Homework: Syntax and Semantics of Programming Language with Variables

Due-date: Mar 13 at 11:59pm
Submit online on Canvas

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 342 instructor or TAs.

Learning Outcomes

- Knowledge and application of Functional Programming
- Ability to understand grammar 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      -> (SSeq)
SSeq         -> Statement | Statement SSeq
Statement    -> Decl | Assign | If | While
Decl         -> (decl Var)
Assign       -> (assign Var ArithExpr)
If           -> (if CondExpr (SSeq))
While        -> (while CondExpr (SSeq))

ArithExpr    -> Number | Var | (Op ArithExpr ArithExpr)
Op           -> + | -  | * | /
CondExpr     -> BCond | (or CondExpr CondExpr) |
                        (and CondExpr CondExpr) | (not CondExpr)
BCond        -> (gt ArithExpr ArithExpr) | (lt ArithExpr ArithExpr) |
                (eq ArithExpr ArithExpr)
Var          -> symbol
```

- $S =$ `Program`

1. Write a function, `synchk` which takes as input an program and returns true if and only if the program belongs to $\mathcal{L}$ of the above grammar. To check for a variable, use the racket command `symbol?`.

2. Write a function `sem` which takes as input a syntactically correct program as per the above grammar, an environment and computes the semantics, which is another environment. The environment is a list of variable-value pairs captured in Racket as follows:
   `'((x 0) (y 10) (z 20))` is an environment containing the valuations of variables `x`, `y` and `z`.

   If a variable is used in an expression and is not declared before being used, then you can assume that the input environment will include a value information for that variable.

   The operational semantics of arithmetic and conditional expressions are already presented in class. The meaning of the other constructs are enumerated informally below. You will develop the operational semantics from the following, and use it to develop the implementation of `sem`.

   (a) Block Context: A valid program as per the above grammar consists of at least one block; outermost block being the sequence of statements in the program itself.

   The function `sem` *outputs the variable-value information of the input environment and variable-value information for all variables in the outer scope.*

   The statement sequence inside an if-statement and a while-statement result a new nested block context.

   For instance,

```
(
 (decl x)
 (assign x 3)
 (decl y)
 (assign y 10)
 (if (gt x 2)
   (
     (decl x)
     (assign x y)
     (assign x (+ x y))
   )
 )
 (assign x (+ x 1))
)
```

   There are two block contexts, the outermost context consists of all statements of the program. The statements

```
  (
     (decl x)
     (assign x y)
     (assign x (+ x y))
  )
```

result in a context nested inside the outermost context. There can be multiple nestings depending on the nesting depth of the if-statements, while-statements and the number of times a while-statement is unfolded.

(b) The variable-value pair stored in the environment keeps track of the contexts in the environment. The specific strategy for keeping track of the context is your choice. For instance, in the above program the semantics of `x` is 10 in `(+ x y)` in the rhs of the statement `(assign x (+ x y))`.

(c) Semantics of a statement in the context of an environment e is the a new environment e' resulting from the updates to the variable-values in e or addition of new variable-value pairs to e.

(d) Semantics of sequence of statements $(s_1\ s_2 \ldots s_n)$ in the context of an environment e_0 is a new environment e_n, where for $i \in [1, n]$, semantics of s_i in the context of e_{i-1} is e_i.

(e) Semantics of a declaration statement for a variable x, adds the fact that a new variable x is initialized to 0 to the environment.

You can assume that a syntactically correct program will not have two declarations with identical variable name within the same context. For instance, in this assignment you will never have programs of the form:

```
(
  (decl x)
  (decl x)
)
```

or

```
(
  (decl x)
  (if (gt x 1)
     (
      (assign x 10)
     )
  )
  (decl x)
)
```

or

