Κεφάλαιο 10 : Συναρτησιακές Γλώσσες

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Γ. Γαροφαλάκης, Σ. Σιούτας

Εισαγωγή

- Οι διαφορές των ΓΠ στις *συντακτικές δομές* τους, είναι πολύ μεγαλύτερες από τις διαφορές τους στις εννοιολογικές δομές. Π.χ. το στοιχείο του πίνακα Α στη θέση 1, γράφεται:

  A(1) στις FORTRAN, COBOL, PL/1, Ada

  A[1] στις Pascal, C

  A<1> στη SNOBOL

- Στόχος συντακτικού:

  Κανόνες επικοινωνίας της πληροφορίας μεταξύ προγραμματιστή και μεταφραστή/διερμηνέα.

Ιστορικά Στοιχεία

- Τα μοντέλα του προστακτικού και του συναρτησιακού προγραμματισμού προέκυψαν από το έργο των Alan Turing, Alonzo Church, Stephen Kleene, Emil Post, κλπ. ~δεκαετία 1930
  - διαφορετικοί φορμαλισμοί για την έννοια του αλγορίθμου ή της *αποτελεσματικής διαδικασίας,* βασισμένοι στα αυτόματα, τη συμβολική επεξεργασία, τους αναδρομικούς ορισμούς συναρτήσεων, και τη συνδυαστική
- Τα αποτελέσματα αυτά οδήγησαν τον Church στην εικασία ότι *οποιοδήποτε* διαισθητικά αποδεκτό μοντέλο υπολογισμών θα είναι και αυτό εξίσου ισχυρό
  - η εικασία αυτή είναι γνωστή *ως θέση του Church*

Ιστορικά Στοιχεία

- Το μοντέλο υπολογισμών του Turing ήταν η *μηχανή Turing*, ένα είδος αυτόματου στοίβας που χρησιμοποιούσε μια «ταινία» με απεριόριστο πλήθος αποθηκευτικών θέσεων
  - η μηχανή Turing κάνει υπολογισμούς με προστακτικό τρόπο, αλλάζοντας τις τιμές θέσεων στην ταινία της – όπως τα προστακτικά προγράμματα υψηλού επιπέδου εκτελούν υπολογισμούς αλλάζοντας τις τιμές των μεταβλητών

Ιστορικά Στοιχεία

- Το μοντέλο υπολογισμού του Church ονομάζεται *λογισμός λάμβδα*
  - βασίζεται στην έννοια των παραμετρικών εκφράσεων (κάθε παράμετρος εισάγεται με μια εμφάνιση του γράμματος λ – από όπου και το όνομα της σημειογραφίας)
  - ο λογισμός λάμβδα ήταν η έμπνευση για το συναρτησιακό προγραμματισμό
  - ο υπολογισμός γίνεται με την αντικατάσταση παραμέτρων σε εκφράσεις, ακριβώς όπως σε ένα συναρτησιακό πρόγραμμα υψηλού επιπέδου ο υπολογισμός γίνεται με τη μεταβίβαση ορισμάτων σε συναρτήσεις

Ιστορικά Στοιχεία

- Οι μαθηματικοί διατύπωσαν τη διάκριση μεταξύ
  - μιας *κατασκευαστικής* απόδειξης (που δείχνει πώς λαμβάνεται ένα μαθηματικό αντικείμενο με κάποια επιθυμητή ιδιότητα)
  - μιας *μη κατασκευαστικής* απόδειξης (που απλώς δείχνει ότι ένα τέτοιο αντικείμενο πρέπει να υπάρχει, π.χ. με απαγωγή σε άτοπο)
- Ο λογικός προγραμματισμός είναι στενά συνδεδεμένος με την έννοια της κατασκευαστικής απόδειξης, αλλά σε ένα πιο αφηρημένο επίπεδο:
  - ο προγραμματιστής στο λογικό μοντέλο προγραμματισμού, γράφει ένα σύνολο *αξιωμάτων* που επιτρέπει στον *υπολογιστή* να ανακαλύψει μια κατασκευαστική απόδειξη για κάθε συγκεκριμένο σύνολο εισόδων

