ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Μεταγλωττιστές

8ο εξάμηνο, Ακαδημαϊκή περίοδος 2012

Υλοποίηση της γλώσσας **Llama**

Κωνσταντίνος Αθανασίου 03108132

Νίκος Γιανναράκης 03108054

Ζωή Παρασκευοπούλου 03108152

12/11/2013

Περιεχόμενα

1 Εισαγωγή

Η Llama είναι μια γλώσσα η οποία συνδυάζει στοιχεία συναρτησιακού και προστακτικού προγραμματισμού και συντακτικά μοιάζει αρκετά με τη γλώσσα Caml. Η εργασία έχει υλοποιηθεί στη γλώσσα OCaml. Αναλυτικά οι προδιαγραφές τις γλώσσας βρίσκονται εδώ.

2 Χρήση

Για τη μεταγλώττιση του μεταγλωττιστή χρειάζονται τα παρακάτω:

- OCaml 4.0.0 ή νεότερη έκδοση
- Ocamlgraph ή μέσω OPAM
- CamlP5 ή μέσω CamlP5

Η διαδικασία μεταγλώττισης είναι η παρακάτω:

```
$ make depend
$ make
```

Εναλλακτικά μπορεί εξαχθεί native εκτελέσιμο εκτελώντας τα παρακάτω

```
$ make depend
$ make Llama.opt
```

Προαιρετικά, μπορεί να εξαχθεί το API του compiler σε αρχεία html μέσω της εντολής:

```
$ make doc
```

Για την μεταγλώττιση ενός αρχείου πηγαίου κώδικα Llama απαιτούνται τα παρακάτω βήματα:

```
$ ./Llama file.lla
```

ή

```
$ ./Llama.opt file.lla
```

Επιπλέον παρέχονται οι εξής επιλογές:
Για την εκτέλεση του μεταγλωττισμένου αρχείου $file.asm$ χρειάζονται τα παρακάτω:

- dosbox
- masm
- Η βιβλιοθήκη χρόνου εκτέλεσης llama.lib που περιέχεται στην υλοποίηση μας

Η συμβολομετάφραση μπορεί να γίνει εκτελώντας την παρακάτω εντολή στο περιβάλλον του dosbox:

```
> ml.exe file.asm llama.lib
```

οπότε και παράγεται το εκτελέσιμο αρχείο $file.exe$. Το εκτελέσιμο που παράγεται έχει μέγεθος stack $32KB$ το οποίο μπορεί να τροποποιηθεί μέσω κατάλληλου ορίσματος στον Microsoft Linker.

Option	Description
-O	Εκτέλεση βελτιστοποιήσεων
-i	Παραγωγή ενδιάμεσου κώδικα
-g	Παραγωγή γράφου ελέγχου ροής σε .dot αρχείο
-help	Προβολή διαθέσιμων επιλογών

Πίνακας 1: Command-line options

3 Υλοποίηση

Στην ενότητα αυτή θα παρουσιάσουμε συνοπτικά κάποια βασικά στοιχεία της υλοποίησης.

3.1 Συντακτικός και Γραμματικός Αναλυτής

Για τον συντακτικό αναλυτή χρησιμοποιήθηκε το εργαλείο Ocamllex και για τον γραμματικό αναλυτή το εργαλείο Ocamlyacc.

3.2 Σημασιολογική Ανάλυση

Ο σημασιολογικός αναλυτής της Llama εισάγει τις απαραίτητες πληροφορίες στο symbol table και ελέγχει κατά πόσο οι μεταβλητές του προγράμματος χρησιμοποιούνται ορθά, για παράδειγμα αν έχουν δηλωθεί πριν τη χρήση. Επιπλέον μέσω του αλγορίθμου εξαγωγής τύπων, κάνει έλεγχο τύπων και ανακατασκευάζει τους τύπους που δεν έχουν δηλωθεί από τον προγραμματιστή με κάποιο type annotation.

