαδική ή εκτελέσιμη μορφή, αλλά σε μία άλλη γλώσσα περιγραφής υλικού προκειμένου να δύναται η χρήση ήδη υπαρχόντων εργαλειών με τα προγραμματα που γράφει ο χρήστης στη γλώσσα Iris. Η γλώσσα που έχει επιλεχθεί ως στόχος μετάφρασης σε αυτή την υλοποίηση είναι η Verilog. Ο μεταγλωττιστής όμως δεν μεταφράζει ακριβώς μία προς μία τις έννοιες της γλώσσας σε Verilog, αυτό θα ήταν αδύνατο λόγω των διαφορών προγραμματιστικών προτύπων που διέπουν τις δύο αυτές γλώσσες. Ο σκοπός το μεταγλωττιστή λοιπόν, πέραν της λεκτικής, γραμματικής και σημασιολογικής ανάλυσης, πρέπει να λυνει τις πολυμορφικές εκφράσεις και τις εκφράσεις εξόδου σε σωρεία μονάδων και πεδίων Verilog, μία για κάθε εκδοχή που χρησιμοποιείται, προκειμένου το πρόγραμμα Verilog να συμπεριφέρεται με τον αναμενόμενο τρόπο. Για την υλοποίηση του μεταγλωττιστή χρησιμοποιείται η γλώσσα Rust, μία γλώσσα χαμηλού επιπέδου που όμως διασφαλίζει μέσω των αυστηρών κανόνων σημασιολογίας της την ασφάλεια κατά τον χρόνο εκτέλεσης από προβλήματα όπως προβληματική χρήση μνήμης, μεταξύ άλλων, καθώς επίσης

προσφέρει εκτενείς δυνατότητες αφαιρετικού προγραμματισμού που ακολουθούν τις αρχές του συναρτησιακού, αντικειμενοστραφή και δηλωτικού προγραμματισμού, καθώς και του πολυμορφισμού.

(Υποκεφάλαιο 5.1) Εργαλεία Παραγωγής Λεκτικού και Γραμματικού Αναλυτή

Η rust, μία σχετικά νέα γλώσσα προγραμματισμού, έχει παραδόξως μαζέψει μία μεγάλη κοινότητα από γύρω της λόγω των μοναδικών της ιδεών. Γι' αυτόν τον λόγο, υπάρχει σωρεία ποιοτικών και μοντέρνων βιβλιοθηκών και λοιπών εργαλειών τα οποία είναι φτιαγμένα για χρήση στη γλώσσα. Περεταίρω, όλα τους βρίσκονται πακεταρισμένα σε μία συγκεκριμένη μορφή (ονόματι "crate"), και είναι εύκολο να ενσωματωθούν σε κάθε πρότζεκτ το οποίο χρησιμοποιεί rust χάρη στο έξοχο σύστημα χτισίματος της, το "cargo", και της σελίδας αποθήκης "crates.io" από την οποία μπορεί ο καθένας να επιλέξει τα εργαλεία που του ταιριάζουν.[6] Τα εργαλεία λοιπόν που χρησιμοποιούνται για τον σκοπό της λεκτικής ανάλυσης και του γραμματικού αναλυτή είναι δύο, το "Lalrpop"[7] για γραμματική ανάλυση και το "Logos"[8] για λεκτική ανάλυση. Τα δύο αυτά εργαλεία λειτουργούν μαζί έξοχα μέσω των αφαιρετικών τους μηχανισμών, οι οποίοι διέπονται και υλοποιούνται μέσω των διεπαφών χαρακτηριστικών (αγγ. "trait") της rust.[6][7][8]

(Ενότητα 5.1.1) "Logos"

Το πακέτο logos προσφέρει εργαλεία για την δημιουργία πολύ γρήγορων αναλυτών μέσω των μηχανισμών μακροεντολών της rust. Έτσι, ο αναλυτής γράφεται μέσα σε κανονικό, συμβατικό αρχείο rust, και όχι σε ξεχωριστό αρχείο με διαφορετική σύνταξη. Αυτό είναι χρήσιμο γιατί απλουστευει την αποσφαλμάτωση κατά τον χρόνο μεταγλώττισης. Λειτουργεί με τη δήλωση μιας απαρίθμησης, της οποίας η κάθε παραλλαγή αντιστοιχεί σε μία λέξη-στόχο του λεκτικού αναλυτή και μπορεί να περιέχει δεδομένα τα οποία χαρακτηρίζουν τη λέξη (εξού λοιπόν γιατί η απαρίθμηση είναι μονοειδής). Σε κάθε παραλλαγή αντιστοιχείται μία "ταμπέλα" ("attribute") η οποία περιέχει μία κανονική έκφραση η οποία αντιστοιχεί σε αυτή τη παραλλαγή, και μία αναφορά στη συνάρτηση η οποία εξάγει τα περιεχόμενα δεδομένα από το κείμενο της λέξης σε περίπτωση που η συγκεκριμένη παραλλαγή είναι μονοειδής. Δοσμένης λοιπόν και καλώς ορισμένης της εν λόγω απαρίθμησης και των συναρτήσεων εξαγωγής δεδομένων, ο λεκτικός αναλυτής δημιουργείται κατά τη φάση της μετάφρασης μέσω του μηχανισμού μακροεντολών της rust, ο οποίος διέπεται από τις εν λόγω ταμπέλες.