Έννοιες Συναρτησιακού Προγραμματισμού

- Οι γλώσσες συναρτησιακού προγραμματισμού όπως η Lisp, η Scheme, η FP, η ML, η Miranda, και η Haskell είναι μια προσπάθεια να πραγματωθεί ο λογισμός λάμβδα του Church σε μια πρακτική μορφή σαν γλώσσα προγραμματισμού
- η βασική ιδέα: τα πάντα γίνονται με τη σύνθεση συναρτήσεων
  - δεν υπάρχει μεταβλητή κατάσταση
  - δεν υπάρχουν παρενέργειες

Έννοιες Συναρτησιακού Προγραμματισμού

- Απαραίτητες δυνατότητες, αρκετές από τις οποίες λείπουν από κάποιες προστακτικές γλώσσες
  - συναρτήσεις πρώτης κατηγορίας και υψηλότερης τάξης
  - ισχυρός πολυμορφισμός
  - ισχυρές λειτουργίες για λίστες
  - αναδρομή
  - δομημένα αποτελέσματα συναρτήσεων
  - πραγματικά γενικές σταθερές σύνθετου τύπου
  - συλλογή σκουπιδιών

Έννοιες Συναρτησιακού Προγραμματισμού

- Πώς κάνουμε κάτι σε μια συναρτησιακή γλώσσα;
  - Η αναδρομή (ειδικά η αναδρομή ουράς) αντικαθιστά την επανάληψη
  - Γενικά, μπορείτε να έχετε το ίδιο αποτέλεσμα με μια σειρά αναθέσεων

```
x := 0          ...
x := expr1      ...
x := expr2      ...
```

με την `f3(f2(f1(0)))`, όπου κάθε f περιμένει την τιμή του x σαν παράμετρο, η f1 επιστρέφει expr1, και η f2 επιστρέφει expr2

Έννοιες Συναρτησιακού Προγραμματισμού

- Η αναδρομή αντικαθιστά με επιτυχία ακόμα και τους βρόχους

```
x := 0; i := 1; j := 100;
while i < j do
    x := x + i*j; i := i + 1;
          j := j - 1
end while
return x
```

γίνεται `f(0,1,100)`, όπου

```
f(x,i,j) == if i < j then
f (x+i*j, i+1, j-1) else x
```

Έννοιες Συναρτησιακού Προγραμματισμού

- Το να θεωρούμε όμως ότι η αναδρομή είναι μια απευθείας, μηχανική αντικατάσταση της επανάληψης είναι ο λάθος τρόπος να βλέπουμε τα πράγματα
  - Πρέπει να συνηθίσουμε να σκεπτόμαστε σε αναδρομικό στυλ
- Πιο σημαντική έννοια και από την αναδρομή είναι οι ***συναρτήσεις υψηλότερης τάξης***
  - Παίρνουν μια συνάρτηση ως παράμετρο, ή επιστρέφουν μια συνάρτηση ως αποτέλεσμα
  - Χρήσιμες στην κατασκευή πραγμάτων

Έννοιες Συναρτησιακού Προγραμματισμού

- Εκδόσεις της LISP
  - Αμιγής Lisp (η πρώτη Lisp)
  - Interlisp, MacLisp, Emacs Lisp
  - Common Lisp
  - Scheme

Έννοιες Συναρτησιακού Προγραμματισμού

- Η αμιγής Lisp είναι αμιγώς συναρτησιακή· όλες οι άλλες Lisp έχουν προστακτικά χαρακτηριστικά
- Όλες οι πρώτες Lisp έχουν δυναμική εμβέλεια
  - Δεν γνωρίζουμε αν αυτό ήταν ηθελημένο ή συνέβη κατά λάθος
- Η Scheme και η Common Lisp έχουν στατικές εμβέλειες
  - Η Common Lisp επιτρέπει τη δυναμική εμβέλεια για ρητά δηλωμένες *ειδικές* συναρτήσεις
  - Η Common Lisp είναι τώρα η πρότυπη Lisp
    - Πολύ μεγάλη·και πολύπλοκη (η Ada του συναρτησιακού προγραμματισμού)

