

## Semantica operativa

Siano  $env$  un ambiente e sia  $env \triangleright exp$  la notazione per indicare che l'espressione  $exp$  viene valutata rispetto ad  $env$ . Dette inoltre  $exp \Rightarrow evT$  la valutazione dell'espressione ad un denotabile e  $:type$  l'annotazione di tipo, al fine di estendere il linguaggio con insiemi e stringhe definisco la seguente serie di regole della semantica operativa.

### Regole di inferenza

$$\text{Estring: } \frac{s : string}{env \triangleright Estring(s) \Rightarrow String(s)}$$

$$\text{Concat: } \frac{env \triangleright e1 \Rightarrow String(x : string), \quad env \triangleright e2 \Rightarrow String(y : string)}{env \triangleright Concat(e1, e2) \Rightarrow String(x \wedge y)}$$

dove con “ $\wedge$ ” ho rappresentato l'operazione di concatenazione di stringhe

$$\text{EmptySet: } \frac{env \triangleright type \Rightarrow String(t : string)}{env \triangleright EmptySet(type) \Rightarrow \emptyset : t}$$

$$\text{Singleton: } \frac{env \triangleright type \Rightarrow String(t : string), \quad env \triangleright el \Rightarrow v : t}{env \triangleright Singleton(el, type) \Rightarrow \{v\} : t}$$

$$\text{Set: } \frac{env \triangleright type \Rightarrow String(t : string), \quad (\forall i. i \in els \wedge (env \triangleright i \Rightarrow v_i : t))}{env \triangleright Set(els, type) \Rightarrow \{v_i : i \in els\} : t}$$

IsEmpty, divisa tra valutazione true e false:

$$\frac{env \triangleright type \Rightarrow String(t : string), \quad env \triangleright e \Rightarrow \emptyset : t}{env \triangleright IsEmpty(e) \Rightarrow Bool(true)} \quad , \quad \frac{env \triangleright type \Rightarrow String(t : string), \quad env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \quad \{v_i : i \in els\} \neq \emptyset}{env \triangleright IsEmpty(e) \Rightarrow Bool(false)}$$

Contains, divisa tra valutazione true e false:

$$\frac{env \triangleright type \Rightarrow String(t : string), \quad env \triangleright elem \Rightarrow w : t, \quad env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \quad w \in \{v_i : i \in els\}}{env \triangleright Contains(e, elem) \Rightarrow Bool(true)} \quad , \quad \frac{env \triangleright type \Rightarrow String(t : string), \quad env \triangleright elem \Rightarrow w : t, \quad env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \quad w \notin \{v_i : i \in els\}}{env \triangleright Contains(e, elem) \Rightarrow Bool(false)}$$

$$\text{Insert: } \frac{env \triangleright type \Rightarrow String(t : string), \quad env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \quad env \triangleright elem \Rightarrow w : el\_type, \quad el\_type = t}{env \triangleright Insert(e, elem) \Rightarrow \{v_i : i \in els\} \cup \{w\} : t}$$

$$\begin{array}{c}
\text{env} \triangleright \text{type} \Rightarrow \text{String}(t : \text{string}), \\
\text{env} \triangleright e \Rightarrow \{v_i : i \in \text{els}\} : t, \\
\text{Remove:} \quad \text{env} \triangleright \text{elem} \Rightarrow w : \text{el\_type}, \\
\quad \text{el\_type} = t \\
\hline
\text{env} \triangleright \text{Remove}(e, \text{elem}) \Rightarrow \{v_i : i \in \text{els}\} \setminus \{w\} : t
\end{array}$$

