Αρχές Γλωσσών Προγραμματισμού

Άσκηση 3η

Στοιχεία ομάδας:

1)Σπάχου Ευαγγελία

1115201200169

2)Σταμούλου Κωνσταντίνα

1115201200173

3)Κατούνας Γεώργιος

1115201200061

Στα πλαίσια της συγκεκριμένης άσκησης, της οποίας το ζητούμενο ήταν η υλοποίηση ενός διερμηνέα μιας επέκτασης του λ-λογισμού,

δημιουργήσαμε της εξής συναρτήσεις:

-fv :: Term -> [String] -> [String] : Επιστέφει τη λίστα με της ελεύθερες μεταβλητές του λ-όρου τύπου Term (1^{ο} όρισμα), χρησιμοποιώντας τις

(f :: Term -> String) και (returnfv :: [String] -> [String] -> [String]) για την έυρεση των ελεύθερων μεταβλητών του κάθε υποόρου και την τελική λίστα με τις ελεύθερες μεταβλητές από μία φορά την καθεμία αντίστοιχα.

-replace :: Term -> String -> Term -> Term : Πραγματοποιεί την αντικατάσταση μιας μεταβλητής (string, 2^{ο} όρισμα στον τύπο) με έναν λ-όρο τύπου Term (3^{ο} όρισμα στον τύπο). Έχουμε υλοποιήσει όλες τις

περιπτώσεις που περιγράφονται στην διαφάνεια 29, κεφ.9 του μαθήματος. Για την τελευταία περίπτωση που αλλάζει την όνομα στη δεσμευμένη μεταβλητή του όρου, χρησιμοποιούμε τη συνάρτηση (findDiffBounded :: String -> Int -> Term -> Term -> String) η οποία παίρνει τη δεσμευμένη μεταβλητή και επιστρέφει στη θέση της μία άλλη που ικανοποιεί τις συνθήκες. Η αναζήτηση της νέας μεταβλητής γίνεται με τη χρήση του chr g, όπου g ένας ακέραιος που ξεκινάει με το 97 (‘a’).

-reduce :: Term -> Result : Η reduce παίρνει σαν όρισμα ένα Term και επιστρέφει ένα Result, όπως αυτό είναι ορισμένο στο βοηθητικό αρχείο. Η reduce καλεί την red με ορίσματα το Term 2 κενές λίστες, μία για τους ενδιάμεσους λ-όρους και μία για τις μετατροπές και 0 που για τον counter που μετράει πόσα reductions έγιναν.

-red :: Term -> Int -> [Term] -> [String]-> Result : Η red χρησιμοποιεί τις (eta :: Term -> Term) και (beta :: Term -> Term), οι οποίες υλοποιούν τα ήττα και βήτα reductions αντίστοιχα. Πάντα γίνεται πρώτα το ποίο αριστερό reduction. Στην περίπτωση που ένας λ-όρος δεν έχει κανονική μορφή αλλά μετά από κάποιο βήτα reduction ο όρος δεν αλλάζει (π.χ: ο όρος ((\\x.xx)(\\x.xx)) μετά απο βήτα reduction μένει ίδιος και επομένως δεν έχει κανονική μορφή), η reduce επιστρέφει false , κενές τις 2 λίστες και -1 στον counter. Σε κάθε άλλη περίπτωση που ο λ-όρος δεν έχει κανονική μορφη το πρόγραμμα δεν τερματίζει.

<u>Πέρα από το ζητούμενο της άσκησης υλοποίησαμε και τα εξής:</u>

Τα λογικά true, false, not, cond ως εξής:

true = "(\\x.\\y.x)"

false = "(\\x.\\y.y)"

notnot = "(\\z.z(\\x.\\y.y)(\\x.\\y.x))"

condition = "(\\z.\\x.\\y.zxy)"

-logicnot :: String -> String : Παίρνει ένα string (true ή false) και επιστρέφει τον λ-όρο της άρνησής του.

Π.χ. logicnot true επιστρέφει "\\x.\\y.y" (που είναι ο λ-όρος του false)

cond :: String -> String -> String -> String : Παίρνει τρία String, το πρώτο θα είναι μία λογική τιμή (true/false) και τα επόμενα δύο ο,τιδήποτε. Ανάλογα με το πρώτο που αποτελεί τη συνθήκη, επιστρέφει αντίστοιχα το δεύτερο ή το τρίτο string.

Π.χ. cond true "m" "n" επιστρέφει "m"

returnTerm :: Result -> Term : Παίρνει έναν όρο τύπου Result και επιστρέφει από αυτόν το πρώτο του τμήμα που είναι όρος τύπου Term.

-first :: String -> String -> String : Παίρνει δύο strings και επιστρέφει το πρώτο από αυτά.

Π.χ. first "n" "m" επιστρέφει "n"

-second :: String -> String -> String : Παίρνει δύο strings και επιστρέφει το δεύτερο από αυτά.

Π.χ. second "n" "m" επιστρέφει "m"

pairs :: String -> String -> String : Παίρνει δύο strings και τα συνενώνει με τον λ-όρο του pair, επιστρέφοντας ένα ενιαίο string. Χρησιμοποιείται από τις first και second.

-data Cc = C Int deriving(Show,Eq) : Ο τύπος τον Church numerals

-c :: Cc -> String : Επιστρέφει τον λ-όρο του αριθμοειδούς, χρησιμοποιώντας την -findNum για να βρει τον αριθμο των f. Καλείται με τον εξής τρόπο:

Π.χ. Για το c0: c (C 0)

Για το c1: c (C 1)

Κτλ.

-successor :: Cc -> String : Της δίνουμε ένα αριθμοειδές και επιστρέφει τον λ-όρο του επόμενου.

π.χ. successor (C 3) επιστρέφει "\\f.\\x.f(f(f(f(x))))"

-add :: Cc -> Cc -> String : Παίρνει δύο αριθμοειδή και επιστρέφει τον λ-όρο του αθροίσματός τους.

Π.χ. add (C 1) (C 3) επιστρέφει "\\f.\\x.f(f(f(f(x))))"

-mult :: Cc -> Cc -> String : Επιστρέφει το γινόμενο δύο αριθμοειδών.

Π.χ. mult (C 2) (C 3) επιστρέφει "\\f.\\x.f(f(f(f(f(x)))))"

-expon :: Cc -> Cc -> String : Παίρνει δύο αριθμοειδή και επιστρέφει το αριθμοειδές που προκύπτει αν υψώσουμε τον εκθέτη του πρώτου στον εκθέτη του δεύτερου.

Π.χ. expon (C 2) (C 3) επιστρέφει "\\f.\\x.f(f(f(f(f(f(f(x)))))))"