Σχεδίαση και Υλοποίηση της Γλώσσας Προγραμματισμού lambda-cases

Δημήτρης Σαριδάκης Μπίτος

Εθνικό Μετσόβιο Πολυτεχνείο

7 Ιουλίου 2024

Μ′ αρέσει πολύ η Haskell

Τα πάντα είναι τιμές (όροι) και έχουν κάποιο τύπο:

- Σταθερές
- Συναρτήσεις
- Είσοδος/Έξοδος
  - Άρα μπορούν να είναι ορίσματα συναρτήσεων, στοιχεία λίστας κτλ

Οι τύποι τα λένε όλα.

Βοηθητικός μεταγλωττιστής:

- Μεταγλωττίζεται; Δουλεύει! (Συνήθως)
- Δεν μεταγλωττίζεται; Οι τάδε τύποι δεν ταιριάζουν.

Γιατί να γράψω κώδικα σε άλλη γλώσσα;

Γιατί δεν είναι η πιο διαδεδομένη γλώσσα;

Building εργαλεία όχι τόσο καλά:

- Φαίνεται να υπάρχει βελτίωση (απ'όσο λένε online)
- Δεν αφορά την διπλωματική

Δύσκολη στην εκμάθηση για αρχάριο. Ίσως παίζουν ρόλο:

- Όχι πολύ περιγραφικές λέξεις κλειδιά
- Όχι πολύ περιγραφικά ονόματα βασικών συναρτήσεων
- Γραμματική λάμδα λογισμού

Τι θα άλλαζα για μένα;

Μπορούν να συμπτυχθούν κομμάτια που γράφω πολύ συχνά;

- Ορισμοί Τιμών
- LambdaCase extension

Μπορούν να αλλάξουν κομμάτια ώστε να μοιάζουν περισσότερο στα αντίστοιχα άλλων γλωσσών όπου είναι πιο κατανοητά;

- Τελεία για attributes/members/fields
- Εφαρμογή συνάρτησης με ορίσματα σε παρένθεση

Υπάρχει κάτι καινούργιο που θα μπορούσα να προσθέσω;

- Ορίσματα στην αρχή ή στην μέση του ονόματος της συνάρτησης
- Ανώνυμες Παράμετροι
- Τύποι Δύναμης

Εφαρμογή Συνάρτησης Με Παρενθέσεις

Haskell	Icases
`f x`	`f(x)`
`g x y z`	`g(x, y, z)`
`putStrLn "Hello World!"`	`print("Hello World!")`

Ορίσματα πριν ή στην μέση:

`show x`	`(x)to_string`
`mod x y`	`(x)mod(y)`
`map f l`	`apply(f)to_all_in(l)`

Ανώνυμες Παράμετροι

Σε οποιαδήποτε συνάρτηση μπορούν να λείπουν οποιαδήποτε από τα ορίσματα: κάτω παύλα.

Τα υπόλοιπα είναι παράμετροι.

Νέα συνάρτηση με είσοδο τα κενά ορίσματα.

`f(_, 1.61, 42)`	`x => f(x, 1.61, 42)`
`f(3.14, _, 42)`	`x => f(3.14, x, 42)`
`f(_, 1.61, _)`	`(x, y) => f(x, 1.61, y)`

Ανώνυμες Παράμετροι

```
greetings : ListOf(String)s
  = ["hey!", "hello!", "hi!"]

length_of(_) : String => Int

apply(_)to_all_in(_)
  : (T1 => T2) x ListOf(T1)s => ListOf(T2)s



apply(length_of(_))to_all_in(_)
  : ListOf(String)s => ListOf(Int)s

apply(_)to_all_in(greetings)
  : (String => T1) => ListOf(T1)s

>>> apply(length_of(_))to_all_in(greetings)
  : ListOf(Int)s
  = [4, 6, 3]
```

Το ίδιο σε Haskell

```
greetings :: [String]
greetings = ["hey!", "hello!", "hi!"]

length :: String => Int

map :: (a -> b) -> [a] -> [b]


map length :: [String] -> [Int]

flip map :: [a] -> (a -> b) -> [b]

flip map greetings :: (String -> a) -> [a]

>>> map length greetings
  = [4, 6, 3]
```

Ανώνυμες Παράμετροι: tuples και λίστες

Αντίστοιχα μπορούμε να αφήσουμε κενά στοιχεία tuple ή λίστας.