(Ενότητα 5.1.2) "Lalrpop"

Το lalrpop είναι ένας μοντέρνος γεννήτορας γραμματικών αναλυτών με τη δική του σύνταξη, η οποία μεταφράζεται σε rust. Παραδόξως με το όνομά του, δεν παράγει γραμματικές LALR, αλλά LR(1). Προσφέρει στους χρήστες του την επιλογή μεταξύ ενός

αυτόματα δημιουργημένου λεκτικού αναλυτή με βάση regex, ή τη χρήση ενός εξωτερικού λεκτικού αναλυτή μέσω μια δήλωσης διεπαφής χαρακτηριστικού επαναληπτικότητας (αγγ. “iterator trait”) προκειμένου να διαβάζει μία μία τις λεκτικές μονάδες του λεκτικού αναλυτή της αρέσκειας του χρήστη. Περεταίρω προσφέρει το ισχυρό σύστημα τύπων και τη διασφάλιση της rust σε κάθε κομμάτι διαδικασιακού κώδικα. Τέλος, προσφέρει τη δυνατότητα της δημιουργίας “μακροεντολών”· πρακτικά πρότυπα κανόνων τα οποία παραμετροποιούνται με βάσει αφηρημένους κανόνες ως παραμέτρους, τους οποίους καλείται ο χρήστης να ορίσει κατά τη χρήση του προτύπου ως κανόνα συναρτήσει των κανόνων επιχειρημάτων που επιλέγει. Το χαρακτηριστικό αυτό προσφέρει τη δυνατότητα χρήσης προγραμματιστικών μοτίβων τα οποία μπορούν να χρησιμοποιηθούν για σκοπούς όχι μόνο λακωνικότητας, αλλά και λύσης συγκρούσεων shift/reduce ή reduce/reduce, ή ακόμη και δήλωση προτεραιότητας εκφράσεων.

(Υποκεφάλαιο 5.2) Λεκτική Ανάλυση

Ο λεκτικός αναλυτής είναι γενικά απλός, με την εξαίρεση του μηχανισμού αυτόματου χωρισμού γραμμών. Ο μηχανισμός αυτός δύναται να χωρίζει γραμμές δηλώσεων χωρίς την απαίτηση από τον χρήστη να λέει που ακριβώς τελειώνει η γραμμή, και χωρίς επίσης να χωρίζει τις εκφράσεις στα δύο δημιουργώντας γραμματικά λάθη. Λειτουργεί με την παρεμβολή της λεκτικής μονάδας “;” στο τέλος των γραμμών που ικανοποιούν συγκεκριμένες προϋποθέσεις. Οι προϋποθέσεις αυτές εξαρτώνται από τις λεκτικές μονάδες που προηγούνται ή ακολουθούν μία λεκτική μονάδα νέας γραμμής “\n”. Συγκεκριμένα, είναι:

1. Δεν ακολουθείται ή προηγείται λεκτικής μονάδας τελεστή έκφρασης. Τέτοιες μονάδες είναι οι “+”, “-”, “*”, “/”, “%”, “&&”, “&”, “||”, “|”, “^^”, “^”, “!”, “~” και “.”
2. Δεν ακολουθείται από τη λεκτική μονάδα “}” και δεν προηγείται από τη λεκτική μονάδα “{“
3. Δεν βρίσκεται μέσα σε παρενθέσεις ή αγκύλες, τα άγκιστρα εξαιρούνται αυτού του κανόνα ακόμα κι αν βρίσκονται μέσα σε παρενθέσεις ή αγκύλες.

Εάν ισχύουν και οι τρεις αυτοί κανόνες ταυτόχρονα, εισάγεται η λεκτική μονάδα “;” προκειμένου να χωριστεί η γραμμή από την επόμενη. Είναι προφανές ότι οι κανόνες αυτοί βασίζονται βαθιά στη γραμματική της γλώσσας αυτής και δεν θα λειτουργήσουν σε οποιαδήποτε γλώσσα.

Η υλοποίηση της παραπάνω τεχνικής βασίζεται πάνω σε έναν συμβατικό λεκτικό αναλυτή Logos και τον επαναλήπτη του, μέσω της υλοποίησης ενός επαναλήπτη που καλύπτει τον προαναφερθέντα αναλυτή και ελέγχει τις παραπάνω προϋποθέσεις. Συγκεκριμένα, υλοποιείται ως ένα ρυθμιστικό απόθεμα (αγγ. “buffer”) τριών λεκτικών μονάδων, των οποίων η δεύτερη θεωρείται η τρέχουσα. Σε κάθε επανάληψη, και οι τρεις αυτές λεκτικές μονάδες χρησιμοποιούνται προκειμένου να αποφασιστεί εάν η τρέχουσα μονάδα αντικατασταθεί με την “;”, μαζί με το σύστημα μέτρησης επιπέδων παρενθέσεων/αγκύλων το οποίο αναλαμβάνει τη προαναφερθείσα προϋπόθεση 1. Το σύστημα αυτό μετράει τα επίπεδα παρενθέσεων ή αγκυλών στα οποία βρίσκεται η τρέχουσα

μονάδα, προσθαφαιρώντας έναν μετρητή επιπέδων, οποίος βρίσκεται σε μία στοίβα για τα επίπεδα άγκιστρων. Η στοίβα αυτή είναι μέρος της δομής του επαναλήπτη. Συγκεκριμένα, αν η κορυφή της στοίβας έχει τον αριθμό 0, επιτρέπεται η εισαγωγή της μονάδας ";", ενώ εάν όχι δεν επιτρέπεται. Ο αριθμός αυτός αυξάνεται κατά ένα με κάθε μονάδα "(" ή "[", και μειώνεται κατά ένα με κάθε μονάδα ")", "]". Επίσης, κάθε μονάδα "{" προσθέτει στη στοίβα τον αριθμό 0, επιτρέποντας προσωρινά ξανά την εισαγωγή ";", και κάθε μονάδα "}" αφαιρεί από τη κορυφή της στοίβας. Το παρακάτω παράδειγμα εξηγεί τη λειτουργία του συστήματος

Λεκτική Μονάδα	Πράξη	Στοίβα			Απόφαση (κορυφή == 0)
(	+1	1			όχι
[	+1	2			όχι
{	>>	2	0		ναι
{	>>	2	0	0	ναι
(	+1	2	0	1	όχι
)	-1	2	0	0	ναι
}	<<	2	0		ναι
(	+1	2	1		όχι
)	-1	2	0		ναι
}	<<	2			όχι
]	-1	1			όχι
)	-1	0			όχι

(Υποκεφάλαιο 5.3) Γραμματική Ανάλυση

Η γραμματική ανάλυση χρησιμοποιεί εκτεταμένα το χαρακτηριστικό των μακροεντολών[7] του εργαλείου lalrpop, για διάφορους λόγους. Η χρήση τους όμως ακολουθεί μερικά μοτίβα τα οποία όχι μόνο διευκολύνουν τη κατανόηση του προβλήματος που λύνουν, αλλά μπορούν να αναπαραχθούν και σε άλλες εφαρμογές.