3.2.1 Εξαγωγή Τύπων

Η εξαγωγή τύπων βασίζεται στην παραγωγή περιορισμών κατά τη διάσχιση του ast, υπό τη μορφή εξισώσεων τύπων, και κάθε άγνωστος τύπος αναπαριστάται από μια μοναδική μεταβλητή τύπου. Στο τέλος της διάσχισης, καλείται η συνάρτηση unify, η οποία λύνοντας τους περιορισμούς αναθέτει σε κάθε μεταβλητή τύπου έναν γνωστό τύπο. Σε περίπτωση που οι περιορισμοί δεν είναι επιλύσιμοι επιστρέφεται μήνυμα λάθους, το οποίο επισημαίνει τους δύο τύπους που δεν μπορούν να ενοποιηθούν.
Στην γλώσσα Llama οι τελεστές σύγκρισης υποστηρίζονται μόνο για τους τύπους int, float και char. Προκειμένου να ελέγξουμε στατικά αυτόν τον περιορισμό δημιουργούμε έναν καινούριο τύπο ord που αναπαριστά τους τύπους που υποστηρίζουν τελεστές διάταξης. Έτσι προκύπτει ο παρακάτω κανόνας τυποποίησης:

$$\frac{\Gamma \vdash e_1 : ord \quad \Gamma \vdash e_2 : ord}{\Gamma \vdash e_1 \diamond e_2 : bool, \diamond \in \{<, >, \leq, \geq\}}$$

Μετά την δημιουργία των αντίστοιχων constraints, η συνάρτηση unify συσσωρεύει τους τύπους, που προκύπτει από τους περιορισμούς ότι πρέπει να υποστηρίζουν διάταξη, και αφού λύσει όλους τους περιορισμούς και εφαρμόσει όλες τις αντικαταστάσεις που προκύπτουν για τις μεταβλητές τύπων, ελέγχει αν οι συσσωρευμένοι τύποι είναι από τους 3 τύπους που υποστηρίζουν διάταξη και διαφορετικά επιστρέφει μήνυμα λάθους. Με ανάλογο τρόπο ελέγχεται τόσο ότι ο τύπος επιστροφής των συναρτήσεων δεν είναι

τύπος συνάρτησης όσο και ότι οι τύποι των τελουμένων των τελεστών $=, <>, ==, != $, δεν είναι τύποι συνάρτησης ή πίνακα, ενέργειες που απαγορεύονται στη Llama .
Επιπλέον για τους πίνακες, πέρα από τους περιορισμούς που αφορούν τους τύπους τον στοιχείων, υπάρχουν περιορισμοί σχετικά με τον αριθμό των διαστάσεων τους. Έτσι σε μια έκφραση $a[e_1, \ldots, e_n]$ πέρα από τον περιορισμό να είναι int ο τύπος των $e_1, \ldots, e_n$, προκύπτει ο περιορισμός οι διαστάσεις του a ναι είναι ίσες με n. Επιπλέον σε μια έκφραση $dim\ i\ a$ θα πρέπει $0 < i \leq n$ όπου n οι διαστάσεις του a. Ο έλεγχος για το αν ικανοποιούνται οι περιορισμοί αυτοί γίνεται από τη συνάρτηση unify αμέσως μετά την επίλυση των περιορισμών που προκύπτουν για τους τύπους.

3.3 Πίνακες

Οι πίνακες σε σημασιολογικό επίπεδο αναπαρίστανται ως μία μεταβλητή με τύπο, η οποία αποτελεί τον τύπο των στοιχείων που περιλαμβάνει, καθώς επίσης και τον αριθμό των διαστάσεων του. Οι διαστάσεις του πίνακα είναι γνωστές εξ αρχής, αντίθετα ο τύπος των στοιχείων του γίνεται γνωστός μέσω της εξαγωγής τύπων. Καθώς ο αριθμός των διαστάσεων του πίνακα αποτελεί μέρος του τύπου του πίνακα, αναφορά σε υπο-πίνακες (δηλαδή όχι σε συγκεκριμένα στοιχεία) δεν επιτρέπεται από το σύστημα τύπων. Οι πίνακες στον ενδιάμεσο κώδικα αναπαρίστανται μέσω της τετράδας

