Interprete di un linguaggio didattico

Programmazione 2 (2020/2021)

Il progetto consiste nella realizzazione e discussione di un interprete di un linguaggio didattico funzionale. Viene sviluppato in Ocaml e aggiunge alle funzionalità di base, viste a lezione, la gestione di insiemi omogenei (non ordinati).

Nel dettaglio:

Verrà usato un formalismo molto simile a quello visto a lezione.
Sia Γ un ambiente di valutazione e di tipo e sia $\Gamma \vdash e : (\tau, v)$ l'associazione dell'espressione e al tipo τ e al valore v nell'ambiente Γ. Si noti che l'associazione $v \to t$ è banalmente effettuata, tramite pattern matching dalla funzione *typeof* utilizzata nell'interprete (riga 112). Con un abuso di notazione, indicheremo con $e_t : (\tau_e, \tau_n)$ l'associazione tra il *typeInfo* (*TSet, TBool, TInt...*, vedi riga 4) e_t e il tipo esprimibile (*Set, Bool, Int...*) ad esso relativo τ_n. Inoltre utilizzeremo $e_s : ([\tau_s, \tau_i], v_s)$ per indicare l'associazione tra un' espressione (e_s) relativa ad un set, il tipo (τ_s) esprimibile del set (*Set*) e il valore interpretato (v_s) del set, inoltre τ_i è il tipo degli elementi del set.
Sia τ_1 un tipo esprimibile del linguaggio tale per cui è possibile avere un set di elementi di tipo τ_1 (nota: si è scelto di utilizzare soltanto insiemi di interi e stringhe) e sia τ_0 un tipo esprimibile del linguaggio, diverso da τ_1 tale per cui **non** è possibile avere un set di elementi di tipo τ_0. Mostriamo adesso le regole di valutazione per il tipo di dato Set, costruttori ed operazioni:

Costruttori

- Empty: $\dfrac{\Gamma \vdash e_t : (\tau_e, \tau_1)}{\Gamma \vdash Empty(e_t) \to \emptyset}$ Singleton: $\dfrac{\Gamma \vdash e_1 : (\tau_e, \tau_1), e_2 : (\tau_1, v)}{\Gamma \vdash Singleton(e_t, e_2) \to \{v_2\}}$

Operazioni di base

- Insert: $\dfrac{\Gamma \vdash e_1 : (\tau_1, v_1), e_s : ([\tau_s, \tau_i], v_s) \land \tau_i = \tau_1}{\Gamma \vdash Insert(e_1, e_s) \to v_s \cup \{v_1\}}$

- Remove: $\dfrac{\Gamma \vdash e_1 : (\tau_1, v_1), e_s : ([\tau_s, \tau_i], v_s) \land \tau_i = \tau_1 \land v_1 \in v_s}{\Gamma \vdash Remove(e_1, e_s) \to v_s - \{v_1\}}$

- Contains: $\dfrac{\Gamma \vdash e_1 : (\tau_1, v_1), e_s : ([\tau_s, \tau_i], v_s) \land v_1 \in v_s}{\Gamma \vdash Contains(e_1, e_s) \to Bool(true)}$

  $\dfrac{\Gamma \vdash e_1 : (\tau_1, v_1), e_s : ([\tau_s, \tau_i], v_s) \land v_1 \notin v_s}{\Gamma \vdash Contains(e_1, e_s) \to Bool(false)}$

- IsEmpty: $\dfrac{\Gamma \vdash e_s : ([\tau_s, \tau_i], v_s) \land v_s = \emptyset}{\Gamma \vdash IsEmpty(e_s) \to Bool(true)}$ $\dfrac{\Gamma \vdash e_s : ([\tau_s, \tau_i], v_s) \land v_s \neq \emptyset}{\Gamma \vdash IsEmpty(e_s) \to Bool(false)}$

- IsSubSet: $\dfrac{\Gamma \vdash e_{s1} : ([\tau_{s1}, \tau_i], v_{s1}), e_{s2} : ([\tau_{s2}, \tau_i], v_{s2}) \land v_{s1} \subseteq v_{s2}}{\Gamma \vdash IsSubSet(e_{s1}, e_{s2}) \to Bool(true)}$

  $\dfrac{\Gamma \vdash e_{s1} : ([\tau_{s1}, \tau_i], v_{s1}), e_{s2} : ([\tau_{s2}, \tau_i], v_{s2}) \land v_{s1} \nsubseteq v_{s2}}{\Gamma \vdash IsSubSet(e_{s1}, e_{s2}) \to Bool(false)}$

