Αρχές Γλωσσών Προγραμματισμού : Άσκηση 3η
Ημερομηνία Παράδοσης Άσκησης : 28/2/2016 , 23:59 μμ

Να υλοποιηθεί σε **Haskell** ένας διερμηνέας για μια επέκταση του λ-λογισμού. Το συντακτικό των λ-όρων θα πρέπει να είναι όπως το γνωστό συντακτικό του λ-λογισμού με τη διαφορά ότι το λ θα αναπαρίσταται με backslash (που στη Haskell γράφεται "\\" λόγω της αναγκαίας χρήσης escape character). Ο διερμηνέας σας θα πρέπει να παράγει την **κανονική μορφή** της λ-έκφρασης που δίνει ο χρήστης (αν αυτή υπάρχει). Για παράδειγμα αν η είσοδος στον διερμηνέα είναι η ((\x. (\y. x)) (\z. z)) τότε η έξοδος θα πρέπει να είναι (\y. (\z. z)). Προς δική σας διευκόλυνση οι μεταβλητές θεωρήστε πως αποτελούνται από ακριβώς 1 χαρακτήρα (παρόλο που στην μοντελοποίηση που ακολουθεί χρησιμοποιείται ο τύπος String και όχι ο τύπος Char).

Η μοντελοποίηση των λ-όρων σε Haskell πρέπει να γίνει με τον παρακάτω τρόπο :

```
data Term = Var String | Application Term Term | Abstraction String Term deriving(Show , Eq)
```

Συνολικά καλείστε να γράψετε :

1) Μια συνάρτηση **reduce :: Term -> Result** που παίρνει για όρισμα ένα λ-όρο και επιστρέφει ένα Result. Το Result πρέπει να μοντελοποιεί :
   - την κανονική μορφή του λ-όρου που δόθηκε ως όρισμα
   - τον αριθμό των reductions που απαιτήθηκαν για τον υπολογισμό αυτής
   - την λίστα από τους ενδιάμεσους λ-όρους που παρήχθησαν κατά τον υπολογισμό αυτής
   - την λίστα από τις συμβολοσειρές που αντιστοιχούν στο είδος του reduction που έγινε στο αντίστοιχο βήμα του υπολογισμού αυτής (η λίστα αυτή μπορεί να περιέχει τις συμβολοσειρές "beta" , "eta")

Ο ορισμός του Result πρέπει να είναι ο παρακάτω :

```
data Result = Res Term Int [Term] [String] deriving(Show , Eq)
```

Υπόδειξη : Αρχικά φροντίστε η συνάρτηση reduce που θα γράψετε να υπολογίζει σωστά την κανονική μορφή του λ-όρου που παίρνει ως όρισμα (αυτό αποτελεί το βασικό ζητούμενο της άσκησης σε πρώτο στάδιο). Συνεπώς υλοποιείστε αρχικά μια συνάρτηση **reduce :: Term -> Term** και βεβαιωθείτε πως υπολογίζει σωστά την κανονική μορφή του λ-όρου. Έπειτα βελτιώστε σταδιακά την υλοποίησή σας ώστε η reduce να υπολογίζει και να επιστρέφει όλα όσα σας ζητούνται παραπάνω (δηλαδή να επιστρέφει ένα Result).

2) Μια μικρή αναφορά (report) μεγέθους το πολύ 2 σελίδων στην οποία να εξηγείτε τη λειτουργικότητα του κώδικα σας τον τρόπο με τον οποίο υπολογίζεται η κανονική μορφή του λ-όρου καθώς και ότι άλλο θεωρείτε πως πρέπει να αναφερθεί από την πλευρά σας.

Παράδοση Ασκήσεων: Η παράδοση πρέπει να γίνει μέχρι τις 23:59 μμ , την 28/2/2016. Ο κώδικας της άσκησης θα πρέπει να έχει αποσταλεί με e-mail σε μορφή attachment στο **gvastakis@di.uoa.gr** (με κοινοποίηση στο **prondo@di.uoa.gr**) μέχρι την παραπάνω ημερομηνία. Καθυστερημένες ασκήσεις δε θα βαθμολογηθούν. Σε περιπτώσεις αντιγραφής όλες οι εμπλεκόμενες ασκήσεις θα μηδενίζονται. Η εργασία μπορεί να γίνει είτε ατομικά είτε σε ομάδες των δύο ατόμων.

