

Project 3 Overview

Trevor Mee

04/11/2025

Project 3

COP4020 - Programming Languages

Table of Contents

Table of Contents.....	2
1 Accepted Expression Structures.....	3
2 Example Test Runs.....	4
3 Approach Description.....	7
3.1 evalu8 Description.....	7
3.2 Auxiliary Function Descriptions.....	8

1 Accepted Expression Structures

The expressions that can be evaluated by the meta-evaluator are structures as follows:

- **Lists (Operations or Function Calls):** The first element of a list is the operation or function name, while the rest of the elements are arguments or parameters. The meta-evaluator performs the operation on the arguments or calls the function with the provided arguments
 - **Operations:** These include addition (+), subtraction (-), multiplication (*), division (/), equality (eq?), and less than (<). Note that I had to implement the division operation in Part 2 of this project to handle the updated binding environment of `(foo . (lambda (x y) (/ (+ x y) 2)))`.
 - **Function Calls:** Functions can either be built-in or user-defined. User-defined functions are defined with a lambda.
 - **Lambda Expressions:** Represent anonymous functions with parameters and a body.
- **Numerical Literals:** These can be integers or floating-point numbers (e.g, 5, 3.2). They are immediately returned when encountered during evaluation.
- **Identifiers:** These represent variables or function names and are evaluated by looking up their value in the binding environment. If the identifier is not found, an error message is outputted to the console.

2 Example Test Runs

Below are 8 examples, with 4 expected to work correctly and 4 producing errors:

Working Examples

1) `(evaluator (+ a 5) env)`

a) Expected Output: 17

b) Explanation: a is 12, so $12 + 5 = 17$

2) `(evaluator (* (+ a 5) 3) env)`

a) Expected Output: 51

b) Explanation: $(+ a 5)$ is 17, so $17 * 3 = 51$

3) `(evaluator (foo 5 9) env)`

a) Expected output: 7

b) Explanation: foo is a function $(\lambda (x y) (/ (+ x y) 2))$. Evaluating $(foo 5 9)$ returns 7

4) `(evaluator ((lambda (x y) (* x (+ x y))) 2 3) env)`

a) Expected output: 10

b) Explanation: The lambda expression is evaluated with $x = 2$ and $y = 3$, which returns $2 * (2 + 3) = 10$

- The following image depicts the successful run of the above 4 examples:

```

119 ; Test cases
120 (display (evaluator8 '(+ a 5) env))      ; expect 17
121 (newline)
122 (display (evaluator8 '(* (+ a 5) 3) env)) ; expect 51
123 (newline)
124 (display (evaluator8 '(foo 5 9) env))      ; expect 7
125 (newline)
126 (display (evaluator8 '((lambda (x y) (* x (+ x y))) 2 3) env)) ; expect 10
127 (newline)

```

Welcome to [DrRacket](#), version 8.2 [cs].
 Language: racket/base, with debugging; memory limit: 128 MB.

```

17
51
7
10
>

```

Error Examples

1) (evaluator8 '(* (+ a 5) c) env))

a) Expected output: “Error: c is not bound”



b) Explanation: The identifier ‘c’ is not bound to a value in the environment.

```

119 ; Error Test cases
120 (display (evaluator8 '(* (+ a 5) c) env)) ; expect "Error: C is not bound"
121 (newline)

```

Welcome to [DrRacket](#), version 8.2 [cs].
 Language: racket/base, with debugging; memory limit: 128 MB.

  Error: c is not bound

```
>
```



2) ((evaluator8 '(lambda (x) (+ a x)) '((a . 5))) 3)

a) Expected output: “Error: lambda is not bound”

b) Explanation: The lambda expression is not correctly evaluated as a function call.

```
119 ; Error Test cases
120 (display (evalu8 '(lambda (x) (+ a x)) '((a . 5))) 3)
121 (newline)
---
```

Welcome to [DrRacket](#), version 8.2 [cs].
Language: racket/base, with debugging; memory limit: 128 MB.



