Homework: Syntax and Semantics with Variables

Due-date: Feb 24 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
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
```

- $S =$ `Program`

Note the usage of `VarExpr`, which is different from the one we discussed in class—the above grammar allows multiple variable assignments under the same "var"-directive. The semantics is given as follows:

$$[\![\texttt{(var ((Variable Expr1)) Expr2}]\!]_\sigma = [\![\texttt{Expr2}]\!]_{\texttt{(Variable }[\![\texttt{Expr1}]\!]_\sigma\texttt{)} \circ \sigma}$$

$$[\![\texttt{(var ((Variable Expr1) VarAssignSeq) Expr2}]\!]_\sigma = [\![\texttt{(var (VarAssignSeq) Expr2)}]\!]_{\texttt{(Variable }[\![\texttt{Expr1}]\!]_\sigma\texttt{)} \circ \sigma}$$

1. Write a function, `synchk` which takes as input an program and returns true if and only if the program can be generated by the above grammar. To check for a variable, use the racket command `symbol?`.

2. Write a function `eval` which takes as input a program, an environment and returns the semantics of the expression. Follow the operational semantics and its environment-based encoding discussed in class. If the program contains any *free* variable, then the `eval` function returns `'(Cannot Evaluate)`. 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 is a sample program file:

```
;; This is for eval
(define prog2
  '(var ((x z) (y (+ x 1)))
        (+ x y)
   )
)

;; (eval prog2 '((z 10))) will return 21
;; (eval prog2 '()) will return '(Cannot Evaluate)
```

You will write the functions necessary for this assignment in a separate file `<netid>hw3.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>hw3.rkt` in the same folder; which will allow you to access the definitions of the input program easily. Comment your code–what each function (including the helpers you write) is realizing.

You are required to submit only the `<netid>hw3.rkt` file on the blackboard.

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.