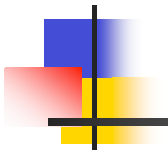
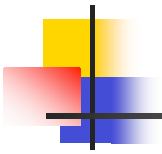


Il Linguaggio Prolog

liste, aritmetica,
strutture dati



LISTE in Prolog

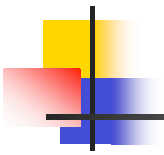
In Prolog una lista è rappresentata con un termine così definito:

- [] rappresenta la lista vuota
- [t1|t2] rappresenta la lista il cui primo elemento è rappresentato da t1 ed il resto è rappresentato da t2.

Sono definite delle abbreviazioni:

[a, b] per [a | [b|[]]]

[a,b| X] per [a | [b| X]]



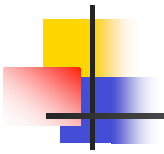
esempio: "lungh"

- lungh (L, N) "N è la lunghezza della lista L"

```
lungh([], 0).
lungh([ _ |L], s(N)) :- lungh(L, N).
```

Il trattino "_" rappresenta una variabile anonima che unifica con qualsiasi termine.

```
?- lungh( [X|[Y|[a|[]]]]).
?- lungh([a,b,Z]).
?- lungh([a|x]).
```

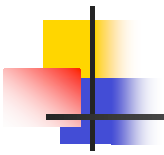


esempio: "member"

- member(X, L) "X appartiene alla lista L"

```
member(X, [X| _ ]).
member(X, [ _ |L]) :- member(X,L).

?- member( a, [c|[b|[a|[]]]]).
?- member( a, [a|[b|[a|[]]]]).
?- member( a, [c|[Z|[a|[]]]]).
?- member( a, [c|Z]).
```

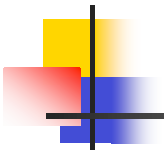


esempio: "append"

- append(L1, L2, L) L è la concatenazione di L1 ed L2

```
append([], L, L).
append([X|L1], L2, [X|L]) :- append(L1,L2,L).
```

```
?- append([a,b], [c], Z).
?- append([X,a,Y], [b,c], Z).
?- append([a,b],[c|Y], Z).
?- append([X],[c|Y], [a,c,e,f]).
```

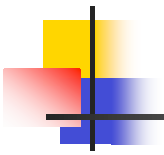


Tipizzazione

- Attenzione Prolog NON è un linguaggio tipato.

```
?- member( a, [a|non_sono_una_lista]).
```
ha successo!

```
append([], non_sono_una_lista, L).
```
ha successo con c.a.s. {L/non_sono_una_lista}



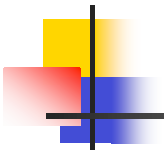
Il predicato lista

- Possiamo controllare la tipizzazione da programma definendo il predicato: lista(L)

```
lista([]).
lista([ _ |L]) :- lista(L).
```

ed introdurre una versione "tipata" di append:

```
t_append([], L, L).
t_append([X|L1], L2, [X|L]) :- t_append(L1,L2,L).
```

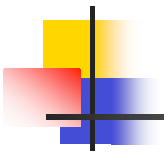


esempio: "delete"

- delete(X,LB,LS) la lista LS si può ottenere da LB cancellando un elemento (che unifica con) X.

```
delete(X,LB,LS) :-   append(F, [X|R], LB),
                     append(F,R,LS).

delete(X, [X|R], R).
delete(X, [Y|L], [Y|R]) :- delete(X,L,R).
```



esercizio

Per ciascuno dei seguenti goal descrivere il comportamento dell'interprete Prolog.

?- delete(a, [c,d,a,b,a], Z).
?- delete(a, [X|Z], [a,b,a,c]).
?- delete(X, [a,f,a,c], Z).
?- delete(a, [b,X,c], Y).
?- delete(a, [b|X], Y).

Aritmetica

- Tra le costanti predefinite dal linguaggio troviamo gli interi, i numeri in virgola mobile ed i numeri in una base a scelta:
  - interi: 0, 1, 9977, -79393
  - virgola mobile: 4.5e3, 1.0, -0.5e+7
  - base x': 2'101, 8'174
- Troviamo poi molti funtori (+, _, *, …) che ci permettono di costruire le usuali espressioni aritmetiche o le funzioni più comuni (abs(.), max(.), ..) (vedi manuale www.siscs.se/isl/sicstus.html).

built-in's aritmetici

- Vi sono poi dei predicati predefiniti che ci permettono di calcolare espressioni aritmetiche, ma …

  attenzione! ogni variabile che compare in una espressione che deve essere valutata deve essere legata ad un termine ground nel momento della valutazione.

