Αρχές Γλωσσών Προγραμματισμού

Χειμερινό Εξάμηνο 2015-2016
Παρουσίαση Άσκησης 3

Γλώσσα λ- όρων

Χρησιμοποιώντας αφηρημένη σύνταξη σε μορφή BNF και θεωρώντας ότι η συντακτική κλάση των μεταβλητών παριστάνεται με το μη-τερματικό σύμβολο ⟨var⟩, η γλώσσα Λ των λ-όρων περιγράφεται ισοδύναμα ως εξής:

$$
\begin{array}{rcl}
\langle \text{term} \rangle & ::= & \langle \text{var} \rangle \\
& | & (\langle \text{term} \rangle \ \langle \text{term} \rangle) \\
& | & (\lambda \langle \text{var} \rangle . \langle \text{term} \rangle)
\end{array}
$$

Παράδειγμα 9.1 Οι παρακάτω είναι λ-όροι:

$(x\,y)$

$(\lambda x.\, x)$

$(\lambda x.\, (\lambda y.\, (x\,y)))$

$(((\lambda x.\, x)\, y)\, (\lambda x.\, z))$

$((\lambda x.\, (\lambda y.\, z))\, (\lambda x.\, x))$

$(\lambda x.\, ((\lambda y.\, y)\, (\lambda z.\, x)))$

□

Μοντελοποίηση σε Haskell

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

- Στα πλαίσια της άσκησης , οι μεταβλητές θεωρήστε πως αποτελούνται από ακριβώς 1 χαρακτήρα (παρόλο που στην μοντελοποίηση χρησιμοποιείται ο τύπος String και όχι ο τύπος Char)

- Το deriving που αναφέρεται στον παραπάνω ορισμό αγνοήστε το εντελώς …

.

Keep in mind …

1. Οι εξωτερικές παρενθέσεις δε γράφονται:

$$\lambda x.\, x \quad \text{είναι συντομογραφία του} \quad (\lambda x.\, x)$$

2. Η εφαρμογή είναι αριστερά προσεταιριστική:

$$F\, M_1\, M_2\, \ldots\, M_n \quad \text{είναι συντομογραφία του} \quad (\ldots((F\, M_1)\, M_2)\, \ldots\, M_n)$$

3. Η αφαίρεση εκτείνεται όσο περισσότερο είναι δυνατό, δηλαδή ως το επόμενο κλείσιμο παρένθεσης ή το τέλος του όρου:

$$\lambda x.\, M_1\, M_2\, \ldots\, M_n \quad \text{είναι συντομογραφία του} \quad \lambda x.\, (M_1\, M_2\, \ldots\, M_n)$$

Παράδειγμα 9.2 Ακολουθώντας τις παραπάνω συμβάσεις, οι λ-όροι του παραδείγματος 9.1 γράφονται σε απλοποιημένη μορφή ως εξής:

$x\, y$

$\lambda x.\, x$

$\lambda x.\, \lambda y.\, x\, y$

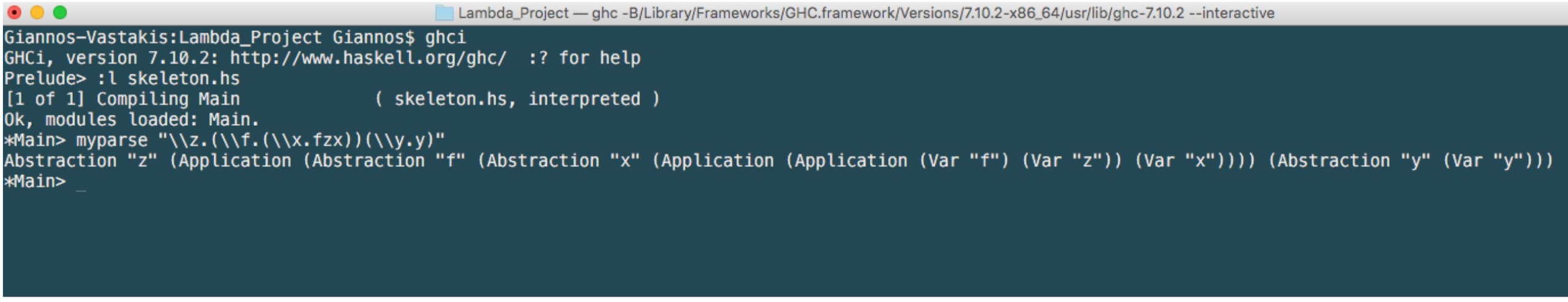
$((\lambda x.\, x)\, y)\, (\lambda x.\, z)$

$(\lambda x.\, \lambda y.\, z)\, (\lambda x.\, x)$

$\lambda x.\, (\lambda y.\, y)\, (\lambda z.\, x)$

□

Parsing in Haskell



- Για δική σας διευκόλυνση σας δίνεται η συνάρτηση myparse :: String -> Term για να κάνετε parse λ-όρους

- Γράφουμε κολλητά και **χωρίς κενά** όπως φαίνεται στο παραπάνω παράδειγμα

- Το backslash στη Haskell απαιτεί χρήση escape character όπως επίσης φαίνεται από πάνω

```
Lambda_Project — ghc -B/Library/Frameworks/GHC.framework/Versions/7.10.2-x86_64/usr/lib/ghc-7.10.2 --interactive
*Main> :l skeleton.hs
[1 of 1] Compiling Main             ( skeleton.hs, interpreted )
Ok, modules loaded: Main.
*Main> myterm
Application (Abstraction "x" (Abstraction "y" (Var "x"))) (Abstraction "z" (Var "z"))
*Main> inputString
"(\\x.\\y.x)(\\z.z)"
*Main> myparse inputString
Application (Abstraction "x" (Abstraction "y" (Var "x"))) (Abstraction "z" (Var "z"))
*Main> myparse inputString == myterm
True
*Main> prettyprint myterm
"(\\x.\\y.x)(\\z.z)"
*Main> prettyprint myterm == inputString
True
*Main>
```

- Όμοια σας δίνεται και η συνάρτηση prettyprint :: Term -> String η οποία κάνει το αντίστροφο από ότι η myparse

Αφού μου έδωσε τον Parser εγώ τι θα κάνω ?

