CSCI3136
Assignment 7

Instructor: Alex Brodsky

Due: 3:00pm, Friday, March 17, 2014

In this assignment we will expand the Scheme Evaluator to handle variables. We will first need to change the grammar slightly as indicated in Figure 1. The only difference from the previous

$$
\begin{array}{rcl}
S & \rightarrow & \mathit{Atoms} \\
\mathit{Atoms} & \rightarrow & \epsilon \\
\mathit{Atoms} & \rightarrow & \mathit{Atom}\ \mathit{Atoms} \\
\mathit{Atom} & \rightarrow & '\ \mathit{Atom} \\
\mathit{Atom} & \rightarrow & \mathit{List} \\
\mathit{Atom} & \rightarrow & \texttt{id} \\
\mathit{Atom} & \rightarrow & \texttt{int} \\
\mathit{List} & \rightarrow & (\ \mathit{ListBody}\ ) \\
\mathit{ListBody} & \rightarrow & \mathit{Atoms} \\
\mathit{ListBody} & \rightarrow & \mathit{Proc}\ \mathit{Atoms} \\
\mathit{Proc} & \rightarrow & \texttt{let} \\
\mathit{Proc} & \rightarrow & \texttt{define}
\end{array}
$$

Figure 1: A less simplified grammar for Scheme.

assignment is that a *List* now consists of a *ListBody* that can expand to either *Atoms* or *Proc Atoms* where *Proc* expands to the terminals `let` or `define`, which are the two new terminals in our grammar.

Both `define` and `let` allow us to bind variable names *id*s to values. The syntax for `define` is

(`define` *id atom)*

For example, the expressions

```
( define foo 42 )
( define bar ( * 7 13 ) )
```

bind `foo` to the value 42 and bind `bar` to 91, which is the evaluation of `( * 7 13 )`. Specifically, `define` takes two parameters, an *id*, *which is not evaluated*, and an expression, whose evaluation is then bound to the *id*. The evaluation of a `define` expression adds the binding to the current reference envirnonment and yields the empty string `""`.

The syntax for `let` is

(`let` *((id atom) ...) atom ...)*

where the `...` denotes one or more of the preceding item. For example, the expressions:

```
( let ( ( x 42 ) )
   ( + x 13 )
)
( let ( ( x 42 ) ( y 13 ) )
   ( + x y )
   ( let ( ( z 7 ) )
      ( + x 13 )
      ( - x y z )
   )
)
( let ( ( x ( * 2 21 ) ) )
   ( let ( ( x 7 ) )
      ( + x 13 )
   )
   ( + x x )
)
```

evaluate to 55, 22, and 84, respectively. Specifically, `let` takes 2 or more arguments. The first argument is a list of bindings where each binding consists of an *id* and an expression. The remaining arguments are expressions that are evaluated using the bindings defined in the first argument. The evaluation of the `let` expression is the evaluation of the last argument. The bindings are only active inside the `let` expression.

In the example above, the first `let` expression binds x to 42 and then evaluates `( + x 13 )` to yield 55.

The second `let` expression binds x to 42 and y to 13, evaluates `( + x y )` and then evaluates the next expression, which is also a `let` expression. This expression binds z to 7 and then evaluates the expressions `( + x 13 )` and `( - x y z )`. The evaluation of the latter expression is also the evaluation of the overall expression.

Lastly, the third expression binds x to the evaluation of `( * 2 21 )` and evaluates the next two expression. The first expression is also a `let` expression, which rebinds x to 7, and evaluates `( + x 13 )`. Then, the second expression `( + x x )` is evaluated, using the outer binding of x. Thus, the overall evaluation yields 84.

1. **[10 marks]** A recursive descent parser, called `suscm.py`, (written in Python) that parses and evaluates programs using the above context-free grammar is provided as part of this assignment. However, this parser does not implement `let` or `define`. Furthermore, the evaluation phase has been decoupled from the parsing phase. That is, `suscm.py` first parses the input, creating am list of expressions, and then evaluates each of the expressions in a separate pass.

Consider the Scheme binding operations `define`, `let`, `let*`, and `letrec` that were discussed in class. For each of these operations determine whether it is necessary to separate the parsing and the evaluation into two distinct passes in order to implement each of these operations. Justify each of your answers.

2. **[5 marks]** Suppose we implemented bindings in `suscm.py` by using a single global referencing environment. Would this implementation correctly evaluate Scheme expressions? If yes, justify why. If no, give an example of a Scheme expression that could not be properly evaluated.

3. **[35 marks]** Using the provided evaluator `suscm.py`, or your own version of an evaluator (implemented in a language of your choice) implement the `let` and `define` operations. Note: the provided implementation will parse `let` and `define` expressions, but does not evaluate them. So, all you need to do is modify the evaluation phase of the evaluator (unless you are creating your own).

   A test solution (in the form of a Java class file: `SchemeEval2.class`) is provided for you to test your new evaluator.

   Since the choice of language is up to you, you must provide a standard script called `runme.sh` to run your parser. See previous assignment for an example.

   To submit this part of the assignment please use SVN as well as in hard-copy. On the hard-copy please note the login id of the person who submitted the programming portion of the assignment for the group. Please see the Resource or Assignment page of the course website for how to use SVN. Remember, you need to include a script called `runme.sh` that will let the marker run your code.

CSCI3136: Assignment 7

Winter 2014

Student Name	Login ID	Student Number	Student Signature

	Mark	
Question 1		/10
Question 2		/5
Question 3		/35
Functionality	/20	
Structure	/15	
Total		**/50**

Comments:

Assignments are due by 3:00pm on the due date before class and must include this cover page. Assignment *must* be submitted into the assignment boxes on the second floor of the Goldberg CS Building (by the elevators).

Plagiarism in assignment answers will not be tolerated. By submitting their answers to this assignment, the authors named above declare that its content is their original work and that they did not use any sources for its preparation other than the class notes, the textbook, and ones explicitly acknowledged in the answers. Any suspected act of plagiarism will be reported to the Facultys Academic Integrity Officer and possibly to the Senate Discipline Committee. The penalty for academic dishonesty may range from failing the course to expulsion from the university, in accordance with Dalhousie Universitys regulations regarding academic integrity.