

ARITMETICA E RICORSIONE

- Non esiste, in logica, alcun meccanismo per la *valutazione* di funzioni, operazione fondamentale in un linguaggio di programmazione
- I numeri interi possono essere rappresentati come termini Prolog.
 - Il numero intero N è rappresentato dal termine:

$$\underbrace{s(s(s(\dots s(0)\dots)))}_{N \text{ volte}}$$

```
prodotto(X, Y, Z) "Z è il prodotto di X e Y"  
prodotto(X, 0, 0).  
prodotto(X, s(Y), Z) :- prodotto(X, Y, W),  
                           somma(X, W, Z).
```

- Non utilizzabile in pratica: predicati predefiniti per la valutazione di espressioni

PREDICATI PREDEFINITI PER LA VALUTAZIONE DI ESPRESSIONI

- L'insieme degli atomi Prolog contiene tanto i numeri interi quanto i numeri floating point. I principali operatori aritmetici sono simboli funzionali (operatori) predefiniti del linguaggio. In questo modo ogni espressione può essere rappresentata come un termine Prolog.
- Per gli operatori aritmetici binari il Prolog consente tanto una notazione prefissa (funzionale), quanto la più tradizionale notazione infissa

TABELLA OPERATORI ARITMETICI

Operatori Unari	<code>-</code>, <code>exp</code>, <code>log</code>, <code>ln</code>, <code>sin</code>, <code>cos</code>, <code>tg</code>
Operatori Binari	<code>+</code>, <code>-</code>, <code>*</code>, <code>\</code>, <code>div</code>, <code>mod</code>

- `+(2,3)` e `2+3` sono due rappresentazioni equivalenti. Inoltre, `2+3*5` viene interpretata correttamente come `2+(3*5)`

PREDICATI PREDEFINITI PER LA VALUTAZIONE DI ESPRESSIONI

- Data un'espressione, è necessario un meccanismo per la valutazione
- Speciale predicato predefinito **is**.

T is Expr (**is** (T, Expr))

- T può essere un atomo numerico o una variabile
 - Expr deve essere un'espressione.
- L'espressione **Expr** viene valutata e il risultato della valutazione viene unificato con T

Le variabili in **Expr** DEVONO ESSERE ISTANZIATE al momento della valutazione

ESEMPI

```
:- X is 2+3.  
yes X=5
```

```
:- X1 is 2+3, X2 is exp(X1), X is X1*X2.  
yes X1=5 X2=148.413 X=742.065
```

```
:- 0 is 3-3.  
yes
```

```
:- X is Y-1.  
No
```

Y non è istanziata al momento della valutazione

(NOTA: Alcuni sistemi Prolog danno come errore
Instantion Fault)

```
:- X is 2+3, X is 4+5.  
no
```

ESEMPI

```
:- X is 2+3, X is 4+1.  
yes      X=5
```

- In questo caso il secondo goal della congiunzione risulta essere:

```
:- 5 is 4+1.
```

che ha successo. `x` infatti è stata istanziata dalla valutazione del primo `is` al valore 5.

```
:- X is 2+3, X is X+1.  
no
```



NOTA: non corrisponde a un assegnamento dei linguaggi imperativi. Le variabili sono *write-once*

ESEMPI

Nel caso dell'operatore `is` l'ordine dei goal cioè rilevante.

(a) `:- X is 2+3, Y is X+1.`

(b) `:- Y is X+1, X is 2+3.`

- Mentre il goal (a) ha successo e produce la coppia di istanziazioni `X=5, Y=6`, il goal (b) fallisce.
- Il predicato predefinito "is" è un tipico esempio di un predicato predefinito non reversibile; come conseguenza le procedure che fanno uso di tale predicato non sono (in generale) reversibili.