Subset, divisa nei casi  $a \subseteq b$  e  $a \not\subseteq b$  (se i tipi fossero diversi non potrei effettuare l'operazione):

$$\begin{array}{c}
e1 = \text{Set}(\text{el1}, \text{type1}), \\
e2 = \text{Set}(\text{el2}, \text{type2}), \\
\text{env} \triangleright e1 \Rightarrow \{v_i : i \in \text{el1}\} : t_1, \\
\text{env} \triangleright e2 \Rightarrow \{w_j : j \in \text{el2}\} : t_2, \\
t_1 = t_2, \\
\{v_i : i \in \text{el1}\} \subseteq \{w_j : j \in \text{el2}\} \\
\hline
\text{env} \triangleright \text{Subset}(e1, e2) \Rightarrow \text{Bool}(\text{true})
\end{array}
, \quad
\begin{array}{c}
e1 = \text{Set}(\text{el1}, \text{type1}), \\
e2 = \text{Set}(\text{el2}, \text{type2}), \\
\text{env} \triangleright e1 \Rightarrow \{v_i : i \in \text{el1}\} : t_1, \\
\text{env} \triangleright e2 \Rightarrow \{w_j : j \in \text{el2}\} : t_2, \\
t_1 = t_2, \\
\{v_i : i \in \text{el1}\} \not\subseteq \{w_j : j \in \text{el2}\} \\
\hline
\text{env} \triangleright \text{Subset}(e1, e2) \Rightarrow \text{Bool}(\text{false})
\end{array}$$

Per l'ordinamento ho usato l'usuale ordinamento per gli interi, la convenzione  $\text{Bool}(\text{false}) < \text{Bool}(\text{true})$  per i booleani e l'ordinamento lessicografico per le stringhe

$$\begin{array}{c}
e = \text{Set}(\text{els}, \text{type}), \\
\text{env} \triangleright e \Rightarrow \{v_i : i \in \text{els}\} : t, \\
\text{SetMin:} \quad \{v_i : i \in \text{els}\} \neq \emptyset, \\
\quad w = \min \{v_i : i \in \text{els}\} \\
\hline
\text{env} \triangleright \text{SetMin}(e) \Rightarrow w
\end{array}
, \quad
\begin{array}{c}
e = \text{Set}(\text{els}, \text{type}), \\
\text{env} \triangleright e \Rightarrow \{v_i : i \in \text{els}\} : t, \\
\{v_i : i \in \text{els}\} = \emptyset \\
\hline
\text{env} \triangleright \text{SetMin}(e) \Rightarrow \text{Unbound}[t]
\end{array}$$
  

$$\begin{array}{c}
e = \text{Set}(\text{els}, \text{type}), \\
\text{env} \triangleright e \Rightarrow \{v_i : i \in \text{els}\} : t, \\
\text{SetMax:} \quad \{v_i : i \in \text{els}\} \neq \emptyset, \\
\quad w = \max \{v_i : i \in \text{els}\} \\
\hline
\text{env} \triangleright \text{SetMax}(e) \Rightarrow w
\end{array}
, \quad
\begin{array}{c}
e = \text{Set}(\text{els}, \text{type}), \\
\text{env} \triangleright e \Rightarrow \{v_i : i \in \text{els}\} : t, \\
\{v_i : i \in \text{els}\} = \emptyset \\
\hline
\text{env} \triangleright \text{SetMax}(e) \Rightarrow \text{Unbound}[t]
\end{array}$$

## Semantica degli operatori funzionali

Per quanto riguarda i predicati l'interprete supporta soltanto funzioni non ricorsive, in quanto non è possibile effettuare il bind del parametro formale di una funzione ricorsiva senza valutare anche il corpo del LetRec a causa del modo in cui sono definite, a meno di non forzare il bind prima della valutazione dell'espressione (come ho fatto nei test sulle funzioni ricorsive), ma dovrei poter stabilire che la funzione è ricorsiva senza valutarla e chiaramente ciò comporterebbe una notevole complicazione sia nelle regole che nella valutazione. La stessa cosa vale per la funzione che Map prende come argomento.

