Homework: Syntax and Semantics with References

Due-date: Apr 06 at 11:59pm
Submit online on Blackboard LS

Homework must be individual's original work. Collaborations and of any form with any students or other faculty members are not allowed. If you have any questions and/or concerns, post them on Piazza and/or ask our TA or me.

Learning Outcomes

- Knowledge and application of Functional Programming
- Ability to understand syntax specification
- Ability to understand the impact of references in programming languages
- Ability to design software following requirement specifications (operational semantics)

Questions

Consider the grammar G of a language $\mathcal{L}$, where $G = (\Sigma, V, S, P)$ such that

- Σ is a set of terminals: anything that does not appear on the left-side of the product rules P presented below
- V is the set of non-terminals appearing in the left-side of the production rules P presented below

```
Program      -> Expr
Expr         -> Number | Variable | OpExpr | FExpr | ApplyF
                |DRef | WRef | Ref
OpExpr       -> ArithExpr | CondExpr | VarExpr
ArithExpr    -> (Op Expr Expr)
Op           -> + | -  | * | /
CondExpr     -> (CCond Expr Expr)
CCond        -> BCond | (or CCond CCond) | (and CCond CCond) | (not CCond)
BCond        -> (gt Expr Expr) | (lt Expr Expr) | (eq Expr Expr)
VarExpr      -> (var VarAssign Expr)
VarAssign    -> (VarAssignSeq)
VarAssignSeq -> (Variable Expr) | (Variable Expr) VarAssignSeq
Variable     -> symbol
FExpr           -> (fun FAssign Expr)
FAssign         -> ((FName FormalParams) Expr)
FormalParams    -> () | (FormalParamList)
FormalParamList -> Variable | Variable FormalParamList
ApplyF          -> (apply (FName Args))
Args            -> () | (ArgsList)
ArgList         -> Expr | Expr ArgList
```

```
FName            -> symbol
DRef       -> (deref Expr)
WRef       -> (wref Expr Expr)
Ref        -> (ref Expr) | (free Expr)
```

- $S =$ `Program`

  Write a function `eval` which takes as input a syntactically correct program as per the above grammar (that also satisfies the meta-syntax rules of Homework assignment 4), an environment, a heap and computes the semantics (a pair containing the result of evaluation along with the resultant heap) of the expression. The environment is a list of variable-value pairs and the heap is a list of location-value pairs. Follow the operational semantics discussed in class (Note that we are still using the VarExpr from Homework 3). Comment your code to show how the operational semantics is encoded in your implementation.

  (a) Assume that the evaluations of expressions in `deref`, `wref`, `free` and `ref` do not depend on the heap. For example, `(deref (ref 10))` will not be present in your program.

  (b) As discussed in class, there are several situations when the evaluation can end up in an exception. In any expression, if evaluation of any subexpression results in an exception, then the evaluation of the expression also results in an exception. For instance, the evaluation of the addition of two expression (operands) results in exception, if the evaluation of one of the operands results in exception. The following exceptions should be considered:

  i. reading from an address corresponding to a free memory location: `'(exception fma)`

  ii. reading from an address that does not correspond to a memory location: `'(exception ooma)`

  iii. there exists no address corresponding to free memory location: `'(exception oom)`

  If the evaluation of a program results in any of the above exception, the eval should return the exception along with the resultant heap. For instance:

```
(define prog1
  '(var ((x (deref 1))) (+ x 1)))
;; (eval prog1 '() '((1 free))) will result in
;; '((exception fma) ((1 free)))

(define prog2
  '(var ((x (ref 32)))
        (var ((y (+ x 1)))
             (deref y))))
;; (eval prog2 '() '((1 free))) will result in
;; '((exception ooma) ((1 32)))
```

Organization and Submission Guidelines You will have a file named `program.rkt`, which will contain the **input programs**.

You will write the functions necessary for this assignment in a file named `<netid>.rkt`. At the top of this file, you must include

```
#lang racket
(require "program.rkt")
(provide (all-defined-out))
```

Put the `program.rkt` and `<netid>.rkt` in the same folder; which will allow you to access the definitions of the input program and the heap definition easily.

You are required to submit the `<netid>.rkt` file only on the blackboard. If you netid is asterix, then the name of the file should be asterix.rkt. You must follow the instructions for submission. For instance, do not submit the file with any other extension, do not submit any other file, do not submit any zip files, remove/comment any inclusion of test code including trace directives that get automatically executed when your code is tested for evaluation, remove incomplete code.

In terms of the racket language features you can use, the same rules continue to apply for this assignment as well: list, numbers, boolean and typical operations over them, if-then-else and cond. If you have doubts, please post/ask.

If you do not follow the requirements or submission guidelines of the assignment, then your submission may not graded. If you have doubts about the instruction, please post/ask.