Caratteristiche di Caml

- è un linguaggio funzionale implementazione del λ-calcolo (Church A. 1932): usa la definizione e l'applicazione di funzioni come concetti essenziali.
  - Integrità referenziale
  - Ordine superiore: le funzioni sono valori del linguaggio (gli unici nel λ-calcolo puro).

Caratteristiche del sistema

- È un sistema interattivo: (di base ma esistono anche compilatori)
  - Si entra in un ambiente in cui il sistema stampa un prompt (#) e resta in attesa di una richiesta da parte dell'utente.
  - L'utente può immettere una frase in Caml (quali le frasi corrette?).
  - Il sistema calcola, stampa il risultato e un nuovo prompt.

Caratteristiche di Caml

- Frasi corrette in Caml:
  - Espressioni: applicazioni di funzioni definite nell'ambiente.
  - Definizioni di funzioni:
    1. globali: let rec …;; (let)
    2. locali: let rec … in …;;
  - Ambiente:
    - contiene le definizioni delle funzioni
    - è arricchito dal let rec o let (1)
  - Gli identificatori (nomi di variabili) sono associati a valori (funzioni), mai modificabili.

Integrità referenziale

- Integrità referenziale:
  - le espressioni denotano valori
  - Il significato di tutte le espressioni (in Caml e in tutti i linguaggi funzionali) è SOLO il valore. Qualunque metodo venga utilizzato per ottenerlo non produce altri effetti (non vero per gli altri linguaggi di programmazione)
  - Il valore (o forma più semplice di una espressione) è in realtà una rappresentazione di tale valore (rappresentazione dell'intero 10).

Riduzione di espressioni

- Valutazione di un'espressione, è un processo di riduzione o semplificazione (espressione canonica o in forma normale).
  - (2+3*4) → 2+12 → 14.
  - double (3+2) → double (5) → 5+5 → 10
- A volte ci sono più sequenze di riduzione possibili (ma in genere (?) calcolano lo stesso risultato):
  - es. : double (3+2) → (3+2)+ (3+2) → 5+(3+2) → 5+5 → 10

I tipi in Caml

- Ogni espressione ha un tipo (linguaggio fortemente tipato).
- I tipi delle espressioni vengono calcolati dal sistema (inferenza dei tipi). Non è necessario dichiarare i tipi.
- I tipi primitivi sono:
  - int: 0, 1,2,...-1,-2.. operazioni + , - , * , ecc
  - float: 1.2, 3.2, -4.5,... operazioni +. , -. , ecc.
  - boolean: true, false, operazioni & , or , not,
  - char, ‘a’,’b’,..
  - string: “Anna”,.. operazioni ^,s.[n], length

I tipi in Caml

- Tipi complessi:
  - Predefiniti:
    - Funzioni type -> type
    - Prodotto cartesiano: (type * type *..)
    - Liste: type list
  - Definibili dall'utente:
    - Tipo unione con costruttori di tipo: per valori eterogenei, ogni componente dell'unione ha un costrutore.

I tipi funzione

- Funzioni; type -> type
  - una funzione molte definizioni.
- applicazione parziale: f1 x y=x+y f2(x,y)=x+y
- currizzazione:
  - curry f x y=f(x,y)
  - uncurry f (x,y)= f x y
- operatori infissi e prefissi. + e (+) ecc.

Inferenza dei tipi

compose è una funzione che ha 3 argomenti cosi` definita:

let rec compose f g x = f(g(x));;

- cosa calcola? inferiamo il tipo.
  1. f:a'
  2. g:b':
  3. x:c'
  4. compose :a'->b'->c'->m
- da g(x) abbiamo che b'=c'->d'
- da f(g(x) abbiamo che a'=d'->e'
- sostituendo in 1,2 e 4 abbiamo:
  - f:d'->e' e g: c'->d'
  - compose:(d'->e')(c'->d')->c'->e'

Inferenza dei tipi

let rec const x y =x

let rec foo f g x y =if f(x) then g(y) else f(y)

let rec subst f g x =f x (g x)

uncurry compose

compose uncurry curry

compose curry uncurry

Il principio d'induzione

Permette di dimostrare una proprietà P su un dominio S sulla quale è definita una relazione di ordinamento ($\prec$). (riflessiva, antisimmetrica e transitiva)

Un esempio classico (e semplice) è l'insieme N dei naturali, con la relazione di ordinamento $<$ definita come $x < y$ se $y=x+1$. 0 è l'elemento minimo.