$$Array, a, [i_1, i_2, \ldots, i_n], ptr$$

όπου a είναι ένας πίνακας, $[i_1, i_2, \ldots, i_n]$ είναι μία λίστα κάθε στοιχείο της οποίας είναι το αποτέλεσμα μιας έκφρασης τύπου int και αντιστοιχεί στη θέση της αντίστοιχης διάστασης, και ptr είναι ένας δείκτης προς το στοιχείο του πίνακα που προσδιορίζεται. Αυτή η αναπαράσταση μας διευκολύνει στην παραγωγή τελικού κώδικα. Στο τελικό πρόγραμμα οι πίνακες αποθηκεύονται στο heap και στο stack αποθηκεύεται ένας δείκτης προς την αρχή του πίνακα. Στο heap ο πίνακας έχει την παρακάτω αναπαράσταση:

d_1	d_2	$\ldots$	d_n	$a_{00\ldots0}$	$a_{00\ldots1}$	$\ldots$	$a_{10\ldots0}$	$\ldots$	$a_{(n-1)(n-1)\ldots(n-1)}$

όπου d_i είναι το μέγεθος της i-οστής διάστασης και $a_{ij\ldots k}$ είναι τα στοιχεία του πίνακα αποθηκευμένα κατά γραμμές. Έτσι το μέγεθος κάθε διάστασης είναι προσβάσιμο σε χρόνο $\mathcal{O}(1)$. Η πρόσβαση σε ένα στοιχείο του πίνακα είναι πιο ακριβή διαδικασία και εξαρτάται από τον αριθμό των διαστάσεων του πίνακα. Για την πρόσβαση σε ένα στοιχείο χρειάζεται να υπολογιστεί δυναμικά η παρακάτω έκφραση

$$i_1 \cdot (d_2 \cdot d_3 \cdot \ldots d_n) + i_2 \cdot (d_3 \cdot \ldots \cdot d_n) + \ldots i_n$$

όπου i_k είναι το index που αντιστοιχεί σε κάθε διάσταση και d_j είναι το μήκος κάθε διάστασης.

3.4 Συναρτήσεις Υψηλής Τάξης

Στη Llama οι συναρτήσεις μπορούν να περαστούν ως παράμετροι σε μία συνάρτηση ή να ανατεθούν σε κάποια μεταβλητή, με τον περιορισμό ότι δεν μπορεί να γίνει μερική εφαρμογή και ότι μια συνάρτηση δεν μπορεί να επιστρέψει συνάρτηση ως αποτέλεσμα. Μια μεταβλητή ή μια παράμετρος συνάρτηση, αναπαριστάται στο stack ως ένα ζεύγος (closure) που αποτελείται από ένα δείκτη προς τον κώδικα της συνάρτησης και ένα δείκτη προς το περιβάλλον της τελευταίας. Το περιβάλλον της συνάρτησης περιέχει τιμές για κάθε ελεύθερη μεταβλητή που εμφανίζεται στον κώδικα της. Έτσι σε κάθε συνάρτηση που παίρνει ως όρισμα μία άλλη συνάρτηση μπορούμε να θεωρήσουμε ότι περνάμε ένα όρισμα ακόμα που είναι το περιβάλλον της συνάρτησης-παραμέτρου.
Με αυτόν το τρόπο γίνεται δυνατή η μεταγλώττιση και η εκτέλεση του παρακάτω προγράμματος

code_ptr
env_ptr

Πίνακας 2: Αναπαράσταση closure

```
let f =
    let mutable a in
        a := 0;
        let g (x : unit) =
            incr a;
            !a
        in
        g


let main =
    print_int (f ());    (* 1 *)
    print_string "\n";
    print_int (f ());    (* 2 *)
    print_string "\n";
    print_int (f ());    (* 3 *)
    print_string "\n";
    print_int (f ());    (* 4 *)
    print_string "\n"
```

