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

Σχεδίαση και Υλοποίηση μιας Εικονικής Μηχανής για WebAssembly

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΛΕΞΑΝΔΡΟΣ ΙΩΑΝΝΟΥ

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

Σχεδίαση και Υλοποίηση μιας Εικονικής Μηχανής για WebAssembly

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΛΕΞΑΝΔΡΟΣ ΙΩΑΝΝΟΥ

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 8η Ιουλίου 2020.

Νικόλαος Σ. Παπασπύρου Καθηγητής Ε.Μ.Π.	Κωσταντίνος Σαγώνας Αν. Καθηγητής Ε.Μ.Π.	Γεώργιος Γκούμας Επίκ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2020

Αλέξανδρος Ιωάννου

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

Copyright © Αλέξανδρος Ιωάννου, 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

Το διαδίκτυο αποτελεί πλέον μια προγραμματιστική πλατφόρμα. Η WebAssembly είναι ένα πρότυπο γλώσσας χαμηλού επιπέδου, το οποίο επιτρέπει την ανάπτυξη εφαρμογών υψηλής επίδοσης και πολυπλοκότητας στο διαδίκτυο, και όχι μόνο. Αυτή τη διπλωματική εργασία ασχολείται με την σχεδίαση και ανάπτυξη μιας εικονικής μηχανής για WebAssembly με τρόπο εύκολα κατανοητό, επεκτάσιμο και συντηρήσιμο. Επίσης, κατασκευάζεται και παρουσιάζεται ένας μηχανισμός ελέγχου της ορθότητας της μηχανής, ικανός να επεκταθεί και ο ίδιος ανάλογα με τις μελλοντικές ανάγκες.

Λέξεις κλειδιά

WebAssembly, Εικονική Μηχανή, C++, bytecode.

Abstract

The Web has become a programming platform. WebAssembly is a low-level language specification, which allows the development of highly efficient and complex applications for the Web and other platforms. In this diploma thesis, we present the design and implementation of a virtual machine for WebAssembly which is simple, easily understood, expandable and maintainable. We also construct a mechanism for testing the machine's correctness, which is also expandable to suit future needs.

Key words

WebAssembly, Virtual Machine, C++, bytecode.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που υποστήριξαν τη δουλεία μου και με βοήθησαν στην ποιοτική και ποσοτική βελτίωσή της. Ευχαριστώ θερμά τον επιβλέποντα καθηγητή αυτής της εργασίας κ. Νικόλαο Παπασπύρου, για τη συνεχή καθοδήγησή και εμπιστοσύνη του. Θα ήθελα να ευχαριστήσω τους φίλους και συμφοιτητές που ήταν δίπλα μου καθ όλη τη διάρκεια των σπουδών μου. Θα ήθελα τέλος να ευχαριστήσω την οικογένειά μου και κυρίως τους γονείς μου, οι οποίοι με υποστήριξαν και έκαναν δυνατή την απερίσπαστη ενασχόλησή μου τόσο με την εκπόνηση της διπλωματικής μου, όσο και συνολικά με τις σπουδές μου.

Αλέξανδρος Ιωάννου,

Αθήνα, 8η Ιουλίου 2020

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-2-20, 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

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός της εργασίας

Στόχος της εργασίας είναι η υλοποίηση μιας εικονικής μηχανής (virtual machine) για WebAssembly σε C++ 17. Η υλοποίηση είναι όσο το δυνατόν πιστή στο πρότυπο της WebAssembly και συνοδεύεται από μηχανισμό εκτενούς ελέγχου (testing). Επίσης, στόχος της εργασίας είναι η εύκολη κατανόηση του πηγαίου κώδικα και δευτερευόντως η αποδοτικότητα. Μία τέτοια "εκπαιδευτική" υλοποίηση, θα επιτρέψει τον πειραματισμό σε ιδέες που αφορούν συστήματα τύπων (type systems), επαλήθευση προγραμμάτων (verification) και άλλα, για WebAssembly. Επιπρόσθετα, η WebAssembly είναι ένα συνεχώς εξελισσόμενο πρότυπο οπότε είναι ιδιαίτερα σημαντικό η υλοποίηση να είναι συντηρήσιμη και επεκτάσιμη. Τέλος, η υλοποίηση θα επιλύσει τις βασικές προκλήσεις του προβλήματος και θα δομηθεί κατάλληλα έτσι ώστε να είναι εύκολη η προσθήκη νέας λειτουργικότητας.

1.2 Ιστορική αναδρομή

Το διαδίκτυο (World Wide Web – WWW, συχνά αναφέρεται και ως "Web") δημιουργήθηκε ως ένα απλό δίκτυο ανταλλαγής αρχείων και σήμερα αποτελεί την πιο διαδεδομένη πλατφόρμα ανάπτυξης εφαρμογών, προσβάσιμη από πληθώρα λειτουργικών συστημάτων και συσκευών. Από την αρχή αυτής της πορείας, η μόνη οικουμενικά υποστηριζόμενη γλώσσα για διαδικτυακές εφαρμογές είναι η Javascript, παρά τις προσπάθειες άλλων τεχνολογιών να την αντικαταστήσουν. Η Javascript έχει ωριμάσει σημαντικά τόσο στην δομή της όσο και στην επίδοση των εικονικών μηχανών της. Η αποκλειστική υποστήριξη της Javascript την έχει καταστήσει ως τον μοναδικό στόχο μεταγλώττισης (compilation target) για το Web, παρότι η γλώσσα δεν έχει σχεδιαστεί για κάτι τέτοιο.

Το 2013 εμφανίζεται το asm.js [asmj] ένα υποσύνολο της Javascript σχεδιασμένο ώστε να επιτρέψει σε προγράμματα γραμμένα σε γλώσσες όπως η C, να λειτουργήσουν ως web εφαρμογές, διατηρώντας τα χαρακτηριστικά της επίδοσής τους. Τα προγράμματα αυτά μεταγλωττίζονται με χρήση του Emscripten [Zaka11], ενός μεταγλωττιστή από LLVM (Low Level Virtual Machine) IR σε Javascript, πιο συγκεκριμένα σε asm.js. Οι πρώτες δοκιμές επίδοσης τέτοιων προγραμμάτων έδειχναν διπλασιασμό του χρόνου εκτέλεσής τους σε σχέση με την native έκδοση, κάτι που εξακολουθεί να είναι πολύ ταχύτερο από την επίδοση των αντίστοιχων προγραμμάτων γραμμένων σε συνηθισμένη Javascript.

Το 2017 δημιουργείται η WebAssembly [Haas17, wasm] και πολύ σύντομα υποστηρίζεται από τους πιο δημοφιλείς browsers. Η WebAssembly επιτρέπει σημαντικά υψηλότερες επιδόσεις από αυτές της Javascript, χωρίς αυτό να σημαίνει ότι δύναται να την αντικαταστήσει. Αντιθέτως, η WebAssembly είναι μια γλώσσα η οποία λειτουργεί σαν ένα ενσωματωμένο σύνολο από συναρτήσεις που μπορούν να υπολογίσουν ταχύτατα το αποτέλεσμά τους. Στο περιβάλλον ενός browser, η Javascript είναι αναγκαία έτσι ώστε να γίνει χρήση των λειτουργιών που μπορεί να παρέχονται από ένα WebAssembly module. Επιπλέον, η WebAssembly δεν κάνει καμία υπόθεση για το περιβάλλον ενσωμάτωσής της. Λόγω αυτού έχει βρει σημαντικές εφαρμογές εκτός των browsers. Η ευρεία χρήση της, ειδικά σε ζητήματα όπως έξυπνα συμβόλαια (smart contracts), έχει ελκύσει το ενδιαφέρον πολλών ερευνητών για συστηματικές και μηχανικές αποδείξεις της ασφάλειάς της [Watt18], κάτι που ενισχύει την πρόβλεψη ότι η WebAssembly θα μεταρρυθμίσει ριζικά τον τρόπο με τον οποίο θα αναπτύσσεται το λογισμικό

στο μέλλον.

1.3 Δομή της εργασίας

Το υπόλοιπο της εργασίας οργανώνεται ως εξής. Στο Κεφάλαιο 2 γίνεται μια περιγραφή του προτύπου και των πλεονεκτημάτων του, τόσο σε Web περιβάλλον όσο και σε διαφορετικά σενάρια, και δίνεται μια σκιαγράφηση των βασικών δομικών του στοιχείων. Στο Κεφάλαιο 3 παρουσιάζεται η υλοποίηση της εικονικής μηχανής, εστιάζοντας σε σημαντικές κλάσεις και περιγράφοντας ορισμένες σχεδιαστικές επιλογές. Το Κεφάλαιο 4 αναλύει τον μηχανισμό ελέγχου (testing) και παρουσιάζει τα αποτελέσματά του. Τέλος, το Κεφάλαιο 5 συνοψίζει την εργασία και παρέχει πιθανές μελλοντικές επεκτάσεις της υλοποίησης.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 WebAssembly

Η WebAssembly είναι ένα ανοικτό πρότυπο για δυαδική κωδικοποίηση εκτελέσιμων προγραμμάτων το οποίο διασφαλίζει φορητότητα, ασφάλεια, ταχεία εκτέλεση και συνοπτική αναπαράσταση. Αν και το όνομα παραπέμπει στο διαδίκτυο, στην πράξη η WebAssembly προσφέρει όλα όσα χρειαζόμαστε για εφαρμογές απομακρυσμένου υπολογισμού. Πιο συγκεκριμένα η WebAssembly προσφέρει:

- **Ασφάλεια**

  Η WebAssembly δεν παρέχει κανένα τρόπο έτσι ώστε το πρόγραμμα να αλληλεπιδρά με το περιβάλλον εκτέλεσης. Λειτουργίες I/O, πρόσβαση σε πόρους, κλήσεις στο λειτουργικό σύστημα είναι εφικτές μόνο μέσω συναρτήσεων που παρέχονται από τον embedder και εισάγονται στο WebAssembly module. Έτσι έχουμε ένα πλήρως απομονωμένο περιβάλλον εκτέλεσης στο οποίο μπορούμε κατ' επιλογήν να παρέχουμε πόρους.

- **Φορητότητα**

  Η WebAssembly είναι ανεξάρτητη από το υλικό, την γλώσσα και την πλατφόρμα. Μπορεί να μεταφραστεί σε όλες τις αρχιτεκτονικές τόσο για desktop υπολογιστές, όσο και για φορητές συσκευές (κινητά). Δεν δείχνει προτίμηση προς κάποια συγκεκριμένη γλώσσα ή μοντέλο προγραμματισμού και μπορεί να ενσωματωθεί σε προγράμματα περιήγησης, να λειτουργεί σε μια αυτούσια εικονική μηχανή ή να ενταχθεί σε κάποιο άλλο περιβάλλον. Τέλος τα παραγόμενα προγράμματα μπορούν να επικοινωνούν με το περιβάλλον τους με ένα απλό και οικουμενικό τρόπο.

- **Ταχύτητα**

  Το πρότυπο είναι σχεδιασμένο έτσι ώστε να εκμεταλλεύεται τις δυνατότητες του σύγχρονου υλικού. Επίσης, τα παραγόμενα προγράμματα είναι συνοπτικά με αποτέλεσμα να μεταδίδονται γρήγορα μέσα στο δίκτυο. Κάθε WebAssembly module μπορεί να υποστεί επεξεργασία ανεξάρτητα από τα υπόλοιπα σε ένα πέρασμα. Η αποκωδικοποίηση, επιβεβαίωση και μετάφραση του κάθε module μπορεί να γίνει με σταδιακή ανάγνωση από το input stream χωρίς αναμονή για την ολοκλήρωση της λήψης.

Τα πλεονεκτήματα του προτύπου, έχουν επιτρέψει σε πολλούς προγραμματιστές να μεταφράσουν προγράμματα γραμμένα σε γλώσσες όπως η C++ και να τα εντάξουν στο διαδίκτυο. Πλέον υπάρχει ένας τρόπος για εφαρμογές όπως παιχνίδια με υψηλού επιπέδου γραφικά, επεξεργασία εικόνων κ.α. να μεταφερθούν στην πλατφόρμα του διαδικτύου.

