KAZAKH BRITISH TECHNICAL UNIVERSITY

Programming language theory, 2021 srping.
Home work 4

Start date: 23.04.2021
Deadline: 30.04.2021, 9:00
Total weight: 10 points

Introduction

So far, we have implemented few languages and have proved important properties. Let us sum up our notion of syntax and evaluation on the example of **BN** and its typed version (booleans and naturals, with *if then else*, *succ*, *pred*, *iszero*) and **lambdaBN** (functions + BN). To do so, you will be given some ready implemented language in O'Caml. In this homework, you are not going to implement lexers and parsers, don't worry. But to finish up this HW successfully, we need you to understand the concept of evaluation and show some practical skills in O'Caml.

Task 0

Step 0. https://try.ocamlpro.com/ - recommend to complete tutorial for O'Caml. Alternatively, https://ocaml.org/learn/tutorials/ this you might want to cover.

Step 1. Obtain ***simplebool*** archives via this link. These are the implementations of lambda-**BN** (without naturals)**.** It is complete implementation described in Ch. 10. Omit the parsers and lexers, main code is written in files *syntax.ml and core.ml.*
Run it with command *make.* This will compile the programm into executable *f.* In the *test.f* file there are examples of input strings.
Here are some notes to understand implementation:

a) **.ml* files contain code, *.mli cotain declarations (same as .cpp and .h files in C++).
b) *syntax.ml* – ignore *info* entries, they are needed for parsers. You can think of *TmSmth* as nodes in derivation of that term. Also, *printtm_ATerm* deserve a look. This is how evaluated term is printed for you.
c) *core.ml* – main function here is *eval1* – it produces small evaluation step by the rules described in our semantics. *typeof* function is just provides algorithm for type derivation of a term. Note that *typeof* isn't evaluating terms.

Task 1

Answer questions:

1. What is the output for test "if succ(0) then true else (if true the succ(2) else succ(3));" in **untyped** implementation? Why? Why term in brackets won't reduce?
2. Perform same for **typed** implementation and explain result.
3. What evaluation strategy is implemented? Reason it, show how you understood it.
4. *eval* function is implemented as repeatedly calling *eval1* of term. How it knows when it reached the normal form (value or stuck state)?
5. Exc 8.3.5.
6. Exc 8.3.6.

Note, that function *typeof* actually plays role of *typechecker,* and we proved already on the lectures, that well typed terms are actually can be evaluated to value in SAFE systems.

Format: list of answers line by line in report.

Task 2

Now you are to finally code some part of our lovely system. Take a look at exercise 3.5.16 from one of the previous homeworks.

Task: Add a new syntactic element *wrong*. Modify implementation, so that stuck terms will return wrong statement. Note, you may be needed to rewrite **typeof** function as well.

Format: in your PDF report you should describe what you've done. When submitting your homework you should include archive of codebase (don't include report into the archive).