3.5 Τύποι Οριζόμενοι από τον Προγραμματιστή

3.5.1 Αναπαράσταση

Στο εμπρόσθιο τμήμα του μεταγλωττιστή οι τύποι που ορίζονται από τον προγραμματιστή, και συγκεκριμένα οι κατασκευαστές αυτών, αναπαρίστανται και αντιμετωπίζονται με τρόπο παρόμοιο (αλλά όχι ίδιο) με τις συναρτήσεις. Με κάθε κατασκευαστή τύπου σχετίζεται ένας κωδικός (tag), ο οποίος μαζί με τον τύπο του κατασκευαστή προσδιορίζουν μονοσήμαντα τον κατασκευαστή, μία λίστα με τύπους που αντιστοιχούν στον τύπους των παραμέτρων του κατασκευαστή, και φυσικά ο τύπος τον οποίο κατασκευάζει.
Οι τύποι που ορίζονται από τον προγραμματιστή χρησιμοποιούνται στο μεταγλωττιστή μόνο για τη διενέργεια σημασιολογικών ελέγχων. Στην αναπαράσταση τους στη μνήμη του υπολογιστή χρησιμοποιούνται μόνο οι κατασκευαστές. Τα αντικείμενα που κατασκευάζονται από κατασκευαστές τύπων που ορίζονται από τον προγραμματιστή αποθηκεύονται στο heap. Στο stack αποθηκεύεται μόνο ένας δείκτης προς το πρώτο byte του αντικειμένου όπως αυτό έχει αποθηκευθεί στη μνήμη. Τα πρώτα 2 byte αποτελούν το tag του κατασκευαστή και στις επόμενες θέσεις αποθηκεύονται τα ορίσματα αυτού, εφόσον υπάρχουν. Σημειώνουμε ότι το όρισμα μπορεί να είναι κάποιος δείκτης προς κάποιον άλλο κατασκευαστή. Σε επίπεδο ενδιάμεσου κώδικα χρησιμοποιήσαμε μία επιπλέον τετράδα για την πρόσβαση στις παραμέτρους κάθε κατασκευαστή, παρόμοια με την τετράδα $Array$ για την πρόσβαση στα στοιχεία ενός πίνακα αλλά πιο στενά συνδεδεμένη με την αναπαράσταση των κατασκευαστών τύπων στη μνήμη. Πιο συγκεκριμένα η τετράδα

$$Constr, c, offset, ptr$$

αποθηκεύει στο δείκτη ptr τη διεύθυνση $addr(c) + offset$, όπου $addr(c)$ είναι ο δείκτης στο πρώτο byte της αναπαράστασης του c στο heap.

