Solution

The semantics is given in a more readable form. It could easily be expressed in the λ-calculus notation used in CS571 class.

1 Domains

```
F-ENV:  function-name -> clause-list
ENV :  variable -> value
value : null UNION simple-value X value
simple-value : integer
```

2 Types:

```
Interpret-program:: program -> value
Create-funct-environment:: F-ENV -> function-list -> F-ENV
Interpret-expr:: F-ENV -> ENV -> expression -> value
Interpret-function-call:: F-ENV -> function-name -> value
apply-clause-list:: F-ENV ->  clause-list -> value
apply-pattern:: pattern -> value -> < Boolean, ENV >
apply-simple-pattern::  pattern -> value -> < Boolean, ENV >
val:: pattern -> value
```

3 Methods

```
Interpret-program(program) ==
                Interpret-expr( F-Env , empty, program.expression),
                        where F-Env ==
                                Create-funct-environment(empty,
                                                 program.function-list)

Create-funct-environment( FEnvironment, empty)== FEnvironment

Create-funct-environment( FEnvironment, function-list)==
        [function-list.function.function-name ->
                                 function-list.function.clause-list] +
        Create-funct-environment( FEnvironment, function-list.function-list)

Interpret-expr(FEnv, Env, e) ==
                if e is variable then e(Env),
                if e is constant then e,
                if e is e1 : e2  then
                     Interpret-expr(FEnv, Env, e1) : Interpret-expr(FEnv, Env, e2),
                if e is fname(e1) then
                        Interpret-function-call(FEnv, fname,
                                        Interpret-expr(FEnv, Env, e1))
```

```
Interpret-function-call(FEnv, fname, Val) ==
                apply-clause-list( FEnv, Clause-List, Val),
                        where Clause-List == fname(FEnv)

apply-clause-list( FEnv, Clause-List, Val) ==
                if successful-pattern-matching then Interpret-expr(FEnv, Env, e),
                        where e == Clause-List.clause.expression,
                              < successful-pattern-matching, Env > ==
                                 apply-pattern(Clause-List.clause.pattern, Val)

                otherwise  apply-clause-list( FEnv, Clause-List.clause-list, Val)

apply-pattern(Pattern, Val) ==
        if Pattern is simple-pattern then apply-simple-pattern(Pattern, Val)
        otherwise  if Val is simple-value then < FALSE, empty >
                otherwise  < P1 AND P2, E1 + E2 >
                        where < P1, E1 > == apply-pattern(Pattern.simple-pattern, Val.simple-value),
                              < P2, E2 > == apply-pattern(Pattern.pattern, Val.value)

apply-simple-pattern(Pattern, Val) ==
                        if Pattern is variable then <TRUE, [ Pattern.variable -> Val] >,
                        otherwise < (Val = val(Pattern)), empty >

val(Pattern) ==  if Pattern is constant then Pattern.constant
```