Secondo Progetto Intermedio

Filippo Costa

Dicembre 2020

1. Introduzione

Questo progetto consiste in un interprete in OCaml di un semplice linguaggio di programmazione a paradigma funzionale. L'intero interprete è contenuto in un unico file sorgente con nome `interpreter.ml`. Il file `original.ml` contiene la versione originale dell'interprete del linguaggio (senza le modifiche apportate dal sottoscritto). Entrambi i file contengono ciascuno una batteria di test per verificare il corretto funzionamento del rispettivo interprete.

Per eseguire la batteria di test si può usare il comando `ocaml`:

```
$ ocaml original.ml
$ ocaml interpreter.ml
```

Rispetto alla versione originale, `interpreter.ml` aggiunge il supporto per *stringhe* e *insiemi*. Le stringhe sono sequenze immutabili di caratteri e gli insiemi sono collezioni immutabili e senza ordine di oggetti omogenei. Gli insiemi possono essere parametrizzati esclusivamente su `Int`, `Bool`, o `String` (i.e. non è possibile costruire insiemi di insiemi o insiemi di funzioni).

2. Regole operazionali per `Set`

Si riportano in seguito le regole operazionali per:

- l'introduzione del tipo di dato `Set` nel linguaggio didattico;
- le classi di operazioni previste dal tipo di dato `Set`.

$$\frac{env \rhd e \implies t\colon String, \quad t \in \{"string", "int", "bool"\}}{env \rhd \texttt{Eset}(e) \implies \emptyset\colon Set_t}$$

$$\frac{env \rhd e_1 \implies t\colon String, \quad t \in \{"string", "int", "bool"\}, \quad env \rhd e_2 \implies v\colon t}{env \rhd \texttt{Singleton}(e_1, e_2) \implies \{v\}\colon Set_t}$$

`ForAll`, `Exists`, `Filter`:

$$\frac{env \rhd s \implies A\colon Set_t, \quad env \rhd p \implies P\colon t \to Bool}{\begin{array}{l} \forall x \in A \implies P(x) \vdash env \rhd \texttt{ForAll}(p, s) \implies \top \\ \forall x \in A \implies \neg P(x) \vdash env \rhd \texttt{Exists}(p, s) \implies \bot \\ \exists x \in A \mid P(x) \vdash env \rhd \texttt{Exists}(p, s) \implies \top \\ \exists x \in A \mid \neg P(x) \vdash env \rhd \texttt{ForAll}(p, s) \implies \bot \\ env \rhd \texttt{Filter}(p, s) \implies B\colon Set_t, \quad B \subset A, \quad \forall x \in A \implies (P(x) \iff x \in B) \end{array}}$$

`Union`, `Intersection`, `SetDifference`, `IsSubsetOf`:

$$\frac{env \rhd s_1 \implies A\colon Set_t, \quad env \rhd s_2 \implies B\colon Set_t}{\begin{array}{l} env \rhd \texttt{Union}(s_1, s_2) \implies A \cup B \\ env \rhd \texttt{Intersection}(s_1, s_2) \implies A \cap B \\ env \rhd \texttt{SetDifference}(s_1, s_2) \implies A \backslash B \\ A \subset B \vdash env \rhd \texttt{IsSubsetOf}(s_1, s_2) \implies \top \\ A \not\subset B \vdash env \rhd \texttt{IsSubsetOf}(s_1, s_2) \implies \bot \end{array}}$$

`SetAdd`, `SetRemove` and `IsIn`:

$$\frac{env \rhd s \implies A\colon Set_t, \quad env \rhd e \implies v\colon t}{\begin{array}{l} env \rhd \texttt{SetAdd}(s, e) \implies A \cup \{v\} \\ env \rhd \texttt{SetRemove}(s, e) \implies A \backslash \{v\} \\ v \in A \vdash env \rhd \texttt{IsIn}(e, s) \implies \top \\ v \notin A \vdash env \rhd \texttt{IsIn}(e, s) \implies \bot \end{array}}$$

`IsEmpty`:

$$\frac{env \rhd s \implies A\colon Set_t}{\begin{array}{l} A = \emptyset \vdash env \rhd \texttt{IsEmpty}(s) \implies \top \\ A \neq \emptyset \vdash env \rhd \texttt{IsEmpty}(s) \implies \bot \end{array}}$$

`Min` and `Max`:

$$\frac{env \rhd s \implies A\colon Set_t, \quad A \neq \emptyset}{\begin{array}{l} env \rhd \texttt{Min}(s) \implies v\colon t, \quad \forall x \in A \implies x > v \\ env \rhd \texttt{Max}(s) \implies v\colon t, \quad \forall x \in A \implies v > x \end{array}}$$

`Map`:

$$\frac{env \rhd s \implies A\colon Set_{t_1}, \quad env \rhd e \implies f\colon t_1 \to t_2}{\begin{array}{l} env \rhd \texttt{Map}(e, s) \implies B\colon Set_{t_2} \\ \forall a \in A \implies (\exists b \in B \mid b = f(a)) \\ \forall b \in B \implies (\exists a \in A \mid b = f(a)) \end{array}}$$