TERMINI ED ESPRESSIONI

- Un termine che rappresenta un'espressione viene valutato solo se è il secondo argomento del predicato **is**

```
p(a, 2+3*5) .  
q(X, Y) :- p(a, Y), X is Y.  
:- q(X, Y).  
yes X=17 Y=2+3*5 (Y=+(2, *(3, 5)))
```


Valutazione di Y

NOTA: Y non viene valutato, ma unifica con una struttura che ha $+$ come operatore principale, e come argomenti 2 e una struttura che ha $*$ come operatore principale e argomenti 3 e 5

OPERATORI RELAZIONALI

- Il Prolog fornisce operatori relazionali per confrontare i valori di espressioni.
- Tali operatori sono utilizzabili come goal all'interno di una clausola Prolog ed hanno notazione infissa

OPERATORI RELAZIONALI

>, <, >=, =<, ==, /=  *disuguaglianza*



uguaglianza

CONFRONTO TRA ESPRESSIONI

- Passi effettuati nella valutazione di:

Expr1 REL Expr2

dove **REL** e' un operatore relazionale e **Expr1** e **Expr2** sono espressioni

- vengono valutate **Expr1** ed **Expr2**
- NOTA: le espressioni devono essere completamente istanziate
- I risultati della valutazione delle due espressioni vengono confrontati tramite l'operatore **REL**

CALCOLO DI FUNZIONI

- Una funzione può essere realizzata attraverso relazioni Prolog.
- Data una funzione f ad n argomenti, essa può essere realizzata mediante un predicato ad $n+1$ argomenti nel modo seguente

– $f : x_1, x_2, \dots, x_n \rightarrow y$ diventa
 $f(x_1, x_2, \dots, x_n, Y) :- \langle \text{calcolo di } Y \rangle$

- Esempio: calcolare la funzione fattoriale così definita:

$\text{fatt} : n \rightarrow n !$ (n intero positivo)

$\text{fatt}(0) = 1$

$\text{fatt}(n) = n * \text{fatt}(n-1)$ (per $n > 0$)

$\text{fatt}(0, 1).$

$\text{fatt}(N, Y) :- N > 0, N1 \text{ is } N-1, \text{fatt}(N1, Y1), Y \text{ is } N*Y1.$

CALCOLO DI FUNZIONI

- Esempio: calcolare il massimo comun divisore tra due interi positivi

`mcd: x, y → MCD(x, y) (x, y interi positivi)`

`MCD(x, 0) = x`

`MCD(x, y) = MCD(y, x mod y) (per y > 0)`

`mcd(X, Y, Z)`

`"Z è il massimo comun divisore di X e Y"`

`mcd(X, 0, X) .`

`mcd(X, Y, Z) :- Y > 0, X1 is X mod Y,
mcd(Y, X1, Z) .`

ESEMPI

- Calcolare la funzione

`abs(x) = |x|`

`abs(X, Y)`

`"Y è il valore assoluto di X"`

`abs(X, X) :- X >= 0.`

`abs(X, Y) :- X < 0, Y is -X.`

- Si consideri la definizione delle seguenti relazioni:

`pari(X) = true se X è un numero pari`
`false se X è un numero dispari`

`dispari(X) = true se X è un numero dispari`
`false se X è un numero pari`

`pari(0).`

`pari(X) :- X > 0, X1 is X-1, dispari(X1).`

`dispari(X) :- X > 0, X1 is X-1, pari(X1).`

RICORSIONE E ITERAZIONE

- Il Prolog non fornisce alcun costrutto sintattico per l'iterazione (quali, ad esempio, i costrutti *while* e *repeat*) e l'unico meccanismo per ottenere iterazione è la definizione ricorsiva.
- Una funzione f è definita per *ricorsione tail* se f è la funzione "più esterna" nella definizione ricorsiva o, in altri termini, se sul risultato della chiamata ricorsiva di f non vengono effettuate ulteriori operazioni
- La definizione di funzioni (predicati) per ricorsione tail può essere considerata come una definizione per *iterazione*
 - Potrebbe essere valutata in spazio costante mediante un processo di valutazione iterativo.