tag	arg_1	...	arg_n

Πίνακας 3: Αναπαράσταση κατασκευαστή στη μνήμη

3.5.2 Υλοποίηση Δομικής Ισότητας

Για την υλοποίηση τη δομικής ισότητας υλοποιούμε μια συνάρτηση για κάθε τύπο ορισμένο από τον προγραμματιστή σε επίπεδο ενδιάμεσου κώδικα. Η συνάρτηση αυτή δέχεται δύο ορίσματα και αφού ελέγξει ότι το tag τους είναι ίδιο, συνεχίζει στον ένα προς ένα έλεγχο των ορισμάτων των κατασκευαστών για ισότητα, χρησιμοποιώντας την κατάλληλη συνάρτηση, ανάλογα με τον τύπο τους. Όταν ο προγραμματιστής χρησιμοποιήσει τους τελεστές δομικής ισότητας ή ανισότητας με τελούμενα τύπου οριζόμενου από το προγραμματιστή, τότε καλείται η ανάλογη συνάρτηση με ορίσματα τα δύο τελούμενα προς σύγκριση. Σημειώνεται ότι αν ένας κατασκευαστής έχει ορίσματα τύπου πίνακα η συνάρτησης που δεν υποστηρίζουν ισότητα και ο προγραμματιστής προσπαθήσει να συγκρίνει δύο τελούμενα που προέρχονται από αυτό τον κατασκευαστή τότε αν η διαδικασία της σύγκρισης φτάσει μέχρι το σημείο που πρέπει να συγκριθούν τα παραπάνω ορίσματα θα προκληθεί σφάλμα εκτέλεσης. Αν ο έλεγχος για ισότητα αποτύχει νωρίτερα, δηλαδή κάποια ορίσματα που προηγούνται δεν είναι ίσα, τότε θα επιστραφεί false, ή true αντίστοιχα στην περίπτωση που τα τελούμενα ελέγχονται για ανισότητα.

3.5.3 Εκφράσεις match

Για την υλοποίηση των εκφράσεων match εισάγουμε την παρακάτω τετράδα:

$$Match, e, c, l$$

Η τετράδα αυτή ελέγχει αν το τελούμενο e προέρχεται από τον κατασκευαστή c και αν αυτό ισχύει, εκτελεί άλμα προς την τετράδα l. Εάν η προς έλεγχο έκφραση δεν ταιριάζει με κανένα από τα πρότυπα μιας έκφρασης match τότε προκαλείται σφάλμα εκτέλεσης και τυπώνεται ανάλογο μήνυμα. Σημειώνεται επίσης ότι οι εκφράσεις προς έλεγχο στο match μπορεί να είναι και οποιουδήποτε άλλου τύπου της γλώσσας.

3.6 Control Flow Graph

Για την υλοποίηση του γράφου ροής ελέγχου αρχικά χωρίσαμε της τετράδες του ενδιάμεσου κώδικα σε Blocks έτσι ώστε να υπάρχουν άλματα μόνο προς την πρώτη εντολή κάθε Block και οι εντολές άλματος (γενικότερα οι εντολές αλλαγής ροής του προγράμματος) να βρίσκονται μόνο στο τέλος κάποιου Block. Σε κάθε Block πέρα από τις αριθμημένες τετράδες που το αποτελούν, μέσω ενός record αποθηκεύουμε και επιπλέον πληροφορίες που διευκολύνουν την ανάλυση του προγράμματος σε αυτή τη μορφή, όπως για παράδειγμα το αν ορίζεται κάποια συνάρτηση μέσα σε αυτό το Block ή σε ποια συνάρτηση ανήκουν οι τετράδες του κάθε Block και αν κάποιο Block περιλαμβάνει το τέλος μίας συνάρτησης (Endu).
Κάθε κόμβος του γράφου ροής ελέγχου είναι ένα Block με ένα παραπάνω στοιχείο, συγκεκριμένα ένα set που περιλαμβάνει τον αριθμό των τετράδων που περιέχονται στο Block. Στην πραγματικότητα θα έπρεπε και αυτή η πληροφορία να περιλαμβάνεται στο παραπάνω record. Με βάση τις πληροφορίες που έχουμε

κρατήσει σε κάθε κόμβο η κατασκευή του γράφου ροής ελέγχου είναι σχετικά απλή, αρκεί να τηρήσουμε τους παρακάτω κανόνες.

- Αν στο τέλος κάποιου Block υπάρχουν εντολές διακλάδωσης (if, match, jump) τότε προσθέτουμε μια ακμή προς το Block που περιέχει την εντολή προς την οποία γίνεται το άλμα. Το Block αυτό μπορεί να βρεθεί σε χρόνο $\mathcal{O}(m \cdot \log n)$ όπου m ο αριθμός των Blocks και n ο αριθμός των τετράδων σε κάθε Block.