Z is X	il valore di X viene unificato con Z;
X =:= Y	i valori di X ed Y sono uguali;
X=\= Y	i valori di X ed Y sono diversi;
X > Y	il valore di X è maggiore del valore di Y.

Il predicato " is": esempi

- p(X) :- X is 3+5.
- ?- p(X).
- X=8

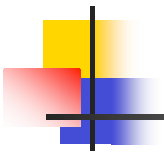
- p(X) :- Y is 3+5, X is Y.
- ?- p(X).
- X=8

- p(X) :- X is Y, Y is 3+5.
- ?- p(X).
- errore

?- 7 is 3+4
yes

?- 8 is 3+4
no

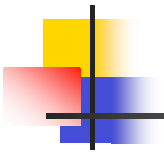
?- 3+4 is 3+4
no - perché la parte destra viene valutata a 7 che non è unificabile con la parte sinistra.



"lungh" con aritmetica

- lungh (L, N) "N è la lunghezza della lista L"

```
lungh([], 0).
lungh([ _ |L], N1) :- lungh(L, N), N1 is N+1.
```



esempio: merge

- merge(L1,L2,L) L è la fusione delle liste ordinate di interi L1 ed L2.

```
merge([],Y,Y).
merge(Y,[],Y).
merge([X|L1],[Y|L2],[X|L]) :- X ≤ Y,
   merge(L1,[Y|L2],L).
merge([X|L1],[Y|L2],[Y|L]) :- X >Y,
   merge([X|L1],L2,L).
```

esempio: split

- split(L,L1,L2) L= L1.L2 e lungh(L1)=lungh(L2)

```
split([X|L],[X|L1],L2) :- split(L, L2, L1).
split([],[],[]).
```

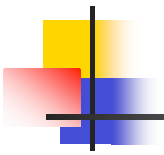
split([a,b,c],L1,L2)
= {L1/[a|L1']} => split([b,c],L2,L1')
= {L2/[b|L2']} => split([c],L1',L2')
= {L1'/[c|L1"]} => split([],L2',L1")
= {L2'/[],L1"/[]} =>'

esempio: mergesort

- merge_sort(L,LSort) LSort è l'ordinamento di L

```
merge_sort([],[]).
merge_sort([X],[X]).
merge_sort([X,Y|R],Lsort) :- split([X,Y|R], L1, L2),
    merge_sort(L1,L1Sort),
                    merge_sort(L2,L2Sort),
                    merge(L1Sort, L2Sort, LSort).
```

n.b. se lungh(L) ≥2 allora, dopo split, lungh(L1)<lungh(L) e lungh(L2)<lungh(L).

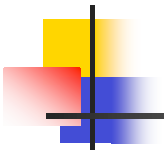


strutture dati

- La sintassi di Prolog ci permettere di definire ed utilizzare strutture di dati: dobbiamo decidere la rappresentazione (come insieme di termini) e poi definire i costruttori, i distruttori ed i predicati.

Esempio: albero binario

- rappresentiamo l'albero vuoto con la costante "void"
- usiamo il funtore "tree" di arietà 3 per descrivere gli alberi non vuoti: tree(Root,Left,Right).



Albero binario: esempi

- empty(T) test su albero vuoto

```
empty(void).
```

- build_tree(X,T1,T2, T)

```
build_tree(X, T1, T2, tree(X,T1,T2)).
```

- left_tree(T,L)

```
left_tree(tree(Root,Left,Right), Left).
```

Costruzione della lista in preordine

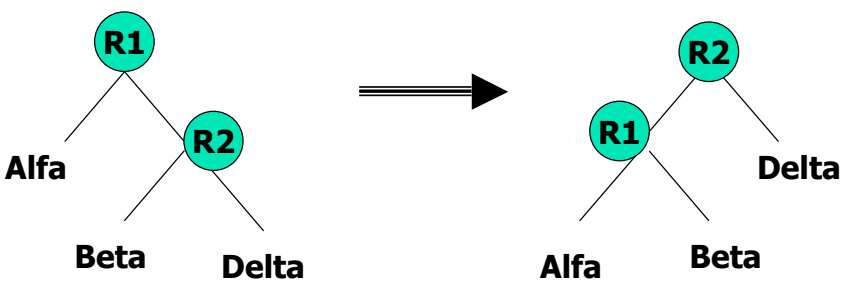
- pre_visit(T,L) L è la lista delle chiavi di T ottenuta da una visita in preordine.

