Semantica operazionale

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

Regole di inferenza

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

Concat: $$\frac{\begin{array}{c} env \triangleright e1 \Rightarrow String(x : string), \\ env \triangleright e2 \Rightarrow String(y : string) \end{array}}{env \triangleright Concat(e1, e2) \Rightarrow String(x \wedge y)}$$

dove con "^" ho rappresentato l'operazione di concatenazione di stringhe

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

Singleton: $$\frac{\begin{array}{c} env \triangleright type \Rightarrow String(t : string), \\ env \triangleright el \Rightarrow v : t \end{array}}{env \triangleright Singleton(el, type) \Rightarrow \{v\} : t}$$

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

IsEmpty, divisa tra valutazione true e false:

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

Contains, divisa tra valutazione true e false:

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

Insert: $$\frac{\begin{array}{c} env \triangleright type \Rightarrow String(t : string), \\ env \triangleright e \Rightarrow \{v_i : i \in els\} : t, \\ env \triangleright elem \Rightarrow w : el\_type, \\ el\_type = t \end{array}}{env \triangleright Insert(e, elem) \Rightarrow \{v_i : i \in els\} U \{w\} : t}$$

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

Subset, divisa nei tre casi: a ⊆ b, a ⊈ b e se i tipi di a e di b sono diversi, quindi a non può essere sottoinsieme di b

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

$$\frac{\begin{array}{c} e1 = Set(el1, type1), \\ e2 = Set(el2, type2), \\ env \triangleright type1 \Rightarrow String(t_1 : string), \\ env \triangleright type2 \Rightarrow String(t_2 : string), \\ t_1 \neq t_2 \end{array}}{env \triangleright \text{Subset}(e1, e2) \Rightarrow Bool(false)}$$

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

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

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

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 il corpo del LetRec. La stessa cosa vale per la funzione che Map prende come argomento.

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

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

Exists, divisa tra i due casi:

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

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

Filter:

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

Map:

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

Map supporta anche funzioni del tipo f: type1 → 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.