Secondo Progetto Intermedio

Leonardo Scoppitto Matricola 545615

Dicembre 2020

Introduzione ed esecuzione

Il secondo progetto intermedio consiste nell'estendere un linguaggio funzionale didattico per introdurre il tipo di dato `Set`. Un `Set` è una collezione omogenea di dati, non ordinati e senza duplicati.
Per eseguire il test è sufficiente spostarsi da terminale all'interno della cartella del progetto e digitare il comando `ocaml TestInterprete.ml`, inoltre consiglio di espandere la finestra del terminale a tutto schermo così da leggere meglio l'output, essendo molto denso di scritte.
Alternativamente è possibile copiare e incollare il contenuto del sorgente `TestInterpreteNoPrintf.ml` nella finestra interattiva del tool online Try OCaml Pro (ci metterà qualche secondo a valutare tutto il codice).
Il contenuto dei file è quindi il seguente:

- `TestInterprete.ml` → è il file contenente la batteria di test da eseguire per ottenere in output una prova grafica di tutte le operazioni eseguibili sui set.
- `TestInterpreteNoPrintf.ml` → contiene lo stesso codice di `TestInterprete.ml`, ma sono state rimosse le chiamate alla funzione `Printf.printf`, così da avere la parte di codice della batteria di test più leggibile. Inoltre alcune funzioni sono state commentate in modo da chiarirne il funzionamento.

Dettagli implementativi

Il tipo `Set`, dal punto di vista pratico, non è altro che una coppia `(lista, tipo)` in cui la lista può essere inizializzata come vuota o con un solo elemento (*Singleton*), mentre il tipo è una stringa che identifica, appunto, il tipo di ogni elemento della lista, così da garantire l'omogeneità dell'insieme e l'efficienza nel typechecking, infatti vale la proprietà:

$$\forall elemento \in lista, \quad typecheck(elemento, tipo) = True$$

Costruttori

Come già detto, un set può essere creato come insieme vuoto chiamando `Empty(type_)` e specificando il tipo dell'insieme, mentre chiamando il costruttore `Singleton(a, type_)` viene creato un insieme di un singolo elemento di tipo `type_`

Operazioni di base

`Union(s1, s2)`, `Intersection(s1, s2)` e `Difference(s1, s2)`

Le tre funzioni prendono come parametri due set, s1 e s2, verificano che i tipi siano compatibili ed infine restituiscono, rispettivamente, un set contenente una lista con gli elementi di entrambi i set senza duplicati (proprietà garantita dalla funzione `list_as_set`), un set contenente una lista con gli elementi comuni di s1 e s2 e un set contenente la differenza fra s1 e s2.

Operazioni sui Set

Inserimento e rimozione

Entrambe le funzioni (`Insert(s, toAdd)` e `Rm(s, toDel)`) prendono in input un set e un elemento. Dopo aver effettuato il typechecking dell'elemento da inserire o rimuovere, `Insert` effettua una ricerca dello stesso all'interno del set e, se non viene trovato nessun valore uguale, si effettua un inserimento in testa alla lista del set, mentre `Rm` passa la lista a una funzione di appoggio `delete`, che scorrerà tutta la lista rimuovendo eventualmente l'elemento richiesto.

`IsEmpty(s)`, `IsIn(s, query)`, `IsSubset(s1, s2)`

`IsEmpty` prende come parametro un set e tramite patter matching valuta se la lista del set è vuota, restituendo `Bool(true)` se la proprità è verificata, `Bool(false)` altrimenti.
`IsIn` prende in inglesso due parametri, un set e un valore, ed esegue una ricerca del valore all'interno della lista del set, restituendo `Bool(true)` se la proprità è verificata, `Bool(false)` altrimenti.
`IsSubset` prende come parametri di ingresso due set, s1 ed s2, e verifica se ogni elemento di s1 è contenuto in s2, restituendo `Bool(true)` se la proprità è verificata, `Bool(false)` altrimenti.

`Getmin(s)` e `Getmax(s)`

Entrambe prendono in input un set, la cui lista associata viene passata come parametro a due funzioni di supporto (`findmin` e `findmax`), che ricorsivamente confrontano la testa della lista con l'output della funzione applicata alla coda.

Operazioni di carattere funzionale

`For_all(f, s)`

Questa operazione prende in input un predicato `f` e un set `s` e applica `f` a tutti gli elementi di `s`: finché `f(element) = true`, la funzione va avanti, altrimenti si ferma restituendo `Bool(false)`.

`Exists(f, s)`

Questa operazione prende in input un predicato `f` e un set `s` e applica `f` a tutti gli elementi di `s`: se `f(element) = false`, la funzione va avanti, altirmenti si ferma, o nel caso in cui trova un elemento di `s` tale che `f(element) = true`, restituendo `Bool(true)`, o se arriva in fondo alla lista senza aver trovato nessun elemento tale che `f(element) = true`, restituendo `Bool(false)`.

`Filter(f, s)`

Questa operazione prende in input un predicato `f` e un set `s` e applica `f` a tutti gli elementi di `s`: ogni elemento di `s` tale che `f(element) = true` viene inserito all'interno di un nuovo set, che verrà restituito al termine della procedura.

`Map(f, s)`

Questa operazione prende in input un predicato `f` e un set `s` e applica `f` a tutti gli elementi di `s`: vengono inseriti all'interno di un nuovo tutti gli elementi di `s` dopo che vengono valutati sul predicato `f` (quindi gli elementi `newElement = f(element)`), dopodiché il nuovo set verrà restituito.