Advanced Functional Programming

Boris Düdder

17. 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:

$$M \rightarrow_\beta M_1 \rightarrow_\beta M_2 \rightarrow_\beta \ldots$$

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.

6 Assignment

For those of you who didn't participate in the course *Introduction to Functional Programming*. H.C. Andersen (1805-1875) is a Danish author who wrote plays, travelogues, novels, poems, but perhaps is best known for his fairy tales. An example is Little Claus and Big Claus (Danish: Lille Claus og store Claus), which is a tale about a poor farmer, who outsmarts a rich farmer. A translation can be found here: `http://andersen.sdu.dk/vaerk/hersholt/LittleClausAndBigClaus_e.html`. It starts like this:

> "LITTLE CLAUS AND BIG CLAUS a translation of Hans Christian Andersen's 'Lille Claus og Store Claus' by Jean Hersholt. In a village there lived two men who had the selfsame name. Both were named Claus. But one of them owned four horses, and the other owned only one horse; so to distinguish between them people called the man who had four horses Big Claus, and the man who had only one horse Little Claus. Now I'll tell you what happened to these two, for this is a true story".

The assignment is to design a spell checker that is based on the text by H.C. Andersen. We will hand out a file `assignment6.fsx`, which you should fill out.

1. Please read and understand the data structure Trie that has been handed out in `assignment6.fsx` and understand how a word tree is created in this Trie and how the operations are performed.
2. We need to create an `autoComplete` functionality based on a lookup of a specific word in the trie. This function `lookup` should have the signature `let lookup (prefix: string) (trie: Trie<char>) : Trie<char> Option`. It should return an option of the (sub)trie found by the lookup. `autoComplete` will use `lookup` to check if the word beginning with the `prefix` exists, after which it should return a sequence of strings of words which `prefix` in the trie. The signature of `autoComplete` is `let autoComplete (prefix: string) (trie: Trie<char>) : string seq`.
3. Implement `spellCheck` with the signature `let spellCheck (word: string) (trie: Trie<char>) : bool` with the functionality to check if a given word is contained in the trie.
4. Implement `genText` with the signature `let genText (len: int) (trie: Trie<char>) : string` which generates a string text of length `len` of random words that are generated by the function `randWord` with the signature `let randWord (trie: Trie<char>) : char list`. `randWord` should retrieve a random word from the given trie.
5. Test your implementation with the given tests and, potentially, add extra tests.

7 Submission Details of Assignment 5 and 6

Please hand in one *zip* file, where the name of the zip file and the folder is each group member's student ID, example 20191234567_20197654321. The zip file should contain one single *.fsx* solution file and one report as *pdf* document, the report should have the same name as the zip file and folder. Please include all files such that the solution is runnable from the unzipped folder. Please exclude *bin* and other folders.

Hand in the zip file by email to *sahy@di.ku.dk*, no later than 8:00 PM. You must send your hand in using your UESTC email, because it will not be received otherwise. Assignment 5 must be handed in 2019-07-17 and Assignment 6 must be handed in 2019-07-19. The subject of the email must be the same as the naming of the zip file, folder, report and source code, example 20191234567_20197654321.

We expect the report to follow the guidelines that are available on the server. Assignment 5 must be no more than 2 pages. Assignment 6 must be no more than 5 pages.

8 Tips

Each of the following recommendations will withdraw points, if not followed.

- Make sure you hand in your submission correctly, according to the given requirements.
- Do not exceed the page limit.
- Make sure to include all sections.
- The Analysis and Design should **not** contain program code.
- The Design section should provide rationalization of the design decisions.
- The Program Description should provide a high-level explanation of program constructs and their relations.
- The sections Testing and Experiments, and Conclusion should contain a discussion and clear reflections.