RICORSIONE E ITERAZIONE

- Si dice *ottimizzazione della ricorsione tail* valutare una funzione tail ricorsiva f mediante un processo iterativo ossia caricando un solo record di attivazione per f sullo stack di valutazione (esecuzione).
- In Prolog l'ottimizzazione della ricorsione tail è un po' più complicata che non nel caso dei linguaggi imperativi a causa del:
 - non determinismo
 - della presenza di punti di scelta nella definizione delle clausole.

RICORSIONE E ITERAZIONE

$p(X) \quad :- \quad c1(X), g(X) .$
(a) $p(X) \quad :- \quad c2(X), h1(X, Y), p(Y) .$
(b) $p(X) \quad :- \quad c3(X), h2(X, Y), p(Y) .$

- Due possibilità di valutazione ricorsiva del goal $:-p(Z)$.
 - se viene scelta la clausola (a), si deve ricordare che (b) è un punto di scelta ancora aperto. Bisogna mantenere alcune informazioni contenute nel record di attivazione di $p(Z)$ (i punti di scelta ancora aperti)
 - se viene scelta la clausola (b) (più in generale, l'ultima clausola della procedura), non è più necessario mantenere alcuna informazione contenuta nel record di attivazione di $p(Z)$ e la rimozione di tale record di attivazione può essere effettuata

QUINDI...

- In Prolog l'ottimizzazione della ricorsione tail è possibile solo se la scelta nella valutazione di un predicato "p" è deterministica o, meglio, se al momento della chiamata ricorsiva (n+1)-esima di "p" non vi sono alternative aperte per la chiamata al passo n-esimo (ossia alternative che potrebbero essere considerate in fase di backtracking)
- Quasi tutti gli interpreti Prolog effettuano l'ottimizzazione della ricorsione tail ed è pertanto conveniente usare il più possibile ricorsione di tipo tail.

RICORSIONE NON TAIL

- Il predicato `fatt` è definito con una forma di ricorsione semplice (non tail).
- Casi in cui una relazione ricorsiva può essere trasformata in una relazione tail ricorsiva

```
fatt1(N, Y) :- fatt1(N, 1, 1, Y) .
```

```
fatt1(N, M, ACC, ACC) :- M > N.
```

```
fatt1(N, M, ACCin, ACCout) :- ACCtemp is ACCin*M,  
M1 is M+1,  
fatt1(N, M1, ACCtemp, Accout) .
```

*Accumulatore
in ingresso*

*Accumulatore
in uscita*

RICORSIONE NON TAIL

- Il fattoriale viene calcolato utilizzando un argomento di accumulazione, inizializzato a 1, incrementato ad ogni passo e unificato in uscita nel caso base della ricorsione.

$$ACC_0=1$$

$$ACC_1= 1 * ACC_0 = 1 * 1$$

$$ACC_2= 2 * ACC_1 = 2 * (1*1)$$

...

$$ACC_{N-1}= (N-1) * ACC_{N-2} = N-1 * (N-2 * (\dots * (2 * (1 * 1)) \dots))$$

$$ACC_N = N * ACC_{N-1} = N * (N-1 * (N-2 * (\dots * (2 * (1 * 1)) \dots)))$$

RICORSIONE NON TAIL

- Altra struttura iterativa per la realizzazione del fattoriale

```
fatt2(N, Y)
```

```
    "Y è il fattoriale di N"
```

```
fatt2(N, Y) :- fatt2(N, 1, Y).
```

```
fatt2(0, ACC, ACC).
```

```
fatt2(M, ACC, Y) :- ACC1 is M*ACC,  
                    M1 is M-1,  
                    fatt2(M1, ACC1, Y).
```

RICORSIONE NON TAIL

- Calcolo del numero di Fibonacci: definizione

`fibonacci(0) = 0`

`fibonacci(1) = 1`

`fibonacci(N) =`

`fibonacci(N-1) + fibonacci(N-2) per N >1`

- Programma Prolog

```
fib(0,0).
```

```
fib(1,1).
```

```
fib(N,Y) :- N>1,  
            N1 is N-1,  
            fib(N1,Y1),  
            N2 is N-2,  
            fib(N2,Y2),  
            Y is Y1+Y2.
```