Classificazione dei termini

Il Prolog è un linguaggio a **tipizzazione dinamica**

Predicati predefiniti per riconoscere il tipo di un termine

var, nonvar

```
?- var(X).
true.

?- var(padre(X)).
false.

?- X=Y, Y=23, var(X).
false.
```

integer, float, number
atom, atomic

```
?- atom(pippo).
true.

?- atom(3).
false.

?- atomic(3).
true.
```

Accesso ai componenti delle strutture: functor e arg

functor(?Term, ?Name, ?Arity): True when Term is a term with functor Name/Arity

```
?- functor(f(a,b,c),F,N).  %% il funtore e' f/3
F = f,
N = 3.

?-  functor(X,f,3).
X = f(_G282, _G283, _G284).

?- functor(T, a, 0).
T = a.
```

arg(?Arg, +Term, ?Value): Value is unified with the Arg-th argument of Term.

```
?- arg(2,f(a,b,c),X).
X = b.

?- arg(N,f(a,b,c),c).
N = 3.
```

Accesso ai componenti delle strutture: Univ

Il predicato chiamato Univ è un operatore (priorità 700, tipo xfx), che si scrive

`=..`

?Term =.. ?List: List is a list whose head is the functor of Term and the remaining elementss are the arguments of the term. Either side of the predicate may be a variable, but not both. This predicate is called 'Univ'.

```
?- f(a,b,c) =.. X.
X = [f, a, b, c].
```

```
?- 3+2 =.. X.
X = [+, 3, 2].
```

```
?- Term =.. [funtore, argomento(1), argomento(2)].
Term = funtore(argomento(1), argomento(2)).
```

```
?- f(a,b,c) =.. [F|Rest], Newterm =.. [g|Rest].
F = f,
Rest = [a, b, c],
Newterm = g(a, b, c).
```

Metaprogrammazione

Programmi e dati sono rappresentati in modo uniforme

- Un atomo è un termine: la struttura di un atomo è uguale a quella di un termine (una struttura)
- Una congiunzione di atomi è un termine: la virgola è un funtore
- Una clausola è un termine: `:-` è un funtore
- Un fatto `A` è una clausola della forma `A :- true`.

I programmi possono essere trattati come dati

Meta-programma: programma che tratta altri programmi come dati

Accesso alle clausole del programma

clause(+Head, ?Body): True if Head can be unified with a clause head and Body with the corresponding clause body. Gives alternative clauses on backtracking. For facts, Body is unified with the atom true.

```
?- listing(concat).
concat([], A, A).
concat([A|B], C, [A|D]) :-
concat(B, C, D).
true.

?- clause(concat(X,Y,Z), Body).
X = [],
Y = Z,
Body = true ;
X = [_G556|_G557],
Z = [_G556|_G560],
Body = concat(_G557, Y, _G560).
```

Un semplice **meta-interprete** per il Prolog puro

Meta-interprete: interprete per un linguaggio scritto nel linguaggio stesso

```
solve(true) :- !.
solve((G1,G2)) :-
        !, solve(G1), solve(G2).
solve(Goal) :-
        clause(Goal,Body),
        solve(Body).
```

Costruzione della dimostrazione di un goal

Nucleo di un **sistemi esperto** in Prolog (mira a riprodurre il ragionamento di "esperti" in un particolare dominio, ad esempio per la diagnostica medica). Deve avere la capacità di spiegare all'utente il proprio ragionamento.

```
?- op(300,xfx,'=>').

explain(true,true) :- !.
explain((G1,G2),(Prova1,Prova2)) :-
        !, explain(G1,Prova1), explain(G2,Prova2).
explain(Goal,(Prova => Goal)) :-
        clause(Goal,Body),
        explain(Body,Prova).

genitore(ron,bob). genitore(pam,bob).  genitore(bob,pat).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- explain(nonno(X,pat),P).
X = ron,
P = (true=>genitore(ron,bob), true=>genitore(bob,pat))=>nonno(ron,pat) ;
```

Costruzione della dimostrazione di un goal

Nucleo di un **sistemi esperto** in Prolog (mira a riprodurre il ragionamento di "esperti" in un particolare dominio, ad esempio per la diagnostica medica). Deve avere la capacità di spiegare all'utente il proprio ragionamento.

```
?- op(300,xfx,'=>').

explain(true,true) :- !.
explain((G1,G2),(Prova1,Prova2)) :-
        !, explain(G1,Prova1), explain(G2,Prova2).
explain(Goal,(Prova => Goal)) :-
        clause(Goal,Body),
        explain(Body,Prova).

genitore(ron,bob). genitore(pam,bob).  genitore(bob,pat).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- explain(nonno(X,pat),P).
X = ron,
P = (true=>genitore(ron,bob), true=>genitore(bob,pat))=>nonno(ron,pat) ;
X = pam,
P = (true=>genitore(pam,bob), true=>genitore(bob,pat))=>nonno(pam,pat)
```

Manipolazione della base di dati

Il Prolog ha quattro comandi di base per la manipolazione della base di dati (il programma).

Per aggiungere informazioni:

- **asserta(+Term)**: Assert a fact or clause in the database. Term is asserted as the first fact or clause of the corresponding predicate.
- **assertz(+Term)** o **assert(+Term)**: Equivalent to asserta/1, but Term is asserted as the last clause or fact of the predicate.

