Tipi di Dato

Abbiamo visto che OCaml offre vari tipi di dato

- int
- float
- char
- string
- bool
- ⋮

} tipi primitivi

- $\alpha, \beta, \ldots$ } variabili di tipo $\Rightarrow$ polimorfismo

- $T \to S$
- T list
- T option

} costruttori di tipo

Altri tipi di dato interessanti:

- record
- tuple
- tipi di dato algebrici
- ⋮

Record

```
type studente = {
    nome : string
    matricola : int
}
```

definizione di tipo (cf. dichiarazione)

campi

```
let george = {
    nome = "george boole"
    matricola = 01
}
```

~> val george : studente

george.matricola ~> 01 : int

Sintassi

tipo record: $\{ campo_1 : T_1 ; \ldots ; campo_n : T_n \}$

$campo_1$, $campo_n$: identificatori.

espressioni: $t ::= \ldots \mid \{ campo_1 = t_1 ; \ldots ; campo_n = t_n \} \mid e.campo$

$campo_1$, $campo_n$: identificatori.

valori: $v ::= \ldots \mid \{ campo_1 = v_1 ; \ldots ; campo_n = v_n \}$

Statica

$$\frac{t_1 : T_1 \quad \dots \quad t_m : T_m}{\{ c_1 = t_1 ; \dots c_m = t_m \} : \{ c_1 : T_1 ; \dots ; c_m : T_m \}}$$

Dinamica

$$\frac{t_1 \Rightarrow v_1 \quad \dots \quad t_m \Rightarrow v_m}{\{ c_1 = t_1 ; \dots c_m = t_m \} \Rightarrow \{ c_1 = v_1 ; \dots ; c_m = v_m \}}$$

$$\frac{t \Rightarrow \{ c_1 : v_1 ; \dots ; c_m ; v_m \}}{t . c_i \Rightarrow v_i}$$

NB. I record sono immutabili:

```
type persona = {
    nome : string
    eta : int
}

val pippo = {
    name = "pippo"
    eta = "99"
}
```

Non posso fare

```
pippo.name = pippo.name + 1
```

Non posso aggiungere campi → cambiamento tipo

Posriamo pensare ad un record come ad una forma basica ed immutabile di classe

```
type calcolatrice = {
    capacita : int          -> "campi"
    add: int -> int -> int option   -> "metodi"
}

val calc1 = {
    capacita = 100
    add = fun n m ->
            let p = n+m
            in if p < 100
               Then some p
               else none
}
```

Possiamo chiamare

```
calc1.add 10 10
```

(~> some 20)

NB. Non possiamo aggiornare la capacità

Tuple

Le tuple sono record i cui campi sono dati: dalla posizione nella tupla

⇓

ordine è cruciale (irrilevante nel record)

```
type punto = float * float
let zero = (.0, .0)
let plus_1 (p: punto) : punto =
    match p with
        (x,y) -> (x+.1, y+.1)
```

} possiamo usare pattern matching anche per i record

Un tipo particolare di tupla sono le coppie

$$T_1 * T_2$$

Data $p : T_1 * T_2$, possiamo usare

$p.fst$

$p.snd$

per accedere agli elementi della coppia

Domanda. Possiamo pensare a fst, snd come funzioni su coppie: qual è il loro tipo

Domanda. Possiamo pensare al tipo tupla $int * string * bool$ come zucchero sintattico per un tipo coppia?

Sintassi

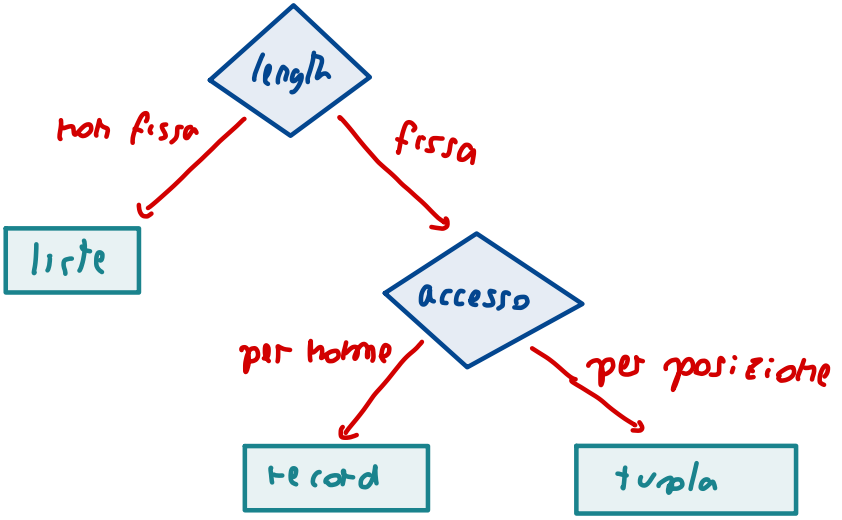
tipi: $T ::= \ldots \mid T_1 + \ldots * T_m$

