Μεταγλωττιστές 2025-26

Ενδιάμεσες αναπαραστάσεις (IR): Κώδικας τριών διευθύνσεων – Three-address code (TAC)

(παράδειγμα για τη γλώσσα των αριθμητικών εκφράσεων)

Μορφές ενδιάμεσων αναπαραστάσεων

- Οι μεταγλωττιστές χρησιμοποιούν διάφορα είδη ενδιάμεσων αναπαραστάσεων (IR) – συχνά περισσότερα από ένα ταυτόχρονα
  - Δένδρα/γράφοι: έχουμε ήδη δει τα Abstract Syntax Trees (ASTs)
  - «Επίπεδες μορφές»: λίστες απλών εντολών, μοιάζουν με εντολές assembly
    - Σε διάφορα επίπεδα, υψηλότερα (προς την αρχική γλώσσα) ή χαμηλότερα (προς τον επεξεργαστή-στόχο)
  - Υβριδικές μορφές: Μορφή γράφου, όπου οι κόμβοι περιέχουν λίστες εντολών

Three-address code (TAC)

- Μια «επίπεδη» μορφή ενδιάμεσης αναπαράστασης
  - Λίστα εντολών της μορφής:

    result ← source1 operator source2

    - Τα result, source1, source2 ουσιαστικά είναι διευθύνσεις (addresses) πηγών/προορισμού
- Είναι μια «σειριοποίηση» ενός AST (tree lowering)
  - Πιο κοντά στα χαρακτηριστικά ενός επεξεργαστή
    - Μοιάζει με το μοντέλο RISC (σε θεωρητικό επίπεδο)
  - Μπορεί να αποτελέσει τη βάση για περαιτέρω βελτιστοποιήσεις ή παραγωγή κώδικα μηχανής
  - Είναι ο «πρόγονος» των πιο εξελιγμένων μορφών IR που χρησιμοποιούν οι σύγχρονοι μεταγλωττιστές

Three-address code (TAC) (2)

- Τι άλλο περιέχει
  - Σε κάποιες περιπτώσεις μια εντολή περιέχει δύο μόνο διευθύνσεις:

    result ← source1

    result ← operator source1

    - Για μονομελείς τελεστές (όπως το NOT) ή όταν μια τιμή μεταφέρεται (αντιγράφεται) από την πηγή στον προορισμό
  - Εντολές διακλάδωσης (με ή χωρίς συνθήκη)

    goto L1

    ifzero source1 goto L2

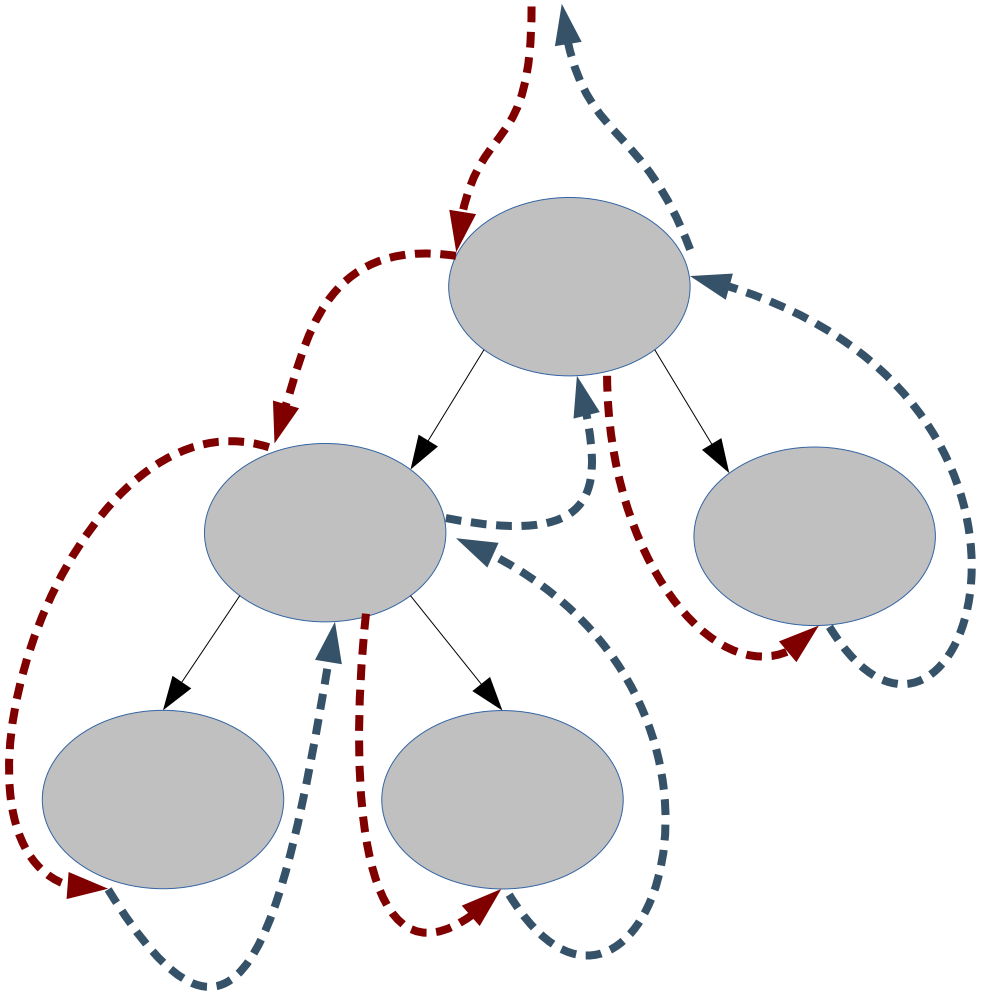
    - L1, L2 είναι ετικέτες (labels), διευθύνσεις δηλαδή σε διαφορετικά σημεία του κώδικα