Προς δική σας διευκόλυνση

Στα πλαίσια αυτής της άσκησης δε θα χρειαστεί να γράψετε εσείς κάποιον Parser που να δέχεται για όρισμα μια συμβολοσειρά και να επιστρέφει τον αντίστοιχο λ-όρο. Εσείς καλείστε να γράψετε συναρτήσεις με σκοπό τον υπολογισμό της κανονικής μορφής ενός λ-όρου (που έχει γίνει parse με κάποιον τρόπο ήδη θεωρήστε το ως ένα black-box).

Στο βοηθητικό αρχείο που σας δίνεται μαζί με την εκφώνηση υπάρχει υλοποιημένος ένας Parser , με σκοπό να σας βοηθήσει. Συγκεκριμένα σας δίνεται η συνάρτηση myparse :: String -> Term η οποία παίρνει για όρισμα μια συμβολοσειρά και επιστρέφει τον αντίστοιχο λ-όρο. Για παράδειγμα :

```
> myparse  “(\\x.\\y.x)(\\z.z)"
Application (Abstraction "x" (Abstraction "y" (Var "x"))) (Abstraction "z" (Var “z"))
```

Αντίστοιχα με τη συνάρτηση myparse σας δίνεται και η συνάρτηση prettyprint με υπογραφή τύπου prettyprint :: Term -> String που κάνει το αντίστροφο (δηλαδή παίρνει για όρισμα έναν λ-όρο και επιστρέφει την αντίστοιχη αναπαράσταση του σε συμβολοσειρά). Η συνάρτηση αυτή θα σας βοηθήσει στον έλεγχο των κανονικών μορφών που θα παράγονται (για να είναι πιο ευανάγνωστες).

Για να χρησιμοποιηθούν τα παρακάτω πρέπει να έχετε εγκατεστημένα :
1) Έκδοση ghc 7.10.2 (με αυτήν την έκδοση έχει ελεγχθεί ο Parser αλλά λογικά θα δουλεύει και με νεότερες εκδόσεις)
2) Cabal : υπάρχει περίπτωση το Cabal να το έχετε ήδη εγκατεστημένο από την εγκατάσταση της Haskell. Αν όχι για την εγκατάσταση και της Haskell (ghc) αλλά και του Cabal μεταβείτε στη σελίδα https://www.haskell.org/downloads#platform και επιλέξτε την έκδοση για το λειτουργικό σας σύστημα.

Έπειτα αφού εγκαταστήσετε τη Haskell και το Cabal εισάγετε στη γραμμή εντολών τις 2 παρακάτω εντολές :

```
cabal update
cabal install parsec
```

Τέλος αφού έχετε κάνει όλα τα παραπάνω σωστά δοκιμάστε να φορτώσετε το βοηθητικό αρχείο που σας δίνεται (θα πρέπει να φορτωθεί χωρίς κανένα πρόβλημα).

Σύνδεσμοι που ίσως σας φανούν χρήσιμοι για την Άσκηση

Maybe - https://hackage.haskell.org/package/base-4.8.1.0/docs/Data-Maybe.html

Either - https://hackage.haskell.org/package/base-4.8.1.0/docs/Data-Either.html

Tuples - https://en.wikibooks.org/wiki/Haskell/Lists_and_tuples#Tuples

Case Expressions - http://learnyouahaskell.com/syntax-in-functions#case-expressions

Case Expressions - http://stackoverflow.com/a/2226292

Where - http://learnyouahaskell.com/syntax-in-functions#where

Let - http://learnyouahaskell.com/syntax-in-functions#let-it-be

Algebraic data types - http://www.seas.upenn.edu/~cis194/lectures/03-ADTs.html

(Στον παραπάνω σύνδεσμο **αγνοήστε** την πρώτη παράγραφο με τίτλο Libraries)

Σύγκριση ορισμού Δέντρων σε Haskell και Java - http://book.realworldhaskell.org/read/defining-types-streamlining-functions.html#deftypes.recursive

(Στον παραπάνω σύνδεσμο δείτε από την παράγραφο για Recursive types μόνο τη σύγκριση του ορισμού μεταξύ Haskell και Java - δεν έχει απαραίτητα σχέση με την άσκηση).