(Ενότητα 5.3.1) Μακροεντολές Λιστών και συλλογής σφαλμάτων

Οι μακροεντολές λιστών παράγουν κανόνες που για ένα κανόνα-επιχείρημα R αντιστοιχίζουν πολλές φορές τον κανόνα R με ένα διαχωριστικό κανόνα-επιχείρημα Sep. Ανάλογα την υλοποίηση της μακροεντολής, μπορεί να ανέχεται τα δοσμένα διαχωριστικά

πολλές φορές ανάμεσα σε δύο διπλανές εμφανίσεις του κανόνα R, ή ακόμη και εμφανίσεις των διαχωριστικών μετά ή πριν όλων των κανόνων. Οι μακροεντολές αυτές είναι πολύ χρήσιμες στη γραμματική της γλώσσας, καθώς χρησιμοποιούνται εκτεταμένα σε πολλά σημεία της γραμματικής, όπως των λιστών εισόδων που χωρίζονται με κόμμα, ή των πεδίων μέσα σε μια μονάδα, που χωρίζονται με ";". Λόγω της εκτεταμένης χρήσης τους, οι μακροεντολές λιστών είναι υπεύθυνες για συλλογή συντακτικών λαθών και την ανάνηψη από αυτά. Ο αλγόριθμος κατά τον οποίο ακολουθεί είναι απλός: εάν αντί για R ή Sep πάρουμε ένα συνακτικό λάθος, αυτό αναφέρεται στη λίστα λαθών και συνεχίζεται η συντακτική ανάλυση της λίστας αδιαφορώντας για το λάθος που μόλις βρέθηκε. Έτσι, καθώς οι λίστες είναι αναδρομικά τοποθετημένες στη γραμματική, στη χειρότερη περίπτωση θα απορριφθεί το μικρότερο δυνατό κομμάτι κώδικα στην πιο εσωτερική λίστα της αναδρομής, και η ανάνηψη θα συνεχίσει με την εν λόγω λίστα μέχρι να μπορεί να γίνει μείωση (αγγ. reduce) στον από πάνω κανόνα.

(Ενότητα 5.3.2) Μακροεντολές Μονοειδών

Οι μακροεντολές μονοειδών είναι αρκετά απλές, καθώς δεν προσφέρουν κάτι παραπάνω στην γραμματική προς αντιστοίχηση, δηλαδή για έναν κανόνα-επιχείρημα R αντιστοιχούν μονάχα τον ίδιο τον κανόνα R. Ο τρόπος που διαφοροποιούνται από τον κανόνα αυτόν είναι στις διαδικαστικές λειτουργίες που κάνουν, που συγκεκριμένα είναι το τύλιγμα του αποτελέσματος του κανόνα R σε μία μονοειδή δομή. Χρησιμοποιείται για την εύκολη μετατροπή τύπων μέσω του δηλωτικού προγραμματισμού.

(Ενότητα 5.3.3) Μονοειδείς Μακροεντολές Προτεραιότητας

Το εργαλείο Lalrpop δεν έχει εγγενή τρόπο δήλωσης της προτεραιότητας και της προσεταιριστικότητας των κανόνων. Γι' αυτόν τον λόγο, χρησιμοποιείται ένα μονοειδές μοτίβο προκειμένου όχι μόνο να επιλυθούν προβλήματα συγκρούσεων κανόνων, αλλά και την ξεκάθαρη δήλωση της προτεραιότητας αυτών. Γι' αυτό, κάθε κανόνας ο οποίος έχει κάποια προτεραιότητα (συνήθως κανόνες εκφράσεων) γίνεται μακροεντολή με ένα κανόνα επιχείρημα "Next", το οποίο χρησιμοποιείται μέσα στην έκφραση σύνταξης σαν τον κανόνα αμέσως μικρότερης προτεραιότητας, και εισάγεται επίσης και μία εκδοχή μόνο με τον κανόνα Next. Έτσι, αυτή η μακροεντολή αντιστοιχεί πάντα μονάχα την έκφραση σύνταξης με τον κανόνα Next ή τον ίδιο τον κανόνα Next σκέτο. Στο τέλος, δημιουργείται ένας κανόνας για όλες τις μακροεντολές αυτής της ιεραρχίας προτεραιότητας (πχ. "Expression") και τίθεται ως η μακροεντολή με τη μεγαλύτερη προτεραιότητα, και ως επιχείρημα παίρνει την μακροεντολή με την αμέσως επόμενη προτεραιότητα, η οποία επίσης παίρνει το ανάλογο επιχείρημα, και ούτω καθεξής. Αυτή η σειρά από μονοειδής μακροεντολές ορίζει και τη σειρά της προτεραιότητας, και σε περίπτωση που θέλουμε να την αλλάξουμε το μόνο που έχουμε να κάνουμε είναι να αλλάξουμε τη σειρά κατά την οποία αναγράφονται.

