COMP301 Project 2 report

Berke Can Rizai 69282
Aybars İnci 69032

Part A)

(1):

1. Syntax and data types
2. Values
3. Environment
4. Behavior specification
5. Behavior implementation

(2):

1. Syntax and data types → (data types) data-structures.rkt, (syntax) lang.rkt
2. Values → data-structures.rkt
3. Environment → environments.rkt
4. Behavior specification → interp.rkt
5. Behavior implementation → interp.rkt

Part B)

(1):

Initial environment created with variables x, y and z.
[x=4, y=3, z=6]*p*

(2):

p ranges over environments.
[ ] Denotes empty environment.
[var = val]*p* denotes (extend-env var val *p*)
[var1 = val1, var2 = val2]*p* abbreviates [var1 = val1]([var2 = val2] *p*) and so on.

For example;
init-env denotes the environment, [x = 4, y = 3, z = 6]
To init-env, we can add a new variable j = 14. After the addition, abbreviation would be
[j = 14]*(init-env)*
[j = 14, x = 4, y = 3, z = 6] where the value of j is 14, value of x is 4, value of y is 3 and z is 6.

Part C)

Expressed values are as follows, we deduce this information from the syntax of the MYLET language. We see that only number, boolean or string can be generated by the program in any of the expressions.
number, boolean, string

ExpVal = number + boolean + string
Denoted values are the set of values that we can assign to any expression in the program.
Denoted values are as follows,
number, boolean, string
DenVal = number + boolean + string
These are the same as expressed values as a result of the grammar of MYLET.

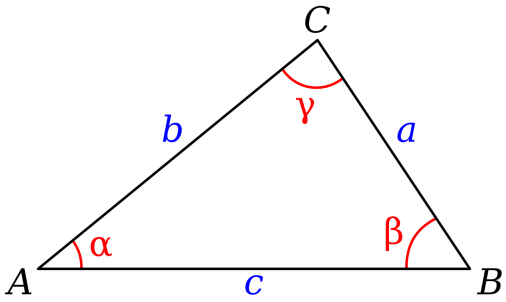
Part D)
(1): Added bool-exp.
(2): Added str-exp.
(3): Added op-exp.
(4): Added comp-exp.

(5): Implemented my-cond.
Our implementation is as follows, we iterate over all the elements in the conditions list. If any condition is true, we change the else statement to the exp that is associated with that condition. With this, if any of the conditions are evaluated as true later in the list, else is changed. When we come to the last condition where the list is empty and cond1 is the only condition that is left, we check it and if it is true, then the last true condition is cond1 and else it is the else-exp that we have changed earlier. If none of the conditions are met, we just return else-exp without changing.

(6):
Expression ::= calcNxEdge-exp (Expression, Expression, Expression)

calcNxEdge-exp (exp1 exp2 exp3)

This calcNxEdge is calculating the other edge of a given triangle. Exp1 is the length of the one of the edges and exp2 is the other edge, exp3 is the angle between these two edges.



In this example, exp1 = a, exp2 = b and Y is the exp3.
calcNxEdge (a, b, Y) = c

It works with the Pythagoras theorem. Theorem we have implemented is as follows,

$$c^2 = a^2 + b^2 - 2ab\cos\gamma,$$

In the code, we first get the values of expressions in the environment with value-of and assign the v1, v2 and v3 to them. Later, we assign an identifier to numeric value v1, b to v2 and deg

which is degree to numeric value of v3. We have also defined pi with many decimal places to make the program accurate in calculations, since we couldn't find pi as a predefined variable. In the calculation line, we first multiply the given degree with pi then divide this by 180 in order to make the degree-radian conversion. Then we take cos of the number with cos function and rest is trivial, a^^2 + b^^2 is summed up then we deduct 2.a.b.cos(Y) from that equation and finally take the square root of the whole equation. Thus, we end up with the result as a number that is precise. C is the returned value.

In the lang where we define the grammar, it is the same as the op grammar where each expression is split with the "," and all arguments are inside ( ).

Part E)

One way to find would be just print the trace and check from the variables. Since in the first implementation, we would be able to see the true evaluated value in the trace. However, this is not a good way or maybe possible way to find out.

Another way would be to give the large number of conditions to this function and only make the last condition to be true so that only the last expression is evaluated. For example, if we give it 1_000_000 conditions and 1_000_000 expressions and run it, in the first implementation, it would iterate over all one million items which would take more time than the second implementation where we just check the last one and return expression since 1 millionth condition is true. The difference between execution times would be huge.
Input for conditions would be like;
((#false), (#false),(#false),(#false)...(#true)) where we have 999999 #false exps and one #true.

Workload:
Part A: Collaborated
Part B: Aybars İnci
Part C: Berke Can Rizai
Part D:

(1) Bool-exp implemented by Berke Can Rizai
(2) Str-exp implemented by Aybars İnci
(3) Op-exp implemented by Aybars İnci
(4) Comp-exp implemented by Berke Can Rizai
(5) We both worked on the my-cond implementation
(6) We both worked on the calcNxEdge-exp implementation
(7) In the tests, Berke implemented three of the basic tests and rest of the tests were created by Aybars.

Part E: We both collaborated on this problem. The first way was developed by Aybars and second was developed by Berke.