2.2 WebAssembly έξω από τον browser

Όπως έχει ήδη αναφερθεί, το πρότυπο δεν κάνει καμία υπόθεση σχετικά με το περιβάλλον εκτέλεσης. Έτσι η ιδέα σχετικά με την υλοποίηση μιας εικονικής μηχανής έξω από τον browser είναι αρκετά

δημοφιλής [nonw]. Εφόσον η WebAssembly δεν εξαρτάται από Web APIs μπορεί να χρησιμοποιηθεί ως ένα φορητό binary format σε πολλές πλατφόρμες, παρέχοντας τα οφέλη της σε αυτές.

Πιο συγκεκριμένα, η WebAssembly δίνει την δυνατότητα εκτέλεσης κώδικα σε ένα απομονωμένο περιβάλλον (sandbox). Αυτό είναι ιδιαίτερα σημαντικό για λόγους ασφαλείας και ελεγχόμενης εκτέλεσης. Επίσης, ένας από τους στόχους του προτύπου είναι η παραγωγή μικρών binary αρχείων. Ίσως για τα σύγχρονα συστήματα αυτό δεν είναι σημαντικό, λόγω των μεγάλων χωρητικοτήτων στους δίσκους, αλλά είναι ιδιαίτερα ωφέλιμο για εφαρμογές block-chain στις οποίες μπορεί να αποθηκεύονται χιλιάδες προγράμματα από χιλιάδες χρήστες, όπως τα έξυπνα συμβόλαια του Etherium [ewas]. Η παραπάνω εφαρμογή έχει ανάγκη για τα μικρά binaries που παράγει η WebAssembly καθώς και για την ασφάλεια με την οποία μπορούν να δομηθούν αποδείξεις ορθότητας σε αυτή. Έτσι το Etherium έχει δημιουργήσει ένα υποσύνολο της WebAssembly, γνωστό ως ewasm (Etherium WebAssembly) κατάλληλο ακριβώς για τις ανάγκες του. Μια ακόμα εφαρμογή της WebAssembly βρίσκεται στο AccTEE [Golt19], έναν μηχανισμό που παρέχει ένα περιβάλλον εκτέλεσης προγραμμάτων σε cloud συστήματα. Το AccTEE προσφέρει λύσεις στα προβλήματα εμπιστοσύνης μεταξύ του προγραμματιστή-καταναλωτή και του παρόχου της cloud υπηρεσίας, δημιουργώντας ένα απομονωμένο περιβάλλον εκτέλεσης δύο κατευθύνσεων (two-way sandbox), το οποίο παρέχει αξιοπιστία εκτέλεσης και καταγραφής των πόρων που χρησιμοποιήθηκαν (metering)

Τέλος, η WebAssembly αποτελεί μια νέα, εξελισσόμενη τεχνολογία, η οποία επεκτείνεται και βελτιώνεται μέσω υιοθέτησης διαφόρων proposals. Ενδιαφέροντα θέματα έχουν τεθεί όπως garbage collection [GC] και άλλα.

2.3 Δομικά στοιχεία και έννοιες του προτύπου

Module Κάθε δυαδικό αρχείο (binary) που έχει δημιουργηθεί σύμφωνα με το πρότυπο της WebAssembly αποτελεί ένα module. Ένα module περιέχει ορισμούς για συναρτήσεις, global μεταβλητές, πίνακες και μνήμες. Τα modules μπορούν να "εξάγουν" τους ορισμούς που περιέχουν, δηλαδή να τους "κοινοποιούν" σε άλλα modules. Για να γίνει αυτό, πρέπει στο module – παραλήπτη να καθορίζεται:

α) το όνομα του module από το οποίο θα "εξαχθούν" οι ορισμοί και

β) οι ορισμοί που θα "εξαχθούν".

Το module αποτελεί μία στατική αναπαράσταση ενός προγράμματος. Για να εκτελεστεί το πρόγραμμα απαιτείται το instantiation του module, κατά το οποίο δίνονται όλοι οι ορισμοί που εισάγονται και το module αποκτά δυναμικά χαρακτηριστικά όπως μνήμη και στοίβα εκτέλεσης. Στο πλαίσιο αυτής της εργασίας καλύπτεται η αποκωδικοποίηση του module σύμφωνα με τους κανόνες του προτύπου, και ο στατικός έλεγχος της σημασιολογικής ορθότητάς του (validation). Η στατική ανάλυση περιέχει την δημιουργία της δομής στη μνήμη με βάση την πληροφορία που περιέχει το αρχείο και την επιβεβαίωση της συντακτικής και σημασιολογικής ορθότητάς του με βάση τους σημασιολογικούς κανόνες του προτύπου.

Συναρτήσεις Ο κώδικας οργανώνεται σε επιμέρους συναρτήσεις. Κάθε συνάρτηση λαμβάνει μία λίστα από τιμές σαν παραμέτρους εισόδου και επιστρέφει μία λίστα του ίδιου τύπου σαν αποτέλεσμα. Οι συναρτήσεις μπορούν να καλούν άλλες συναρτήσεις καθώς και να κάνουν αναδρομικές κλήσεις. Τέλος, δεν επιτρέπεται ορισμός συνάρτησης στο εσωτερικό κάποιας άλλης.

Εντολές Το μοντέλο εκτέλεσης που υιοθετεί η WebAssembly είναι αυτό της στοίβας. Οι εντολές που βρίσκονται μέσα στις συναρτήσεις τροποποιούν, εισάγουν και εξάγουν τιμές από μια στοίβα τελουμένων (operand stack). Αυτός ο τρόπος οργάνωσης (στοίβα) επιλέχθηκε διότι έχει φανεί πως έτσι παράγονται μικρότερα αρχεία. Το σύστημα τύπων της γλώσσας μας επιτρέπει να καθορίσουμε στατικά το μέγεθος της στοίβας σε κάθε σημείο της εκτέλεσης, οπότε οι υλοποιήσεις της γλώσσας μπορούν να την παρακάμψουν εντελώς και να μεταφράσουν τη ροή δεδομένων μεταξύ των εντολών.

Βασικοί Τύποι Η WebAssembly περιέχει 4 βασικούς τύπους οι οποίοι είναι παρόντες και στο υλικό: ακέραιους αριθμούς και αριθμούς κινητής υποδιαστολής, 32 και 64 bit. Οι ακέραιοι 32 bits μπορούν να χρησιμοποιηθούν σαν δείκτες σε κάποια μνήμη ή σε κάποιο πίνακα. Οι περισσότερες εντολές εφαρμόζουν κάποιον τελεστή πάνω σε τιμές αυτών των τύπων καθώς επίσης υπάρχουν και εντολές για μετατροπή μεταξύ τύπων και reinterpret των δυαδικών ψηφίων. Το πρότυπο δεν κάνει κάποιο διαχωρισμό μεταξύ προσημασμένων και μη τιμών. Εν αντιθέσει, όπως συνηθίζεται και στο υλικό, αν υπάρχει ενδιαφέρον για το πρόσημο τότε χρησιμοποιούνται οι καταλήξεις **_s** (signed, συμπλήρωμα ως προς 2) ή **_u** (unsigned) στις αντίστοιχες εντολές.

Τοπικές μεταβλητές Κάθε συνάρτηση μπορεί να ορίσει τοπικές μεταβλητές των τύπων που περιγράφηκαν παραπάνω. Οι εντολές **`local.get`** και **`local.set`** χρησιμοποιούνται για το διάβασμα και γράψιμο τους. Το πρότυπο παρέχει και την **`local.tee`** η οποία επιτρέπει το γράψιμο της αντίστοιχης μεταβλητής, ενώ επιστρέφει την προϋπάρχουσα τιμής της στη στοίβα. Σημαντικό είναι ότι και οι παράμετροι της συνάρτησης θεωρούνται τοπικές μεταβλητές.

Global μεταβλητές Κάθε module μπορεί επίσης να ορίζει global μεταβλητές οι οποίες μπορούν να είναι σταθερές ή όχι. Για το διάβασμα ή το γράψιμο σε αυτές χρησιμοποιούνται οι εντολές **`global.get`**, **`global.set`**. Η αρχικοποίησή τους γίνεται από μία σταθερή έκφραση η οποία υπολογίζεται χωρίς πρόσβαση σε κάποια συνάρτηση, πίνακα, μνήμη, τοπική μεταβλητή ή μη σταθερή global μεταβλητή.

Μνήμη Κάθε module μπορεί να ορίζει το πολύ μία μνήμη. Κάθε μνήμη είναι γραμμική και έχει αρχή και τέλος. Μια μνήμη μπορεί να χρησιμοποιείται από περισσότερα από ένα modules, υπό την προϋπόθεση ότι το module στο οποίο ορίζεται θα την ”εξάγει” στα υπόλοιπα modules που θα την χρησιμοποιούν. Το αρχικό μέγεθος μίας μνήμης μπορεί να μεγαλώνει δυναμικά με την εντολή **`memory.grow`**. Επίσης το πρόγραμμα μπορεί να μάθει το μέγεθος της μνήμης ανά πάσα στιγμή με την χρήση της εντολής **`memory.size`**. Στο module μπορεί να υπάρχει ένα data segment το οποίο αρχικοποιεί τα περιεχόμενα της μνήμης.

Πίνακας Όπως και με τη μνήμη έτσι και με τον πίνακα κάθε module μπορεί να ορίζει το πολύ έναν. Οι πίνακες χρησιμοποιούνται για έμμεση κλήση συναρτήσεων. Υποστηρίζουν τη χρήση δεικτών σε συναρτήσεις (function pointers) κάτι που χρησιμοποιείται αρκετά σε γλώσσες όπως η C++. Ο πίνακας του module μπορεί να αρχικοποιείται από ένα element segment.

Κεφάλαιο 3

Υλοποίηση

Η υλοποίηση είχε ως βασικό στόχο την συντηρησιμότητα και την επεκτασιμότητά του τελικού προϊόντος στο μέλλον. Για την επίτευξη αυτού του στόχου κύριο μέλημα ήταν η σύνταξη και οργάνωση του πηγαίου κώδικα με τρόπο ώστε να είναι εύκολα κατανοητός και η αποφυγή σύνθετων αρχιτεκτονικών. Στο πλαίσιο της παρούσας εργασίας αναπτύχθηκε ένα στιβαρό υπόβαθρο κώδικα, το οποίο:

i. αντιμετωπίζει αποδοτικά τις βασικές προκλήσεις της σχεδίασης και υλοποίησης μιας πλήρους εικονικής μηχανής για WebAssembly και

ii. είναι κατασκευασμένο έτσι ώστε να μπορεί εύκολα να αξιοποιηθεί για προσθήκες νέας λειτουργικότητας.

Τέτοιες προσθήκες μπορεί να προκύψουν είτε για "συγχρονισμό" της εικονικής μηχανής με την εξέλιξη της WebAssembly όπως αυτή υλοποιείται με την υιοθέτηση προτάσεων (proposals), είτε για ενσωμάτωση νέων ιδεών ως προς συγκεκριμένα τμήματα της υπάρχουσας λειτουργικότητας της εικονικής μηχανής.

Η ανάπτυξη της εικονικής μηχανής έγινε σε Linux περιβάλλον με χρήση της γλώσσας C++. Εκτός των πηγαίων αρχείων, που είναι γραμμένα σε C++, χρησιμοποιήθηκαν Makefiles τόσο για την κατασκευή του εκτελέσιμου (cwasm) της εικονικής μηχανής όσο και για τις ανάγκες του μηχανισμού ελέγχου. Επίσης για τον μηχανισμό ελέγχου χρησιμοποιήθηκαν bash και python scripts. Ο κώδικας διατίθεται ανοικτά στον σύνδεσμο github.com/ioanalex/cwasm. Επιλεγμένα τμήματά του παρατίθενται τα Παραρτήματα του παρόντος κειμένου.

Οι λειτουργίες που έχουν υλοποιηθεί είναι η ανάγνωση και κατασκευή ενός WebAssembly Module και ο στατικός του έλεγχος (validation). Για αυτές τις λειτουργίες έχουν σχεδιαστεί οι κλάσεις Reader και Validator. Και οι δύο κλάσεις επιδρούν σε ένα αντικείμενο τύπου Module το οποίο παρουσιάζεται παρακάτω.