  *Error: lambda is not bound*

3) (evalu8 '(fakeFunction 5) env)

- a) Expected output: “Error: fakeFunction is not bound”
- b) Explanation: “fakeFunction” is not defined in the binding environment.

```
120 (display (evalu8 '(fakeFunction 5) env))
121 (newline)
122
```

Welcome to [DrRacket](#), version 8.2 [cs].
Language: racket/base, with debugging; memory limit: 12



  *Error: fakeFunction is not bound*

4) (evalu8 '(> a 5) env))

- a) Expected output: “Error: > is not bound”
- b) Explanation: The greater than (>) operator was not defined as an operation we could use for our meta-evaluator.

```
119 ;/Error Test cases
120 (display (evalu8 '(> a 5) env))
121 (newline)
122 |
```

Welcome to [DrRacket](#), version 8.2 [cs].
Language: racket/base, with debugging; memory

  *Error: > is not bound*

>

3 Approach Description

The approach behind the meta-evaluator function (evalu8) and auxiliary functions used are described in the following two subsections.

3.1 evalu8 Description

The approach behind the meta-evaluator function (evalu8) is recursive and works by breaking down expressions into their components. Here is a step-by-step explanation:

- Base Case(s)
 - If the expression is a number (numerical literal), it is directly returned as the result.
 - If the expression is a symbol (an identifier), it is looked up in the environment to retrieve its value.
- Recursive Case
 - If the expression is a list, it is assumed to represent either an operation or function call.

- The first element (op) is checked to determine the type of operation or function call.
 - For operations, such as '+', '-', '*', '/', 'eq?', and '<' the evaluator recursively evaluates the operands and applies the correct operation.
 - For function calls, the meta-evaluator looks up the function definition, evaluates the arguments, and then applies the function body in an updated binding environment that includes the function's parameters.
- User Functions:
 - If the first element (op) is a user-defined function and is bound in the binding environment, then the function parameters are paired with the arguments, the function body is evaluated in an update environment, and the result is returned.
- Lambda Expressions:
 - If the first element (op) is equal to 'lambda', we call the auxiliary function (userFunction) and pass in the whole lambda expression (op), the arguments (cdr exp), and the current environment as parameters. The result is returned to the console.

3.2 Auxiliary Function Descriptions

I defined two auxiliary functions to assist the main meta-evaluator function (evalu8): lookupId and userFunction.

- **lookupId:**
 - **Brief:** The function searches for an identifier (or symbol) in the binding environment and returns the value if found.
 - **Parameters:**
 - **id:** the identifier (symbol) to look up.
 - **env:** the binding environment in which the identifier may be defined.
 - **Returns:** If an identifier is successfully found in the environment, it is returned. If the identifier is not found, an error message is outputted to the console.
- **userFunction:**
 - **Brief:** This function handles the evaluation of user-defined functions. It takes a function expression, a list of arguments, and the current environment, then creates a new environment where the parameters are bound to the evaluated arguments and the function body is evaluated.
 - **Parameters:**
 - **exp:** the expression representing the user-defined function.
 - **args:** the arguments passed to the user-defined function.
 - **env:** the current binding environment
 - **Returns:** The result of evaluating the user-defined function.
 - **More Details:**
 - `params (cadr exp)` stores the list of parameters.

- `exp` is a list where the 1st element is a function name, the 2nd element is a list of parameters (`cadr`), and the third element is the function body (`caddr`).
- `body (caddr exp)` extracts the function body.
- `updatedEnv` creates a new environment where...
 - `map (lambda (x) (eval8 x env)) args` evaluates each argument in the current environment
 - `map const params (...)` pairs each parameter with its evaluated argument
 - `append (...)` updates the current environment with the new parameters
- `(eval8 body updatedEnv)` evaluates the function body with the updated environment