COMP-301 PROJECT-2 WRITTEN PART

Part-A:

5 components of the language are:

1-) Syntax and datatypes:

Note: For the syntax of the MYLET language, please see Figure 1 of the COMP-301 Project-2 PDF document.

We defined and handled this component of a language (syntax and datatypes) in the racket file which is called “interp.rkt”.

2-) Environment :

Note: For the details about the language component called environment, please see Comp-301 Lecture 13 Slides.

We defined and handled this component of the language in the racket file called “environments.rkt” .

3-) Values (Expressed Values and Denoted Values):

Note: Please see Part-C of this document for the definitions of the expressed values and the denoted values in a language.

We defined and handled this language component (values) in the racket file which is called “data-structures.rkt”.

4-) Behavior specification:

We defined and handled this language component (Behavior Specification) in the racket file which is called “lang.rkt”.

5-) Behavior implementation : (Scanning, Parsing, Evaluation)

We defined and handled the scanning and parsing parts of this component in the racket file which is called “lang.rkt”. Furthermore; we defined and handled the evaluation part of the behavior implementation in the racket file which is called “interpt.rkt”.

Part-B:

Initially, we have the empty-env : ρ

After the addition of x :

[x=4]ρ

After the addition of y :

[x=4]

  [y=3] ρ

After the addition of z :

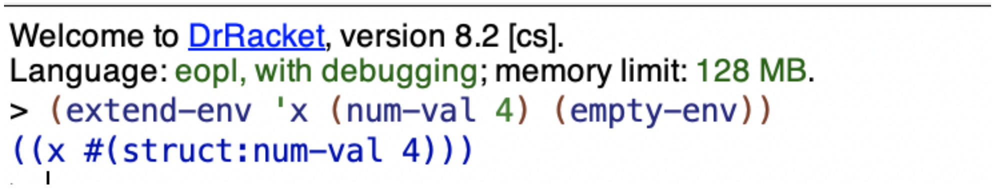
[x=4]

  [y=3]

    [z=6]ρ

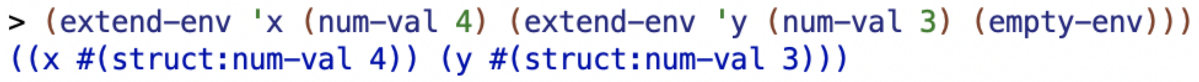
<u>Outputs from the dr racket for Part-B:</u>

After the addition of x:



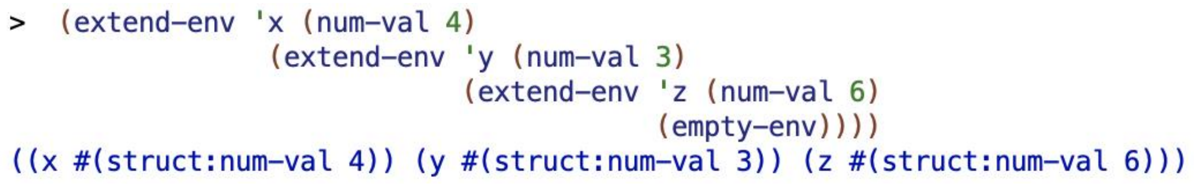
```
Welcome to DrRacket, version 8.2 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> (extend-env 'x (num-val 4) (empty-env))
((x #(struct:num-val 4)))
```

After the addition of y:



```
> (extend-env 'x (num-val 4) (extend-env 'y (num-val 3) (empty-env)))
((x #(struct:num-val 4)) (y #(struct:num-val 3)))
```

After the addition of z:



```
>  (extend-env 'x (num-val 4)
                 (extend-env 'y (num-val 3)
                             (extend-env 'z (num-val 6)
                                         (empty-env))))
 ((x #(struct:num-val 4)) (y #(struct:num-val 3)) (z #(struct:num-val 6)))
```

Part-C:

Expressed values (The definition): Expressed values of a language stand for the possible values of the expressions in that language.