3.1 Module

Το module ορίζεται ως μία δομή με όλα τα πεδία που απαιτούνται (Σχήμα 3.1). Επιπρόσθετα, στο πεδίο `imported_globals` αποθηκεύεται το πλήθος των εισαγόμενων globals κάτι που είναι αναγκαίο για τον στατικό έλεγχο.

3.2 Εντολές

Κάθε εντολή στο binary αρχείο του module αντιστοιχίζεται με ένα αντικείμενο της κλάσης `Instr`. Για να μην υπάρχει επανάληψη πληροφορίας, λεπτομέρειες όπως ο τύπος της εντολής ή το όνομά της αποθηκεύονται σε ένα ξεχωριστό διάνυσμα τύπου `InstrProfile` το οποίο ονομάζεται profiles. Αυτό το διάνυσμα λειτουργεί σαν κατάλογος όλων των εντολών και η αρχικοποίησή του γίνεται σε δύο βήματα. Πρώτα, αρχικοποιείται το διάνυσμα `p_profiles` στο αρχείο `instructions.cpp` (Σχήμα 3.2) και αυτό το διάνυσμα αντιγράφεται στο `profiles`, στο αρχείο `main.cpp`.

```
struct Module {
  vec<type::Func> types;
  vec<Func> funcs;
  vec<Table> tables;
  vec<Memory> mems;
  vec<Global> globals;
  vec<Elem> elem;
  vec<Data> data;
  Start start;
  vec<Import> imports;
  vec<Export> exports;
  int imported_globals = 0;
  ...
}
```

Σχήμα 3.1: Δομή για το WebAssembly module – ast.hpp

```
vec<InstrProfile *> p_profiles = {
new InstrFactory<0x00, Unreachable>("unreachable", {ptype(1, true, false)},
                                    {ptype(2, true, false)}),
new InstrFactory<0x01, Nop>("nop", {}, {}),
new InstrFactory<0x02, Block>("block [t?]", {}, {ptype(0, true, false)}),
new InstrFactory<0x03, Loop>("loop [t?]", {}, {ptype(0, true, false)}),
new InstrFactory<0x04, If>("if [t?]", {type::Value::i32},
                           {ptype(0, true, false)}),
new InstrFactory<0x05, Nop>("else", {}, {}),
new InstrFactory<0x0B, End>("end", {}, {}),
...
```

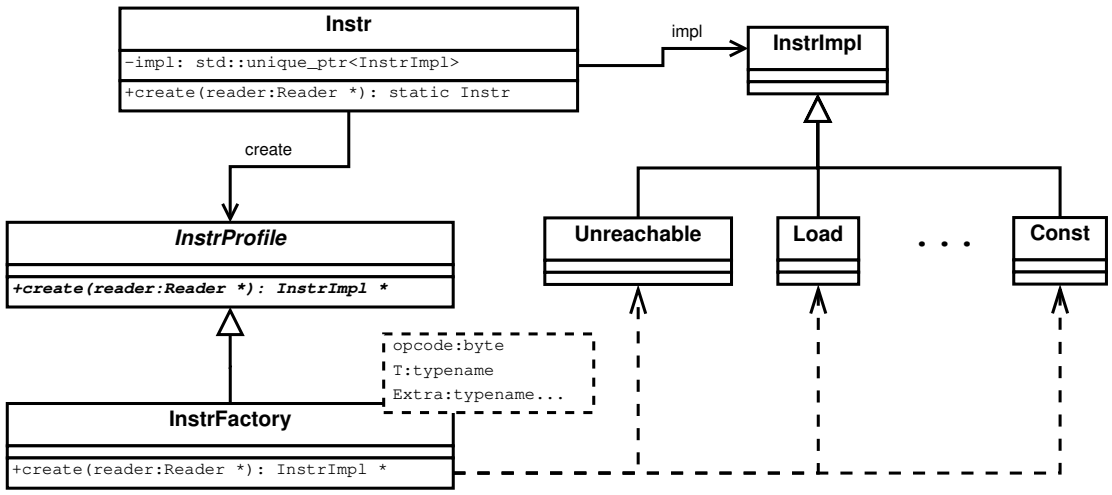
Σχήμα 3.2: Τμήμα από την αρχικοποίηση του καταλόγου των εντολών – instructions.cpp

Οι εντολές στην WebAssembly μπορούν να διαφέρουν σημαντικά μεταξύ τους και η κατασκευή του κάθε αντικειμένου εξαρτάται από την είσοδο. Έτσι επιλέχθηκε η χρήση του Factory μοντέλου για την κλάση **`Instr`**. Κάθε **`Instr`** περιέχει ένα αντικείμενο **`InstrImpl`**. Αυτή η κλάση αποτελεί την αρχή μιας ιεραρχίας κλάσεων οι οποίες μοντελοποιούν κάθε εντολή – Σχήμα 3.4. Για την κατασκευή του κατάλληλου αντικειμένου χρησιμοποιείται το διάνυσμα **`profiles`**. Όπως φαίνεται και στο σχήμα 3.2 το διάνυσμα **`profiles`** στην πραγματικότητα περιέχει αντικείμενα InstrFactory. Αυτή η κλάση κληρονομεί από την **`InstrImpl`** και υλοποιεί την abstract συνάρτηση **`create`**. Επίσης η κλάση παραμετροποιείται με το opcode της εντολής και με έναν τύπο, επομένως, κάθε στοιχείο του διανύσματος profiles μπορεί να κατασκευάσει την κατάλληλη κλάση με τη μέθοδο create που υλοποιεί.

3.3 Αποκωδικοποίηση και η κλάση Reader

Στην κλάση Reader (ο κώδικάς της δίνεται στο Παράρτημα A) ενθυλακώνεται όλη την πληροφορία σχετικά με το διάβασμα από το binary αρχείο του module. Για τη δημιουργία ενός αντικειμένου της κλάσης απαιτείται το όνομα του αρχείου διότι, ο κατασκευαστής καλεί τη μέθοδο **`load`** η οποία το ανοίγει και διαβάζει τα περιεχόμενά του στο διάνυσμα **`bytes`**. Έτσι κάθε Reader αντικείμενο αντιστοιχίζεται με ένα και μόνο αρχείο.

Για την ανάγνωση η κλάση υλοποιεί μεθόδους για κάθε τύπο δεδομένων, καθώς και μεθόδους για πιο σύνθετες δομές, όπως για το διάβασμα όλων των συναρτήσεων. Σε πολλούς από τους ελέγχους διαπιστώθηκε πως τα αρχεία εισόδου μπορεί να διακόπτονται απρόοπτα με αποτέλεσμα να χρειάζεται



Σχήμα 3.3: Factory για την κλάση Instr.

συνεχώς να γίνεται έλεγχος σχετικά με τα διαθέσιμα bytes προς ανάγνωση. Για να αποφευχθεί αυτή η επανάληψη άμεση πρόσβαση στο διάνυσμα έχουν μόνο οι ακόλουθες συναρτήσεις:

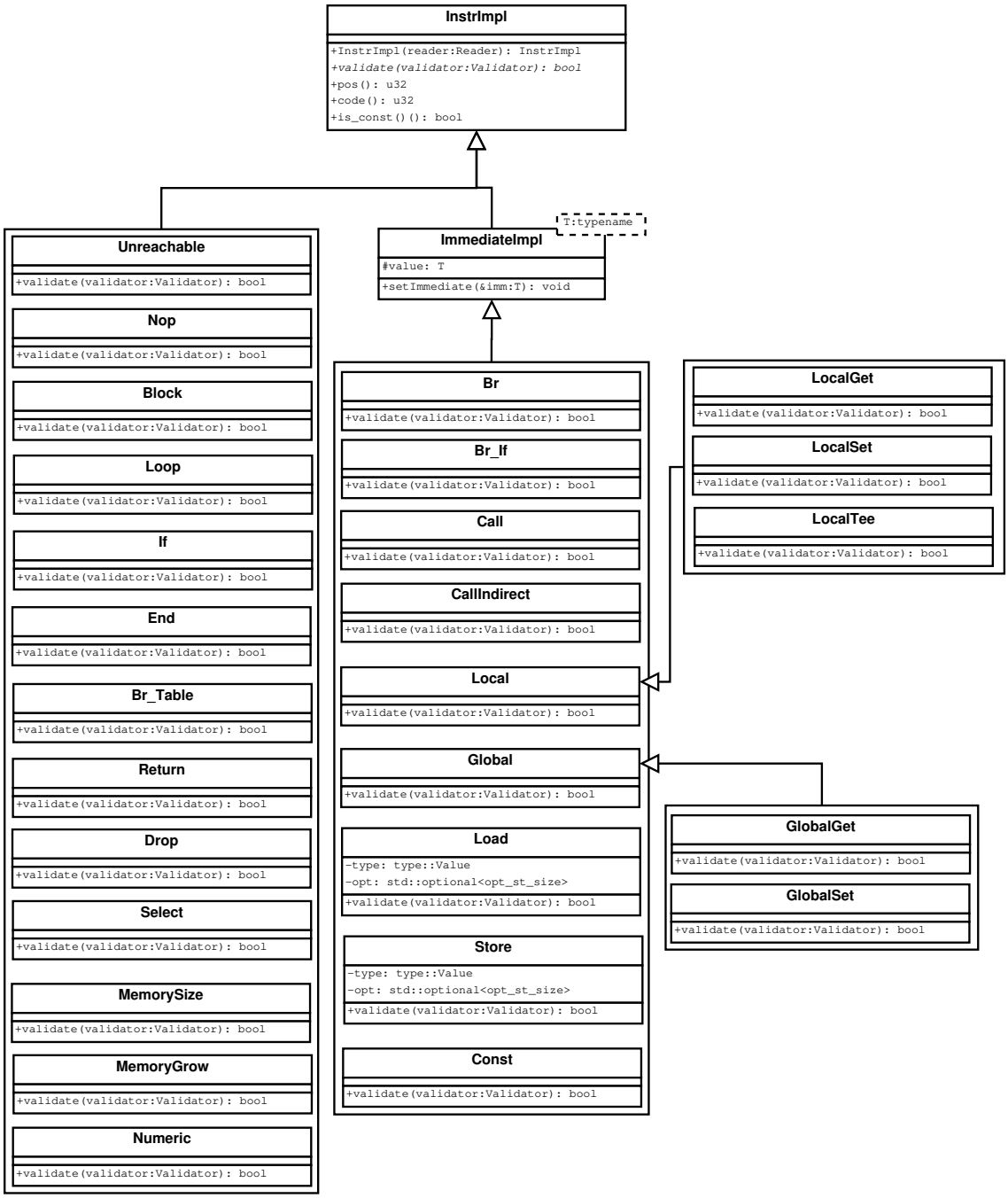
`peek_byte`	επιστρέφει το επόμενο byte χωρίς να το καταναλώνει
`read_byte`	επιστρέφει και καταναλώνει το επόμενο byte
`read_u32`	διαβάζει έναν ακέραιο αριθμό 32 bit που δεν έχει κωδικοποιηθεί
`read_name`	διαβάζει μια συμβολοσειρά και ελέγχει ότι είναι utf8

Περιορίζοντας την πρόσβαση στο διάνυσμα **`bytes`**, περιορίζεται και η ανάγκη για ελέγχους. Στις παραπάνω συναρτήσεις γίνονται δύο διαφορετικοί τύποι διαβάσματος. Αρχικά συναρτήσεις όπως η **`read_byte`** διαβάζουν σειριακά ένα ή περισσότερα bytes και χρησιμοποιούν την μέθοδο **`.at()`** που προσφέρουν τα διανύσματα στην C++. Η μέθοδος αυτή σε περίπτωση που δεν υπάρχει επόμενο byte, θα εγείρει μία εξαίρεση. Συναρτήσεις όπως η **`read_name`** διαβάζουν ένα τμήμα του **`bytes`**, οπότε χρειάζεται διαφορετικός έλεγχος. Σε αυτόν τον έλεγχο χρησιμοποιείται η συνάρτηση **`skip(u32 k=1)`** η οποία αυξάνει τη θέση στο διάνυσμα κατά **`k`** και επιστρέφει την προηγούμενο θέση. Σε περίπτωση που η τελική θέση είναι εκτός του διανύσματος, παράγει ένα σφάλμα και το πρόγραμμα τερματίζει.