- Εσείς θα πρέπει να γράψετε μια συνάρτηση reduce που να **υπολογίζει και να επιστρέφει την κανονική μορφή ενός λ-όρου** (και ορισμένες πληροφορίες ακόμα όπως θα δούμε παρακάτω)

- Αρχικά για δική σας διευκόλυνση γράψετε μια συνάρτηση reduce που να υπολογίζει σωστά την κανονική μορφή του λ-όρου . Στη συνέχεια θα βελτιώσετε την αρχική σας υλοποίηση ώστε να υπολογίζει και να επιστρέφει επίσης :

  - πόσα reductions έγιναν κατά τον υπολογισμό αυτό

  - τους ενδιάμεσους λ-όρους που παρήχθησαν κατά τον υπολογισμό αυτό

  - τη σειρά με την οποία έγιναν τα reductions κατά τον υπολογισμό

Ένα παράδειγμα υπολογισμού της κανονικής μορφής

Ενδιάμεσοι λ-όροι που παρήχθησαν

Παράδειγμα 9.9 Ο όρος $\lambda z.\,(\lambda f.\,\lambda x.\,f\,z\,x)\,(\lambda y.\,y)$ στο παράδειγμα 9.8 είδαμε ότι δεν είναι σε κανονική μορφή. Όμως, εφαρμόζοντας διαδοχικά βήματα μετατροπής προκύπτει:

$$\begin{array}{lll} \lambda z.\,(\lambda f.\,\lambda x.\,f\,z\,x)\,(\lambda y.\,y) & \to_\beta & \lambda z.\,\lambda x.\,(\lambda y.\,y)\,z\,x \\ & \to_\beta & \lambda z.\,\lambda x.\,z\,x \\ & \to_\eta & \lambda z.\,z \end{array}$$

Η κανονική μορφή του λ-όρου

Σειρά με την οποία έγιναν τα reductions

Συνολικά έγιναν 3 reductions

και άρα $\lambda z.\,(\lambda f.\,\lambda x.\,f\,z\,x)\,(\lambda y.\,y) \twoheadrightarrow \lambda z.\,z$. Καθώς ο όρος $\lambda z.\,z$ είναι σε κανονική μορφή, ο αρχικός όρος έχει αυτή την κανονική μορφή. □

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

- Τελικά εσείς πρέπει να γράψετε μια συνάρτηση **reduce :: Term -> Result**
- Θα σας είναι πιο εύκολο αν ξεκινήσετε φτιάχνοντας μια συνάρτηση **reduce :: Term -> Term** που υπολογίζει και επιστρέφει την κανονική μορφή του λ-όρου που παίρνει ως όρισμα και έπειτα την “βελτιώσετε” ώστε να επιστρέφει Result αντί για Term (θα δυσκολευτείτε πολύ περισσότερο αν πάτε από την αρχή να τα υλοποιήσετε όλα μαζί)

Απαιτούμενα για χρήση της myparse και της prettyprint

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 skeleton.hs (θα πρέπει να φορτωθεί χωρίς κανένα πρόβλημα).

Disclaimer

- Καλό είναι πριν ξεκινήσετε με την εργασία να έχετε κατανοήσει αρκετά καλά τη θεωρία του λ-λογισμού (ελεύθερες μεταβλητές , αντικατάσταση , μετατροπές , κανονικές μορφές , normal order reduction).

- Ουσιαστικά θα υλοποιήσετε αυτά που θα έχετε μελετήσει σε θεωρητικό επίπεδο για το λ-λογισμό

- Μιας και η εργασία είναι μέσα στην εξεταστική το διάβασμα σας για την τελική εξέταση έχει μια αρκετά μεγάλη επικάλυψη με την παρούσα εργασία

- Οι σημειώσεις του μαθήματος για το λ-λογισμό αποτελούν άριστη πηγή πληροφόρησης για τα πλαίσια της εργασίας

Καλά όλα αυτά αλλά από Haskell τι πρέπει να ξέρω για την εργασία ?

- Στην εκφώνηση της άσκησης υπάρχουν σύνδεσμοι σε ορισμένα πράγματα που θα χρειαστείτε για την εργασία , τα οποία όμως δεν απέχουν πολύ σε δυσκολία από όσα έχετε διδαχθεί στο μάθημα (θα χρειαστεί **guards , pattern matching** και **case expressions**)

1. Guards - http://learnyouahaskell.com/syntax-in-functions#guards-guards
2. **Case Expressions** - http://learnyouahaskell.com/syntax-in-functions#case-expressions
3. Case Expressions - http://stackoverflow.com/a/2226292
4. Where - http://learnyouahaskell.com/syntax-in-functions#where
5. Let - http://learnyouahaskell.com/syntax-in-functions#let-it-be
6. Maybe - https://hackage.haskell.org/package/base-4.8.1.0/docs/Data-Maybe.html
7. Either - https://hackage.haskell.org/package/base-4.8.1.0/docs/Data-Either.html
8. Tuples - https://en.wikibooks.org/wiki/Haskell/Lists_and_tuples#Tuples
9. Algebraic data types - http://www.seas.upenn.edu/~cis194/lectures/03-ADTs.html