Πηγές και προορισμός

- Στις θέσεις των πηγών και του προορισμού μπορούν να εμφανίζονται τα εξής
  - Ονόματα μεταβλητών του αρχικού προγράμματος
    - Αντιστοιχούν σε πληροφορία θέσης της μεταβλητής κατά την εκτέλεση ή διερμηνεία
  - Σταθερές (αριθμητικές ή άλλες) ως πηγές
  - Προσωρινές μεταβλητές που κρατούν τα ενδιάμεσα αποτελέσματα προηγούμενων πράξεων
    - Ως καταχωρητές ενός επεξεργαστή, αλλά με άπειρο αριθμό
    - Τα διάφορα στάδια βελτιστοποίησης διευκολύνονται όταν κάθε ανάθεση αποτελέσματος (←) γίνεται σε διαφορετική προσωρινή μεταβλητή
    - Στη μετατροπή σε πραγματικό κώδικα μηχανής, ο άπειρος αυτός αριθμός θα αντιστοιχιστεί στους πραγματικούς καταχωρητές του επεξεργαστή

Δημιουργία TAC από το δένδρο AST

- Διάσχιση δένδρου με DFS και δημιουργία TAC σε κάθε κόμβο
  - Χρήση τιμών από «πατέρα» κόμβο (ορίσματα συνάρτησεων) – *inherited attributes*
  - Υπολογισμός νέων τιμών με συνδυασμό επιστρεφόμενων τιμών από τους κόμβους «παιδιά» - *synthesized attributes*
- Στην πραγματικότητα η δημιουργία TAC μπορεί να γίνει κατά τη συντακτική ανάλυση
  - Τυπικός ορισμός απ' όπου δανειζόμαστε τους όρους *inherited/synthesized attributes*



TAC για τη γλώσσα των αριθμητικών εκφράσεων

- Θα πρέπει να υποστηρίζονται οι λειτουργίες και οι τύποι δεδομένων της γλώσσας
  - Ένας και μοναδικός τύπος μεταβλητών/σταθερών (float)
  - Οι βασικές αριθμητικές πράξεις (+ - * /)
  - Αριθμητικές συγκρίσεις (< <= > >= == !=)
  - Λογικοί τελεστές (and, or, not)
  - Απλές (scalar) μεταβλητές
    - Όχι pointers arrays, structures, objects, κ.ο.κ
  - Συναρτήσεις προσδιορισμένες κατά τη μεταγλώττιση
    - Όχι δυναμικές κλήσεις (ορισμός κατά την εκτέλεση)
    - Θα προστεθούν αργότερα
  - Οι βασικές δομές control-flow (if, if-else, while)