- GetMax: $\dfrac{\Gamma \vdash e_s : ([\tau_s, \tau_i], v_s) \land v_s = \{v_i \text{ tc } v_1 > v_2 > \cdots > v_n\}}{\Gamma \vdash GetMax(e_s) \to v_1}$

- GetMin: $\dfrac{\Gamma \vdash e_s : ([\tau_s, \tau_i], v_s) \land v_s = \{v_i \text{ tc } v_1 > v_2 > \cdots > v_n\}}{\Gamma \vdash GetMin(e_s) \to v_n}$

Operatori di natura "funzionale"

- For_all: $\dfrac{\Gamma \vdash f : (\tau_f, v_f), e_s : ([\tau_s, \tau_i], v_s) \wedge \forall v_i \in v_s \implies f(v_i) = true}{\Gamma \vdash \text{Forall}(f, e_s) \to Bool(true)}$

  $\dfrac{\Gamma \vdash f : (\tau_f, v_f), e_s : ([\tau_s, \tau_i], v_s) \wedge \exists v_i \in v_s \text{ tc } f(v_i) = false}{\Gamma \vdash \text{Forall}(f, e_s) \to Bool(false)}$

- Exists: $\dfrac{\Gamma \vdash f : (\tau_f, v_f), e_s : ([\tau_s, \tau_i], v_s) \wedge \forall v_i \in v_s \implies f(v_i) = false}{\Gamma \vdash \text{Exists}(f, e_s) \to Bool(false)}$

  $\dfrac{\Gamma \vdash f : (\tau_f, v_f), e_s : ([\tau_s, \tau_i], v_s) \wedge \exists v_i \in v_s \text{ tc } f(v_i) = true}{\Gamma \vdash \text{Exists}(f, e_s) \to Bool(true)}$

- Filter: $\dfrac{\Gamma \vdash f : (\tau_f, v_f), e_s : ([\tau_s, \tau_i], v_s) \wedge v_1, \ldots, v_n \in v_s \implies f(v_i) = true}{\Gamma \vdash \text{Filter}(f, e_s) \to \{v_1, \ldots, v_n\}}$

- Map: $\dfrac{\Gamma \vdash f : (\tau_f, v_f), e_s : ([\tau_s, \tau_i], v_s) \wedge v_1, \ldots, v_n \in v_s}{\Gamma \vdash \text{Map}(f, e_s) \to \{f(v_1), \ldots, f(v_n)\}}$

Per l'eventuale descrizione delle varie regole si rimanda alla discussione del progetto in quanto non è possibile riassumere le spiegazioni delle regole in una sola pagina di relazione. Si vuol far notare, comunque, che si tratta di banali regole di inferenza in cui vengono quindi eplicitate le premesse e le conseguenze nell'utilizzo dei vari costruttori/operatori.

Scelte progettuali in evidenza

- Gli insiemi possono contenere soltanto interi o stringhe. E' stato deciso di non accettare insiemi di booleani in quanto un insieme non può contenere duplicati e un insieme di soli valori booleani è risultato, ad un primo approccio, inutile.
- Le funzioni For_all, Exists, Filter accettano una funzione (predicato) come primo parametro. Dato che il type checking è dinamico, al momento della chiamata della funzione sul primo elemento dell'insieme viene effettuato il controllo del tipo di ritorno. Se il tipo di ritorno non è booleano, allora viene lanciata un'eccezione.
- La funzione Map accetta una funzione come parametro. Se la funzione ritorna un tipo che non è accettato come tipo per gli elementi di un insieme allora viene lanciata un'eccezione. Il controllo avviene alla fine della chiamata della funzione su tutti gli elementi dell'insieme. Sarebbe stato possibile implementare un type-checking statico per ottimizzare alcuni casi, quello appena discusso è un esempio di questi.
- L'ambiente polimorfo è utile in quanto permette l'associazione tra un identificatore ed un qualsiasi tipo di dato.
- E' possibile allo sviluppatore che utilizza il linguaggio didattico scatenare eccezioni (ma non gestirle tramite try...catch...) tramite l'utilizzo dell'espressione *Raise*.

Test

In allegato al codice dell'interprete è possibile trovare un file contenente un'esaustiva batteria di test. Per provare l'efficacia dell'interprete è possibile usare l'interprete ocaml da terminale o, in alternativa, è possibile usare la pagina web https://try.ocamlpro.com/ (al momento in versione beta).

Lorenzo Amorelli