Νέα συνάρτηση με είσοδο τα κενά στοιχεία.

```
(42, _) : T1 => Int x T1
```

```
(_, 3.14, _) : T1 x T2 => T1 x Real x T2
```

```
[42, _] : Int => ListOf(Int)s
```

```
[_, 3.14, _] : Real^2 => ListOf(Real)s
```

Αντίστοιχα σε εκφράσεις τελεστών: παρακάτω.

Ορισμοί tuple_type και postfix functions

tuple_type αντίστοιχα:

- structs σε C
- classes σε OOP: μόνο attributes
- records σε Haskell

Δημιουργείται αυτόματα ένα postfix function για κάθε field:

- Κατευθείαν με όρισμα: `some_person.last_name`
- Συνάρτηση Μόνη της: `_.last_name`

Ορισμοί tuple_type και postfix functions

```
tuple_type Name
value (first_name, last_name) : String^2

awesome_guy: Name
  = ("Leonhard", "Euler")

>>> awesome_guy.last_name
  : String
  = "Euler"

>>> _.last_name
  : Name => String

>>> apply(_.last_name)to_all_in(_)
  : ListOf(Name)s => ListOf(String)s
```

postfix functions για tuples που έχουν τύπο γινόμενο

"_.1st", "_.2nd", "_.3rd", ... για tuples που έχουν τύπο γινόμενο.

```
tuple : Real x String
  = (1.618, "golden ratio")

origin : Real^3
  = (0, 0, 0)

>>> tuple.2nd
  = "golden ratio"

>>> origin.2nd
  = 0
```

".change" postfix function

Συναρτήση αλλαγής στοιχείων tuple

```
state.change{counter = counter + 1}
point.change{z = 2.718}

tuples : ListOf(Int^2)s
  = [(1, 2), (3, 4), (5, 6)]
>>> apply(_.change{1st = 1st + 1})to_all_in(tuples)
  = [(2, 2), (4, 4), (6, 6)]

name : Name
  = ("Jacob", "Bernoulli")
change_first_name_to(_) : String => Name
  = name.change{first_name = _}
>>> change_first_name_to("Daniel")
  = ("Daniel", "Bernoulli")
```

Ορισμοί or_type και prefix functions

or_type αντίστοιχα:

- C, C++ κτλ: enum types αλλά πιο γενικά
- Haskell: τα πάντα είναι data, συμπεριλαμβάνουν tuple types, or types

Οι τιμές τους χωρίζονται σε περιπτώσεις (με ή χωρίς εσωτερικές τιμές).

Δημιουργείται αυτόματα ένα prefix function για κάθε περίπτωση με εσωτερική τιμή:

- Κατευθείαν με όρισμα: `the_value:1`
- Συνάρτηση Μόνη της: `the_value:_`

Pattern matching με συνάρτηση "cases".

Ορισμοί or_type και prefix functions

```
or_type Bool
values true | false

or_type Possibly(T1)
values the_value:T1 | no_value

or_type Result(T1)OrError(T2)
values result:T1 | error:T2

>>> the_value:1
  : Possibly(Int)
>>> the_value:_
  : T1 => Possibly(T1)
>>> result:1
  : Result(Int)OrError(T1)
```

Τελεστές: Εφαρμογής Συνάρτησης

Τελεστές: Σύνθεσης Συναρτήσεων

Τελεστές: Αριθμητικοί

Τελεστές: Σχεσιακοί και Λογικοί

Τελεστές: Περιβάλλοντος

Εκφράσεις Συναρτήσεων

Εκφράσεις Συναρτήσεων "cases"

pattern matching

LambdaCase extension

Ορισμοί Τιμών

Σύγκριση με Haskell

Εκφράσεις "where"

Παραδείγματα

Τύποι

Αντιστοιχία με Haskell

Πατσούκλια Τύπων

type στην Haskell

Παραδείγματα

Λογική Τύπων

Μηχανισμός ad hoc πολυμορφισμού στην lcases.

Αντιστοιχοί στα type classes.

Ορισμοί Προτάσεων Τύπων

Ατομικές, class

Μετονομασίας

Θεωρήματα Τύπων

instance

Υλοποίηση Parser

Βιβλιοθήκη Parsec

Μετάφραση σε Haskell

Συμπεράσματα

Τι έχει γίνει

Τι θα ήταν καλό να γίνει