(Υποκεφάλαιο 5.4) Σημασιολογική Ανάλυση

Η σημασιολογική ανάλυση είναι το αμέσως επόμενο επίπεδο ανάλυσης. Εδώ, όλες οι δομές που εντοπίστηκαν στο επίπεδο της γραμματικής ανάλυσης ελέγχονται ως προς την ορθότητά τους, αν η γραμματική ανάλυση δεν έχει εντοπίσει συντακτικά λάθη. Επίσης, εδώ ετοιμάζονται όλες οι σχέσεις μεταξύ τον δομών προκειμένου να μπορούν αργότερα να μεταφραστούν επιτυχώς.

(Ενότητα 5.4.1) Συντακτικό Δέντρο

Οι δομές που χρησιμοποιούνται για την αποθήκευση των εκφράσεων στη μνήμη όλες υλοποιούν μια διεπαφή χαρακτηριστικού (αγγ. “trait”) “Expression”. Η διεπαφή αυτή έχει όλες τις μεθόδους οι οποίες μπορούν να χρησιμοποιηθούν για την μελέτη του δέντρου. Συγκεκριμένα, προσφέρει μεθόδους για την εύρεση του τύπου μιας έκφρασης, την προσπάθεια μετατροπής μιας έκφρασης σε σταθερά, την θέση της έκφρασης στο πρόγραμμα, την μονάδα στη οποία αναγράφεται η έκφραση, κ.ο.κ.

Δόμηση και Μοτίβα

Η κάθε δομή που αντιπροσωπεύει έκφραση μπορεί να είναι είτε απλή, στην οποία περίπτωση δεν περιέχει άλλες εκφράσεις μέσα της, ή σύνθετη, δηλαδή περιέχει άλλες εκφράσεις, όπως π.χ. πράξεις τελεστών. Όλες οι δομές αυτές υλοποιούν τη διεπαφή Expression. Κάθε δομή που υλοποιεί τη διεπαφή Expression μπορεί να μετατραπεί σε έναν πολυμορφικό τύπο ονόματι “ExBox”, όπου η ακριβής έκφραση δεν είναι γνωστή, όμως δύναται η χρήση των παραπάνω μεθόδων. Ο τύπος αυτός δεν είναι παρά μόνο ένας εικονικός πίνακας μεθόδων (αγγ. “vtable”) παραγόμενος από το αντικείμενο που υλοποιεί τη διεπαφή “Expression”, μέσα σε δύο μονοειδείς δομές, την “Ranged”, η οποία προσθέτει πληροφορίες όσον αφορά τη θέση της έκφρασης στο πρόγραμμα, και την “Box”, η οποία αντιπροσωπεύει έναν δείκτη στον σωρό της μνήμης, και αναλαμβάνει να ελευθερώσει τη κατανεμημένη μνήμη του σωρού με τρόπο RAII. Χωρίς “Box” τα δεδομένα θα έπρεπε να αποθηκεύονται στη στοίβα, το οποίο δημιουργεί προγραμματιστικές δυσκολίες. Ο ίδιος ο τύπος ExBox υλοποιεί Expression, ουσιαστικά ανακατευθύνοντας στις ανάλογες μεθόδους του εικονικού πίνακα μεθόδων μέσα του. Οι σύνθετες εκφράσεις έχουν μέσα τους πεδία για εκφράσεις-παιδιά με τύπο ExBox, καθώς δεν είναι απαραίτητα ξεκάθαρο το είδος της έκφρασης μέσα. Τα δέντρα εκφράσεων πρακτικά δομούνται ως σύνολα σύνθετων εκφράσεων με σχέση παιδί-πατέρα, και τα φύλλα να είναι απλές εκφράσεις.

Διάβαση του Δέντρου

Κατά την υλοποίηση της διεπαφής Expression, η δομή αυτή δεν μπορεί να ξέρει ακριβώς τις εκφράσεις-παιδιά της, μπορεί όμως να καλέσει αναδρομικά τις συναρτήσεις που δίνονται στη διεπαφή, προκειμένου να πάρει τις απαραίτητες πληροφορίες. Η αναδρομική

αυτή κλήση των μεθόδων είναι ο τρόπος που υλοποιούνται όλες οι μέθοδοι στη διεπαφή Expression που έχουν να κάνουν με εύρεση πληροφοριών στο δέντρο, όπως η “get_type” ή η “get_module”. Για την επικοινωνία μεταξύ των δέντρων εκφράσεων, χρησιμοποιείται η μονοειδής δομή Scope, η οποία λειτουργεί σαν απλώς συνδεδεμένη λίστα με μοντέλο στοίβας, με κάθε κόμβο να αποτελεί ένα πίνακα αναζήτησης (αγγ. “lookup table”), το οποίο υλοποιείται με πίνακα κατακερματισμού, και αντιπροσωπεύει ένα επίπεδο πεδίου ονομάτων (scope) στο οποίο βρίσκεται η έκφραση. Ο πεδίο ονομάτων μέσα στο οποίο βρίσκεται άμεσα η τρέχουσα έκφραση είναι η κορυφή της στοίβας, και η κεφαλή της λίστας. Για παράδειγμα, αν η έκφραση βρίσκεται σε μία ανώνυμη μονάδα η οποία βρίσκεται σε μία κανονική μονάδα, η κάθε μία από αυτές τις μονάδες αποτελούν πίνακες αναζήτησης για τα αναγνωριστικά/ονόματα που περιέχουν το κάθε ένα, με την κανονική μονάδα να είναι ο κόμβος αμέσως μετά την ανώνυμη μονάδα. Μέσω της δομής Scope δύναται η εύρεση πεδίων και τύπων κατά αναγνωριστικά/ονόματα, πάντα ξεκινώντας από την κεφαλή της λίστας, όπου αν ο τρέχων κόμβος δεν μπορεί να βρει το όνομα που αναγράφεται τον ζητάει από τον επόμενο κόμβο στη λίστα, κ.ο.κ., μέχρι είτε να βρεθεί η δομή που ζητήθηκε, είτε στη χειρότερη περίπτωση να τελειώσουν οι κόμβοι της λίστας και να επιστραφεί λάθος αστοχίας. Η στοίβα αυτή δίνεται εμπλουτισμένη με κάθε αναδρομική κλήση μεθόδων διεπαφής Expression από δομές που αποτελούν χώρους ονομάτων, όπως οι μονάδες, με την δομή που καλεί την μέθοδο να θέτει τον εαυτό της ως την νέα κεφαλή της λίστας, χωρίς όμως να αλλάζει η αναφορά στη λίστα που έχει η ίδια, έτσι ώστε όταν τελειώσει η κλήση της μεθόδου να διαγραφεί και η αναφορά της εμπλουτισμένης λίστας, με αποτέλεσμα την αφαίρεση από την στοίβα με τρόπο RAII. Ο τελευταίος κόμβος της λίστας αυτής είναι πάντα ο γενικός χώρος ονομάτων, που περιέχει όλες τις δηλώσεις των μονάδων και των διεπαφών, και δεν αφαιρείται ποτέ από τη στοίβα κατά την μετάφραση, καθώς όλες οι εκφράσεις βρίσκονται μέσα του εξ’ ορισμού.