Τέλος, κατά σύμβαση οι μέθοδοι που διαβάζουν ένα μη τετμημένο κατασκεύασμα της WebAssembly ονομάζονται **`parse_<όνομα κατασκευάσματος>`**. Επίσης, όσες μέθοδοι διαβάζουν κάποιο από τα τμήματα του module λαμβάνουν ως όρισμα μία αναφορά στο αντίστοιχο τμήμα και έχουν τύπο **`void`** (π.χ. **`void parse_imports(vec<Import> &);`**).

3.4 Επικύρωση και η κλάση Validator

Σε αυτή την κλάση (ο κώδικάς της δίνεται στο Παράρτημα B) υλοποιείται ο μηχανισμός του στατικού ελέγχου. Ο στατικός έλεγχος έχει ως εσωτερική κατάσταση το περιβάλλον επικύρωσης (validation context – Σχήμα 3.6). Το περιβάλλον εκτέλεσης αποτελεί μια δομή η οποία διαθέτει όλες τις πληροφορίες σχετικά με τους τύπους του module. Έτσι κάθε πεδίο της δομής Module βρίσκει το αντίστοιχό του στο περιβάλλον. Βεβαίως εξαιρούνται τα πεδία που είτε δεν φέρουν πληροφορία σχετικά με τους τύπους ή η πληροφορία τους βρίσκεται και σε κάποιο άλλο σημείο. Εξαιρούνται δηλαδή τα:



Σχήμα 3.4: Η ιεραρχία κλάσεων για τις εντολές.

`data` και **`elem`**	διότι αρχικοποιούν το module και δεν περιέχουν πληροφορίες τύπων
`start`	αυτό το πεδίο απλά αναφέρεται σε μία από τις συναρτήσεις
`imports`	όλοι οι εισαγόμενοι ορισμοί βρίσκονται στα αντίστοιχα πεδία, ανάλογα με το είδος τους (func, table, mem, global) [impo]
`exports`	περιέχει δείκτες προς τους ορισμούς που εξάγονται

Το περιβάλλον επικύρωσης διαθέτει επιπρόσθετα και τα πεδία **`locals`**, **`labels`**, **`return_`**. Τα πεδία αυτά χρησιμοποιούνται στο πλαίσιο της επικύρωσης των συναρτήσεων του module. Κατά τη διαδικασία του στατικού ελέγχου του κώδικα μίας συνάρτησης (έλεγχος τύπων) χρειάζεται γνώση των

τοπικών μεταβλητών και του τύπου επιστροφής. Οι ετικέτες (labels) είναι επίσης σημαντικές, λόγω του μηχανισμού που διαθέτει η WebAssembly για τις εντολές ελέγχου. Συνοπτικά, η WebAssembly ακολουθεί την μορφή του δομημένου προγραμματισμού, εντάσσει δηλαδή τις εντολές σε δομημένα block και επιτρέπει εντολές ελέγχου να κάνουν άλματα μόνο σε αυτά. Οι ετικέτες αντιστοιχίζονται με αυτά τα σημεία στον κώδικα και έχουν τύπο ίδιο με τον τύπο του block. Άρα ετικέτες εισάγονται από τις εντολές **`block`**, **`loop`**, **`if/else`**.

3.4.1 Συναρτήσεις και εκφράσεις

Κάθε συνάρτηση αποτελείται από τον τύπο της, τις τοπικές μεταβλητές (σε αυτές ανήκουν και οι παράμετροι) και τον κώδικά της. Ο κώδικας είναι μία έκφραση (Expr) η οποία είναι ισοδύναμη με ένα διάνυσμα από εντολές. Ο αλγόριθμος επικύρωσης μίας έκφρασης [vala] ορίζει δύο στοίβες: η πρώτη είναι η στοίβα των τελουμένων και η δεύτερη είναι η στοίβα ελέγχου. Κάθε εντολή χρησιμοποιεί αυτές τις στοίβες για να ελέγξει την εγκυρότητά της διαφορετικά. Έτσι κάθε κλάση που μοντελοποιεί μια εντολή διαθέτει τη μέθοδο **`bool validate(Validator *validator)`** (Σχήμα 3.4). Αυτή η μέθοδος χρησιμοποιεί τις συναρτήσεις που παρέχει το αντικείμενο validator για τον χειρισμό των στοιβών και επιστρέφει true αν η εντολή είναι έγκυρη. Έτσι η κλάση Validator, δεν χρειάζεται να έχει γνώση σχετικά με το ποιες εντολές υπάρχουν στο πρότυπο και μπορεί να υλοποιήσει τη μέθοδο **`Validator::exprs()`** ανεξάρτητα (Σχήμα 3.5).

```
1  bool Validator::expr(Expr &ex) {
2    for (auto &instr : ex) {
3      if (!(instr.validate(this))) return false;
4    }
5    return true;
6  }
```

Σχήμα 3.5: Η μέθοδος που επικυρώνει εκφράσεις – validate.cpp.

3.4.2 Global μεταβλητές

Η επικύρωση των περισσοτέρων στοιχείων του module, εκτός των συναρτήσεων, είναι σχετικά τετριμμένη διαδικασία. Όμως, η επικύρωση του διανύσματος με τις global μεταβλητές έχει ενδιαφέρον. Σύμφωνα με το πρότυπο [glob] αφού επικυρωθεί ότι ο τύπος του global είναι ορθός και συνεπής με το όρισμά του, ελέγχεται αν το όρισμα είναι σταθερό. Αυτό συμβαίνει μόνο όταν όλες οι εντολές είναι είτε:

i. *t.***`const`**, δηλαδή εντολές που δηλώνουν σταθερές, ή

ii. η εντολή **`global.get`** η οποία να διαβάζει από κάποιο immutable global

Υπάρχει όμως και ένας ακόμα κανόνας ο οποίος δεν είναι εμφανής εκ πρώτης όψεως. Κατά τη διαδικασία επικύρωσης τα ορίσματα των μεταβλητών στο διάνυσμα globals δεν μπορούν να είναι αμοιβαία αναδρομικά. Για αυτό το λόγο, για την επικύρωσή τους χρησιμοποιείται ένα διαφορετικό περιβάλλον στο οποίο υπάρχουν μόνο το εισαγόμενα globals. Για να επιτευχθεί αυτός ο έλεγχος χρησιμοποιείται η μεταβλητή **`imported_globals`**. Σύμφωνα με το πρότυπο τα πρώτα στοιχεία του διανύσματος είναι τα εισαγόμενα. Εφόσον είναι γνωστό το πλήθος τους, μία **`global.get`** εντολή είναι "σταθερή" (δηλαδή, μπορεί να χρησιμοποιηθεί στο όρισμα μιας global μεταβλητής) αν και μόνο αν ο δείκτης στο αντίστοιχο global είναι μικρότερος από την τιμή της μεταβλητής **`imported_globals`**. Έτσι δεν χρειάζεται πραγματικά η δημιουργία ενός νέου περιβάλλοντος επικύρωσης για τα globals και αρκεί μια σύγκριση μεταξύ ακεραίων αριθμών.

```
struct Context {
  vec<type::Func> types;
  vec<type::Func> funcs;
  vec<type::Table> tables;
  vec<type::Memory> mems;
  vec<type::Global> globals;
  vec<type::Value> locals;
  vec<type::Result> labels;
  type::Result return_;
  int imported_globals = 0;
};
```

Σχήμα 3.6: Το περιβάλλον επικύρωσης – validate.hpp.

3.5 Σενάριο χρήσης

Το πρόγραμμα υλοποιεί ένα σενάριο χρήσης, παρέχοντας ταυτόχρονα και γενικεύσεις που μπορούν να χρησιμοποιηθούν για διαφορετικά σενάρια. Σε αυτό το σενάριο, το module φορτώνεται στη μνήμη με τη μορφή ενός αντικειμένου, το οποίο έχει ακριβώς τις πληροφορίες που ορίζει το πρότυπο. Αφού αρχικοποιηθεί η δομή, γίνεται και η αρχικοποίηση μιας δομής Reader. Η κλάση Reader ενθυλακώνει όλες τις λειτουργίες και τις πληροφορίες που χρειάζονται κατά το διάβασμα. Έπειτα το αντικείμενο reader αναλαμβάνει να διαβάσει το binary αρχείο και να το φορτώσει στο module. Κατά της διάρκεια αυτής της διαδικασίας παράγει μηνύματα αποσφαλμάτωσης και ειδοποιήσεις. Σε περίπτωση που εντοπιστεί κάποιο σφάλμα, το πρόγραμμα τερματίζει. Εφόσον το διάβασμα πετύχει το επόμενο βήμα είναι η αρχικοποίηση της δομή Validator για τον στατικό έλεγχο του module. Όμοια με τον reader, ο validator εμπεριέχει την εσωτερική κατάσταση του στατικού ελέγχου καθώς και όλες τις μεθόδους που χρειάζονται. Εφόσον πετύχει και ο στατικός έλεγχος τότε το πρόγραμμα τερματίζει επιτυχώς. Διαφορετικά το πρόγραμμα τερματίζει με μήνυμα σφάλματος.

Η παραπάνω διαδικασία δεν είναι δεσμευτική βέβαια. Οι κλάσεις Reader και Validator μπορούν να χρησιμοποιηθούν σαν μαύρα κουτιά, καθώς δεν επηρεάζουν την εσωτερική κατάσταση του προγράμματος. Αυτός είναι και βασικός κανόνας στην αρχιτεκτονική της εφαρμογής. Κάθε λειτουργικότητα οφείλει να παρέχει ένα black box interface.

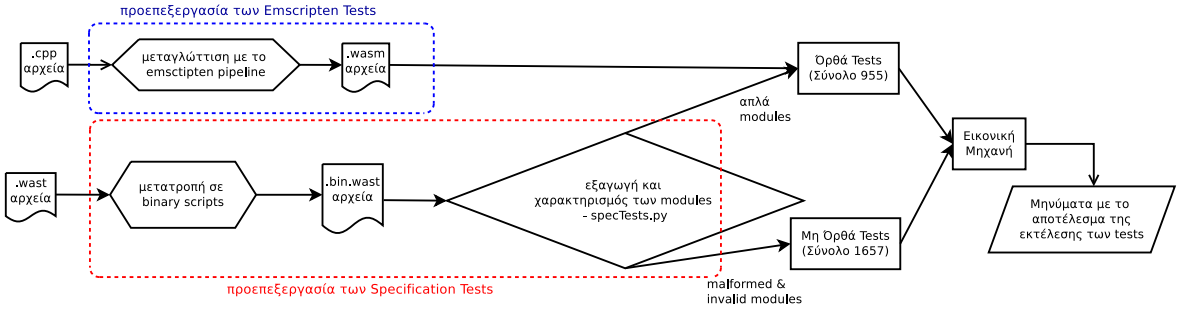
Κεφάλαιο 4

Έλεγχος - Testing

Για τον έλεγχο της ορθής λειτουργίας της εικονικής μηχανής η οποία υλοποιήθηκε στο πλαίσιο της παρούσας Εργασίας, αξιοποιήθηκε διακριτό περιβάλλον δοκιμών, το οποίο επίσης αναπτύχθηκε στο πλαίσιο της Εργασίας. Στην ενότητα αυτή περιγράφεται το περιβάλλον δοκιμών, ως προς το μοντέλο λειτουργίας του, την "είσοδο" που δέχεται, την επεξεργασία που κάνει και τα αποτελέσματα που παράγει (Σχήμα 4.1).