espressioni: $e ::= \ldots \mid (e_1, \ldots, e_m)$

$v ::= \ldots \mid (v_1, \ldots, v_m)$

Semantica. ??

Liste, Record, Tuple



VARIANTI (VARIAMS)

Possiamo usare i tipi varianti per enumerare le costanti di un determinato tipo

```
type colore_primario = Rosso | Blu | Verde
```

lettera maiuscola

stiamo dicendo che tutti i valori di questo tipo sono delle costanti, e le elenchiamo

```
let a = Red   ~> val a : colore_primario
```

Creiamo un variante per figure geometriche

```
type point = float * float
type figure =
  | Circle of point * float
  | Rectangle of point * point
```

costruttore

informazione aggiuntiva

(| Circle of {center: point; radius: float})

↳ tutti e soli i valori di ti o figure p sono della forma

Circle p r oppure Rectangle p q

Domanda Quanti valori contiene primary_color ?

E figure ?

Possiamo utilizzare espressioni di tipo variante usando il pattern matching

```
let centro (f. figura) : point =
    match f with
    | Circle p r -> p
    | Rectangle p q ->
        let (x1,y1) = p in
        let (x2,y2) = q in
        let media = fun a b -> a +. b /. 2. in
        (media x1 x2, media y1 y2)
```

pattern matching

```
match p with
| (x1,y1) -> match q with
             | (x2,y2) ->
             ...
```

Qual è il significato intuitivo di un tipo variante?

```
type figure =
  | Circle of point * float
  | Rectangle of point * point
```

Stiamo dicendo che un valore di tipo figure è, di fatto,
un valore point * float oppure un valore point * point

⇒ abbiamo una sorta di: unione di tipi

I costruttori fungono da tag e fanno sì che l'unione sia disgiunta
somma di tipi T + S

```
type figure =
  | Rectangle of point * point
  | Square of point * point
```

I tipi variant: sono chiamati anche tipi algebrici, essendo somme di prodotti (o record)

figure ≈ (point * float) + (point * point) + (point * point)

SINTASSI

tipi $T ::= \ldots \mid C_1 \ [\text{of } T_1] \mid \ldots \mid C_n \ [\text{of } T_n]$

↳ costruttori

opzionale; diciamo allora che C_n è una costante

NB. OCaml consente l'utilizzo di tipi di dato algebrici tramite type definitions

type <identifier> = C_1 of T_1 | ... | C_n of T_n

↳ utile per ricorsione (vedremo dopo)

espressioni $e ::= \ldots \mid C \mid C\ e \mid$ match e with $p_1 \to e_1 \mid \ldots \mid p_n \to e_n$

valori $v ::= \ldots \mid C \mid C\ v$

STATICA

$$\frac{e : T \qquad [\,\text{type } T = C_1 \text{ of } T_1 \mid \ldots \mid C_n \text{ of } T_n\,]}{C\ e : C_1 \text{ of } T_1 \mid \ldots \mid C_n \text{ of } T_n \quad [T]}$$

DINAMICA

$$\frac{e \Downarrow v}{C\ e \Downarrow C\ v}$$

$$\frac{e \Downarrow C\ v \qquad \text{Match}(v, p) \qquad s[v/p] \Downarrow w}{\text{case}\ C\ e\ \text{of}\ \ldots \mid C\ p \to s \mid \ldots \Downarrow w}$$

↳ estendiamo la grammatica dei pattern con

$p ::= \ldots \mid C\ p \mid \ldots$