(Ενότητα 5.4.2) Έλεγχος Τύπων

Ο έλεγχος τύπων είναι περίπλοκος λόγω της δυνατότητας υπονοούμενου καλουπιάσματος. Αν υπάρχει ένας τρόπος να μετατραπεί ένας τύπος στον ζητούμενο τύπο μέσω μιας σειράς καλουπιασμάτων, πρέπει να γίνει. Επομένως, κάθε φορά που υπάρχει μία αναντιστοιχία τύπων, πρέπει να βρεθεί αν είναι δυνατή η μετατροπή αυτού του τύπου, όσα βήματα κι αν υπάρχουν ανάμεσα. Προκειμένου λοιπόν να δύναται αυτή η μελέτη εν ώρα μετάφρασης, μοντελοποιούνται όλοι οι τύποι σε ένα σχεσιακό μοντέλο, το οποίο περιέχει έναν πίνακα με όλους τύπους, και ένα πίνακας συσχέτισης (αγγ. “join table”) μεταξύ των τύπων, δηλαδή όλων των υλοποιήσεων. Το μοντέλο αυτό μοντελοποιεί πρακτικά έναν γράφο, του οποίου οι κορυφές είναι οι τύποι/κατηγορίες και οι υλοποιήσεις διεπαφών/μορφισμοί καλουπιάσματος είναι οι ακμές. Έτσι, το ανάγουμε το πρόβλημα εύρεσης σειρών μετατροπών από έναν τύπο σε έναν άλλον σε ένα πρόβλημα εύρεσης μονοπατιού σε έναν γράφο, το οποίο λύνεται εύκολα με τη χρήση του αλγόριθμου του Dijkstra. Σε περίπτωση που βρεθεί μονοπάτι, η μετατροπές στο μονοπάτι εννοούνται και η

αναντιστοιχία τύπων λύνεται, διαφορετικά οι τύποι είναι ασυμβίβαστοι και σημειώνεται λάθος σημασιολογίας.

(Υποκεφάλαιο 5.5) Συλλογή Σφαλμάτων

Η συλλογή σφαλμάτων διέπεται από μία διεπαφή-χαρακτηριστικό με το όνομα “Throwable”. Η διεπαφή υλοποιείται από τους τύπους των οποίων τα αντικείμενα αντιπροσωπεύουν σφάλματα. Έχει μεθόδους για την επιστροφή πληροφοριών που σχετίζονται με την εκτύπωση του σφάλματος στον χρήστη, όπως θέση του σφάλματος στο πρόγραμμα, ο τίτλος του σφάλματος, μία μικρή εξήγηση του σφάλματος, και μερικές σημειώσεις που θα μπορούσε ο χρήστης να βρει χρήσιμες. Η διεπαφή αυτή, όπως και η διεπαφή Expression, χρησιμοποιείται ως μοντέλο εκωνικών πινάκων συναρτήσεων, το οποίο σε συνδυασμό με τον μονοειδή τύπο Box χρησιμοποιείται ως πολυμορφικός τύπος για όλα τα πιθανά σφάλματα, προκειμένου να συλλεχθούν σε μία δομή δεδομένων και να εκτυπωθούν. Οι δομές αυτές μπορούν να χρησιμοποιηθούν μαζί με μία δομή που ονομάζεται “PositionBuilder”, η οποία κρατάει μια αναφορά σε ολόκληρο το κείμενο του προγράμματος, και χρησιμοποιείται προκειμένου να εκτυπώσει ένα σφάλμα με τη χαρακτηριστική μορφοποίηση, η οποία εκτυπώνει τον τίτλο, την περιγραφή και τη θέση του σφάλματος, καθώς επίσης και το σφαλμένο απόσπασμα κώδικα ακριβώς από κάτω, μαζί με την σειρά βοηθητικών σημειώσεων που μπορεί να υπάρχει από κάτω.

```
Error: field 'c' is missing at line 6..12
   |
 6 |            module Test () {
 7 |                let method: (a: wire, b: (): IFace): IFace = module (
 8 |                    a: wire,
 9 |                    b: (
10 |                        c: (): IFace
11 |                    ): IFace
12 |                ) {
   |
```

Παράδειγμα εκτυπωμένου σφάλματος

(Ενότητα 5.5.1) Σφάλματα Συνταξης

Τα σφάλματα σύνταξης αναλύονται κατά το βήμα της γραμματικής ανάλυσης, και μπορούν να έχουν πολλές τιμές. Αντιπροσωπεύονται από μία δομή με τις παρακάτω πληροφορίες για το σφάλμα:

