

# Prolog

---

## Esercitazione del 16 Aprile 2010

- *Scopo:*
  1. Acquisire conoscenza dell'ambiente SICStus Prolog
  2. Risolvere alcuni problemi mediante il linguaggio Prolog
    - Prendere dimestichezza con un approccio DICHIARATIVO
    - Ricorsione e ricorsione tail
    - Liste
    - Cut

1

## Istruzioni su SICStus

---

### Avvio di SICStus Prolog

Il sistema viene avviato, al prompt della shell, dal comando “`sicstus`”, o alternativamente, dal menù Avvio di Windows. All'avvio il sistema si identifica e presenta il proprio prompt, con un output del tipo:

```
SICStus 3 #3: Fri Dec 20 10:50:05 MET 1996  
| ?-
```

### Caricamento di programmi

I programmi Prolog possono essere caricati nel sistema mediante il comando “`consult`”:

```
| ?- consult(parenti).  
{consulting /export/users/michela/parenti.pl...}  
{/export/users/michela/parenti.pl consulted, 9 msec  
12000 bytes}  
yes  
| ?-
```

2

# Istruzioni su SICStus

---

## Caricamento di programmi

- In alternativa si può usare l'equivalente comando nel menù "File..."
- E' possibile specificare un file Prolog al momento dell'avvio del sistema:

```
c:\> sicstus -l parenti
```

(si noti che non e' necessario specificare l'estensione del file: SICStus cerca automaticamente file con estensione ".pl")

## Tracing

- Il comando "trace" consente di seguire passo passo la risoluzione delle clausole; tale modalità può poi essere disabilitata con il comando "notrace".

3

# Istruzioni su SICStus

---

## Dichiarazioni dinamiche

- Qualora si vogliano utilizzare i predicati "assert" e "retract" occorre porre l'attenzione su una particolarità del sistema SICStus: per ottimizzare le prestazioni il sistema compila i programmi (o meglio le basi di conoscenza) che vengono caricate. È quindi evidente il problema che si pone nel tentare di modificare a run-time un insieme di regole e fatti già compilati. Affinchè sia possibile aggiungere o togliere clausole (o fatti) ad una relazione è necessario dichiarare che tale relazione è da ritenersi dinamica, cioè non definita staticamente a priori. Ad esempio, la seguente dichiarazione previene la compilazione delle clausole relative alle relazioni **member/2** e **empty/1**:  

```
:- dynamic user: member/2, user : empty/1.
```
- Tale dichiarazione va inserita all'inizio del programma Prolog in cui si definiscono le relazioni in oggetto.

4

## Es. 1

---

Si definisca un predicato in PROLOG chiamato `maxlist` che applicato ad una lista di liste di interi `ListListInt` dia come risultato la lista degli elementi massimi di ogni lista componente di `ListListInt`. Si definisca prima la versione ricorsiva e poi quella ricorsiva-tail.

Esempio:

```
?-  
?- maxlist([[3,10,2], [6,9],[1,2]], X).  
yes, X = [10,9,2]
```

5

## Es. 1 - Soluzione

---

```
maxlist([],[]):-!.  
maxlist([X|Y],[N|T):- max(X,N),maxlist(Y,T).
```

Versione ricorsiva:

```
max([X],X):-!.  
max([X|T],X):- max(T,N),X>=N,!.  
max([X|T],N):- max(T,N).
```

Versione iterativa:

```
max([X|T],M):- max(T,X,M).  
max([],M,M):-!.  
max([H|T],MT,M):- H>MT,!,max(T,H,M).  
max([H|T],MT,M):- max(T,MT,M).
```

6

## Es.1 – Uso del !

---

- Il ! deve essere usato con scrupolosa attenzione
- Consideriamo il programma Prolog

```
max ( [X] , X ) :- ! .
max ( [X|T] , X ) :- max ( T , N ) , X >= N , ! .
max ( [X|T] , N ) :- max ( T , N ) .
```
- Il programma **funziona correttamente** quando invocato con secondo parametro (ovvero il massimo) **VARIABILE**
  - Es.: `max ( [1, 2, 4, 3] , M)`
- Il programma **NON** funziona correttamente quando invocato per testare se un certo valore è il massimo
- In particolare, poiché la terza clausola non contiene la condizione  $X < N$ , il programma restituisce *true*
  - Sia mettendo il valore massimo, es. `max ( [1, 2, 4, 3] , 4)`
  - Sia mettendo l'ultimo valore contenuto nella lista, es. `max ( [1, 2, 4, 3] , 3)`
- Si provino a disegnare gli alberi SLD per vedere cosa accade!