Το περιβάλλον δοκιμών εφαρμόζει διαδικασία black-box testing χρησιμοποιώντας WebAssembly modules. Αυτά τα modules μπορούν να είναι είτε ορθά (δεν έχουν κανένα συντακτικό ή σημασιολογικό σφάλμα), ή μη ορθά και άρα αναμένεται ότι η εικονική μηχανή θα εντοπίσει το σφάλμα είτε στην ανάγνωση είτε στην επικύρωση τους, και θα τερματίσει με κάποιο κωδικό λάθους. Επίσης, σε περίπτωση που υπάρχει κάποιο λάθος στη διαχείριση μνήμης η εικονική μηχανή δύναται να τερματίσει με segmentation fault και το συγκεκριμένο test θεωρείται αποτυχημένο. Έτσι η διαδικασία ελέγχου διαθέτει test cases – προγράμματα εισόδου σε WebAssembly, τα οποία τροφοδοτεί στην εικονική μηχανή. Όταν η εκτέλεση ολοκληρωθεί ελέγχεται ο κωδικός τερματισμού και διαπιστώνεται η επιτυχία ή μη στο συγκεκριμένο test case. Η παραπάνω διαδικασία επαναλαμβάνεται για όλα τα διαθέσιμα προγράμματα εισόδου και εμφανίζει αποτελέσματα σχετικά με την επιτυχία.

Τα προγράμματα εισόδου, στην τρέχουσα υλοποίηση, προέρχονται από δύο πηγές. Αρχικά, έγινε μια συλλογή προγραμμάτων γραμμένων σε cpp, τα οποία μεταγλωττίστηκαν σε WebAssembly με χρήση του emscripten pipeline [emcc]. Το emscripten είναι το πιο διαδεδομένο εργαλείο για παραγωγή WebAssembly προγραμμάτων, οπότε είναι σημαντική η γνώση της συμβατότητας με αυτό. Η δεύτερη πηγή προγραμμάτων είναι η επίσημη σουίτα ελέγχου [test] που παρέχεται στο αποθετήριο του προτύπου. Από τις 3 διαθέσιμες στη σουίτα κατηγορίες ελέγχων, εφαρμογή βρίσκουν τα core tests. Στις ενότητες που ακολουθούν, παρουσιάζεται αναλυτικά η διαδικασία επεξεργασίας των προγραμμάτων εισόδου, η διαδικασία ελέγχου εν γένει καθώς επίσης και τα αποτελέσματα αυτής.



Σχήμα 4.1: Η διαδικασία ελέγχου της εικονικής μηχανής.

4.1 Emscripten tests

Η παραγωγή των προγραμμάτων με το emscripten είναι απλή. Αρχικά χρειάζεται το περιβάλλον εκτέλεσης των ελέγχων να διαθέτει το emscripten. Στη συνέχεια και σε ένα βήμα χρησιμοποιείται το

εκτελέσιμο **em++** για τη μεταγλώττιση των προγραμμάτων. Κάθε πρόγραμμα είναι ένα αρχείο με κατάληξη **.cpp**.Το παραγόμενο (μετά τη μεταγλώττιση με το emscripten) πρόγραμμα εισόδου διατηρεί το ίδιο όνομα, αλλά με κατάληξη **.wasm**. Για την αυτοματοποίηση αυτής της διαδικασίας δημιουργήθηκε ένα Makefile το οποίο μεταγλωττίζει 16 προγράμματα C++ και παράγει 16 test cases.

4.2 Specification tests

Στη περίπτωση της αξιοποίησης των test cases που παρέχονται από το πρότυπο, η πολυπλοκότητα της προεπεξεργασίας αυξάνεται. Αρχικά, τα tests είναι γραμμένα χρησιμοποιώντας το WebAssembly Text Format (wast), ενώ η εικονική μηχανή αναγνωρίζει το binary format (wasm). Επίσης, κάθε test στο αποθετήριο δεν είναι απλά ένα module, αλλά αποτελεί ένα απλό script με ορισμούς module, assertions, actions και άλλα [sexp].

Το πρώτο πρόβλημα μπορεί να λυθεί εύκολα χρησιμοποιώντας τον μεταγλωττιστή που παρέχεται από το πρότυπο, για μετατροπή των **.wast** αρχείων σε **.bin.wast**. Αυτά τα αρχεία περιέχουν τα binaries που πρέπει να ελεγχθούν. Κάθε **.bin.wast** αρχείο αποτελεί ένα αντίγραφο του αντίστοιχου **.wast** αρχείου, με μόνη διαφορά ότι τα modules βρίσκονται σε binary format. Το επόμενο βήμα είναι η εξαγωγή των modules από τα **.bin.wast** αρχεία και ο χαρακτηρισμός τους ανάλογα με τον τρόπο που ορίζονται σε αυτά. Από όλες τις κατηγορίες module που μπορεί να ορίζονται σε ένα τέτοιο script ενδιαφέρον έχουν:

i. τα απλά modules (συντακτικά και σημασιολογικά ορθά),

ii. τα malformed modules (ορίζονται σε ένα assert_malformed statement του script),

iii. και τα invalid modules (αντίστοιχα ορίζονται σε ένα assert_invalid statement)

Τα δύο τελευταία χαρακτηρίζονται ως μη ορθά και άρα περιμένουμε ότι η εικονική μηχανή θα αναγνωρίσει το λάθος και το πρώτο ως ορθό, οπότε η εικονική μηχανή θα τερματίσει χωρίς κάποιο μήνυμα σφάλματος.

Για τη διαδικασία της εξαγωγής αρχικά γίνεται μια τροποποίηση των αρχείων **.bin.wast** με εντολές **sed** και στη συνέχεια χρησιμοποιείται ένα script γραμμένο σε python το οποίο αντλεί όλα τα "ενδιαφέροντα" modules και τα τοποθετεί σε κατάλληλους φακέλους ανάλογα με τον χαρακτηρισμό τους (ορθά ή μη). Αυτή η προεπεξεργασία παράγει συνολικά 2.596 test cases με τα οποία θα τροφοδοτηθεί ο μηχανισμός ελέγχου. Από αυτά, τα 939 χαρακτηρίζονται ως ορθά και τα 1.657 ως μη ορθά.[1].

4.3 Διαδικασία ελέγχου – test.sh

Για το testing παρέχεται ένα Linux Shell Script – test.sh (βλ. Παράρτημα C), το οποίο χειρίζεται όλες τις πηγές προγραμμάτων. Το script μπορεί να χρησιμοποιηθεί για εκτέλεση των emscripten tests (cpp tests), των test που παρέχονται από το πρότυπο (spec tests) ή και των δύο. Επίσης, σε περίπτωση εκτέλεσης των spec tests, το script θα μεταφορτώσει τα tests που αντιστοιχούν στην έκδοση του προτύπου (commit id) που οριζεται στο αρχείο **tests/wasmCommit** (Σχήμα 4.2), και στη συνέχεια θα ακολουθήσει η προαναφερθείσα προεπεξεργασία. Έτσι η εικονική μηχανή μπορεί να ελέγχεται για οποιαδήποτε έκδοση του προτύπου[2].

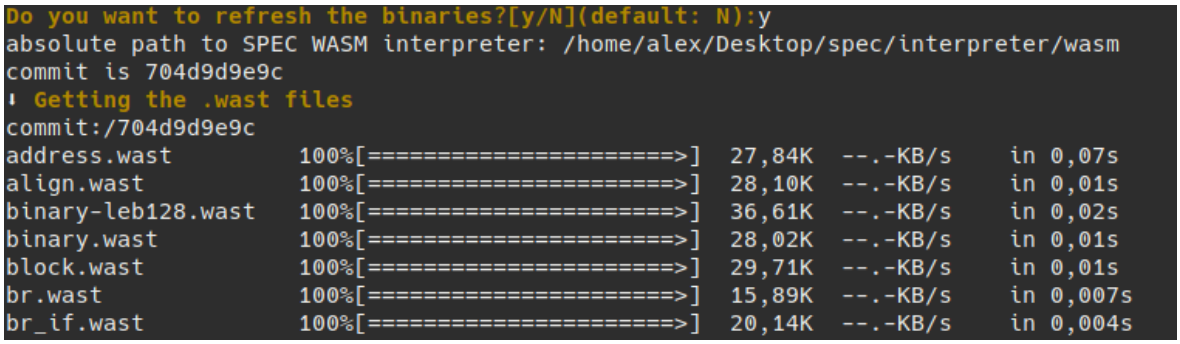
Με την ολοκλήρωση των βημάτων προεπεξερασίας για κάθε πηγή (cpp – Σχήμα 4.3, spec – Σχήματα 4.4, 4.5) όλα τα προγράμματα εισόδου βρίσκονται στον φάκελο **tests/bins** αν χαρακτηρίζονται ορθά[3] και στον φάκελο **tests/bins/fail** αν δεν είναι ορθά.

Με τα προγράμματα εισόδου στους κατάλληλους φακέλους, το script καλεί την εικονική μηχανή, επαναληπτικά τροφοδοτώντας την με αυτά, και ελέγχει τον τερματισμό της. Αρχικά την τροφοδοτεί

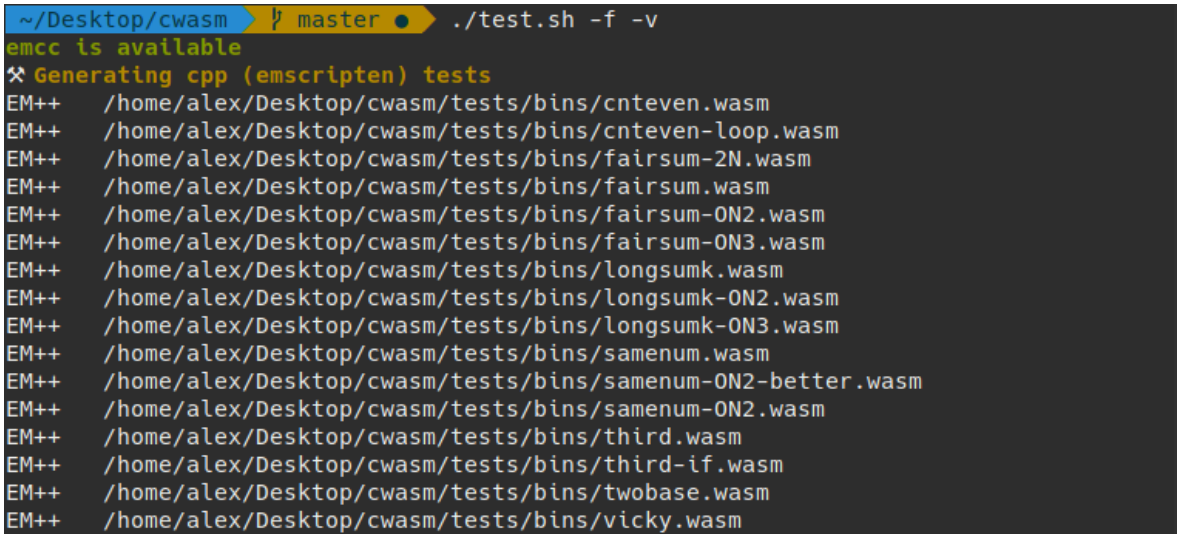
[1] Για την αυτοματοποίηση αυτής της διαδικασίας, χρησιμοποιήθηκε επίσης Makefile

[2] Η εργασία ολοκληρώθηκε ακολουθώντας το commit #704d9d9e9c

[3] Όλα τα emscripten tests τοποθετούνται εκεί



Σχήμα 4.2: Μεταφόρτωση των αρχείων wast από το αποθετήριο της WebAssembly.



Σχήμα 4.3: Κλήση του Makefile για τα emscripten tests.

με όλα τα ορθά προγράμματα εισόδου (Σχήμα 4.6) και αναμένει τον τερματισμό της εικονικής μηχανής χωρίς επιστροφή κωδικού σφάλματος. Αν το πρόγραμμα επιστρέψει κάποιον τέτοιο κωδικό, τότε το test θεωρείται πως έχει αποτύχει. Στη συνέχεια, η μηχανή τροφοδοτείται με τα μη ορθά προγράμματα εισόδου (Σχήμα 4.7). Εδώ αναμένεται να επιστραφεί κωδικός σφάλματος, για να θεωρηθεί επιτυχημένο το test. Επομένως αν κάποιο από τα μη ορθά tests επεξεργαστεί χωρίς κάποιο πρόβλημα, θεωρούμε ότι υπάρχει αποτυχία στο test case.

