HOMEWORK 1

DEFINITION OF THE PROGRAMMING LANGUAGE

The programming language I implemented gives the possibility to use some basic operations as Sum, Minus, Multiplication, And, Or and the comparison operations between two operands. The data types I used are the following:

type expr =

  EInt of int

  | EBool of bool

  | EString of string

  | EVar of string

  | Plus of expr \* expr

  | Minus of expr \* expr

  | Mul of expr \* expr

  | And of expr \* expr

  | Or of expr \* expr

  | Equal of expr \* expr

  | Greater of expr \* expr

  | Lower of expr \* expr

  | Fun of ide\*expr\*permission list

  | Execute of expr\*expr

;;

type ide = string;;

type permission = Read | Write | Execute;;

THE INTERPRETER

The data types helped me in the implementation of the interpreter of the language. In the code the interpreter is represented by the function:

let rec eval (e:expr) (perm\_list: permission list) (env: 'v env)

An example of the syntax of a function is the following:

let f1 = Fun("x", Plus(EVar("x"), EInt(1)), [Read]);;

The parameters of the keyword “Fun” are indeed the variable name, the function body and the permissions. In this case the variable used in the function is “x”, the operation is a Sum and the permission that is needed to execute the function is “Read”.

RUN TIME STACK SIMULATION

To provide a proper variable mechanism I used a run time stack simulation with the definition of an environment and a bind function.

type 'v env = (string \* 'v) list;;

let emptyenv = [];;

let bind (env: 'v env) (x, y) = (x, y)::env;;

The binding of a value to the variable is made during the execution phase as in the following example:

let exe1 = Execute(f1, EInt(5));;

In this case there is the binding of the value 5 to the variable “x”, then the function is executed interpreting the keyword “Execute” as in the following case:

eval exe1 plist myEnv;;

PERMISSION MECHANISM

To execute functions in a secure environment I implemented a permission mechanism. The possible permissions that a user can have are Read, Write and Execute. Before executing the functions, the permissions required (given as parameters to the keyword “Fun”) are compared to the parameters assigned to the user. If the user has all the permissions, the function can be executed. This check is done during the interpretation of the keyword “Fun”. The permissions given to the user can be changed by these functions:

let rec removePermission (p:permission) (p\_list:permission list) : permission list =

  match p\_list with

  | [] -> failwith("No permission to remove!")

  | perm::others -> if p = perm then others else perm :: (removePermission p others)

;;

let rec addPermission (p:permission) (p\_list:permission list) : permission list =

  match p\_list with

  | [] -> [p]

  | perm::others -> if p = perm then failwith ("Permission already existent!") else perm :: (addPermission p others)

;;