PART A

The 5 components of the language are

- Syntax and datatypes
- Values
- Environment
- Behavior specification
- Behavior implementation
    - Scanning
    - Parsing
    - Evaluation

<u>Racket files we define and handle the components</u>
1)syntax&datatypes : lang.rkt
2)values : data-structures.rkt
3)environment : environment.rkt
4)behavior specification : data-structures.rkt
5)behavior implementation: interp.rkt

Syntax and datatypes for MYLET

```
(define-datatype program program?
  (a-program
   (exp1 expression?)))

(define-datatype expression expression?
  (const-exp
   (num number?))
  (str-exp
   (str string?))
  (op-exp
   (exp1 expression?)
   (exp2 expression?)
   (num number))
  (zero?-exp
   (exp1 expression?))
  (if-exp
   (exp1 expression?)
   (exp2 expression?)
   (conds expression?)
   (exps expression?)
   (exp3 expression?))
  (var-exp
   (var identifier?))
  (let-exp
   (var identifier?)
   (exp1 expression?)
   (body expression?)))
```

Values

Expressed values:
ExpVal = Int + String + Bool

Denoted values:
DenVal = Int + String + Bool

Interface for values

Constructors:
num-val: Int → ExpVal
bool-val: Bool → ExpVal
str-val: String → ExpVal

Observers:
expval → num: ExpVal → Int
expval → bool: ExpVal → Bool
expval → string: Expval → String

Environments

For MYLET language we use the same model of environment as LET.

Behavior specification

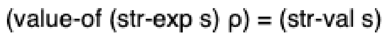
Behavior of Expressions

Constructors:
const-exp: Int → Exp
str-exp: String → Exp
op-exp: Exp x Exp x Int → Exp
zero?-exp: Exp → Exp

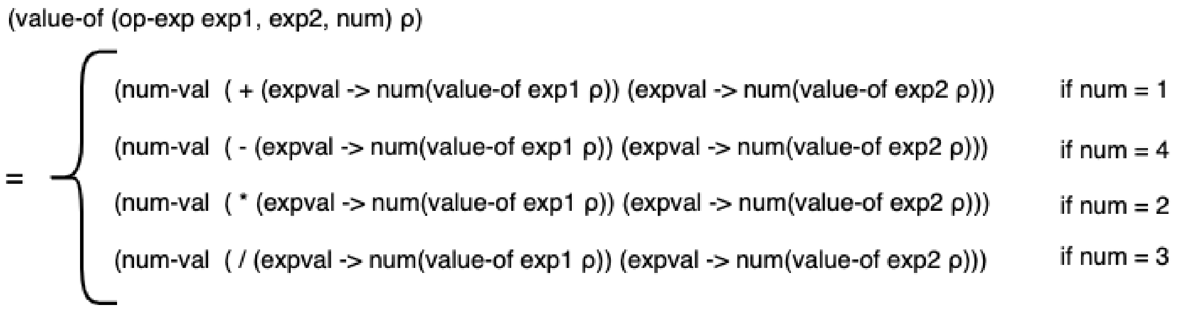
if-exp: Exp x Exp x
var-exp: Var → Exp
let-exp: Var x Exp x Exp → Exp

<u>Observers</u>:
value-of: Exp x Env → ExpVal

Behavior of MYLET methods

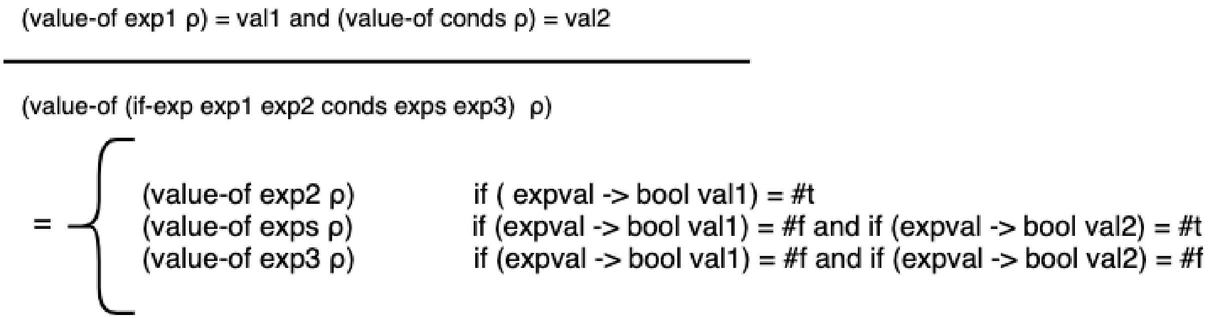


$$(\text{value-of (str-exp s) } \rho) = (\text{str-val s})$$



$$(\text{value-of (op-exp exp1, exp2, num) } \rho)$$

$$= \begin{cases} (\text{num-val} \ \ (+\ (\text{expval -> num(value-of exp1 } \rho))\ (\text{expval -> num(value-of exp2 } \rho))) & \text{if num = 1} \\ (\text{num-val} \ \ (-\ (\text{expval -> num(value-of exp1 } \rho))\ (\text{expval -> num(value-of exp2 } \rho))) & \text{if num = 4} \\ (\text{num-val} \ \ (*\ (\text{expval -> num(value-of exp1 } \rho))\ (\text{expval -> num(value-of exp2 } \rho))) & \text{if num = 2} \\ (\text{num-val} \ \ (/\ (\text{expval -> num(value-of exp1 } \rho))\ (\text{expval -> num(value-of exp2 } \rho))) & \text{if num = 3} \end{cases}$$



$$\frac{(\text{value-of exp1 } \rho) = \text{val1 and (value-of conds } \rho) = \text{val2}}{(\text{value-of (if-exp exp1 exp2 conds exps exp3) } \rho)}$$

$$= \begin{cases} (\text{value-of exp2 } \rho) & \text{if (expval -> bool val1) = \#t} \\ (\text{value-of exps } \rho) & \text{if (expval -> bool val1) = \#f and if (expval -> bool val2) = \#t} \\ (\text{value-of exp3 } \rho) & \text{if (expval -> bool val1) = \#f and if (expval -> bool val2) = \#f} \end{cases}$$

PART B

```
[x=1]
   [y=2]
      [z=3]ρ
```

to abbreviate

```
(extend-env ’x 1
(extend-env ’y 2
(extend-env ’u 3 ρ)))
```

PART C

Expressed values:
ExpVal = Int + String + Bool

Denoted values:
DenVal = Int + String + Bool

Constructors:
num-val: Int → ExpVal
bool-val: Bool → ExpVal
str-val: String → ExpVal

Observers:
expval → num: ExpVal → Int
expval → bool: ExpVal → Bool
expval → string: Expval → String

```
(define-datatype expval expval?
  (num-val (num number?))
  (bool-val (bool boolean?))
  (str-val (str string?))

(define expval->num
   (lambda (val)
     (cases expval val
       (num-val (num) num)
       (else (report-expval-extractor-error ’num val)))))
```

```
(define expval->bool
   (lambda (val)
     (cases expval val
       (bool-val (bool) bool)
       (else (report-expval-extractor-error ’bool val)))))

(define expval->str
    (lambda (val)
      (cases expval val
        (str-val (str) str)
        (else (report-expval-extractor-error ’str val)))))
```

PART D

Our custom expression is an exponential expression with the following syntax:

Expression ::= exponential (Expression, Expression)

custom-exp (exp1, exp2)

Workload Distribution:

All three of us worked together for the most part especially during the implementation of the MYLET language.