Τέλος, εμφανίζονται μηνύματα σχετικά με την επιτυχία στα ορθά, στα μη ορθά και στο σύνολο στων tests (Σχήμα 4.8).

Η εικονική μηχανή επιτυγχάνει σε 2604 από τους 2612 ελέγχους. Οι 8 έλεγχοι στους οποίους αποτυγχάνει είναι μη ορθά προγράμματα εισόδου (προερχόμενα από τα spec tests) τα οποία αναγνωρίζει ως ορθά. Τα προγράμματα αυτά, ανήκουν σε μια κατηγορία ελέγχων της αποκωδικοποίησης leb128. Συγκεκριμένα, τροφοδοτούν την συνάρτηση αποκωδικοποίησης με εισόδους που δεν θα μπορούσαν να είχαν προκύψει από την κωδικοποίηση, για να επιβεβαιωθεί πως δεν θα υπάρχουν προβλήματα υπερχείλισης κλπ. Το πρότυπο προβλέπει εξουθενωτικούς ελέγχους ως προς αυτή τη λειτουργία. Στην τρέχουσα έκδοση της εικονικής μηχανής, έχει ενσωματωθεί ένα μέρος των εν λόγω ελέγχων και όχι το σύνολό τους, με αποτέλεσμα να αποτυγχάνει να εντοπίσει λάθη που υπάρχουν (σκόπιμα) σε αυτά τα 8 προγράμματα. Επόμενες εκδόσεις της εικονικής μηχανής θα μπορούσαν να περιλαμβάνουν εκτενέστερους ελέγχους ως προς την αποκωδικοποίηση των leb128 κωδικοποιημένων ακεραίων.

```
74 files downloaded
⚒ Generating .bin.wast files
"Core dir OK!"
"Wasm Interpreter is probably the right one"
WAST2WASM address.bin.wast
WAST2WASM align.bin.wast
WAST2WASM binary-leb128.bin.wast
WAST2WASM binary.bin.wast
WAST2WASM block.bin.wast
WAST2WASM break-drop.bin.wast
WAST2WASM br_if.bin.wast
```

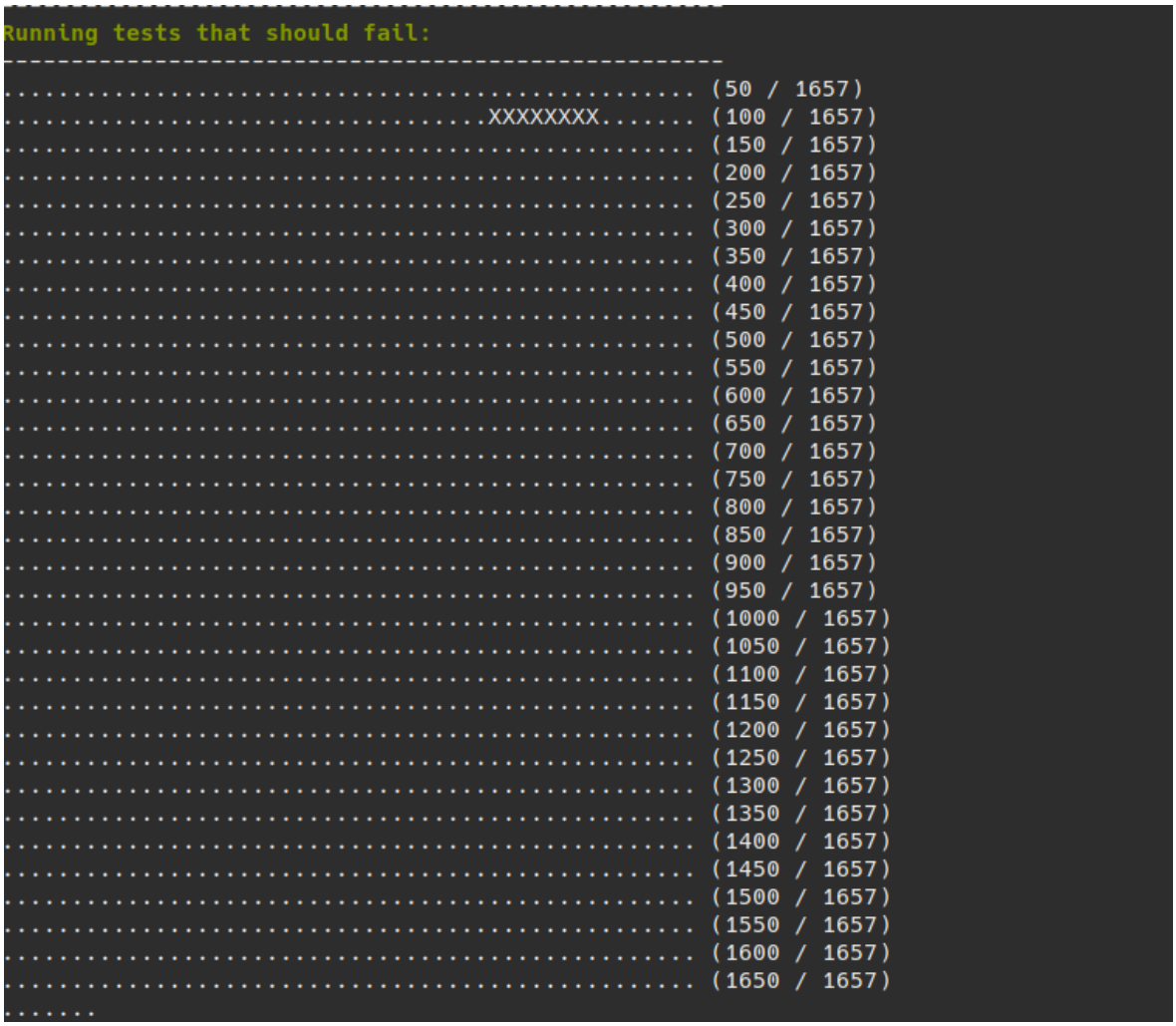
Σχήμα 4.4: Μετατροπή των .wast αρχείων σε .bin.wast.

```
WAST2WASM utf8-custom-section-id.bin.wast
WAST2WASM utf8-import-field.bin.wast
WAST2WASM utf8-import-module.bin.wast
WAST2WASM utf8-invalid-encoding.bin.wast
⚒ Generating spec core tests /home/alex/Desktop/cwasm
👍 all test files were processed correctly
------------------------------------------------------
```

Σχήμα 4.5: Εξαγωγή των modules με χρήση του specTests.py.

```
------------------------------------------------------
Running tests:
------------------------------------------------------
.................................................. (50 / 955)
.................................................. (100 / 955)
.................................................. (150 / 955)
.................................................. (200 / 955)
.................................................. (250 / 955)
.................................................. (300 / 955)
.................................................. (350 / 955)
.................................................. (400 / 955)
.................................................. (450 / 955)
.................................................. (500 / 955)
.................................................. (550 / 955)
.................................................. (600 / 955)
.................................................. (650 / 955)
.................................................. (700 / 955)
.................................................. (750 / 955)
.................................................. (800 / 955)
.................................................. (850 / 955)
.................................................. (900 / 955)
.................................................. (950 / 955)
.....
```

Σχήμα 4.6: Εκτέλεση των ορθών ελέγχων (“.”=επιτυχία,“X”=αποτυχία.)



Σχήμα 4.7: Εκτέλεση των μη ορθών ελέγχων (“.”=επιτυχία,“X”=αποτυχία.)

```
show the failed tests?[y/N](default: N) y

 binary-leb128_module_73.wasm
 binary-leb128_module_74.wasm
 binary-leb128_module_75.wasm
 binary-leb128_module_76.wasm
 binary-leb128_module_77.wasm
 binary-leb128_module_78.wasm
 binary-leb128_module_79.wasm
 binary-leb128_module_80.wasm

         955 / 955 correct tests passed
        1649 /1657 incorrect tests passed

✔ 2604/2612 tests passed!
```

Σχήμα 4.8: Εκτύπωση των tests που απέτυχαν και των ποσοστών επιτυχίας.

Κεφάλαιο 5

Σύνοψη και Μελλοντικές επεκτάσεις

5.1 Σύνοψη

Στόχος της εργασίας ήταν η ανάπτυξη μιας εικονικής μηχανής για WebAssembly. Για το σκοπό αυτό, διερευνήθηκε το πρότυπο της WebAssembly εστιάζοντας στη δυαδική κωδικοποίηση των προγραμμάτων αλλά και τον μηχανισμό επικύρωσής τους. Ακολούθως, σχεδιάστηκε και κατασκευάστηκε η υλοποίηση της εικονικής μηχανής με τρόπο που να διευκολύνει την κατανόηση, την συντήρηση και την επέκτασή της. Στο παρόν κείμενο αναλύθηκαν οι σχεδιαστικές επιλογές και η αρχιτεκτονική της υλοποίησης παρουσιάζοντας τις βασικές κλάσεις και λειτουργίες. Για την επιβεβαίωση της ορθής λειτουργίας της υλοποιημένης εφαρμογής, δομήθηκε μηχανισμός για την εκτέλεση ελέγχων (testing). Ο εν λόγω μηχανισμός χρησιμοποιεί προγράμματα εισόδου με τα οποία τροφοδοτεί την εικονική μηχανή, για να ελέγξει την ορθότητα και την ικανότητά της να ανιχνεύει σφάλματα. Τα προγράμματα προέρχονται από:

- χρήση του emscripten toolchain για μετάφραση προγραμμάτων σε C++, ώστε να επιβεβαιωθεί η συμβατότητα με τα παραγόμενα binaries.
- αξιοποίηση της σουίτας ελέγχου που παρέχεται από το πρότυπο, με την απαραίτητη προεπεξεργασία.

Έχοντας μια υλοποίηση για την οποία υπάρχει ισχυρή βεβαιότητα για την ορθότητά της, δίνεται η ευκαιρία για επέκτασή της για τον "συγχρονισμό" της με το συνεχώς εξελισσόμενο πρότυπο, αλλά και για την υλοποίηση περισσότερων λειτουργιών και ιδεών.

5.2 Μελλοντικές επεκτάσεις

Όπως έχει ήδη αναφερθεί, οι σχεδιαστικές επιλογές για την ανάπτυξη της εικονικής μηχανής, στόχευαν στη συντηρησιμότητα και επεκτασιμότητα του τελικού προϊόντος. Παρακάτω παρουσιάζονται μερικές ιδέες σχετικά με την εξέλιξη της εικονικής μηχανής στο μέλλον:

- **Προσθήκη λειτουργικότητας για την εκτέλεση των modules**

  Στην τρέχουσα έκδοσής της, η εικονική μηχανή μπορεί να κάνει στατικό έλεγχο ενός module, αλλά δεν μπορεί να το εκτελέσει. Η λειτουργία αυτή μπορεί να προστεθεί αρχικά σαν ένας μεταφραστής (interpreter) ο οποίος θα εκτελεί δυναμικά συναρτήσεις από το φορτωμένο στη μνήμη module. Αργότερα μπορεί να υλοποιηθεί ένας JIT compiler, όπως συνηθίζεται για την WebAssembly. Ενδεικτικά θα πρέπει να υλοποιηθεί το instantiation του module και των δομών που απαιτούνται για την εκτέλεσή του. Αφού ολοκληρωθεί αυτό, πρέπει να επεκταθεί ο μηχανισμός ελέγχου για να συμπεριλαμβάνει και τα test cases από τη σουίτα της WebAssembly, τα οποία ελέγχουν την ορθότητα της εκτέλεσης. Εφόσον επιλυθούν όλα τα προβλήματα, μπορεί να χρησιμοποιηθεί το LLVM για υλοποίηση ενός JIT compiler, για να επιτευχθεί η ταχύτερη δυνατή εκτέλεση.

- **Υποστήριξη Text Format**

  Η εικονική μηχανή μπορεί να αντιληφθεί μόνο το binary format της WebAssembly, με αποτέλεσμα να υπάρχει ανάγκη για τον μεταγλωττιστή του προτύπου στη διαδικασία του testing (Σχήμα 4.4). Για να μην υπάρχει αυτή η εξάρτηση στο μέλλον προτείνεται να προστεθεί υποστήριξη και για το text format.