- Τις λεκτικές μονάδες που απορρίφθηκαν λόγω του σφάλματος
- Τη θέση του λάθους στο πρόγραμμα, σε χαρακτήρες
- Τις αναμενόμενες λεκτικές μονάδες
- Την λεκτική μονάδα που αντιθέτως πήραμε και δημιουργήθηκε σφάλμα

- Μια απαρίθμηση (αγγ. “enumeration”), η οποία έχει παραλλαγές για κάθε πιθανό είδος λάθος κατά την γραμματική ανάλυση. Οι παραλλαγές αυτές βασίζονται στις παραλλαγές που δίνονται από το lalrpop, και είναι:
    - Μη έγκυρη λεκτική μονάδα
    - Μη αναγνωρίσιμη λεκτική μονάδα
    - Μη αναμενόμενο τέλος αρχείου
    - Επιπλέων λεκτική μονάδα
    - Άλλο

(Ενότητα 5.5.2) Σφάλματα Σημασιολογίας

Τα σφάλματα σημασιολογίας είναι λίγο πιο περίπλοκα και ως προς την συλλογή και ως προς την μορφή. Παρομοίως τα σφάλματα σύνταξης, περιγράφονται από μια δομή που έχει τη θέση του σφάλματος και μία απαρίθμηση με παραλλαγές για το είδος του σφάλματος, αλλά σε αντίθεση με τα σφάλματα σύνταξης οι περισσότερες πληροφορίες αποθηκεύονται στις παραλλαγές, οι οποίες μπορεί να είναι και μονοειδής, καθώς εδώ η μορφή και τα πεδία πληροφοριών τα οποία χρειάζονται εξαρτώνται άμεσα από το είδος σφάλματος. Οι παραλλαγές αυτές είναι:

- Επαναδήλωση (Redefinition): Ένα όνομα-αναγνωριστικό έχει χρησιμοποιηθεί πάνω από μία φορά σε ένα πεδίο ονομάτων (αγγ. “scope”).
- Ασυμβίβαστοι τύποι (Incompatible Types): Το καλούπιασμα από τύπο σε τύπο έχει αποτύχει, καθώς δεν υπάρχει τρόπος να μετατραπεί ο τύπος της έκφρασης στον ζητούμενο τύπο
- Μη υλοποιημένα πεδία (Unimplemented): Δεν έχουν υλοποιηθεί όλα τα πεδία της διεπαφής στη υλοποίησή της.
- Αδυναμία πράξης (No Operation): Η δυαδική πράξη που δηλώθηκε είναι αδύνατη
- Διφορούμενη Αναφορά (Ambiguous Reference): Η αναφορά που έγινε σε κάποιο πεδίο είναι διφορούμενη και δεν μπορεί να λυθεί
- Δεν βρέθηκε (Not Found): Το ζητούμενο αντικείμενο της αναφοράς δεν μπόρεσε να βρεθεί
- Διφορούμενα Αριθμητικά Εύρη (Intersecting Index): Τα αριθμητικά εύρη των αριθμημένων πεδίων που δηλώθηκαν δεν είναι ξένα μεταξύ τους και επομένως είναι διφορούμενα
- Διφορούμενες υλοποιήσεις (Intersecting Implementation): Υπάρχουν πάνω από μία υλοποίηση της συγκεκριμένης διεπαφής για το συγκεκριμένο τύπο
- Ελλείποντα πεδία εισόδου (Missing Arguments): Τα πεδία εισόδου σε μια μονάδα δεν είναι πλήρη, υπάρχουν δηλαδή πεδία που λείπουν από την έκφραση.
- Δεν Είναι Διεπαφή (Not An Interface): Στην υλοποίηση διεπαφής, το όνομα της διεπαφής το οποίο υλοποιείται δεν αναφέρεται σε διεπαφή, αλλά σε μία δομή διαφορετικού είδους (π.χ. Μονάδα)

- Λάθος Είδους (Not A): Το είδος της δομής που λήφθηκε δεν είναι το είδος της δομής που αναμενόταν
- Τοποθετημένο (Positioned): Μονοειδής, εσωτερική/κρυφή παραλλαγή η οποία χρησιμοποιείται μονάχα για να προσθέτει δεδομένα θέσης μέσα στα δεδομένα της ίδιας της απαρίθμησης.
- Άκυρη ευρετηρίαση (Invalid Index): Ο αριθμός ευρετηρίασης που χρησιμοποιήθηκε είναι εκτός ορίων
- Δεν υπάρχει “self” (NoSelf): Η λέξη-κλειδί self χρησιμοποιήθηκε εκτός μονάδας, διεπαφής ή υλοποίησης.
- Πεδίο χωρίς τύπο (UntypedItem): Το πεδίο δεν έχει τύπο ούτε έκφραση για να εννοηθεί ο τύπος
- Δεν Είναι Πίνακας (NotAnArray): χρησιμοποιήθηκε σύνταξη ευρετηρίασης σε έκφραση που δεν την υποστηρίζει
- Μη Ξεδιπλώσιμο (Unfoldable): Η έκφραση που χρησιμοποιήθηκε ως σταθερά δεν μπορεί να μετατραπεί σε σταθερά
- Ελλειπής τιμή (Missing Value): Η δομή δεν έχει τιμή

Όλες οι παραπάνω παραλλαγές εμφανίζονται σε διαφορετικά σημεία της ανάλυσης και οι παρεχόμενες πληροφορίες τους διαφέρουν από τη μία στην άλλη. Οι πληροφορίες αυτές χρησιμοποιούνται κατά την επεξήγηση του σφάλματος και τις σημειώσεις που εμφανίζονται κατά την εκτύπωσή του.

