Homework: Syntax and Semantics with Functions

Due-date: Mar 20 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 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
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
```

- $S =$ `Program`

1. Write a function `synchk`, which takes as input an program and returns true if and only if the following holds

   - the program can be generated by the above grammar. (To check for a variable, use the Racket command `symbol?`.)
   - the program satisfies two meta-syntax rules described below.
     (a) If the program contains an `ApplyF` expression for some function that is not enclosed in the corresponding `FExpr` for the same function, then the `synchk` function returns false. (see prog1 below)
     (b) If the program contains an `ApplyF` expression for some function with n number of arguments but the enclosing `FExpr` for the same function has $m \neq n$ number of formal parameters, then the `synchk` function returns false. (see prog2 below)

2. Write a function `eval` which takes as input a program for which `synchk` returns true, an environment and returns the semantics of program (i.e., evaluates the program). Follow the operational semantics of static scoping discussed in class. You can assume all variables in the program are either bound inside the program via `varExpr` or their mapping to values are present in the initial environment.

   Comment your code to show how the operational semantics is encoded in your implementation.

Organization and Submission Guidelines You will have a file named `program.rkt`, which will contain the input programs. Here are same sample programs:

```
#lang racket
(provide (all-defined-out))

;; (synchk prog1) returns false (explanation for 1a)
;; f is applied but there is not enclosing fun f
(define prog1
  '(fun ((f1 (x)) ((gt x 0)
                   (* x (apply (f ((- x 1)))))
                   1))
        (apply (f1 (x)))))

;; (synchk prog2) returns false (explanation for 1b)
;; f1 is applied with incorrect number of actual arguments
(define prog2
  '(fun ((f1 (x)) ((gt x 0)
                   (* x (apply (f1 ((- x 1)))))
                   1))
        (apply (f1 ()))))
```

```
;; (eval prog3 '((x 3))) returns 6
;; (eval prog3 '((x 4))) returns 24
(define prog3
  '(fun ((f1 (x)) ((gt x 0)
                   (* x (apply (f1 ((- x 1)))))
                   1))
         (apply (f1 (x)))))
```

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))
```

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