Polimorfismo di sottotipo

Introduciamo una *relazione di sottotipo* $<:$ e la regole:

$$\frac{\Gamma \vdash e : S \quad S <: T}{\Gamma \vdash e : T}\ (\textbf{subsumption})$$

$$\frac{S <: S' \quad S' <: T}{S <: T}\ (\textbf{transitività}) \qquad S <: S\ (\textbf{riflessività})$$

L'idea è permettere l'utilizzo del tipo "più specifico" S dove ci aspettiamo quello "meno specifico" T.

Top

È utile avere un tipo che è supertipo di tutti gli altri (in Java è Object):

$$S <: \text{Top}$$

Record

Sottotipo strutturale:

- un record con più campi è sottotipo di uno con meno campi:

$$\{f_1 = \tau_1, \ldots, f_{n+k} = \tau_{n+k}\} <: \{f_1 = \tau_1, \ldots, f_n = \tau_n\}$$

- l'ordine dei campi è irrilevante:

$$\frac{(f_1, \ldots, f_n) \text{ è una permutazione di } (l_1, \ldots, l_n)}{\{f_1 = \tau_1, \ldots, f_n = \tau_n\} <: \{l_1 = \tau_1, \ldots, l_n = \tau_n\}}$$

- il sottotipo opera ricorsivamente sui campi:

$$\frac{\forall i \in \{1, \ldots, n\}\ .\ S_i <: T_i}{\{f_1 = S_1, \ldots, f_n = S_n\} <: \{f_1 = T_1, \ldots, f_n = T_n\}}$$

Funzioni

Possiamo usare al posto di $f : T_1 \to T_2$ una funzione che accetta valori meno specifici di T_1 e restituisce valori più specifici di T_2:

$$\frac{T_1 <: S_1 \quad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2}$$

Il tipo della funzione è *controvariante* nel tipo del parametro e *covariante* nel tipo di ritorno.

Liste

Covariante:

$$\frac{S <: T}{\text{List } S <: \text{List } T}$$

Ref

In lettura ci aspettiamo un valore di tipo T, quindi il ref può contenere un valore di tipo $S <: T$. Se però in scrittura usassimo $S <: T$, potremmo reinterpretare il contenuto ref come $S' <: T$ e $S \not<: S'$, quindi avremmo un Ref S' che non contiene un S'.

$$\frac{S <: T \quad T <: S}{\text{Ref } S <: \text{Ref } T}$$

In Java gli array (se dichiarati come `T[]` anziché `Array<T>`) sono covarianti, infatti possono verificarsi situazioni problematiche.

Per esempio, due tipi distinti S e T che rispettano la condizione sono due record con stessi campi in ordine diverso.

Oggetti

Nei linguaggi object-based sottotipo strutturale come visto sopra per i record (e con i ref per i campi mutabili), in quelli class-based il tipo di un oggetto corrisponde alla classe da cui è stato istanziato, e:

$$\frac{\texttt{A extends B}}{A <: B}$$

In OCaml (strutturale) il tipo di un oggetto è determinato solo in base ai metodi.