ΑΝΑΦΟΡΑ ΥΛΟΠΟΙΗΣΗΣ

1) Γενική Ιδέα - Main.hs

Έχουμε δημιουργήσει έναν διερμηνέα (interpreter) για την γλώσσα μας (RouReg), γραμμένο σε Haskell. Για τη συγγραφή των προγραμμάτων μας έχει δημιουργηθεί στη Main.hs κατάλληλο interface. Ο κύκλος λειτουργίας για την εκτέλεση ενός προγράμματος είναι ο εξής:

I. Ο κώδικας διαβάζεται ως string και περνίεται στη συνάρτηση process. Από εκεί καλείται η συνάρτηση parseDclr, η οποία είναι η κύρια συνάρτηση του Parser που δημιουργήσαμε με το Happy, για την κατασκευή του συντακτικού δέντρου. Εντός του Parser μας, γίνεται η λεκτική ανάλυση με τη βοήθεια του Lexer που δημιουργήσαμε με το Alex.

II. Εφόσον το συντακτικό δέντρο κατασκευαστεί και είναι “έγκυρο” , δηλαδή δεν προκύψει συντακτικό σφάλμα, περνίεται ως όρισμα στη συνάρτηση execMain, η οποία καλεί με τη σειρά της την runMain, που είναι η υπεύθυνη συνάρτηση του Eval.hs ώστε να δοθεί το αποτέλεσμα της εκτέλεσης του προγράμματος.

2) Συντακτικό Δέντρο – Δομές δεδομένων

Τα προγράμματα γράφονται στη μορφή dclr; dclr; ...dclr . Η αποτίμηση γίνεται μόνο για την τελευταία εντολή dclr που είναι κάτι σαν τη main του ghci. Κάθε dclr είναι της μορφής
VAR Apats '=' Expr , όπου Apats είναι τα ορίσματα της συνάρτησης , VAR το όνομα της, και Expr το σώμα της. Για το συντακτικό μας δέντρο, ο parser “μεταφράζει” το dclr σαν έναν κόμβο ***Assign Name [Apats] Expr,*** όπου Name=VAR, [Apats] = (VAR:Apats) σε μορφή λίστας,
Expr = Lam [Apats] Expr. Τα Expr μπορεί να είναι κόμβοι της μορφής λ-εκφράσεων
Lam [Apats] Expr, είτε της μορφής εφαρμογής συνάρτησης ***App Expr Expr***, είτε της μορφής let-εκφράσεων ***Let [Dclr] Expr***, είτε των απλών εκφράσεων (μεταβλητών και αριθμών) ***Apat Apats***, είτε της μορφής τελεστή ***Op Binop Expr Expr*.**

3) Λεπτομερής Περιγραφή της Eval.hs

Η runMain παίρνει ως είσοδο την λίστα με τους κόμβους dclr(συντακτικό δέντρο) και επιστρέφει το τελικό αποτέλεσμα ή το error που έχει προκύψει. Για να το πετύχει αυτό καλέι την evalMain, με αρχικά “κενο” περιβάλλον. Η evalMain είναι μία συνάρτηση που παίρνει ένα περιβάλλον (env) και μία λίστα απο declarations και υπολογίζει το τελευταίο από αυτά τα declarations (που αντιστοιχεί στη main του ghci). Αυτό το επιτυγχάνει ανανεώνοντας αναδρομικά το περιβάλλον για κάθε declaration και υπολογίζοντας το τελευταίο από αυτά στο περιβάλλον που έχει προκύψει.

Περιβάλλον (type Scope) και τιμές (data Value)

Έχουμε τρεις κατασκευαστές τιμών για τον data type Value: για ακεραίους (Vint int), για λογικές τιμές (Vbool bool) και για closures -δηλαδή για λ-εκφράσεις- (VClosure [Apats] Expr Scope). Όπως και στη Haskell, έτσι και στην RouReg θέλουμε να έχουμε σαν τιμές συναρτήσεις, καθώς μία συνάρτηση μπορεί να επιστρέφει μία άλλη συνάρτηση, αν για παράδειγμα γίνει μερική εφαρμογή

των ορισμάτων της (partial application). Στις λ-εκφράσεις έχουμε μια λίστα από τις εξαρτημένες μεταβήτές (ορίσματα της ανώνυμης συνάρτησης) [Apats], την έκφραση (σώμα συνάρτησης) Expr και το περιβάλλον (Scope) στο οποίο θα αποτιμηθεί η έκφραση αυτή όταν και αν χρειαστεί.

Ως περιβάλλον έχει οριστεί μία map δομή η οποία αντιστοιχίζει ονόματα μεταβήτών σε τιμές, συγκεκριμένα ένα όνομα (string) αντιστοιχίζεται σε μία λίστα τιμών ([Value]). Χρειαζόμαστε λίστα τιμών για να μπορούμε να κάνουμε το pattern matching σε συναρτήσεις με περισσότερα από ένα ορίσματα (arguments), που γίνεται partial application. Για παράδειγμα αν έχουμε μία συνάρτηση:

f 0 1 = 1 ; f x y = x*y ; g = f 0 ; h = g 1 + g 2

κατά την ανάθεση θέλουμε να αντιστοιχίσουμε το όνομα “g” με δύο λ-εκφράσεις [(\1→1), (\y→x*y)] στο περιβάλλον όπου το “χ” αντιστοιχίζεται στο [Int 0], ώστε αν/όταν χρειαστεί, όπως στην h, να πάρουμε τον κατάλληλο όρισμό για το pattern matching στο δέυτερο όρισμα.