Secondo Progetto Programmazione II

Francesco Salvatori
Mat. 581142
f.salvatori5@studenti.unipi.it

June 2021

1 Regole operazionali per l'introduzione di Set

Un insieme è una struttura dati che contiene elementi omogenei di un determinato tipo τ.
In questa implementazione ho deciso di limitare i tipi disponibili ad un sottoinsieme di evT, da qui in poi *Types*: int, bool e string.
Non è possibile definire insiemi di insiemi o insiemi di funzioni.

Empty Set

$$\frac{\tau \in Types}{env \triangleright empty(\tau) \implies (\tau, \emptyset)}$$

Singleton

$$\frac{\tau \in Types \qquad env \triangleright e \implies v \qquad v : \tau}{env \triangleright singleton(\tau, e) \implies (\tau, \{v\})}$$

Of

$$\frac{\begin{array}{c}\tau \in Types \qquad env \triangleright l \implies [e_1, e_2, ..., e_n] \\ env \triangleright \forall e_i \in l, e_i \implies v_i : \tau \qquad V = \{v_1, v_2, ..., v_n\} \\ \forall (v_i = v_j) \in V, i \neq j \implies V = V \setminus \{v_j\}\end{array}}{env \triangleright of(\tau, l) \implies (\tau, V)}$$

2 Operazioni di base

Unione

$$\frac{env \triangleright s_1 \implies Set(\tau_1, vl_1) \qquad env \triangleright s_2 \implies Set(\tau_2, vl_2) \qquad \tau_1 = \tau_2 = \tau}{env \triangleright union(s_1, s_2) \implies Set(\tau, vl_1 \cup vl_2)}$$

Intersezione

$$\frac{env \triangleright s_1 \implies Set(\tau_1, vl_1) \qquad env \triangleright s_2 \implies Set(\tau_2, vl_2) \qquad \tau_1 = \tau_2 = \tau}{env \triangleright intersect(s_1, s_2) \implies Set(\tau, vl_1 \cap vl_2)}$$

Differenza

$$\frac{env \triangleright s_1 \implies Set(\tau_1, vl_1) \qquad env \triangleright s_2 \implies Set(\tau_2, vl_2) \qquad \tau_1 = \tau_2 = \tau}{env \triangleright diff(s_1, s_2) \implies Set(\tau, \{vl_1 - vl_2)\}}$$

Inserimento

$$\frac{env \triangleright e \implies v \qquad v : \tau_1 \qquad env \triangleright s \implies Set(\tau_2, vl) \qquad \tau_1 = \tau_2 = \tau}{env \triangleright insert(e, s) \implies Set(\tau, vl \cup \{v\})}$$

Rimozione di un elemento

$$\frac{env \triangleright e \implies v \qquad v : \tau_1 \qquad env \triangleright s \implies Set(\tau_2, vl) \qquad v \in vl \qquad \tau_1 = \tau_2 = \tau}{env \triangleright remove(e, s) \implies Set(\tau, vl \setminus \{v\})}$$

Controllo se un insieme è vuoto

$$\frac{env \triangleright s \implies Set(\tau, vl)}{env \triangleright isempty(s) \implies Bool(vl = \emptyset)}$$

Member

$$\frac{env \triangleright e \implies v \qquad v : \tau_1 \qquad env \triangleright s \implies Set(\tau_2, vl) \qquad \tau_1 = \tau_2}{env \triangleright member(e, s) \implies Bool(e \in vl)}$$

Sottoinsieme

$$\frac{env \triangleright s_1 \implies Set(\tau_1, vl_1) \qquad env \triangleright s_2 \implies Set(\tau_2, vl_2) \qquad \tau_1 = \tau_2}{env \triangleright subset(s_1, s_2) \implies Bool(vl_1 \subset vl_2)\}}$$

Min

$$\frac{env \triangleright s \implies Set(\tau, vl)}{env \triangleright min(s) \implies min\{vl\}}$$

Max

$$\frac{env \triangleright s \implies Set(\tau, vl)}{env \triangleright max(s) \implies max\{vl\}}$$

3 Operazioni funzionali

For all

$$\frac{\begin{array}{c} env \triangleright s \implies Set(\tau, vl) \qquad env \triangleright p \implies Closure(arg, body, fenv) \\ \forall v_i \in vl\ (env \triangleright Apply(p, v_i)) \implies Bool(r_i) \qquad R = \{r_1, r_2, ..., r_n\} \\ \forall r_i \in R, r_i = Bool(true) \implies res \end{array}}{env \triangleright forall(p, s) \implies Bool(res)}$$

Exists

$$\frac{\begin{array}{c} env \triangleright s \implies Set(\tau, vl) \qquad env \triangleright p \implies Closure(arg, body, fenv) \\ \forall v_i \in vl\ (env \triangleright Apply(p, v_i)) \implies Bool(r_i) \qquad R = \{r_1, r_2, ..., r_n\} \\ \exists r_i \in R, r_i = Bool(true) \implies res \end{array}}{env \triangleright exists(p, s) \implies Bool(res)}$$

Filter

$$\frac{\begin{array}{c} env \triangleright s \implies Set(\tau, vl) \qquad env \triangleright p \implies Closure(arg, body, fenv) \\ \forall v_i \in vl\ (env \triangleright Apply(p, v_i) = Bool(true)) \implies r_i = v_i \quad R = \{r_1, r_2, ..., r_n\} \end{array}}{env \triangleright filter(p, s) \implies Set(\tau, R)}$$

Map

$$\frac{\begin{array}{c} env \triangleright s \implies Set(\tau, vl) \qquad env \triangleright f \implies Closure(arg, body, fenv) \\ \forall v_i \in vl\ (env \triangleright Apply(f, v_i)) \implies v_i' : \tau' \qquad V' = \{v_1', v_2', ..., v_n'\} \end{array}}{env \triangleright map(f, s) \implies List(\tau', V')}$$

4 Scelte implementative

4.1 Information hiding

Ho scelto di implementare gli insiemi mediante la nozione di lista presente in OCaml per semplicità, tuttavia il linguaggio prevede due costrutti che preservano l'information hiding sull'implementazione che vanno ad incapsulare questa informazione: **Elem * exp e End**.
Si usa questo costrutto solo per la costruzione di insiemi a partire da espressioni, mentre tutte le altre operazioni operano direttamente con le liste. In un **Elem** posso mettere una qualsiasi espressione tra quelle definite, **End** sancisce la fine della lista di valori del Set.

4.2 Modifiche all'AST

L'interprete dovrebbe garantire di non essere un JIT compiler poiché la valutazione di un'espressione genera un valore espresso sotto forma di evaluation type, e nel farlo non viene modificato l'albero di sintassi astratta mediante del codice "pre-compilato". Di fatto, nelle operazioni funzionali si richiama una versione in coda all'interprete della Apply in tutto e per tutto uguale nel comportamento ma che opera su **evT e non su espressioni exp.**

5 Esecuzione dei test

Si può testare il completo funzionamento dell'interprete eseguendo il codice del file *test.ml* presente in questo archivio.
È necessario eseguire per prima cosa il codice dell'interprete.[1]
Tra i test è presente una funzione $test\,(e : exp) \rightarrow (evT)$**, che permette di chiamare l'eval di uno specifico test, es:**

```
let t1 = ... ;;

(*Equivalente ad eval t1 [];;*)
test t1;;
```

[1]Il progetto è stato realizzato tramite il text editor Visual Studio Code, ma il codice è stato testato tramite la piattaforma online TryOCaml mediante il solito meccanismo REPL