Giorno 4
Dati

Sommario

1 Polimorfismo

1.1 Idea

Fin ora abbiamo considerato il polimorfismo *parametrico*, quello in cui si hanno tipi con un parametro (e.g. la $< T >$ dei generics in Typescript.)

Vogliamo edesso parlare di **polimorfismo per sottotipo**, ossia vogliamo definire, ad esempio, delle funzioni che accettano come parametro sia un certo tipo T che i suoi "sottotipi", che non sono altro che versioni più specializzate di un certo tipo.

Esempio classico Se si hanno:

- Il record "persona", con i campi "nome" e "cognome"
- Il record "studente" uguale a "persona" ma con in più un campo "matricola"

Allora "studente" può essere visto come **sottotipo** di "persona", scriviamo:

```
studente <: persona
```

1.2 Sottotipi

1.2.1 Subsumption

La regola di *subsumption* ci permette di applicare funzioni che accettano come parametro sia dati di un tipo che dati dei suoi sottotipi.

$$\frac{\Gamma \vdash e : S \quad S <: T}{\Gamma \vdash e : T}$$

Ossia "se un'espressione è di tipo S e $S <: T$ allora e è anche di tipo T"

1.2.2 Definizione della relazione di sottotipo

Data la subsumption, il problema diventa la definizione della relazione di sottotipo. Alcuni linguaggi (class based) utilizzano l'estensione, altri (object based) il subtyping strutturale (i.e. A sottotipo di B se ha i campi di B + altri campi). Un esempio di subtyping strutturale è la relazione di sottotipo per i record:

Subtyping dei record

$$\{l_i = v_i \quad i = 1, \cdots, k+n\} <: \{l_i = v_i \quad i = 1, \cdots, k\}$$

$$\frac{\{k_i = u_i \quad i = 1, \cdots, n\} \text{ permutazione di } \{l_j = v_j \quad j = 1, \cdots, n\}}{\{k_i = u_i \quad i = 1, \cdots, n\} <: \{l_j = v_j \quad j = 1, \cdots, n\}}$$

Caso particolare - il record ha campi di tipo record:

$$\frac{\forall i \quad S_i <: T_i}{\{l : S_i \quad i = 1, \cdots, n\} <: \{l_j : T_j \quad j = 0, \cdots, n\}}$$

e.g. una lista di studenti è sottotipo di una lista di persone.

1.2.3 Top

È conveniente avere un tipo che sia supertipo di ogni tipo: lo chiamiamo ***Top*** (Object in Java)

1.3 Funzioni

1.3.1 Subsumption per le funzioni

Vogliamo che una funzione $S_1 \to S_2$ possa prendere un argomento di tipo $T_1 <: S_1$ e restituire un risultato di tipo $T_2 :> S_2$

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

Diciamo che la relazione di sottotipo è

- **Covariante** per il risultato delle funzioni, dove *covariante* significa che la relazione d'ordine è conservata; in generale per un operatore F:

  $$\frac{S :< T}{F\langle S\rangle <: F\langle T\rangle}$$

  Dove con $F\langle T\rangle$ indico un operatore che prende (o restituisce) tipo T o un **costruttore di tipo** che prende come parametro[1] T.

- **Controvariante** per il parametro della funzione, dove *controvariante* significa che la relazione di sottotipo è invertita.

1.4 Altri esempi

1.4.1 Classi in Java

In java una sottoclasse **non** può cambiare i tipi dei parametri dei **metodi**, ma può rendere più specifico il tipo del risultato (**covariante** sul risultato (come le funzioni prima), e **invariante** sugli argomenti):

$$\frac{S_1 <: T_1}{m : S \to S_1 <: m : S \to T_1}$$

1.4.2 Liste

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

List è un costruttore di tipo covariante.

1.5 Riferimenti e puntatori

1.5.1 Riferimenti vs puntatori

Riferimenti e puntatori sono simili, ma la differenza sta nel fatto che per i puntatori esiste un'**aritmetica dei puntatori**, i.e. si possono fare operazioni aritmetiche sulle locazioni di memoria, mentre i riferimenti permettono solo di creare delle variabili che accedono alla stessa locazione di memoria.

1.5.2 Aliasing

Problema:

$$\lambda r : Ref\ Nat.\lambda s Ref\ Nat.(r := 2, s := 3; !r)$$

Restituisce 2, a meno che s non sia un alias di r.

I compilatori effettuano analisi degli alias per cercare di stabilire quando variabili diverse non si riferiscono alla stessa memoria.

1.5.3 Regole di tipo

$$\frac{\Gamma \vdash e_1 : \tau_1}{\Gamma \vdash ref\ e_1 : ref\ \tau_1} \qquad \frac{\Gamma \vdash e_1 : ref\ \tau_1}{\tau \vdash !e_1 : \tau_1} \qquad \frac{\Gamma \vdash e_1 : ref\ \tau_1 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 := e_2 : unit}$$

[1] Vedi: `en.wikipedia.org/wiki/Type_constructor` e `en.wikipedia.org/wiki/List_(abstract_data_type)`

2 Abstract Data Types

Un **tipo di dato astratto** consiste di un **insieme di dati** e una **collezione di operazioni** per operare sui dati di quel tipo.

- **estendibili**: si possono costruire nuovi ADT
- **Astratti**: Rappresentazione nascosta agli utenti
- **Incapsulati**: Si opera con i dati solo attraverso operazioni dell'ADT

2.1 Specifica ed implementazione

La **specifica** descrive la semantica delle operazioni di un tipo di dato astratto; uno stesso ADT può avere diverse implementazioni, ma questa differenza sarà invisibile al programmatore.

2.1.1 Approcci alla definizione di ADT

- **Signature + Axioms**: si definiscono le operazioni e i loro *tipi*, e delle proprietà astratte da rispettare nella forma di *assiomi*

```
Stack <E>
  Signature                              //Operazioni significative
      new     : -> STACK
      push    : E, STACK
      top     : STACK -> E
      pop     : STACK -> STACK
      isEmpty : STACK -> Bool
      undef_e : -> E                     //Completezza (elem/stack vuoto)
      undef_s : -> STACK

  Axioms                                 // Proprieta' astratte da rispettare
      forall e: E, stk: STACK
      top(push(e, stk)) = e ;
      top(new) = undef_e ;
      pop(push(e, stk)) = stk ;
      pop(new) = undef_s ;
      isEmpty(new) ;
      ~isEmpty(push(e, stk))

```

2.2 Moduli in OCaml

2.2.1 Signature

```
module type BOOL = sig
   type t
   val yes : t
   val no  : t
   val choose : t -> 'a -> 'a -> 'a
end
```

2.2.2 Esempio: Implementazioni equivalenti di BOOL

```
module M1 : BOOL = struct
   type t = unit option
   let yes = Some ()
   let no  = None
   let choose v ifyes ifno =
       match v with
       | Some () -> ifyes
       | None -> ifno
end
```

```
module M1 : BOOL = struct
  type t = int
  let yes = 1
  let no  = 0
  let choose v ifyes ifno =
      match v with
```

```
7           | 1 -> ifyes
8           | 0 -> ifno
9   end
```