```
(
  (decl x)
  (if (gt x 1)
      (
       (decl y)
       (assign x y)
```

```
          (decl y)
        )
    )
```

(f) Semantics of an assignment statement `(assign x Expr)` in environment e, evaluates the expression `Expr` and updates the valuation of `x` in e, that is in the context of the statement.

(g) Semantics of if-statement `(if CondExpr (SSeq))` in environment e (variable-value pairs of the context inside which if-statement is present) considers the valuation of the conditional expression `CondExpr` in the environment e. If the expression evaluates to false, then the resultant semantics is e.

If, on the other hand, the expression evaluates to true, then the semantics of the sequence of statements `(SSeq)` in environment e is computed. Note that, the semantics of the if-statement returns the environment containing the variable-value pairs for the context in which if-statement is present.

(h) Semantics of while-statement `(while CondExpr (SSeq))` in environment e (variable-value pairs of the context inside which while-statement is present) considers the valuation of the conditional expression `CondExpr` in the environment e. If the expression evaluates to false, then the resultant semantics is e.

If, on the other hand, the expression evaluates to true, then the semantics of the sequence of statements `(SSeq)` in the environment e is computed, resulting in a new environment e', which is the variable-value pairs for the context in which while-statement is present. The above steps starting from the evaluation of `CondExpr` in e'. The process terminates only when the `CondExpr` evaluates to false.

Programming Rules

- You are required to submit one file hw4-⟨net-id⟩.rkt[1]. The file must start with the following.

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

In the above, the program.rkt will be used as an input file for our test programs. It will contains programs of the form:

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

(define program1
  '(
```

[1]Your netid is your email-id and please remove the angle brackets, they are there to delimit the variable net-id.

```
    (decl x)
    (assign x 3)
    (decl y)
    (assign y 10)
    (if (gt x 2)
        (
         (decl x)
         (assign x y)
         (assign x (+ x y))
        )
    )
    (assign x (+ x 1))
    )
  )
```

- You are **only allowed** to use functions, if-then-else, cond, list operations, operations on numbers. No imperative-style constructs, such as begin-end or explicity variable assignments, such as get/set are allowed. If you do not follow the guidelines, your submission will not be graded. If you are in doubt that you are using some construct that may violate rules, please contact instructor/TA (post on Piazza).
- You are expected to test your code extensively. If your implementation fails any assessment test, then points will be deducted. Almost correct is equivalent to incorrect solution for which partial credits, if any, will depend only on the number of assessment tests it successfully passes.
- The focus is on correct implementation. In this assignment, we will not assess the efficiency; however, avoid re-computations and use tail-recursion, if possible.

  Here are few example programs to get you started with testing.

```
(define program1
  '(
    (decl x)
    (assign x 3)
    (decl y)
    (assign y 10)
    (if (gt x 2)
        (
         (decl x)
         (assign x y)
         (assign x (+ x y))
        )
    )
    (assign x (+ x 1))
    )
  )


(define program2
 '(
    (decl x)
    (decl z)
    (assign x (+ y 1))
    (if (gt x 1)
        ((assign z 1))
    )
    (if (gt x 2)
        ((assign z 2))
    )
```

The following are the expected results for the functions you are going to implement.

```
> (synchk program1)
#t

> (synchk program2)
#t

> (sem program1 '())
'((y 10) (x 4))

> (sem program1 '((x 20))
'((y 10) (x 4) (x 20)) ;; inputs do not impact the semantics
                       ;; input environment information is
                       ;; not removed

> (sem program2 '((y 10)))
'((x 11) (z 2) (y 10))

> (sem program2 '((y 0)))
'((x 1) (z 0) (y 0))
```

Guidelines

1. Review and learn about operational semantics. It is important to understand how operational semantics is represented and implemented. Unless you write some on your own, this homework (and subsequent ones) is likely to be difficult.

2. In class, we have developed a language and discussed the implementation of operational semantics of that language. Make sure you understand that before proceeding.

3. Learn how to use Racket. If you have not written some definitions in Racket and have not reviewed the solutions to any/some exercises, then it is likely to be difficult to complete this assignment.

4. Starting two days before the deadline to do the above is likely to make it impossible for you to complete this assignment.

5. As always, map out the functions you need to write, write comments that include the specification of the functions you are writing, test each function extensively before using it in another function.