7

## Es. 2

---

Data una lista **L1** e un numero intero **N**, scrivere un predicato Prolog `domanda1 (L1, N, L2)` che restituisca in **L2** la lista degli elementi di **L1** che sono **liste** contenenti **solo due valori** interi positivi fra 1 e 9 la cui somma valga **N**.

Esempio:

```
:- domanda1 (
    [ [3,1], 5, [2,1,1], [3], [1,1,1], a, [2,2] ],
    4,
    L2) .
yes, L2 = [[3,1], [2,2]]
```

8

## Es. 2 - Soluzione

---

```
domanda1([],_,[]).
domanda1([[A,B]|R], N, [[A,B]|S]):-
    A>=1, A<9, B>=1, B<9,
    N is A + B, !,
    domanda1(R,N,S).
domanda1([_|R], N,S):- domanda1(R,N,S).
```

9

## Es. 3

---

Si scriva un predicato Prolog che data una lista ed un elemento **E1** appartenente alla lista, restituisca in uscita l'elemento **successivo** ad E1 nella lista.

Esempio:

```
?- consec(3, [1,7,3,9,11],X).
yes    X=9
```

Nel caso in cui E1 sia l'ultimo elemento il predicato dovrà fallire

10

## Es. 3 - Soluzione

---

```
consec(E1, [E1| [X|_]],X):-!.  
consec(E1, [_|Tail],X):- consec(E1,Tail,X).
```

11

## Es. 4 – tratto dal Compito del 29 marzo 2006

---

Un giorno il mago rosso venne sfidato dal mago verde ad un duello di veleni. Ciascuno dei due doveva portare il veleno più potente che era riuscito a produrre; ciascuno avrebbe bevuto prima il veleno dell'altro e poi il proprio.

Vale infatti la regola che se si beve un veleno e poi uno più potente, allora non si muore, ma il secondo fa da antidoto per il primo.

Il mago rosso accettò la sfida, sapendo che il mago verde aveva un veleno molto più potente del suo: quello del mago verde ha potenza 8, mentre il mago rosso ha solo dell'acqua (che non è un veleno e ha potenza 0) ed un veleno a potenza 1. Il mago rosso, però ha pensato di bere qualcosa *prima* della sfida ...

12

## Es. 4 – tratto dal Compito del 29 marzo 2006

---

Si scriva un predicato Prolog `vivo/1` che prende in ingresso una lista di numeri ed ha successo se bevendo quella sequenza di veleni si **sopravvive**.

Es:

```
?- vivo([1,8,0,1,4]).
```

```
yes
```

```
?- vivo([1,2,4,1]).
```

```
no
```

13

## Es. 4 – Soluzione

---

```
vivo([]).
```

```
vivo([0|L]):- !, vivo(L).
```

```
vivo([A,B|L]):- A < B, vivo(L).
```

14

## Es. vari

---

- Scrivere un predicato `flatten` che “appiattisce” una lista di liste. Ad esempio:

```
:- flatten([ 1,a,[2,3],[],h,f(3),[c,[d,[e]]] ],L) .  
yes, L=[1,a,2,3,h,f(3),c,d,e]
```

Suggerimento: cercare di riformulare il problema in maniera dichiarativa. Nella formulazione, assumere che “l’appiattimento di un elemento è una lista contenente quell’elemento”

- Data una matrice  $N \times N$  rappresentata come lista di liste, si calcoli la somma degli elementi della diagonale principale.
  - Provare a ragionare in termini di “sottomatrici”
- Si definisca un predicato Prolog che calcola i massimi locali (esclusi gli estremi) di una lista, ad esempio:

```
:- max_loc([5,4,7,2,3,6,1,2],X)  
yes, X=[7,6]
```