- **Διαχείριση εξαιρέσεων και μηνυμάτων αποσφαλμάτωσης**

  Προτείνεται να οριστεί και να υλοποιηθεί μια σύμβαση σχετικά με την διαχείριση των σφαλμάτων και των αντίστοιχων μηνυμάτων που εμφανίζονται. Η χρήση κατάλληλων εξαιρέσεων μπορεί να παρέχει αυτή τη λειτουργία. Στην ίδια κατεύθυνση προτείνεται να οριστούν συμβάσεις σχετικά με τα Logs, τα Warnings, τα Debug και τα Infos που εμφανίζονται κατά την εκτέλεση, έτσι ώστε η επέκταση της υλοποίησης να μην οδηγεί σε "θορυβώδη" outputs που δυσχεραίνουν την κατανόηση της λειτουργίας του προγράμματος.

- **Υλοποίηση περισσοτέρων Σεναρίων Χρήσης**

  Θα ήταν χρήσιμο να υλοποιηθούν περισσότερα σενάρια χρήσης, όπως για παράδειγμα η εκτέλεση των ελέγχων της σουίτας απευθείας από την εικονική μηχανή χωρίς κάποια προεπεξεργασία. Εναλλακτικά, για την επίτευξη καλύτερης διεπαφής με τον χρήστη μπορεί να δημιουργηθεί ένα REPL για την εικονική μηχανή το οποίο θα παρέχει όλες τις διαθέσιμες λειτουργίες (αποκωδικοποίηση, επίκυρωση και μελλοντικά εκτέλεση).

- **Υποστήριξη για big endian συστήματα**

  Η WebAssembly χρησιμοποιεί little endian απεικόνιση για τα δεδομένα της, κάτι που δεν χρειάστηκε ιδιαίτερο χειρισμό λόγω της ανάπτυξης σε Linux. Θα ήταν σκόπιμο να τροποποιηθούν κάποιες βασικές μέθοδοι ανάγνωσης, έτσι ώστε να μπορεί η εικονική μηχανή να λειτουργεί και σε περιβάλλοντα που χρησιμοποιούν την big endian απεικόνιση.

Βιβλιογραφία

[asmj] "asm.js Website", https://asmjs.org/. Accessed: 2020-06-25.

[emcc] "Emscripten", https://emscripten.org/. Accessed: 2020-06-25.

[ewas] "Ewasm", https://github.com/ewasm. Accessed: 2020-06-27.

[GC] "Garbage Collection", https://github.com/WebAssembly/gc. Accessed: 2020-06-17.

[glob] "Globals Validation", https://webassembly.github.io/spec/core/valid/modules.html#globals. Accessed: 2020-06-24.

[Golt19] David Goltzsche, Manuel Nieke, Thomas Knauth and Rüdiger Kapitza, "AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting", in *Proceedings of the 20th International Middleware Conference*, pp. 123–135, 2019.

[Haas17] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai and JF Bastien, "Bringing the web up to speed with WebAssembly", in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 185–200, 2017.

[impo] "Imports Structure", https://webassembly.github.io/spec/core/syntax/modules.html#imports. Accessed: 2020-06-24.

[modu] "Module Validation", https://webassembly.github.io/spec/core/valid/modules.html#valid-module. Accessed: 2020-06-24.

[nonw] "Non-Web Embeddings", https://webassembly.org/docs/non-web/. Accessed: 2020-06-17.

[sexp] "Simple Scripts", https://github.com/WebAssembly/spec/tree/master/interpreter#scripts. Accessed: 2020-06-25.

[test] "Core Tests for WebAssembly", https://github.com/WebAssembly/spec/tree/master/test. Accessed: 2020-06-26.

[vala] "Validation Algorithm", https://webassembly.github.io/spec/core/appendix/algorithm.html. Accessed: 2020-06-24.

[wasm] "WebAssembly Website", https://webassembly.org/. Accessed: 2020-06-25.

[Watt18] Conrad Watt, "Mechanising and verifying the WebAssembly specification", in *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 53–65, 2018.

[Zaka11] Alon Zakai, "Emscripten: an LLVM-to-JavaScript compiler", in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pp. 301–312, 2011.

Παράρτημα Α

Κώδικας της κλάσης Reader – Reader.hpp

```
class Reader {
  public:
    Reader(const char *fname) : filename(fname), pos(0) { load(); }

    // Use this method to advance the position on the file
    std::streamsize skip(u32 k = 1) {
      std::streamsize p = pos;
      pos += k;
      if (pos > length) FATAL("trying to skip outside of the buffer");
      return p;
    }

    void expect(byte b) {
      byte next = bytes[pos++];
      ASSERT(next == b, "Expected 0x%02x instead of 0x%02x\n", b, next);
    }

    bool upto(byte b) { return peek_byte() != b; }
    bool upto(byte b1, byte b2) { return peek_byte() != b1 && peek_byte() != b2; }

    bool maybe(byte b) {
      if (bytes[pos] != b) return false;
      ++pos;
      return true;
    }
    bool unless(byte b) { return !maybe(b); }

    // Use this method if you want the value pos.
    std::streamsize get_pos() const { return pos; }

  private:
    std::string filename;
    std::vector<byte> bytes;
    std::streamsize pos;
    std::streamsize length;

    void load() {
      std::ifstream f(filename, std::ios::binary);
      if (!f) return;
      f.seekg(0, f.end);
```

```
    length = f.tellg();
    f.seekg(0, f.beg);
    bytes.resize(length + 1);
    f.read(reinterpret_cast<char *>(&bytes[0]), length);
    bytes[length] = '\0';
    pos = 0;
  }

public:
  // This is the method used for reading a module
  void parse_module(Module &m);

  // Methods for parsing the insides of a Module
  void parse_types(vec<type::Func> &);
  void parse_imports(vec<Import> &);
  void parse_funcs(vec<Func> &);
  void parse_tables(vec<Table> &);
  void parse_mems(vec<Memory> &);
  void parse_globals(vec<Global> &);
  void parse_exports(vec<Export> &);
  void parse_elems(vec<Elem> &);
  void parse_codes(vec<Func> &);
  void parse_datas(vec<Data> &);

  // Methods for parsing other, more basic constructs
  void parse_expr(Expr &);
  u32 parse_idx();
  type::Value parse_valtype();
  type::Elem parse_elemtype();
  type::Limits parse_limits();
  type::Table parse_tabletype();
  type::Memory parse_memtype();
  type::Global parse_globaltype();
  type::Block parse_blocktype();
  Memarg parse_memarg();
  Value parse_value(type::Value, bool);
  // methods that actually read from the binary file
  u32 read_u32() {
    auto prevpos = skip(sizeof(u32));
    u32 u = *reinterpret_cast<u32 *>(&bytes[prevpos]);
    return u;
  }

  byte peek_byte(int offset = 0) const { return bytes.at(pos + offset); }
  byte read_byte() { return bytes.at(pos++); }

  u64 read_LEB(unsigned maxbits, bool sign = false) {
    std::cout << ((sign) ? "signed" : "unsigned") << std::endl;
    const unsigned total_bytes = (maxbits + 7 - 1) / 7;
    u32 received_bytes = 0;

```

```
92        u64 result = 0;
93        u32 shift = 0;
94
95        u64 input;
96        while (true) {
97          received_bytes += 1;
98          if (received_bytes > total_bytes) {
99            FATAL("leb128 overflow\n");
100         }
101         input = read_byte();
102         std::cout << "input = " << std::hex << (int)input << std::dec
103                   << std::endl;
104         result |= ((input & 0x7f) << shift);
105         shift += 7;
106         if ((input & 0x80) == 0) break;
107       }
108
109       // checking for unused bits
110       if (received_bytes == total_bytes) {
111         unsigned zero_bits = total_bytes * 7 - maxbits;
112         byte check = 0xff >> (zero_bits + 1);
113         std::cout << "check = " << std::hex << (int)check << std::dec
114                   << std::endl;
115         std::cout << "input = " << std::hex << (int)input << std::dec
116                   << std::endl;
117         if (!sign && input > check) FATAL("integer too large\n");
118         check = -check;
119         std::cout << "check = " << std::hex << (int)check << std::dec
120                   << std::endl;
121         std::cout << "input = " << std::hex << (int)input << std::dec
122                   << std::endl;
123         if (sign && input > check) FATAL("integer too large\n");
124       }
125
126       if (sign && (shift < maxbits) && (input & 0x40)) {
127         // Sign extend
128         result |= -(1 << shift);
129       }
130       return result;
131     }
132
133     type::Name read_name() {
134       u32 namelen = read_LEB(32);
135       int p = skip(namelen);
136       type::Name name(bytes.begin() + p, bytes.begin() + pos);
137       if (!is_utf8(name)) {
138         FATAL("Malformed utf8 string\n");
139       }
140       return name;
141     }
142
```

```
143  bool is_utf8(std::string &the_string) {
144    unsigned index = 0;
145    std::cout << "checking utf8\n";
146
147    if (the_string.empty()) {
148      warn("the empty string is valid\n");
149      return true;
150    }
151
152    const char *c_string = the_string.c_str();
153    if (!c_string) {
154      warn("!c_string is true\n");
155      return false;
156    }
157
158    const unsigned char *bytes = (const unsigned char *)c_string;
159    while (index < the_string.size()) {
160      if (  // ASCII
161            // use bytes[0] <= 0x7F to allow ASCII control characters
162          bytes[0] <= 0x7F) {
163        warn("ascii\n");
164        index += 1;
165        bytes += 1;
166        continue;
167      }
168
169      if ((  // non-overlong 2-byte
170              (0xC2 <= bytes[0] && bytes[0] <= 0xDF) &&
171              (0x80 <= bytes[1] && bytes[1] <= 0xBF))) {
172        warn("2-byte\n");
173        index += 2;
174        bytes += 2;
175        continue;
176      }
177
178      if ((  // excluding overlongs
179              bytes[0] == 0xE0 && (0xA0 <= bytes[1] && bytes[1] <= 0xBF) &&
180              (0x80 <= bytes[2] && bytes[2] <= 0xBF)) ||
181          (  // straight 3-byte
182              ((0xE1 <= bytes[0] && bytes[0] <= 0xEC) || bytes[0] == 0xEE ||
183                bytes[0] == 0xEF) &&
184              (0x80 <= bytes[1] && bytes[1] <= 0xBF) &&
185              (0x80 <= bytes[2] && bytes[2] <= 0xBF)) ||
186          (  // excluding surrogates
187              bytes[0] == 0xED && (0x80 <= bytes[1] && bytes[1] <= 0x9F) &&
188              (0x80 <= bytes[2] && bytes[2] <= 0xBF))) {
189        warn("excluding\n");
190        index += 3;
191        bytes += 3;
192        continue;
193      }
```

```
194
195         if ((  // planes 1-3
196                 bytes[0] == 0xF0 && (0x90 <= bytes[1] && bytes[1] <= 0xBF) &&
197                 (0x80 <= bytes[2] && bytes[2] <= 0xBF) &&
198                 (0x80 <= bytes[3] && bytes[3] <= 0xBF)) ||
199             (  // planes 4-15
200                 (0xF1 <= bytes[0] && bytes[0] <= 0xF3) &&
201                 (0x80 <= bytes[1] && bytes[1] <= 0xBF) &&
202                 (0x80 <= bytes[2] && bytes[2] <= 0xBF) &&
203                 (0x80 <= bytes[3] && bytes[3] <= 0xBF)) ||
204             (  // plane 16
205                 bytes[0] == 0xF4 && (0x80 <= bytes[1] && bytes[1] <= 0x8F) &&
206                 (0x80 <= bytes[2] && bytes[2] <= 0xBF) &&
207                 (0x80 <= bytes[3] && bytes[3] <= 0xBF))) {
208           warn("planes\n");
209           index += 4;
210           bytes += 4;
211           continue;
212         }
213         std::cout << "bytes[0] = " << std::hex << (int)bytes[0] << std::dec
214                   << std::endl;
215         return false;
216       }
217       return true;
218     }
219   };
```