Κώδικας για υπολογισμό εκφράσεων

- Το «πρόβλημα»:
  - Για να κρατήσουμε τη γραμματική LL(1), έχουμε ενοποιήσει τις αριθμητικές και τις λογικές εκφράσεις
  - Υπάρχουν τώρα δύο περιπτώσεις χρήσης μιας έκφρασης
    - Control-flow: για έλεγχο ροής σε if, if-else, while

      ```
      if a>2 …
      ```

    - Expression value: για υπολογισμό τιμής μέσα σε μια έκφραση

      ```
      b = (a>2)+8
      ```

- Πώς αντιμετωπίζεται;
  - Θα χρησιμοποιήσουμε διαφορετική συνάρτηση για κάθε περίπτωση

Κώδικας για υπολογισμό εκφράσεων (2)

- Εκφράσεις για υπολογισμό τιμής
  - GenTACev(astnode,dest)
    - astnode: η ρίζα του AST για την έκφραση
    - dest: το όνομα του προορισμού (για τοποθέτηση του αποτελέσματος)
- Εκφράσεις για έλεγχο ροής
  - GenTACcf(astnode,ltrue,lfalse)
    - astnode: η ρίζα του AST για την έκφραση
    - ltrue, lfalse: οι ετικέτες όπου πρέπει να μεταφεθεί η ροή του προγράμματος εάν ο υπολογισμός του astnode δίνει αληθές ή ψευδές αποτέλεσμα αντίστοιχα
      - Οι ετικέτες θα έχουν κειμενική μορφή (ονόματα)

Α. Υπολογισμός τιμής έκφρασης

- Αριθμητικές σταθερές/μεταβλητές

```
dest = value/varname
```

- Αριθμητικές πράξεις

```
GenTACev(leftnode,t1)
GenTACev(rightnode,t2)
dest = t1 aop t2
```

- Συγκρίσεις

```
GenTACev(leftnode,t1)
GenTACev(rightnode,t2)
if t1 rop t2 goto L1
dest = 0.0
goto L2
L1: dest = 1.0
L2:
```

Α. Υπολογισμός τιμής έκφρασης

- Ο τελεστής and (python-style semantics)

```
GenTACev(leftnode,t1)
ifzero t1 goto L1
GenTACev(rightnode,t2)
ifzero t2 goto L1
dest = t2
goto L2
L1: dest = 0.0
L2:
```

  - Πώς θα ήταν με C-style semantics;

- Ποιος θα είναι ο αντίστοιχος κώδικας για το or;

Β. Έλεγχος ροής

- Αριθμητικές σταθερές
  - Εδώ γνωρίζουμε ακριβώς αν η σταθερά είναι true (≠0) ή όχι

    `goto Ltrue` ή `goto Lfalse` αντίστοιχα
- Μεταβλητές

  ```
  ifzero varname goto Lfalse
  goto Ltrue
  ```
- Αριθμητικές πράξεις
  - Όπως πριν, με έλεγχο του αποτελέσματος (ifzero…)
- Συγκρίσεις
  - Όπως πριν, με έλεγχο του αποτελέσματος (if t1 rop t2…)

Β. Έλεγχος ροής

- Ο τελεστής and

```
GenTACcf(leftnode,L1,Lfalse)
L1:
GenTACcf(rightnode,Ltrue,Lfalse)
```

- Ο τελεστής or

```
GenTACcf(leftnode,Ltrue,L1)
L1:
GenTACcf(rightnode,Ltrue,Lfalse)
```

- Ποιος θα είναι ο αντίστοιχος κώδικας για το not;

Κώδικας εντολών (statements)

- Συνάρτηση παραγωγής κώδικα εντολών

```
GenTACstmts(ast_list)
```

- id = expr

```
GenTACev(expnode,varname)
```

- print expr

```
GenTACev(expnode,t1)
call @print(t1)
```

- if expr block-statement

```
GenTACcf(leftnode,Ltrue,Lfalse)
Ltrue:
GenTACstmts(bs_ast_list)
Lfalse:
```

  - Ποιος θα είναι ο αντίστοιχος κώδικας για το while και το if-else;