Secondo progetto intermedio
Programmazione 2
A.A. 2019/2020

Francesco Amodeo
Matricola: 560628
Corso A, prof. Ferrari

Il progetto consiste nell'estensione del linguaggio funzionale didattico e il relativo interprete, affrontati durante il corso di Programmazione 2. Si richiede la possibilità di manipolare dizionari, collezioni di valori identificati univocamente da una chiave. Più precisamente un dizionario è una collezione di coppie chiave-valore, dove la chiave è unica e i valori hanno tutti lo stesso tipo.

Allo scopo sono state aggiunte alla sintassi astratta del linguaggio (`exp`) le seguenti operazioni per la manipolazione di dizionari:

- **`Dict`**`(dictionary)` : crea un dizionario a partire dal nuovo tipo della sintassi `dictionary`[1];
- **`Insert`**`(k,v,d)` : inserisce la coppia k-v nel dizionario d;
- **`Delete`**`(d,k)` : rimuove la coppia con chiave k dal dizionario d;
- **`Has_key`**`(k,d)` : verifica se la chiave k è presente nel dizionario d;
- **`Iterate`**`(f,d)` : applica la funzione f ad ogni elemento del dizionario;
- **`Fold`**`(f,d)` : calcola il valore ottenuto, sommando il risultato delle applicazioni della funzione f a tutti gli elementi del dizionario;
- **`Filter`**`(l,d)` : filtra il dizionario d, eliminando da esso tutte le coppie k-v per cui la chiave k non appartiene alla lista l.

Tra i tipi esprimibili è stato aggiunto il tipo **`Dictionary`**, implementato con una lista `(ide * evT) list`. Quest'ultimo corrisponde al valore esprimibile ottenuto dalla valutazione di una espressione di tipo **`Dict`**. La valutazione avviene tramite la funzione `eval`, che definisce l'interprete, la quale usa a sua volta la funzione ausiliare `evalD`, che permette il passaggio dall'espressione che rappresenta il dizionario (definita tramite il tipo `dictionary`) al tipo esprimibile **`Dictionary`**, valutando sequenzialmente i valori (`exp`) di ogni coppia chiave-valore presente nel dizionario. La `evalD` ha inoltre il compito di verificare che il dizionario non presenti chiavi duplicate e che i valori abbiano tutti lo stesso tipo effettivo.
Per ognuna delle operazioni sopra introdotte sono state implementate una o più funzioni ausiliarie, da supporto per la valutazione delle stesse nella `eval`. Un esempio sono la `insert`, che prima di inserire la chiave controlla ricorsivamente che essa non sia già presente o la `fold`, che con la `selectSum` verifica l'applicabilità della **`Fold`** e sceglie l'operatore corretto da applicare per sommare i risultati intermedi. Nella maggior parte dei casi queste funzioni restituiscono un risultato di tipo `evT`, che successivamente sarà restituito dalla `eval` come risultato della valutazione dell'operazione sul dizionario.

Per il supporto a run-time e il controllo dei tipi è stato definito un type checker dinamico il quale, oltre a definire i pattern per i tipi comuni, introduce un pattern per il controllo del tipo `"dictionary"`, ampiamente utilizzato dalla `eval` durante la valutazione delle operazioni su dizionari.

Al fine di testare l'implementazione prodotta, sono stati inseriti dei semplici casi di test che permettono di verificare il comportamento dell'interprete in casi normali e in casi limite. Per far ciò basta valutare singolarmente ogni caso di test in un ambiente REPL invocando la `eval` su uno dei casi di test e in qualsiasi ambiente che sia implementato come una lista `(ide * evT) list`.

---

[1] **`dictionary`** `=` **`Empty`** `|` **`Item`** `of (ide * exp) * dictionary`,
dove `ide = string`