$\forall n \in N \ \#\{y \mid y<n\} = n$ (cardinalità dell'insieme finita)

ad es. n=5 ho k=5 predecessori: 0<1<2<3<4<5 (catena)

Il principio d'induzione

Permette di dimostrare una proprietà P sui naturali:

1. Dimostro P(0)
2. Dimostro P(n+1) dato P(n) (ipotesi induttiva).

   Se è vero P(0) per 1. per 2. è vero P(1), e quindi P(2) ...ecc.

Dimostro: $\sum_{i\in[0,n]} i = n*(n+1)/2$

Il principio d'induzione

Dimostro: $\sum_{i\in[0,n]} i = n*(n+1)/2$

$0+1+2+...n= n*(n+1)$

Caso base: $0=0*(1)/2=0$

Suppongo vero per n $\sum_{i\in[0,n]} i = n*(n+1)/2$ dimostro per n+1

$\sum_{i\in[0,n+1]} i = (n+1)*(n+1+1)/2$ sviluppo (sn)

$(\sum_{i\in[0,n]} i)+n+1= (n+1)*(n+2)/2$ ipotesi induttiva

$(n*(n+1))/2+n+1= (n+1)*(n+2)/2$ sviluppo i prodotti

$(n^2+n)/2+n+1=(n^2+2n+n+2)/2$ stesso denominatore

$(n^2+n+2n+2)/2=(n^2+2n+n+2)/2$

Induzione strutturale

Permette di dimostrare una proprietà P su un dominio S sulla quale è definita una relazione di ordinamento ($\prec$).

Una catena è un sottoinsieme (finito o infinito) di S totalmente ordinato

$$\ldots s_{m-1} \prec s_m \prec s_{m+1} \prec \ldots$$

dove s_i è il predecessore immediato di s_{i+1}

Tutti gli elementi sono confrontabili ($s_i \neq s_j \Rightarrow s_i \prec s_j$ or $s_j \prec s_i$

Induzione strutturale

(S, ‹) è ben fondato se solo se non ammette catene discendenti infinite (finiti predecessori).

Quindi esistono uno (o più) elementi minimali m.

Un elemento m è minimale in S se $\forall$ s$\in$S non è vero che s ‹ m.

Induzione sulle liste

Insieme L di tutte le liste (L,‹), dove ‹ è la relazione

xs ‹ ys se e solo se ∃ x tale che x::xs =ys.

Tale relazione è antisimmetrica e non contiene cicli infatti: xs≠ys non può essere che esistano x1 e x2 tali che xs=x_1::ys e ys=x_2::xs cioè non può essere che xs ‹ ys e ys ‹ xs.

Induzione sulle liste

Il minimo è []

∀ lista $[x_1;\ldots;x_i]$ ho finiti predecessori

$[] \prec [x_1] \prec [x_1;x_2] \prec [x_1;x_2;x_3] \ldots \prec [x_1;x_2;x_3 \ldots x_{i-1}]$

in questo caso ho infinite catene di lunghezza i diversamente dai naturali

Tipi unione (o somma)

- Tipi che raccolgono valori eterogenei:
  - hanno costruttori senza o con argomenti. Es.
    - type coloreB= Rosso | Giallo | Blu;;

      definizione del tipo coloreB che in questo caso ha 3 valori. Rosso, Giallo e Blu che sono costruttori (di valori di tipo colore).
  - I costruttori di tipo iniziano sempre con la maiuscola.
  - funzioni con il match:

```
let convert x = match x with
Rosso -> 1
Giallo -> 2
Blu -> 3;;
```

Tipi unione (o somma)

- Tipi unione con costruttori con argomenti:
  - type alberoBin = Foglia string
    | Node of string * alberoBin * alberoBin
  - definizione del tipo alberoBin per rappresentare alberi con 2 figli:
  - let a=Node (“a”,Node(“b”,(Foglia “c”),(Foglia “d”)), Node(“e”,(Leaf “f”),(Leaf “g”)));;
  - tipo aExp

Tipi polimorfi

- Tipi polimorfi definiti dall'utente parametrici rispetto a uno o più tipi:
  - type 'a alberoBin = Foglia 'a
    | Node of 'a * 'a alberoBin * 'a alberoBin
  - definizione del tipo alberoBin per rappresentare alberi con 2 figli:
  - let a=Node ("a",Node("b",(Foglia "c"),(Foglia "d")), Node("e",(Leaf "f"),(Leaf "g")));;
  - La variabile di tipo 'a è istanziata con il tipo string.
  - il tipo 'a mu per la memoria