$$\begin{array}{c}
e = \text{Set}(\text{els}, \text{type}), \\
\text{env} \triangleright e \Rightarrow \{v_i : i \in \text{els}\} : t, \\
\text{Forall, divisa tra i due casi:} \quad \text{env} \triangleright \text{pred} \Rightarrow \text{FunVal}(\text{arg}, \text{body}, \text{decEnv}) : \text{Bool}(\_) \text{ as } f, \\
\quad (\forall j. j \in \{v_i : i \in \text{els}\} \wedge (\text{decEnv} \triangleright \text{FunCall}(f, j) \Rightarrow \text{Bool}(\text{true}))) \\
\hline
\text{env} \triangleright \text{Forall}(\text{pred}, e) \Rightarrow \text{Bool}(\text{true})
\end{array}$$

$$\begin{array}{c}
e = \text{Set}(els, type), \\
env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \\
env \triangleright pred \Rightarrow \text{FunVal}(arg, body, decEnv) : \text{Bool}(\_) \text{ as } f, \\
(\exists j. j \in \{v_i : i \in els\} \wedge (decEnv \triangleright \text{FunCall}(f, j) \Rightarrow \text{Bool}(false))) \\
\hline
env \triangleright \text{Forall}(pred, e) \Rightarrow \text{Bool}(false)
\end{array}$$

Exists, divisa tra i due casi:

$$\begin{array}{c}
e = \text{Set}(els, type), \\
env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \\
env \triangleright pred \Rightarrow \text{FunVal}(arg, body, decEnv) : \text{Bool}(\_) \text{ as } f, \\
(\exists j. j \in \{v_i : i \in els\} \wedge (decEnv \triangleright \text{FunCall}(f, j) \Rightarrow \text{Bool}(true))) \\
\hline
env \triangleright \text{Exists}(pred, e) \Rightarrow \text{Bool}(true)
\end{array}$$

$$\begin{array}{c}
e = \text{Set}(els, type), \\
env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \\
env \triangleright pred \Rightarrow \text{FunVal}(arg, body, decEnv) : \text{Bool}(\_) \text{ as } f, \\
(\forall j. j \in \{v_i : i \in els\} \wedge (decEnv \triangleright \text{FunCall}(f, j) \Rightarrow \text{Bool}(false))) \\
\hline
env \triangleright \text{Exists}(pred, e) \Rightarrow \text{Bool}(false)
\end{array}$$

Filter:

$$\begin{array}{c}
e = \text{Set}(els, type), \\
env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \\
env \triangleright pred \Rightarrow (\text{Rec}) \text{FunVal}(arg, body, decEnv) : \text{Bool}(\_) \text{ as } f, \\
x = \{j : j \in \{v_i : i \in els\} \wedge (decEnv \triangleright \text{FunCall}(f, j) \Rightarrow \text{Bool}(true))\} \\
\hline
env \triangleright \text{Filter}(pred, e) \Rightarrow x : t
\end{array}$$

Map:

$$\begin{array}{c}
e = \text{Set}(els, type), \\
env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \\
env \triangleright funct \Rightarrow \text{FunVal}(arg, body, decEnv) : new\_t \text{ as } f, \\
(\forall j. j \in \{v_i : i \in els\} \wedge (decEnv \triangleright \text{FunCall}(f, j) \Rightarrow x_j)) \\
\hline
env \triangleright \text{Map}(funct, e) \Rightarrow \{x_j : j \in \{v_i : i \in els\}\} : new\_t
\end{array}$$

Map supporta anche funzioni del tipo  $f: \text{type1} \rightarrow \text{type2}$ , per cui introduco il tipo `new_t` per specificare che il tipo di set ritornato potrebbe essere diverso da quello in input. Questo porta ad un comportamento semanticamente non corretto nel caso di Set vuoti, perché in tal caso non esiste alcun elemento su cui valutare il tipo di ritorno della funzione, per cui mi limito a ritornare un `SetVal` vuoto con il tipo originale. Un modo di ovviare a questo comportamento potrebbe essere quello di provare la chiamata di funzione con un valore di default per ogni tipo all'interno di blocchi `try ... with` concatenati, ma ho deciso di mantenere più semplice la valutazione in questo caso.