Relazione Progetto

Relazione complessiva dei due esercizi

di Pieraccini Francesco, matricola 475944

Primo Esercizio

esercizio1.ml

Qua si trova l'implementazione del linguaggio funzionale dell'esercizio 1.

Sostanzialmente si trattava di implementare il linguaggio funzionale didattico con l'aggiunta di un costrutto try-with ed i relativi pattern. Per fare ciò, prima ho definito i tre pattern fra i tipi di dati nella parte della sintassi: PatComp rappresenta la composizione di pattern, PatEl il pattern elementare e PatDe il pattern di default.

Quindi ho creato una funzione per il pattern matching utilizzato dal costrutto try-with dell'interprete: per il pattern composto vado a verificare la guardia che, se vera, darà un'espressione, se falsa, chiamerà ricorsivamente sé stessa con un nuovo pattern, in caso contrario significa che la guardia non è booleana e quindi lancio un'eccezione; per il pattern elementare, invece, se la guardia è vera, restituisco l'espressione, se la guardia è falsa do errore dicendo che non è stata trovata una corrispondenza nel pattern; infine, il pattern di default che restituisce semplicemente l'espressione.

Per quanto riguarda l'interprete vi ho inserito il costrutto try-with con la sua implementazione, ovvero, come spiegato commentando il file:

"Prima mi creo un nuovo ambiente con l'associazione fra i (identificatore) ed e (espressione), valuto il pattern nella funzione funpat per vedere di che tipo (composto o elementare) è (cioè faccio il pattern matching), infine valuto il pattern risultante nel nuovo ambiente ottenuto."

tests.ml

Una serie di test sui costrutti del linguaggio, con particolare attenzione al try-with, ognuno dei quali è preceduto da un commento e dalla stampa del risultato atteso.

Scoping statico e dinamico:

Per lo scoping statico il corpo della funzione viene valutato nell'ambiente ottenuto legando i parametri formali ai valori dei parametri effettivi (attuali), che saranno noti al momento dell'applicazione, nell'ambiente in cui viene valutata l'astrazione. E questo è il metodo che ho utilizzato come si può vedere da:

```
"Fun (x, a) -> Funval (x, a, r)
Appl ( e1, e2 ) ->
    (match sem (e1, r) with
   Funval ( x, a, r1 ) ->
     sem (a, bind (r1, x, sem (e2,r)))
 | _ -> failwith ("Il primo valore di Appl deve essere un Funval"))".
```

Se si volesse utilizzare invece lo scoping dinamico il corpo della funzione dovrebbe essere valutato nell'ambiente ottenuto legando i parametri formali ai valori dei parametri effettivi, che questa volta saranno noti al momento dell'applicazione, nell'ambiente in cui avverrà l'applicazione.

Quindi dovrei modificare il costrutto dell'applicazione di funzione così:

```
"Appl ( e1, e2 ) ->
    (match sem (e1, r) with
    Funval ( x, a, r ) ->
    sem (a, bind (r, x, sem (e2,r)))
 | _ -> failwith ("Il primo valore di Appl deve essere un Funval"))".
```

Secondo Esercizio

SimpleTw.java

Contiene semplicemente l'interfaccia del testo. Ho scelto di fare una piccola aggiunta che, secondo me, era essenziale e nemmeno difficile da implementare, cioè non permettere ad addUser di aggiungere due volte lo stesso utente. Insieme all'interfaccia nel file ci sono le classi: utente e tag, a cui ho dato due metodi oltre al costruttore per rendere possibile un confronto e per poter leggere la variabile da esse contenuta.

Messaggio.java

È il "nodo" del microblog, è il tipo di ciascun messaggio e contiene tutte le informazioni necessarie a costruirlo (utente, codice, contenuto e tag), più dei metodi per poter visualizzare queste informazioni.

collezioneMessaggi.java

È la classe in cui, sostanzialmente, si svolgono tutte le operazioni del microblog e che contiene tutti i messaggi. Qui ho messo tutti i metodi che ho ritenuto necessari per svolgere i compiti del microblog.

myTw.java

È la classe che implementa l'interfaccia. Ha 3 variabili di istanza: una contiene la password, necessaria per la creazione del microblog e successivamente agli utenti per registrarsi, una collezione di utenti, un Set per l'esattezza (così da implementare la mia scelta di non avere due volte lo stesso utente) che contiene tutti gli utenti registrati, e infine una collezioneMessaggi. La mia scelta è stata quella di far appoggiare ogni metodo su quelli della collezioneMessaggi così da avere una visione più chiara (dato che mi limito a controllare se vi sono eccezioni e a chiamare funzioni di collezioneMessaggi).

provaMyTw.java

Una serie di test a tutti i metodi che erano da implementare nell'interfaccia composti da: un commento iniziale che dice solo ciò che si va a testare (con specificato se mi aspetto una eccezione) e una serie di println su quello che sta succedendo; in caso di errore (o, più che altro, comportamento inatteso) ci sarà una println("Errore"), altrimenti "Metodo + risultato" o "Eccezione + Test passed".