Advanced Functional Programming

Boris Düdder

16. juli 2019

Assignment 1-4 are *individual* assignments and Assignment 5 and 6 are group assignments, where a group consists of two students. You will receive the assignment descriptions for Assignment 4-6 according to the schedule. We expect timely submission of the assignments, late assignments count as failed assignments. We expect genuine solutions and plagiarism will lead directly to fail. Please read the submission guide for assignments 4-6.

1 Assignment

Perform β reduction of the following terms and write the β-contractum: *(Lambda calculus slides, p. 21.)*

1. $\lambda x.x$
2. $(\lambda x.x)y$
3. $(\lambda x.xx)y$
4. $(\lambda x.z)y$
5. $(\lambda x \lambda y.x)z$

2 Assignment

Does the following terms have a β Normalform? Please briefly explain why. β Normalforms are presented in *(Lambda calculus slides, p. 21.)*

1. $\mathbf{I}= \lambda x.x$
2. $\Omega = \omega\omega$ with $\omega = \lambda x.xx$
3. $\mathbf{KI}\Omega$ with $\mathbf{K} = \lambda x\ y.x$
4. $(\lambda x.\mathbf{KI}(x\ x)\lambda y.\mathbf{KI}(y\ y))$
5. $(\lambda x.z(xx))\lambda y.z(yy)$

3 Assignment

A reduction path of a λ term M is a finite or infinite sequence of the following form:

$$M \to_\beta M_1 \to_\beta M_2 \to_\beta \dots$$

We call a term weak normalizing, if it has a Normalform. A term is called strong normalizing, if all of its reduction paths end in a Normalform.

1. Which of the previous five terms are weak normalizing and which of them are strong normalizing?
2. In which of these cases does the term end in different Normalforms?

4 Assignment

Our calculator is missing the functionality to operate with predefined constants, e.g. `pi`. Please extend the calculator from the *Calculator* example, by these functionalities.

1. Extend the discriminated union type `Expr` by defining a type called `Var` which is a `string`.
2. Implement the function `memory`, which should return a `Map<string,float>`. Please include the keys and values: `"pi", 3.14152976`, `"x", 10.0`.
3. Implement the function `lookup`, which takes a key `x` and a storage `stor` of type `Map<string,ConstType>` as arguments and returns the value of key `x` from storage `stor`. If the key it not contained in `stor` then it should fail with an error message.
4. Extend the function `eval` to evaluate variables of type `Var`, such that it performs a lookup on a key in the storage.
5. Run the program and test that your output is correct.

5 Assignment

This assignment is about implementing an interpreter for terms of untyped λ calculus as presented in *(Lambda calculus slides)*. We will use various simplifications:

- We ignore all the α equivalence, as presented *(Lambda calculus slides, p. 16)*.
- We ignore handling free variables and potential collisions *(Lambda calculus slides, p. 19)*.

In order to implement the interpreter follow the steps below.

1. Implement the Embedded Domain Specific Language, `Term`, for the grammar *pre-terms* presented in *(Lambda calculus slides, p. 8.)*.
2. Implement the function `substitute` with the signature: `let rec substitute (term : Term) (var : VarType) (sub : Term)`. The function `substitute` should perform the substitution that is defined in *(Lambda calculus slides, p. 20)*. Ignore the free variable side condition in case 4.
3. Implement the function `reduce` with the signature: `let rec reduce ((term,red) : (Term * bool))`. The function `reduce` should perform the β reduction that is defined in *(Lambda calculus slides, p. 21)*.

   Hints: (1) the application case should be handled by recursive descend in both arguments. (2) Use the second member of the tuple, `red`, to indicate that substitution has been performed.
4. Implement the function `eval` with the signature: `let rec eval (term : Term)`. The function `eval` should perform a β reduction, implemented in `reduce`, until a Normalform is reached. The β Normalform is defined in *(Lambda calculus slides, p. 21)*.

   Hint: a Normalform is reached if no further substitution can be performed. Use the boolean variable `red` for this check.
5. Run the program and test that your output is correct.