RELAZIONE PROGETTO OCAML 2018/2019

"estensione dell'interprete con il nuovo tipo Dictionary"

1. Specifica e analisi del problema:

il progetto si pone come obiettivo quello di estendere il linguaggio didattico funzionale (visto a lezione) in modo che si possa utilizzare anche il tipo di dato "Dictionary",oltre ai dati primitivi "standard". Il nuovo tipo "Dictionary" è un insieme di coppie (chiave, valore) *e ApplyOver, Get,Set,Remove,Clear* rappresentano le operazioni che possono essere effettuate su un dizionario. Se il parametro su cui devono operare non è un dizionario (DictVal) falliranno la loro valutazione.

i. **Sintassi concreta:** Dict::= ε | (Ide,Exp); Dict
Get::= (Dict, Ide)
Set::= (Dict, Ide, Exp)
Remove::= (Dict,Ide)
Clear::= (Dict)
ApplyOver::= (Exp,Dict)

ii. **Estensione:** **type exp** = … | Estring of string | EDict of (ide * exp) list | ApplyOver of exp * exp |
1. Get of exp * ide | Set of exp * ide * exp | Remove of exp * ide | Clear of exp ;;
type evT =… | StringV of string | DictVal of (ide * evT)list ;;

Estring of string : un'espressione può essere una stringa, che, dopo essere stata valutata nella funzione *eval,* diventerà StringV of string. Se prima della valutazione la stringa è Estring "nome", dopo la valutazione assumerà valore StringV "nome".

EDict of (ide * exp) list : un dizionario è un'espressione costituita da una lista di coppie (k,v):

- **k è univoca** ed **è sempre un *identificatore*** (quindi di tipo *string*) . Per verificare l'unicità delle chiavi la funzione ausialiaria *uniqueKey* usa la funzione ausiliaria *member* che cerca una chiave nella lista di coppie. Se *member* restituisce *true*,ovvero la trova, allora *uniqueKey* restituisce *false* e viene lanciato *"error: duplicate key".* Altrimenti le chiavi sono uniche e viene creato un nuovo dizionario.
- **v può essere una qualsiasi *espressione*** (exp), compresa *Edict of (ide * exp) list*

Edict, in quanto *exp*, dopo essere stato valutato, diventerà *un evT*: DictVal of (ide * evT) list, ovvero un dizionario valutato, in cui la lista è valutata tramite la funzione ausiliaria *evallist.*
Esempio:
let d1 = EDict [("k1",Eint1);("k2", Edict [("'k4'", Eint 20);("'k5'" ,Eint 21)]);("k3", Estring "val")];
- : evT = DictVal [("k1", Int 1); ("k2", DictVal [("'k4'", Int 20); ("'k5'", Int 21)]); ("k3", StringV "val")].

ApplyOver of exp * exp : la valutazione di *ApplyOver(f,dict)* restituisce un nuovo dizionario in cui viene applicata la funzione *f* ad ogni elemento di dict, tramite *applyFunToDict:* prima controlla che f (valutata) sia davvero un *FunVal* o *RecFunVal,* in tal caso restituisce la lista aggiornata, ovvero ogni valore associato a ogni chiave del dizionario *dict* di tipo compatibile è stato modificato secondo f. Se, per esempio, il tipo della funzione *f* è : - *int → int* =<*fun*>, allora quest'ultima verrà applicata solo ai valori interi del dizionario e il risultato, ancora intero, sostituirà il valore precedente associato a quella chiave k. Il controllo dei tipi è gestito dall'interprete, in modo che la funzione sia applicata solo ai valori del tipo opportuno, compatibili ai tipi dei parametri della funzione (*typecheck*).
Get of exp * ide : la valutazione di *Get(dict,k)* sul dizionario *dict (exp)* seleziona una chiave *k (ide)* e restituisce il valore associato a k (*evT* corrispondente) se la chiave è presente nel dizionario. Per scorrere le coppie del dizionario, alla ricerca di k, viene usata la funzione ausiliaria *lookup,* la quale restituisce *evT* o *Unbound* (nel caso in cui k non è presente c'è un errore: Unbound value).
Set of exp * ide: la valutazione di *Set(dict,k,v)* sul dizionario *dict (exp)* restituisce un nuovo dizionario, in cui:

- se k è già presente: avviene la sostituzione del valore associato precedentemente a k, con il nuovo v passato come parametro.
- se k non è presente: *esempio di Set(dict,k,v), con dict = [(k1,1);(k2,2)], k=k0 e v=0:* per prima cosa la funzione ausialiaria *reverse* "rovescia" la lista e si ottiene *dict = [(k2,v2);(k1,v1)],* poi la funzione ausiliaria *update* si occupa dell'aggiornamento, inserendo in coda alla lista rovesciata, ovvero *dict = [(k2,v2);(k1,v1);(k0,0)];* ora richiamando la *reverse,* dopo l'aggiornamento, si ottiene il nuovo dizionario con la nuova coppia (k0,0) in testa: *dict = [(k0,0);(k1,v1);(k2,v2)];* la *complessità,* al caso pessimo, è *lineare nel numero di coppie del dizionario*, invece che quadratica se avessi chiamato la *member* (per assicurarmi prima della presenza di k nel dizionario), all'interno della funzione *update* che poi avrebbe eseguito l'effettivo aggiornamento.

Remove of exp * ide: la valutazione di *Remove(dict, k)* sul dizionario *dict (exp)* restituisce un nuovo dizionario, in cui è stata rimossa la coppia (k,v), se è presente, altrimenti restituisce *dict* immutato, grazie alla funzione ausiliaria *remove.*

Clear of exp: *Clear(dict)* sul dizionario *dict* crea un nuovo dizionario vuoto, ottenuto azzerando *dict*, ma non apportando modifiche a *dict* stesso.

2.Testing del programma: in fondo al programma, nella sezione *TEST NUOVI SUL DIZIONARIO*, si trova una batteria di test che coinvolge tutti gli operatori aggiunti. Sotto ogni test ci sono i relativi commenti che riportano il risultato ottenuto dopo l'esecuzione del programma.