Έννοιες Συναρτησιακού Προγραμματισμού

- Η Scheme είναι μια ιδιαίτερα κομψή Lisp
- Άλλες συναρτησιακές γλώσσες
  - ML
  - Miranda
  - Haskell
  - FP
- Η Haskell είναι η κυρίαρχη γλώσσα στην έρευνα στο συναρτησιακό προγραμματισμό

Μια Ανασκόπηση/Σύνοψη της Scheme

- Όπως αναφέρθηκε, η Scheme είναι μια ιδιαίτερα κομψή Lisp
  - Ο διερμηνέας εκτελεί έναν βρόχο ανάγνωσης-αποτίμησης-εκτύπωσης
  - Ό,τι εισάγεται στο διερμηνέα αποτιμάται (αναδρομικά) μόνο μια φορά
  - Αν κάτι είναι μέσα σε παρενθέσεις, είναι κλήση συνάρτησης (εκτός αν είναι σε παράθεση)
  - Οι παρενθέσεις ΔΕΝ υπάρχουν μόνο για ομοαδοποίηση, όπως στις γλώσσες της οικογένειας της Algol
    - Η προσθήκη ενός επιπέδου παρενθέσεων αλλάζει τη σημασία

    `(+ 3 4)` $\Rightarrow$ `7`

    `((+ 3 4))` $\Rightarrow$ `error`

    (το βέλος '$\Rightarrow$' σημαίνει «αποτιμάται σε»)

Μια Ανασκόπηση/Σύνοψη της Scheme

- Scheme:
  - Λογικές τιμές #t και #f
  - Αριθμοί
  - Εκφράσεις λάμβδα
  - Παράθεση

```
(+ 3 4) ⇒ 7
(quote (+ 3 4)) ⇒ (+ 3 4)
'(+ 3 4) ⇒ (+ 3 4)
```

  - Μηχανισμοί για τη δημιουργία νέων εμβελειών

```
(let ((square (lambda (x) (* x x))) (plus +))
(sqrt (plus (square a) (square b))))
let*
letrec
```

Μια Ανασκόπηση/Σύνοψη της Scheme

- Scheme:
  - Εκφράσεις συνθήκης

```
(if (< 2 3) 4 5) ⇒ 4
(cond
 ((< 3 2) 1)
 ((< 4 3) 2)
 (else 3)) ⇒ 3
```

  - Προστακτικές δυνατότητες
    - αναθέσεις
    - ακολουθιακή εκτέλεση (begin)
    - επανάληψη
    - είσοδος-έξοδος (read, display)

Μια Ανασκόπηση/Σύνοψη της Scheme

- Πρότυπες συναρτήσεις της Scheme (η λίστα δεν είναι πλήρης):
  - αριθμητικές
  - λογικοί τελεστές
  - ισοδυναμία
  - τελεστές λιστών
  - symbol?
  - number?
  - complex?
  - real?
  - rational?
  - integer?

Επιστροφή στη Σειρά Αποτίμησης

- Εφαρμοστική σειρά
  - την έχετε συνηθίσει από τις προστακτικές γλώσσες
  - συνήθως γρηγορότερη
- Κανονική σειρά
  - όπως η κλήση κατ' όνομα: δεν αποτιμά την παράμετρο μέχρι να τη χρειαστεί
  - μερικές φορές πιο γρήγορη
  - αν ο υπολογισμός μπορεί γενικά να τερματιστεί, η αποτίμηση θα τελειώσει (θεώρημα Church-Rosser)

Επιστροφή στη Σειρά Αποτίμησης

- Στη Scheme
  - οι συναρτήσεις χρησιμοποιούν εφαρμοστική σειρά που ορίζεται με εκφράσεις λάμβδα
  - οι ειδικές μορφές (γνωστές και ως μακροεντολές) χρησιμοποιούν κανονική σειρά που ορίζεται με συντακτικούς κανόνες
- Μια *αυστηρή* γλώσσα απαιτεί όλα τα ορίσματά της να είναι καλά ορισμένα, επομένως μπορεί να χρησιμοποιηθεί εφαρμοστική σειρά
- Μια *μη αυστηρή* γλώσσα δεν απαιτεί όλα τα ορίσματά της να είναι καλά ορισμένα· χρειάζεται αποτίμηση κανονικής σειράς