(Ενότητα 5.5.3) Διάδοση Σφαλμάτων και Ανάνηψη

Όλα τα σφάλματα σύνταξης δίνονται απευθείας στη δομή “Program”, η οποία περιέχει όλες τις πληροφορίες της μετάφρασης του προγράμματος, και είναι “δημόσια” στο γραμματικό αναλυτή. Επειδή ο γραμματικός αναλυτής αναλαμβάνει τη διάδοση λαθών στα σημεία που επιλέγουμε, ξέρουμε πάντα ότι ένα λάθος θα ανιχνευθεί στον κανόνα λιστών, ο οποίος θα αναλαμβάνει να αναφέρει το σφάλμα στη δομή Program και να ανανήψει χωρίς την αναφορά περιττών παρεμφερών σφαλμάτων.

Η συλλογή και η ανάνηψη στο βήμα της σημασιολογικής ανάλυσης είναι λίγο πιο περίπλοκη γιατί, αντίθετα με τα σφάλματα σύνταξης που τα διαδίδει το lalrpop, πρέπει να δημιουργήσουμε εμείς οι ίδιοι τη διάδοση τους. Για την διάδοση λοιπόν χρησιμοποιείται ο μηχανισμός επιστροφής σφαλμάτων μέσω του τύπου επιστροφής συναρτήσεων της rust. Ο μηχανισμός αυτός λέγεται “Result”, και δεν είναι παρά μόνο μία μονοειδής ενδοσυναρτησιακή απαρρίθμηση, με δύο παραλλαγές: Την “Ok”, η οποία περιέχει τα δεδομένα που αναμένονταν από την συνάρτηση, και την “Err” η οποία περιέχει την δομή που περιγράφει το λάθος που προέκυψε. Η δομή αυτή επιστρέφεται από τις μεθόδους σημασιολογικής ανάλυσης και κάθε φορά που λαμβάνεται πρέπει να χειριστεί αναλόγως προκειμένου να αποκαλυφθούν οι πληροφορίες που επιστράφηκαν, είτε αυτές είναι οι αναμενόμενες είτε το σφάλμα. Για την εύκολη διαχείρισή τους και την διατήρηση των τύπων, μαζί με τις απλές εκφράσεις που προσφέρει η rust, δίνονται ενδοσυναρτησιακές μέθοδοι με

διάφορες λειτουργίες, όπως η μετατροπή της πληροφορίας Ok σε άλλο τύπο κατά μια συνάρτηση (map) ή η “βίαια” αποκάλυψη των περιεχομένων της παραλλαγής Ok, όπου αν η απαρίθμηση τελικά είναι Err ολόκληρο το πρόγραμμα σταματά (unwrap). Με τη χρήση τέτοιων μεθόδων γίνεται απλή η διαχείριση λαθών σε κάθε επίπεδο της ανάλυσης κατά την σημασιολογική ανάλυση, και κάθε μέθοδος μπορεί έτσι να αποφασίσει αν θα διαδώσει το σφάλμα στη συνάρτηση που την κάλεσε, ή αν θα διαχειριστεί το λάθος μόνη της, δηλώνοντάς το στη δομή Program και ανανήπτοντας.

(Υποκεφάλαιο 5.6) Ενδιάμεση Αναπαράσταση και παραγωγή Verilog

Αφού ο κώδικας αναλυθεί συντακτικά και σημασιολογικά και σιγουρευτεί η ορθότητά του, ακολουθεί η μετατροπή στην ενδιάμεση αναπαράσταση και η παραγωγή κώδικα Verilog. Η ενδιάμεση αναπαράσταση είναι μια ενδοσυναρτησιακή δομή η οποία έχει μεθόδους αλλαγής, οι οποίες προσθέτουν αντικείμενα στη δομή με κάθε κλήση. Η δομή αυτή δίνεται μέσα σε μία αναδρομική εκτέλεση της μεθόδου μετατροπής των σημασιολογικών δομών, δίνοντας σαν αποτέλεσμα τη τελική δομή ενδιάμεσης αναπαράστασης. Η δομή αυτή έχει τέτοια μορφή έτσι ώστε να μπορεί να μεταφραστεί σε Verilog εύκολα, ένα προς ένα, μέσω της σύνθεσης ορμαθών (αγγ. “strings”), των οποίων η τελική μορφή εξάγεται και αποθηκεύεται στο αρχείο εξόδου, έτοιμο για χρήση από εργαλεία που υποστηρίζουν Verilog

(Ενότητα 5.6.1) Λύση πολυμορφικών δομών

Η verilog ως γλώσσα δεν υποστηρίζει πολυμορφία[9], επομένως είναι αδύνατη η μετατροπή πολυμορφικών δομών σε verilog μία προς μία. Γι’ αυτόν τον λόγο, η δομή ενδιάμεσης αναπαράστασης αναλαμβάνει να διαφοροποιήσει επί τούτω όλες τις διαφορετικές μορφές των πολυμορφικών δομών και να ξεκαθαρίσει ποια χρησιμοποιείται που. Κατά την δημιουργία, χρησιμοποιείται μία σύμβαση ονομασίας των μονάδων σε verilog που κάνουν εύκολη την οπισθοδρόμηση λογικών λαθών από verilog σε iris μέσω των ονομάτων. Συγκεκριμένα, χ.π.τ.γ.:

- Η κάθε μονάδα που παράγεται από υλοποίηση της διεπαφής “Interface” για έναν τύπο “Type”, ονομάζονται “TypeAsInterface”,
- Η κάθε μονάδα που παράγεται από μία μονάδα “Module” που χρησιμοποιεί μία γκάμα πολυμορφικών τύπων “Type1”, “Type2”, “Type3”, κ.ο.κ. ονομάζεται “ModuleWithType1AsInterfaceAndType2AsInterfaceAndType3AsInterface…” κ.ο.κ.,

Οι κανόνες αυτοί, αν και μπορούν να δημιουργήσουν μερικά πολύ μεγάλα ονόματα, συνήθως δεν το κάνουν, καθώς οι χρήστες συνήθως δεν χρησιμοποιούν παραπάνω από δύο σημεία πολυμορφισμού σε έναν τύπο.