Παράρτημα Β

Κώδικας της κλάσης Validator – validate.hpp

```
class Validator {
  public:
    // constructor
    Validator() {}

  private:
    Context c;  // the validation context

  public:
    const Context& context() const { return c; }
    void InitContext(Module&);
    void PrintContext();
    void UpdateContext(vec<type::Value>&, vec<type::Value>&);
    void UpdateContext(vec<type::Value>&, vec<type::Value>&, type::Result);
    void AddLabel(type::Result);
    void RemoveLabel(type::Result);

  private:
    vec<valtype> opds;  // the operand stack
    vec<frame> ctrls;   // the control stack

  public:
    // a helper function to print the operand and control stacks
    void PrintStacks();
    // functions to access the stacks
    void push_opd(valtype);
    valtype pop_opd();
    valtype pop_opd(valtype);

    void push_opds(vec<valtype>);
    void pop_opds(vec<valtype>);

    void push_ctrl(vec<valtype>, vec<valtype>);
    vec<valtype> pop_ctrl();
    void unreachable();

    long unsigned int ctrls_size();
    frame n_frame(int n);

  public:
```

```
41    bool validate_module(Module&);  // validates a module
42
43  private:
44    bool expr(Expr&);
45    bool limits(type::Limits&, u32);
46    bool func(Func&);
47    bool funcs(Module&);
48    bool tables(Module&);
49    bool mems(Module&);
50    bool globals(Module&);
51    bool elems(Module&);
52    bool datas(Module&);
53    bool start(Module&);
54    bool exports(Module&);
55    bool imports(Module&);
56  };
```

Παράρτημα C

Κώδικας του test script – test.sh

```
#! /bin/bash

print_usage() {
    echo "Usage: $0 [OPTIONS]"
    echo "  OPTIONS:"
    echo "      -h | --help                print this message"
    echo "      -r | --run  [cpp | spec]   select what tests you want to run"
    echo "                                 by default all tests are run"
    echo "      -f | --fail                run tests that should fail. This will"
    echo "                                 not happen by default. Should be used"
    echo "                                 when spec tests are enabled."
    echo "      -v | --verbose             enable verbose output. Use this option"
    echo "                                 to get failed tests list"
    echo ""
}

# gets the wast files from the web and stores them in the provided directory
# usage: getTheWasts <target-dir> <commit-id>?
getTheWasts() {
    if [ "$#" -eq 0 ]; then
        echo "no target dir passed"
        exit 1
    fi

    target="${1}"
    if [ ! -d $target ]; then
        echo "error: ${target} no such directory"
    fi

    # get the commit id
    printf "commit:"
    if [ "$#" -eq 1 ]; then
        echo "latest"
        commit="/master"
    else
        commit="/${2}"
        # curl returns a lot of stuff but we only care the image that
        # displays the 404 message
        stringToFind="<img alt=\"404"
        ret=$(curl -s "https://github.com/WebAssembly/spec/tree${commit}/test/core" \
```

```
        | egrep -o "${stringToFind}")
        if [ "${ret}" == "${stringToFind}" ]; then
            echo "Invalid commit id"
            exit 1
        fi
        echo "${commit}"
    fi

    # create a temporary directory and navigate to it
    work_dir=$(mktemp -d)
    if [ ! -d $work_dir ]; then
        echo "failed to create tmp dir"
        exit 1
    fi
    cd $work_dir

    # get all Wast filenames
    stringToFind="/WebAssembly/spec/blob$commit"
    if [ commit!="/master" ]; then
        stringToFind=$stringToFind"[0-9abcdef]*"
    fi
    stringToFind=$stringToFind"/test/core/[^ >]*wast"

    curl -s https://github.com/WebAssembly/spec/tree${commit}/test/core |
        egrep -o "${stringToFind}" |
        sed 's/blob\///g' |
        sed -e 's/^/raw.githubusercontent.com/g' |
        xargs wget -q --show-progress

    echo $(ls -1 | grep .wast | wc -l) files downloaded

    # clean old wast files
    rm $target/*.wast >/dev/null 2>&1

    # move the wast files to the target dir
    mv *.wast $target/

    rmdir $work_dir
}

RUN_CPP=1
RUN_SPEC=1
RUN_FAIL=0
VERBOSE=0

# check options
while [[ $# -gt 0 ]]; do
    option=$1
    case "$option" in
    "-h" | "--help")
        print_usage
```

```
92          exit 0
93          ;;
94      "-r" | "--run")
95          test $2 = cpp && RUN_SPEC=0
96          test $2 = spec && RUN_CPP=0
97          shift
98          shift
99          ;;
100     "-v" | "--verbose")
101         VERBOSE=1
102         shift
103         ;;
104     "-f" | "--fail")
105         RUN_FAIL=1
106         shift
107         ;;
108     *)
109         echo "Unknown option: $1"
110         print_usage
111         exit 1
112         ;;
113     esac
114 done
115
116 ROOT_DIR=$PWD
117 TEST_DIR=$ROOT_DIR/tests
118 SRCS_DIR=$TEST_DIR/srcs
119 CPP_DIR=$SRCS_DIR/cpp_testfiles
120 CORE_TEST_DIR=$SRCS_DIR/core-tests
121 WASM_DIR=$TEST_DIR/bins
122
123 # colors for output
124 RED='\033[31;1m'
125 GREEN='\033[32;1m'
126 YELLOW='\033[33;1m'
127 NC='\033[0m' # No Color
128
129 # check if bins dir is present
130 if [ ! -d "$WASM_DIR" ]; then
131     echo "MKDIR bins"
132     mkdir $WASM_DIR
133 fi
134
135 # check if bins/fail dir is present
136 if [ ! -d "$WASM_DIR/fail" ]; then
137     echo "MKDIR bins/fail"
138     mkdir "$WASM_DIR/fail"
139 fi
140
141 if [ $RUN_CPP -eq 1 ]; then
142     # check for emscripten compiler
```

```
    em++ -v &>/dev/null
    EXITCODE=$?
    if [ $EXITCODE -eq 0 ]; then
        echo -e "${GREEN}emcc is available${NC}"
    else
        echo -e "${RED}emcc is missing${NC}, make sure to source the emsdk_env.sh"
        exit $EXITCODE
    fi

    # cleaning up older wasm files
    rm $WASM_DIR/*_module_*.wasm &>/dev/null
    rm $WASM_DIR/fail/*.wasm &>/dev/null

    # compile all cpp files to wasm
    echo -e "▫ ${YELLOW}Generating cpp (emscripten) tests${NC}"
    cd $CPP_DIR
    # WASM_DIR=$WASM_DIR make clean &>/dev/null
    WASM_DIR=$WASM_DIR make
    cd $ROOT_DIR
fi
if [ $RUN_SPEC -eq 1 ]; then
    printf "${YELLOW}Do you want to refresh the binaries?[y/N](default: N):${NC}"
    read -r ans

    if [ "${ans}" == "y" ]; then
        printf "absolute path to SPEC WASM interpreter: " && read WASMI

        commit=
        # read the commit in wasmCommit file if it exists
        if [ -f $TEST_DIR/wasmCommit ]; then
            commit=$(head -n 1 $TEST_DIR/wasmCommit)
        fi
        if [ "${commit}" == "" ]; then
            commit=

        fi

        echo "commit is ${commit}"
        # create a temporary dir to store the wasts
        WAST_DIR=$(mktemp -d)
        if [ ! -d $WAST_DIR ]; then
            echo "failed to create tmp dir"
            exit 1
        fi

        # Step 1: get the wast files from the web
        echo -e "▫ ${YELLOW}Getting the .wast files${NC}"
        getTheWasts $WAST_DIR $commit

        # Step 2: use the interpreter to convert them to .bin.wast
        echo -e "▫ ${YELLOW}Generating .bin.wast files${NC}"
```

```
194
195         # cleaning up older wasm files
196         if [ $RUN_CPP -eq 0 ]; then
197             rm $WASM_DIR/*.wasm &>/dev/null
198         else
199             rm $WASM_DIR/*_module_*.wasm &>/dev/null
200         fi
201         rm $WASM_DIR/fail/*.wasm &>/dev/null
202
203         cd $CORE_TEST_DIR
204         WASMI=$WASMI SPEC_CORE_DIR=$WAST_DIR make
205         cd $ROOT_DIR
206
207         # Step 3: get the hex code from each .bin.wast file
208         echo -e "␣ ${YELLOW}Generating spec core tests${NC} $PWD"
209         ./specTests.py
210
211         # delete the WAST_DIR
212         rm $WAST_DIR/*
213         rmdir $WAST_DIR
214
215     fi
216
217     # check that all is good
218     # script that checks that all failed tests are under bins/fail
219     for file in $CORE_TEST_DIR/*; do
220         name=$(basename --suffix=.bin.wast $file)
221         a=$(grep -e 'assert_malformed' -e 'assert_invalid' $file | wc -l)
222         if [ $a -ne 0 ]; then
223             b=$(ls -1 tests/bins/fail/${name}_module_* | wc -l)
224             if [ ! $a -eq $b ]; then
225                 echo -e "\033[31;1m␣ error:\033[0m file: ${name} has not \
226                   been processed correctly"
227                 exit 1
228             fi
229         fi
230     done
231     echo -e "\033[32;1m␣\033[0m all test files were processed correctly"
232 fi
233
234 test $RUN_CPP -eq 0 -a $RUN_SPEC -eq 0 && exit 0
235
236 # run the tests
237 echo "-------------------------------------------------------"
238 echo -e "${GREEN}Running tests:${NC}"
239 echo "-------------------------------------------------------"
240
241 FAILED_TESTS=" "
242
243 col=0
244 passed=0
```

```
count=0
total=$(ls -1 $WASM_DIR/*.wasm | wc -l)

tempfile=$(mktemp)
for prog in $WASM_DIR/*.wasm; do
    ((col++))
    test $col -eq 51 && printf " (${count} / ${total})" && echo "" && col=1

    ((count++))

    ./cwasm $prog &>tempfile
    EXITCODE=$?
    if [ $EXITCODE -eq 0 ]; then
        ((passed++))
        printf '.'
    else
        printf 'X'
        NAME=$(basename $prog)
        FAILED_TESTS="${FAILED_TESTS}\n ${NAME}"
        echo
        cat tempfile
        break
    fi
done
rm $tempfile
echo

col2=0
passed2=0
count2=0
total2=0
if [ $RUN_FAIL -eq 1 ] && [ $(ls -1 $WASM_DIR/fail | wc -l) -ne 0 ]; then
    # run the tests that should fail
    echo "-------------------------------------------------------"
    echo -e "${GREEN}Running tests that should fail:${NC}"
    echo "-------------------------------------------------------"

    total2=$(ls -1 $WASM_DIR/fail/*.wasm | wc -l)
    for prog in $WASM_DIR/fail/*.wasm; do
        ((col2++))
        test $col2 -eq 51 && printf " (${count2} / ${total2})" && echo && col2=1

        ((count2++))

        timeout 2 ./cwasm $prog &>/dev/null
        EXITCODE=$?
        if [ $EXITCODE -ne 0 ]; then
            ((passed2++))
            printf '.'
        else
            printf 'X'
```

```
296                 NAME=$(basename $prog)
297                 FAILED_TESTS="${FAILED_TESTS}\n ${NAME}"
298                 # echo
299                 # cat tempfile
300                 # TODO?: This would be nice in the future
301                 # msg=$(cat $WASM_DIR/fail/msg/$prog)
302                 # echo "test should fail: $msg"
303                 # break
304             fi
305         done
306         echo
307     fi
308
309     # Print results
310     all_passed=$((passed + passed2))
311     all_total=$((total + total2))
312
313     if [ $all_passed -ne $all_total ] && [ $VERBOSE -eq 1 ]; then
314         printf "show the failed tests?[y/N](default: N) " && read ans
315         if [ -n $ans ]; then
316             if [ "${ans}" == "y" ]; then
317                 echo -e $FAILED_TESTS
318             fi
319         fi
320     fi
321
322     echo
323     printf "\t %4d /%4d correct tests passed  \n" $passed $total
324     printf "\t %4d /%4d incorrect tests passed\n" $passed2 $total2
325     echo
326     echo -e "${GREEN}\u2714 ${all_passed}/${all_total} tests passed!${NC}"
327     echo
```