- Αν στο τέλος κάποιου Block υπάρχει κλήση συνάρτησης τότε προσθέτουμε μία ακμή προς το Block που περιέχει τη δήλωση της συνάρτησης την οποία καλούμε και μία ακμή από το Block που περιέχει το τέλος της συνάρτησης που καλέσαμε προς το Block που περιέχει την αμέσως επόμενη εντολή από την κλήση της συνάρτησης. Η εύρεση των Block που περιλαμβάνουν τη δήλωση της συνάρτησης που καλείται και το τέλος αυτής αντίστοιχα γίνεται σε χρόνο $\mathcal{O}(m)$ όπου m ο αριθμός των Blocks και η εύρεση του Block με την επόμενη εντολή σε χρόνο $\mathcal{O}(m \cdot \log n)$ όπου m ο αριθμός των Blocks και n ο αριθμός των τετράδων σε κάθε Block.

- Αν δεν υπάρχει κάποια εντολή αλλαγής ροής όπως οι παραπάνω, τότε αρκεί να προσθέσουμε μία ακμή στο Block που περιλαμβάνει την επόμενη εντολή που μπορεί να βρεθεί σε χρόνο $\mathcal{O}(m \cdot \log n)$, όπου m ο αριθμός των Blocks και n ο αριθμός των τετράδων σε κάθε Block.

Θα πρέπει να σημειωθεί βέβαια ότι η παραπάνω διαδικασία ίσως να είναι πιο πολύπλοκη από ότι δείχνει, ανάλογα με τη γλώσσα την οποία στοχεύει. Για παράδειγμα στη Llama επιτρέπεται το πέρασμα μίας συνάρτησης ως παραμέτρος, πράγμα που δυσκολεύει την κατασκευή ενός γράφου ροής ελέγχου στην περίπτωση που έχουμε κλήση αυτής της συνάρτησης-παραμέτρου. Μπορούν να γίνουν διαφόρων ειδών αναλύσεις ώστε να περιοριστεί το εύρος των τιμών που μπορεί να έχει μία τέτοια μεταβλητή/παράμετρος. Ωστόσο στην υλοποίηση μας διαλέξαμε την τετριμμένη λύση, κάθε τέτοια κλήση μπορεί να είναι προς οποιοδήποτε block που περιέχει μία συνάρτηση. Για την δημιουργία του γράφου χρησιμοποιήθηκε η βιβλιοθήκη Ocamlgraph. Αν θέλει κάποιος να εξετάσει τον γράφο ροής ελέγχου που παράγει ο Alpaca μπορεί να το κάνει δίνοντας το όρισμα -g. Αυτό θα εξάγει το γράφο ροής ελέγχου του δοθέντος αρχείου πηγαίου κώδικα σε μορφή .dot που μπορεί να μετατραπεί σε εικόνα με κάποιο λογισμικό όπως το Graphviz.

3.7 Βελτιστοποιήσεις

Στον Alpaca υπάρχουν δύο στάδια βελτιστοποιήσεων, ένα στην ενδιάμεση μορφή τετράδων και ένα στον τελικό κώδικα. Ωστόσο όλες οι βελτιστοποιήσεις είναι τοπικές καθώς δεν έχει γίνει ανάλυση ροής δεδομένων.

3.7.1 Απλοποίηση υπολογισμών στις τετράδες

Αν και η απλοποίηση αυτή θα μπορούσε να είχε γίνει στο στάδιο παραγωγής ενδιάμεσου κώδικα, θα έκανε το στάδιο αυτό πιο πολύπλοκο και προτιμήθηκε να υλοποιηθεί σε κάποιο επόμενο στάδιο. Σκοπός είναι να μη χρησιμοποιούνται προσωρινές μεταβλητές για την εκτέλεση των διαφόρων πράξεων αλλά να γίνονται απευθείας πάνω στις μεταβλητές που πρέπει.