ΚΕΦΑΛΑΙΟ 6 Συμπεράσματα

Η ανάπτυξη και η υλοποίηση μίας νέας γλώσσας υποδυκνείεται ότι είναι μία πολύπλοκη επιχείρηση λόγω των βαθμών ελευθερίας που την διέπουν, καθώς επίσης και των απαιτήσεων σε γνώσεις θεωρητικής και πρακτικής επιστήμης υπολογιστών. Το θεωρητικό μέρος της ιδέας όμως δεν απαιτεί μονάχα στεγνή μαθηματική λογική, αλλά και διαίσθηση για τις απαιτήσεις ενός σχεδιαστή υλικού από την γλώσσα προγραμματισμού του, η οποία οφείλει να είναι εργονομική, όχι μόνο στον σχεδιασμό της ίδιας αλλά και στον σχεδιασμό του μεταγλωττιστή της.

(Υποκεφάλαιο 6.1) Σχεδιασμός, Σύνταξη και Σημασιολογία

Η γλώσσα ως σύνολο κανόνων σύνταξης και σημασιολογίας διέπεται από απαιτήσεις τις οποίες δεν ικανοποιούν οι καθιερωμένες γλώσσες περιγραφής υλικού, πχ. Verilog. Οι απαιτήσεις αυτές όμως έχουν καθιερωθεί στα πρότυπα των γλωσσών προγραμματισμού εφαρμογών λογισιμικού, και έχουν αναπτυχθεί ιδιαιτέρως, και σε θεωρητικό και σε πρακτικό πλαίσιο. Με παράδειγμα λοιπόν τέτοια πρότυπα σχεδιάστηκε ένα σύνολο συντακτικών και σημασιολογικών κανόνων το οποίο να τα ικανοποιεί και να ακολουθεί τους συμβατική μορφή των γλωσσών οι οποίες ακολοθούν τα εν λόγω πρότυπα. Αυτό που αποτέλσε πρόκληση ήταν η εφαρμογή των προτύπων αυτών, τα οποία εφαρμόστηκαν κυρίως στο πλαίσιο του προγραμματισμού εφαρμογών λογισμικού, στο πλαίσιο του σχεδιασμού υλικού. Η πρόκληση αυτή εκπληρώθηκε μέσω της απεικόνισής των ιδεών των προτύπων αυτών ως κατηγορίες, και των μορφισμών αυτών ως συναρτήσεις της άλγεβρας λάμδα. Η χρήση των θεωρητικών αρχών αυτών λοιπόν βοήθησαν στην αποδόμηση των ιδεών των προτύπων αυτών και στην αναπαραγωγή τους στο πλαίσιο του σχεδιασμού υλικού.

(Υποκεφάλαιο 6.2) Υλοποίηση του Μεταγλωττιστή

Η υλοποίηση ενός μεταγλωττιστή είναι μια πολυπλοκη διαδικασία λόγω όλων των βημάτων που απαιτούνται από αυτόν. Για την υλοποίηση του, χρησιμοποιήθηκε η γλώσσα rust μαζί με τα εργαλεία Lalrpop και Logos προκειμένου να ολοκληρωθεί το βήμα της συντακτικής ανάλυσης. Ως γλώσσα εξόδου επιλέχθηκε η Verilog. Για την σημασιολογική ανάλυση και την παραγωγή τελικού κώδικα χρησιμοποιήθηκαν πολλές τεχνικές, με αναδρομικά, συναρτησιακά χαρακτηριστικά. Κατά την παραγωγή τελικού κώδικα, είναι απαραίτητη η ανάλυση του πηγαίου κώδικα και η λύση των χαρακτηριστικών που δεν υποστηρίζονται στην Verilog, όπως οι πολυμορφικοί τύποι. Η συλλογή και η εμφάνιση σφαλμάτων σε κάθε βήμα υλοποιήθηκε με χρήση δομών οι οποίες περιέιχαν τις απαραίτητες πληροφορίες για την περιγραφή του σφάλματος, όπως θέση, περιγραφή, και σημειώσεις, μεταξύ άλλων. Οι πληροφορίες αυτές μπορούν να χρησιμοποιηθούν σε συνδυασμό με συγκεκριμένες μονοειδείς δομές προκειμένου να επιτευχθεί μία εμπειρία αποσφαλμάτωσης η οποία είναι φιλική προς τον χρήστη.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1]: Awodey, Steve. Category theory. Oxford university press, 2010.

[2]: Barendregt, Henk, Wil Dekkers, and Richard Statman. Lambda calculus with types. Cambridge University Press, 2013.

[3]: Milner, Robin. "A theory of type polymorphism in programming." Journal of computer and system sciences 17.3 (1978): 348-375.

[4]: Γερογιάννης, Β., Κακαρόντζας, Γ., Καμέας, Α., Στάμελος, Γ., Φιτσιλής, Π. (2006). Αντικειμενοστραφής Ανάπτυξη Λογισμικού με τη Uml. Κλειδάριθμος.

[5]: Cardelli, Luca, and Peter Wegner. "On understanding types, data abstraction, and polymorphism." ACM Computing Surveys (CSUR) 17.4 (1985): 471-523.

[6]: Klabnik, Steve, and Carol Nichols. The Rust Programming Language (Covers Rust 2018). No Starch Press, 2019.

[7]: The Lalrpop Developers. “Lalrpop/Lalrpop: LR(1) Parser Generator for Rust.” GitHub, 2015, https://github.com/lalrpop/lalrpop.

[8]: Hirsz, Maciej. “Maciejhirsz/Logos: Create Ridiculously Fast Lexers.” GitHub, 2022, https://github.com/maciejhirsz/logos.

[9]: Mano, M. M., & Cilleti, M. D. (2018). *Digital Design* (6η εκδ.). Παπασωτηρίου.