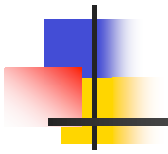
```
pre_visit(void,[]).
pre_visit(tree(Root, Left, Right), [Root|L]) :-
                        pre_visit(Left, L1),
   pre_visit(Right,L2),   append(L1,L2,L).
```

esempio: rotate_left

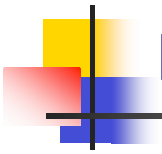
- rotate_left(T,Tr) Tr si ottiene da T tramite rotazione a sinistra

```
rotate_left ( tree(R1,Alfa,tree(R2,Beta,Delta)),
              tree(R2,tree(R1,Alfa,Beta),Delta)).
```





Il predicato ! (cut)

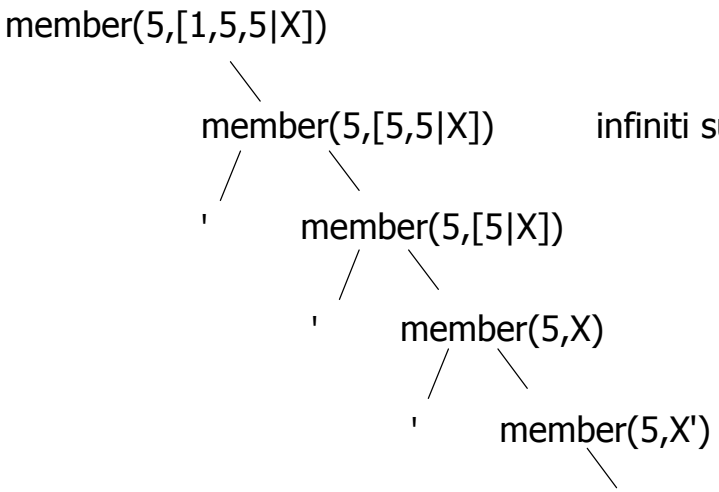


Problemi di efficienza

- Liste differenza e tecnica dell'accumulo ci permettono di evitare inefficienze generate dalla scomposizione e ricomposizione di liste, permettendo di "preparare" il risultato in una struttura ausiliaria (accumulatore) durante la fase di scomposizione ed utilizzando l'unificazione per il calcolo finale.
- Questa non è l'unica fonte di inefficienza, altre gravi inefficienze sono legate alla struttura dell'albero di ricerca (albero SLD) per un goal G. Questo può infatti contenere molti cammini ripetuti o vicoli ciechi.

esempio 1

```
member(X, [X| _ ]).
member(X, [ _ |L]) :- member(X,L).
?- member(5,[1,5,5|X]).
```



infiniti successi tutti uguali

esempio 2

```
confronta(X,Y,minore) :- X<Y.
confronta(X,Y,uguale) :- X=:=Y.
confronta(X,Y,maggiore) :- X>Y.
?- confronta(10,20,X), X=maggiore.
```

Notiamo che

- il goal fallisce (vedi diapositiva successiva)
- accorgendosi che le tre clausole sono mutamente esclusive si sarebbe potuta evitare la ricerca del successo dopo il primo fallimento.

esempio 2

confronta(10,20,X), X=:=maggiore

10<20, minore =:= maggiore
minore =:= maggiore
fail

10=:=20, uguale=:=maggiore
fail

10>20, maggiore=:=maggiore
fail

Il cut

- Il problema con gli esempi precedenti deriva dalla strategia di ricerca dell'interprete Prolog che cerca sempre di ispezionare TUTTO l'albero SLD.
- Per alterare tale strategia viene offerto al programmatore il predicato cut (!) che può essere inserito nel corpo di una qualsiasi clausola.
- L'esecuzione del predicato cut ha l'effetto di "potare" l'albero SLD, tagliando i rami ridondanti o morti secondo le regole seguenti.

esecuzione di ! (cut)

Sia !, G1 un goal che ha il cut (!) come atomo più a sinistra e sia c: H :- **A**, !, **B**. l'istanza della clausola che ha introdotto tale occorrenza del cut.

L'esecuzione del cut rende definitiva ogni scelta fatta nel risolvere l'atomo che unificava con H, ovvero la scelta della clausola c e le scelte fatte per risolvere **A**.

esempio

p(s1) :- **B1**.

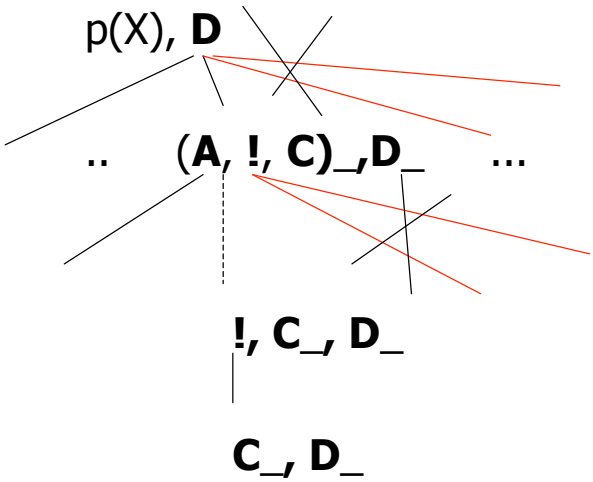
...

p(si):- **A**, !, **C**.