3.7.2 Βελτιστοποιήσεις βασισμένες στο γράφο ροής ελέγχου

Οι βελτιστοποιήσεις που βασίζονται στον γράφο ροής ελέγχου γίνονται κατά κύριο λόγο σε κάθε block ξεχωριστά.

Generated	Optimized
+a,b,$1	+a,b,c
:=,$1,-,c	

Πίνακας 4: Temp Reduction

Απαλοιφή κοινών υποεκφράσεων Για την απαλοιφή κοινών υποεκφράσεων σε κάθε block χρησιμοποιείται η μέθοδος *Value Numbering*. Κάθε έκφραση που υπολογίζεται αποθηκεύεται σε μία καινούργια προσωρινή μεταβλητή, έτσι σε περίπτωση που η τιμή αυτής της έκφρασης χρειαστεί σε κάποιον παρακάτω υπολογισμό - μέσα στο ίδιο Block - τότε χρησιμοποιείται η προσωρινή μεταβλητή που είχε δημιουργηθεί προηγουμένως. Για την αντιμετώπιση των πολλών νέων προσωρινών μεταβλητών και των μη απαραίτητων εντολών ανάθεσης χρησιμοποιούνται οι δύο παρακάτω βελτιστοποιήσεις.

Διάδοση αντιγράφων Όπου είναι εφικτό, χρησιμοποιείται η αρχική μεταβλητή για την οποία υπολογίστηκε μία έκφραση και όχι η προσωρινή μεταβλητή που δημιουργήθηκε παραπάνω. Με αυτό τον τρόπο δίνεται η δυνατότητα να πραγματοποιήσουμε απαλοιφή νεκρού κώδικα σε τοπικό επίπεδο.

Απαλοιφή νεκρού κώδικα Όσες προσωρινές μεταβλητές δημιουργήθηκαν παραπάνω αλλά δεν χρησιμοποιούνται είτε επειδή αντικαταστάθηκαν κατά τη διάδοση αντιγράφων είτε επειδή δε χρειάστηκε, μπορούν να σβηστούν. Αυτό γίνεται με ένα πέρασμα κάθε Block ανάποδα, δηλαδή ξεκινώντας από την τελευταία εντολή προς την πρώτη, σημειώνοντας ποιες προσωρινές μεταβλητές χρειάζονται και σβήνοντας όσες κατά τη δήλωση τους δεν έχουν σημειωθεί ως χρήσιμες.

Απαλοιφή Απροσπέλαστου Κώδικα Από τις παραπάνω βελτιστοποιήσεις (και όχι μόνο) πιθανώς να προκύψουν σημεία του κώδικα που είναι απροσπέλαστα με βάση τη ροή ελέγχου. Ξεκινώντας από το αρχικό Block και εκτελώντας μία αναζήτηση κατά βάθος (DFS) πάνω στο γράφο ροής ελέγχου μπορούμε να συμπεράνουμε ποια blocks είναι απροσπέλαστα και να τα αφαιρέσουμε.

3.7.3 Tail Recursion Elimination

Στον Alpaca δεν βελτιστοποιούνται όλες οι κλήσεις ουράς αλλά μόνο οι κλήσεις ουράς προς την ίδια συνάρτηση, δηλαδή οι αναδρομικές. Η υλοποίηση της απαλοιφής κλήσης ουράς περιλαμβάνει 2 στάδια:

Εντοπισμός κλήσεων ουράς Ο εντοπισμός των σημείων κλήσης ουράς έγινε πάνω στο AST με βάση τους παρακάτω κανόνες:

$$\tau(let_{fun} = e_1, \_) \rightarrow e_1.tail \leftarrow true; \tau(e_1, true)$$

$$\tau(e_1; e_2, true) \rightarrow \tau(e_1, false); e_2.tail \leftarrow true; \tau(e_2, true)$$

$$\tau(begin; e_1; end, true) \rightarrow e_1.tail \leftarrow true; \tau(e_1, true)$$

