Relazione Homework 1

Obiettivo

Lo scopo di questo homework è quello di estendere l'interprete di un semplice linguaggio funzionale di base in un interprete dotato di un meccanismo di access control dinamico basato su stack inspection.
Tale interprete esteso, oltre allo stack per mantenere le associazioni tra variabili e e valori, utilizzerà un'ulteriore struttura dati per i permessi presenti a runtime.
In più, l'interprete esteso farà affidamento su un'entità responsabile di simulare l'algoritmo di stack inspection e quindi di consentire/negare l'esecuzione di un programma o di una specifica funzione.

Implementazione

Le scelte implementative che ho fatto per estendere l'interprete sono le seguenti:

1. **Permessi**: ho definito un nuovo tipo per i permessi, intesi sia come permessi a runtime che come permessi richiesti da una funzione.

```
type permission =
   | Read
   | Write
   | Exec
```

2. **Espressioni**: ho modificato i costrutti del linguaggio, ed in particolar modo la dichiarazione di una funzione, assumendo che durante tale operazione vengano forniti anche i permessi necessari per la funzione stessa.

```
type expr =
   ...
     (* 'string' is the formal parameter, 'expr' is the body *)
     | Fun of string * expr * permission list
   ...
```

   Inoltre, ho aggiunto anche due nuovi costrutti per consentire al programmatore di abilitare/disabilitare un certo permesso globale all'interno di una particolare esecuzione.

```
type expr =
   ...
     (* 'expr' is a function, 'permission' is the permission to enable *)
     | EnableP of expr * permission
     (* 'expr' is a function, 'permission' is the permission to disable *)
     | DisableP of expr * permission
;;
```

3. **Espressioni intermedie**: ho scelto di definire un nuovo tipo, diverso dal tipo delle espressioni *expr*, per rappresentare il costrutto sintattico che realizza la stack inspection. In questo modo, tale costrutto sarà invisibile al programmatore e quindi non utilizzabile direttamente.

```
type iexpr =
      (* 'expr' is a function of which permissions should be checked *)
    | StackInsp of expr
;;
```

4. **Interprete**: ho esteso l'interprete classico *eval* aggiungendo come parametro la lista di permessi globali. Inoltre, ho modificato la valutazione del tipo *Fun* e ho aggiunto la valutazione dei costrutti *EnableP* e *DisableP*.

```
let rec eval (e : expr) (perm_list : permission list) (env : 'v env) : value =
    ...
    | Fun(p, fbody, fperm) -> ieval (StackInsp(e)) perm_list env
    | EnableP(func, p) -> begin match func with
                              | Fun(_, _, _) -> eval func (p::perm_list) env
                              | _ -> failwith("ERROR: Not a function!")
                          end
    | DisableP(func, p) -> begin match func with
                              | Fun(_, _, _) -> eval func (removePermission p perm_list) env
                              | _ -> failwith("ERROR: Not a function!")
                          end
;;
```

La funzione *ieval* invocata durante la valutazione di *Fun* è una funzione di valutazione ausiliaria che mi è servita per rendere il meccanismo di stack inspection invisibile al programmatore. Egli avrà a disposizione la sola funzione *eval* e non potrà quindi valutare direttamente il costrutto *StackInsp* responsabile del controlllo dei permessi.
La funzione *ieval* prende come argomenti un'espressione *iexpr*, una lista di permessi *perm_list* e l'ambiente *env* e verifica che i permessi della funzione al momento della sua dichiarazione siano presenti nella lista di permessi globali: se l'esito del controllo è positivo, viene restituita una chiusura, altrimenti un'eccezione con messaggio di errore.

Test

Nel codice è presente anche una parte dedicata ad alcuni test per controllare che il comportamento dei meccanismi implementati sia corretto.