...

p(sn) :- **Bn**

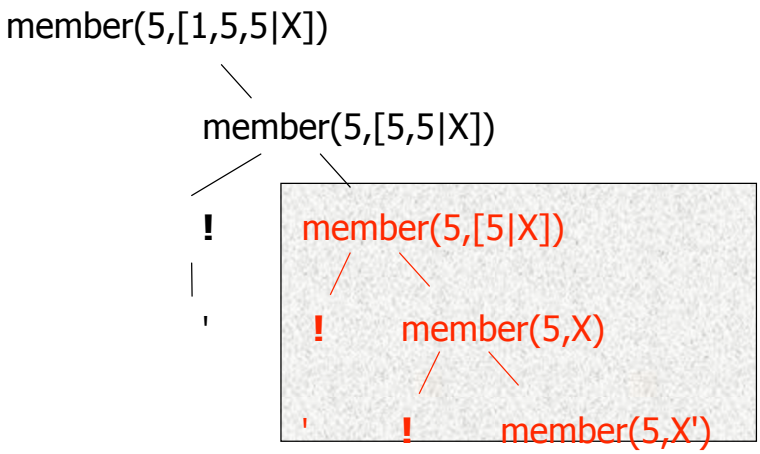
G = p(X), **D**



La parte di albero cancellata non verrà più ispezionata, anche se il goal C_, D_ fallisce.

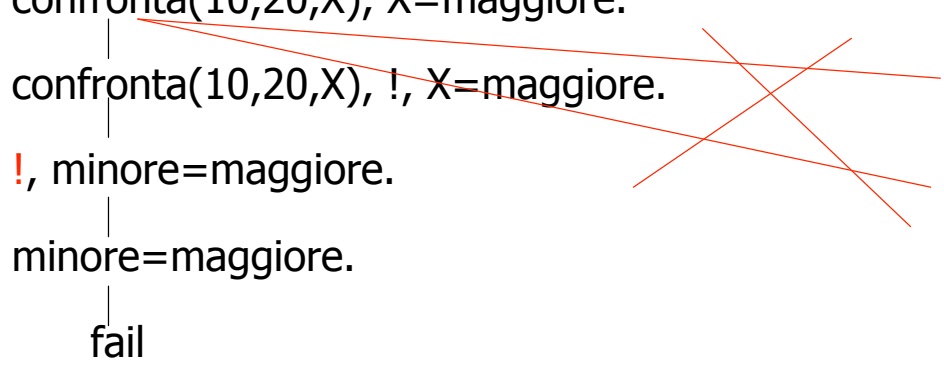
esempio: member con cut

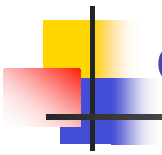
```
member(X, [X|_]) :- !.
member(X,[-|L]) :- member(X,L).
```



esempio: confronta con cut

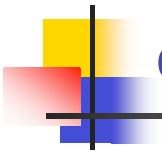
```
confronta(X,Y,minore) :- X<Y, !.
confronta(X,Y,uguale) :- X=:=Y, !.
confronta(X,Y,maggiore) :- X>Y.
```





cut verdi e cut rossi

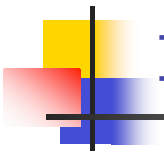
- CUT VERDE. Negli esempi precedenti l'introduzione del cut non ha alterato la corrispondenza tra semantica operazionale e semantica dichiarativa: stesso insieme delle soluzioni con e senza esecuzione del cut.
- CUT ROSSO. Questo non è sempre vero: l'esecuzione del cut può alterare il significato del programma cambiando l'insieme delle soluzioni.



cut rosso: esempio

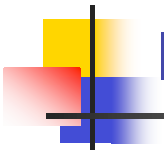
```
red_confronta(X,Y,minore) :- X<Y, !.
red_confronta(X,Y,uguale) :- X=:=Y, !.
red_confronta(X,Y,maggiore).
```

Questo programma produce lo stesso insieme di risposte di quello visto in precedenza (che aveva il controllo X>Y nel corpo dell'ultima clausola) ma la semantica dichiarativa dei due programmi NON è la stessa.