The Expressed Values of the MYLET language= ExpVal = Int+Bool+String

Denoted values (The definition) : Denoted values of a language stand for the possible values of the variables in that language.

The Denoted Values of the MYLET language= DenVal = Int+Bool+String

Part-D: (Custom Expression)

6)

A brief explanation:
When a>0, b>0, and c>0; (a,b,c) is a valid pythagorean triple if c^2 is equal to a^2+b^2. Here; a, b, and c should be positive integers. If c^2 is not equal to a^2+b^2, then the (a, b, c) triple is not a valid pythagorean triple. Moreover; if a or b or c is not a positive integer but a^2+b^2=c^2, then (a,b,c) is not a valid pythagorean triple.

Syntax for the pisagor-exp (The Concrete Syntax and The Abstract Syntax):

The Concrete Syntax of the pisagor-exp:

Expression : := pisagor-exp (Expression, Expression, Expression)

The Abstract Syntax of the pisagor-exp:

```
pisagor-exp (exp1 exp2 exp3)
```

A detailed explanation about pisagor-exp

pisagor-exp is an expression which takes 3 expressions as parameters. pisagor-exp returns a boolean value. This expression checks whether a number triple is a valid pythagorean triple. If the expressions inside the pisagor-exp makes a valid pythagorean triple, then "pisagor-exp" returns #t. If the expressions inside the pisagor-exp does not make a valid pythagorean triple, then "pisagor-exp" returns #f.

Let’s consider the (3,4,5) triple, then we get 3*3+4*4 = 5*5 so this is a valid right-triangle and thus (3,4,5) is a valid pythagorean triple.

Let’s consider the (6,8,11) triple, then we get 6*6+8*8 = 100. However, 11*11 is 121. Since 100 is not equal to 121, (6,8,11) is not a valid pythagorean triple.

NOTE: The definition of a pythagorean triple, and a triple which contains the lengths of the sides of a right triangle are not the same. (a,b,c) is a pythagorean triple if a^2+b^2=c^2; and if a, b, and c are positive integers. For example; since sqrt(10) is not a positive integer, (1,3,sqrt(10)) is not a valid pythagorean triple. However, the numbers in (1, 3, sqrt(10)) can be the sides of a right triangle. Since the expressed values and the denoted values contain Int (and does not contain decimals & square roots of the positive integers) in MYLET language, we could not include a test case where a , b, or c is not a positive integer but the equality a^2+b^2=c^2 holds.

The Expressed Values of the MYLET language= ExpVal = Int+Bool+String
The Denoted Values of the MYLET language= DenVal = Int+Bool+String

<u>Part-E:</u>
No; by only trying different test cases, it is not possible to understand the correct implementation among the two implementations which are explained in this question. Since both of the implementations (top-to-bottom and bottom-to-top) return the value which corresponds to the last-true evaluated condition (regardless of the way of iteration), the different test cases written for these implementation do not help us to understand the correct implementation among the two implementations (top-to-bottom implementation and bottom-to-top implementation).

(my-cond1 "my-cond1 comp(9,'greater',5) then 7, comp(1,'equal',1) then 10, else 15" 10)

(bottom-to-top)

Starting from the first condition; since the last-true evaluated condition is comp(1,'equal',1),

So, my-cond-1 for the above test case will return 10.

--

(my_cond2 "my-cond1 comp(9,'greater',5) then 7, comp(1,'equal',1) then 10, else 25" 10)

(top-to-bottom):

Starting from the last condition; since the last-true evaluated condition is comp(1,'equal',1),

So, my-cond-2 for the above test case will also return 10

<u>Brief explanation about the work load breakdown :</u>

We did all the work together. We helped each other a lot about brain-storming, bug-checking, spell checking, implementation of the coding parts, and so on. We have also discussed, and analyzed the questions in this project together before we start the project.