Επιστροφή στη Σειρά Αποτίμησης

- Η οκνηρή αποτίμηση προσφέρει τα πλεονεκτήματα και των δυο κόσμων
- Αλλά δεν είναι καλή όταν υπάρχουν παρενέργειες.
  - χρήση των delay και force στη Scheme
  - η delay δημιουργεί μια "υπόσχεση"

Συναρτήσεις Υψηλής Τάξης

- Συναρτήσεις υψηλότερης τάξης
  - Δέχονται ως όρισμα μια συνάρτηση, ή επιστρέφουν ως αποτέλεσμα μια συνάρτηση
  - Πολύ χρήσιμες για την δημιουργία πραγμάτων
  - Currying (το όνομα προέρχεται από τον Haskell Curry, από τον ίδιο προέρχεται και το όνομα της Haskell)
    - Για λεπτομέρειες, δείτε το λογισμό λάμβδα σε επόμενες διαφάνειες
    - Η ML, η Miranda, και η Haskell διευκολύνουν ιδιαίτερα τον ορισμό συναρτήσεων με currying

Θεωρητικές Βάσεις : Λογισμός - λ

- Λογισμός λάμβδα:
  - Μια σημειογραφία/μοντέλο υπολογισμών που βασίζεται στον αμιγή συντακτικό χειρισμό συμβόλων, τα πάντα είναι συναρτήσεις
  - Αναπτύχθηκε από τον Alonzo Church στη δεκαετία του 1930 σαν μοντέλο υπολογισιμότητας
  - Ο Church άνηκε σε ένα πλήθος ανθρώπων μεταξύ των οποίων συγκαταλέγονται οι Chomsky, Turing, Kleene, και Rosser

Θεωρητικές Βάσεις : Λογισμός - λ

- Μπορούμε να ορίσουμε πράγματα όπως οι ακέραιοι με βάση μια διακεκριμένη συνάρτηση (όπως η ταυτοτική) που να αναπαριστά το μηδέν, και μια συνάρτηση του «επόμενου» που μας δίνει όλους τους υπόλοιπους αριθμούς
- Γίνεται εύκολο να οριστούν οι αριθμητικοί τελεστές στη σημειογραφία
  - Στην πράξη αυτό είναι κουραστικό
    - θα υποθέσουμε την ύπαρξη της αριθμητικής και διακεκριμένων *σταθερών συναρτήσεων* για τους αριθμούς

Θεωρητικές Βάσεις : Λογισμός - λ

- Παράδειγμα εκφράσεων λάμβδα

```
id              λx.x
const           λx.2
plus            λx.λy.x + y
square          λx.x * x
hypot           λx. λy.sqrt
(plus (square x) (square y))
```

Θεωρητικές Βάσεις : Λογισμός - λ

- Αναδρομικά, μια έκφραση λάμβδα είναι
  - (1) ένα όνομα
  - (2) μια αφαίρεση που αποτελείται από ένα λαμβδα, ένα όνομα, μια τελεία, και μια έκφραση λάμβδα
  - (3) μια εφαρμογή που αποτελείται από δυο διπλανές λάμβδα εκφράσεις (η παράθεση σημαίνει εφαρμογή συνάρτησης), ή
  - (4) μια έκφραση λάμβδα σε παρενθέσεις

Θεωρητικές Βάσεις : Λογισμός - λ

- Συνήθως η εφαρμογή προσεταιρίζει από αριστερά προς τα δεξιά, επομένως η `f A B` είναι `(f A) B`, και όχι `f (A B)`

- Επίσης, η εφαρμογή έχει υψηλότερη προτεραιότητα από την αφαίρεση, επομένως η λ`x.A B` είναι λ`x.(A B)`, και όχι `(`λ`x.A) B` – π.χ., ML

- Οι παρενθέσεις χρησιμοποιούνται για σαφήνεια, ή για την παράβαση των κανόνων: εξ ορισμού τις χρησιμοποιούμε σε κάθε αφαίρεση που χρησιμοποιείται ως συνάρτηση ή παράμετρος:

  `(`λ`f.f 2) (`λ`x.plus x x)`

Θεωρητικές Βάσεις : Λογισμός - λ

- Αυτοί οι κανόνες σημαίνουν ότι η εμβέλεια της τελείας φτάνει δεξιά μέχρι την πρώτη δεξιά παρένθεση που δεν της αντιστοιχεί αριστερή παρένθεση, ή μέχρι το τέλος της έκφρασης αν δεν υπάρχει τέτοια παρένθεση
  - Στην (λx. λy. λz.e) a b c, η αρχική συνάρτηση δέχεται μια παράμετρο και επιστρέφει μια συνάρτηση (μιας παραμέτρου) που επιστρέφει μια συνάρτηση (μιας παραμέτρου)
  - Για την αναγωγή της έκφρασης, αντικαθιστάτε την a σε κάθε x στη λy. λz.e, στη συνέχεια αντικαθιστάτε την b σε κάθε y στην υπόλοιπη έκφραση, και την c σε κάθε z σε ό,τι μένει στο τέλος

Θεωρητικές Βάσεις : Λογισμός - λ

- Παράδειγμα:

```
(λx. λy.x + y) 3 4
λy.(3 + y) 4
(3 + 4)
7
```

- Ελεύθερες και δεσμευμένες μεταβλητές: μια μεταβλητή είναι δεσμευμένη αν εισάγεται από ένα λάμβδα
  - Για παράδειγμα, στη λx. λy.(* x y) έχουμε δυο ένθετες εκφράσεις λάμβδα
    - η x είναι ελεύθερη στην εσωτερική (λy.(* x y)), αλλά δεσμευμένη στην εξωτερική

Θεωρητικές Βάσεις : Λογισμός - λ

- Η αποτίμηση των εκφράσεων λάμβδα γίνεται μέσω
  - (1) αντικατάστασης των παραμέτρων (*βήτα αναγωγή*)
    `(λx.times x x) y  =>  times y y`
  - (2) μετονομασίας των μεταβλητών (*άλφα μετατροπή*) (συχνά για να αποφύγουμε τις συγκρούσεις ονομάτων)
    `(λx.times x x) y ==(λz.times z z) y`
  - (3) απλοποίηση «εκτός σειράς» (*ήτα αναγωγή*)
    `(λx.f x)  =>  f`
    - Ο τελευταίος κανόνας είναι δυσνόητος· ΔΕΝ είναι ο ίδιος με τη βήτα αναγωγή

Απόψεις περί Συναρτ. Προγραμματισμού

- Πλεονεκτήματα των συναρτησιακών γλωσσών
  - η έλλειψη παρενεργειών διευκολύνει την κατανόηση των προγραμμάτων
  - η έλλειψη ρητής σειράς αποτίμησης (σε μερικές γλώσσες) προσφέρει την πιθανότητα παράλληλης αποτίμησης (π.χ. MultiLisp)
  - η έλλειψη παρενεργειών και ρητής σειράς αποτίμησης απλοποιούν κάποια πράγματα για το μεταγλωττιστή
  - τα προγράμματα συχνά είναι εκπληκτικά μικρά
  - η γλώσσα μπορεί να είναι εξαιρετικά μικρή αλλά ισχυρή

Απόψεις περί Συναρτ. Προγραμματισμού

- Προβλήματα
  - δύσκολο (αλλά όχι αδύνατο!) να υλοποιηθούν αποτελεσματικά σε υπολογιστές von Neumann
    - πολλές αντιγραφές δεδομένων μέσω παραμέτρων
    - (φαινομενική) ανάγκη να δημιουργείται ένας νέος πίνακας όταν πρέπει να αλλάξει ένα στοιχείο του
    - συχνή χρήση δεικτών (προβλήματα χώρου/χρόνου και τοπικότητας)
    - συχνές κλήσεις διαδικασιών
    - η αναδρομή χρησιμοποιεί σημαντικό χώρο
    - απαιτεί συλλογή σκουπιδιών
    - απαιτεί ένα διαφορετικό τρόπο σκέψης από τον προγραμματιστή
    - δύσκολο να ενσωματωθεί η είσοδος-έξοδος στο αμιγώς συναρτησιακό μοντέλο