Il predicato fail

Il predicato fail è un altro predicato predefinito che, contrariamente al cut, quando viene eseguito genera un fallimento.



la sequenza !, fail

```
not_member(X, []).
not_member(X, [X|_]):- !,fail.
not_member(X,[-|L]) :- not_member(X,L).
```

Quando X unifica con la testa della lista [X|-] viene prima eseguito il cut che taglia tutte le scelte successive e poi il fail che fa abortire la computazione (effetto desiderato).

E' un modo per realizzare la negazione (… vedremo).

Esempio: set

Vogliamo definire un predicato

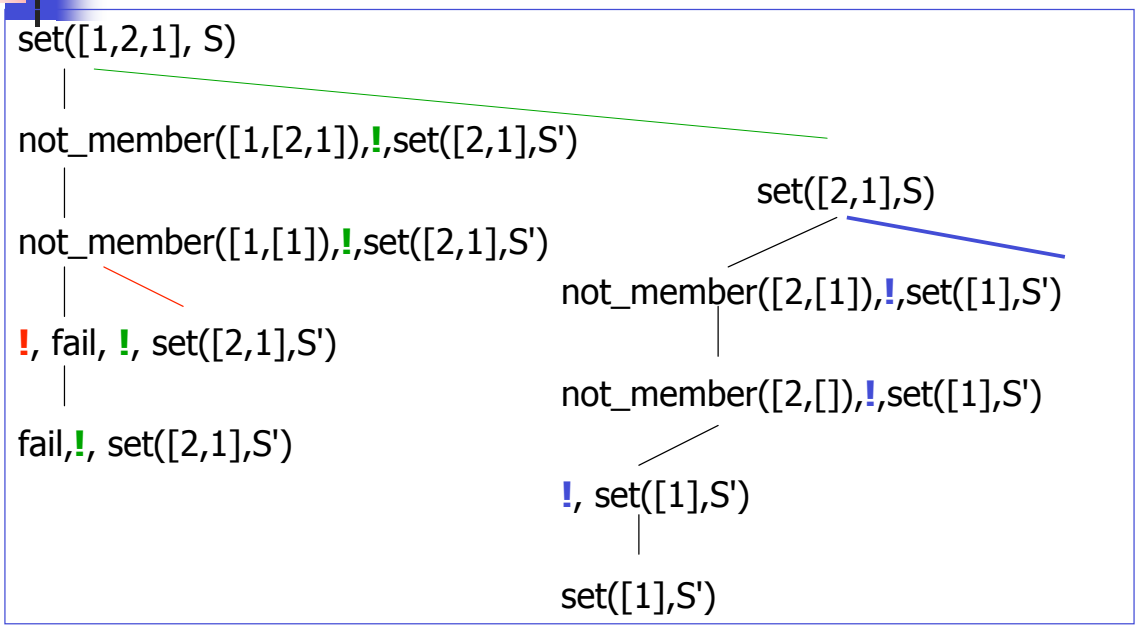
set(L,S) dove S ed L contengono gli stessi elementi ma S non ha duplicati.

```
set([],[]).
set([X|L],[X|S]) :-
   not_member(X,L),!,set(L,S).
set([X|L], S) :- set(L,S).
```

associazione cut-taglio

- In un albero SLD possono occorrere più cut, ciascuno (se eseguito) può generare delle potature, è importante una giusta associazione.

esempio



Esercizio

- Definire un predicato flatten(L,L') dove L è una lista di liste ed L' è la sua versione "appiattita",
  es. L=[[a,b],[c],[[d,e],f]] L=[a,b,c,d,e]

```
flatten([X|L],F) :-  flatten(X,F1), flatten(L,F2),
                     append(F1,F2,F).
flatten(X,[X]) :- constant(X), X =!=[].
flatten([],[]).
```

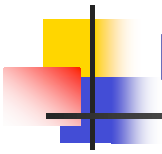
flatten_dl

Sostituiamo ogni lista risultato con una lista differenza

```
flatten_dl([X|L],F-Z) :-  flatten_dl(X,F1-Y),
                          flatten_dl(L,F2-W), append_dl(F1-Y,F2-W,F-Z).
flatten_dl(X,[X|Z]-Z) :- constant(X), X =!=[].
flatten_dl([],Z-Z).
```

Risolvendo append nella prima clausola Y=F2, W=Z e F1=F:

```
flatten_dl([X|L],F-Z) :- flatten_dl(X,F-F2), flatten_dl(L,F2-Z).
flatten_dl(X,[X|Z]-Z) :- constant(X), X =!=[].
flatten_dl([],Z-Z).
```



Esercizio

- Definire il tipo di dato coda con le operazioni: enqueue e dequeue.