$$\tau(if\ e_1\ then\ e_2\ else\ e_3, true) \rightarrow \tau(e_1, false); e_2.tail \leftarrow true; \tau(e_2, true); e_3.tail \leftarrow true; \tau(e_3, true)$$

$$\tau(if\ e_1\ then\ e_2, true) \rightarrow \tau(e_1, false); e_2.tail \leftarrow true; \tau(e_2, true)$$

$$\tau(match\ e_1\ with\ p_i \rightarrow e_i, true) \rightarrow \tau(e_1, false); e_i.tail \leftarrow true; \tau(e_i, true)$$

$$\tau(let\ x\ =\ e_1\ in\ e_2, true) \rightarrow \tau(e_1, false); e_2.tail \leftarrow true; \tau(e_2, true)$$

Η διαδικασία τ υποθέτει ότι το πεδίο $tail$ κάθε έκφρασης είναι $false$. Στη συνέχεια θέτει την τιμή $true$ σε κάθε πεδίο $tail$ με βάση τους παραπάνω κανόνες.

Υλοποίηση κλήση ουράς Η ιδέα είναι ότι μπορούμε να χρησιμοποιήσουμε το υπάρχον εγγράφημα δραστηριοποίησης, αποφεύγοντας το κόστος δημιουργίας (κυρίως σε μνήμη) ενός καινούργιου. Για να γίνει κλήση ουράς αντί για κανονική κλήση αρκεί στις κλήσεις συνάρτησης που έχουν σημειωθεί ως $tail$ και επίσης αναφέρονται στην ίδια συνάρτηση με αυτή που εκτελείται, να αντικαταστήσουμε τις τιμές των παραμέτρων με τις νέες τιμές μέσω ανάθεσης και να κάνουμε $jump$ στην αρχή της συνάρτησης. Η διαδικασία αυτή υλοποιείται στον ενδιάμεσο κώδικα τετράδων και στη συνέχεια μεταφράζεται σε τελικό κώδικα χωρίς κάποια αλλαγή στην παραγωγή τελικού κώδικα.

3.7.4 Βελτιστοποιήσεις τελικού κώδικα

Peephole Οι βελτιστοποιήσεις που έγιναν στον τελικό κώδικα ακολουθούν τη λογική των λεγόμενων Peephole optimizations. Γίνονται πολλαπλά περάσματα στον τελικό κώδικα ψάχνοντας για γνωστά μοτίβα εντολών (1 έως και 3 εντολές) τα οποία στη συνέχεια μετατρέπονται σε πιο απλές και γρήγορες εντολές. Εκμεταλλευόμενοι το ισχυρό pattern-matching της OCaml η υλοποίηση αυτών των βελτιστοποιήσεων είναι αρκετά εύκολη σε σχέση με αυτές του γράφου ροής ελέγχου. Είναι ωστόσο και αρκετά αποτελεσματικές, δεδομένου του απλοϊκού αλγορίθμου παραγωγής τελικού κώδικα που ακολουθήσαμε. Στον παρακάτω πίνακα παρουσιάζουμε πολλές από τις βελτιστοποιήσεις που υλοποιήσαμε. Επειδή γίνονται παραπάνω από ένα περάσματα απλοποιούνται και άλλοι συνδυασμοί αυτών.

Generated	Optimized
add target, 1	inc target
sub target, 1	dec target
add/sub target, 0	-
mov r1, 0 sub r1, val	mov r1,val neg r1
mov si, bp push si	push bp
mov r1, imm mov size ptr [r2], r1	mov size ptr [r2], imm
mov mem1, r1 mov r2, mem1	mov mem1, r1 mov r2, r1
mov mem1, r1 mov r1, mem1	mov mem1, r1
mov r1, val op r2, r1	op r2, val
jcond target1 jmp target2	jrevcond target2
mov r1, imm mov r2, size ptr [r3] mov size ptr [r2], r1	mov r2, size ptr [r3] mov size ptr [r2], imm

Πίνακας 5: Peephole Optimizations