Per cancellare informazioni:

- **retract(+Term)**: When Term is an atom or a term it is unified with the first unifying fact or clause in the database. The fact or clause is removed from the database.
- **retractall(+Head)**: All facts or clauses in the database for which the head unifies with Head are removed. If Head refers to a predicate that is not defined, it is implicitly created as a dynamic predicate.

Danno la possibilità di cambiare il significato dei predicati a tempo di esecuzione.

Predicati statici e dinamici

I predicati il cui significato può essere modificato a runtime si dicono **dinamici**

(Gli altri vengono chiamati statici)

Alcuni interpreti Prolog (tra cui SWI) richiedono una dichiarazione dei predicati dinamici

```
:- dynamic felice/1, padre/2.
```

dynamic è una direttiva, che può essere inserita nel programma (non come query)

Esempio

```
:-dynamic felice/1.

?- assert(felice(giovanni)).
true.

?- assertz(felice(vittorio)).
true.

?- asserta(felice(anna)).
true.

?- assert(felice(X) :- stupido(X)).
true.

?- listing(felice).
:- dynamic felice/1.

felice(anna).
felice(giovanni).
felice(vittorio).
felice(A) :-
        stupido(A).
true.
```

Memoisation (o caching)

Memorizzazione dei risultati dei calcoli, quando possono servire di nuovo

Esempio: i numeri di Fibonacci

```
:- dynamic(fib/2).

fib(1,1).
fib(2,1).
fib(N,F) :-
        N>2,
        N1 is N-1, fib(N1,F1),
        N2 is N-2, fib(N2,F2),
        F is F1+F2.
```

Poco efficiente: quando si calcola fib(N2,F2), si deve ricalcolare fib(N1,F1).

Memoisation (o caching)

Memorizzazione dei risultati dei calcoli, quando possono servire di nuovo

Esempio: i numeri di Fibonacci

```
:- dynamic(fib/2).

fib(1,1).
fib(2,1).
fib(N,F) :-
        N>2,
        N1 is N-1, fib(N1,F1),
        N2 is N-2, fib(N2,F2),
        F is F1+F2,
        asserta(fib(N,F)).   %% si memorizza il risultato
```

Ogni volta che viene calcolato un valore fib(N,F), viene memorizzato, in modo da non doverlo calcolare un'altra volta

Esecuzione di fib

```
?- fib(5,N).
N = 5

?- listing(fib).
:- dynamic fib/2.

fib(5, 5).
fib(4, 3).
fib(3, 2).
fib(1, 1).
fib(2, 1).
fib(A, D) :-
        A>2,
        B is A+ -1,
        fib(B, E),
        C is A+ -2,
        fib(C, F),
        D is E+F,
        asserta(fib(A, D)).
true.
```

Cancellazione delle clausole aggiunte

Cancellazione di tutti i fatti fib(N,F) con N>2

```
cancella :-
    clause(fib(N,F),true),
    N > 2,
    retract(fib(N,F)),
    fail.
cancella.
```

fail forza il backtracking a cercare di soddisfare di nuovo clause(fib(N,F),true)

```
?- cancella.
true.

?- listing(fib).

fib(1, 1).
fib(2, 1).
fib(A, B) :-
        A>2,
        C is A-1,
        fib(C, D),
        E is A-2,
        fib(E, F),
        B is D+F,
        asserta(fib(A, B)).

Yes
```

retractall

```
?- listing(felice).
:- dynamic felice/1.

felice(anna).
felice(giovanni).
felice(vittorio).
felice(A) :-
        stupido(A).
true.

?- retractall(felice(_)).
true.

?- listing(felice).
:- dynamic felice/1.
true.
```

Raccogliere tutte le soluzioni di un goal: findall

Il Prolog ha tre predicati per generare tutte le soluzioni di un dato goal e riportarli in una lista: **findall, bagof, setof**.

- **findall(+Term,+Goal,-L)**: costruisce una lista L di tutte le istanze di Term che si ottengono soddisfando Goal in tutti i modi possibili.
  Ha sempre successo: se Goal fallisce, L è la lista vuota.

```
?-  findall((X,Y),(member(X,[a,b]),member(Y,[1,2])),Set).
Set = [ (a, 1), (a, 2), (b, 1), (b, 2)].

?- findall(X+Y,(member(X,[a,b]),member(Y,[1,2])),Set).
Set = [a+1, a+2, b+1, b+2].

?- findall(f(X,Y),(member(X,[a,b]),member(Y,[1,2])),Set).
Set = [f(a, 1), f(a, 2), f(b, 1), f(b, 2)].

?- findall(X,member(X,[]),Set).
Set = [].
```

findall: esempi

```
filter(P,L,Result) :-
        findall(X,(member(X,L),Cond=..[P,X],Cond),Result).
pari(X) :- 0 is X mod 2.

?- filter(pari,[20,3,12,41,100],Result).
Result = [20, 12, 100].
```

findall: esempi

```
filter(P,L,Result) :-
        findall(X,(member(X,L),Cond=..[P,X],Cond),Result).
pari(X) :- 0 is X mod 2.

?- filter(pari,[20,3,12,41,100],Result).
Result = [20, 12, 100].


intersect(S1,S2,S) :-
        findall(X, (member(X,S1),member(X,S2)), S).
```

findall: esempi

```
filter(P,L,Result) :-
        findall(X,(member(X,L),Cond=..[P,X],Cond),Result).
pari(X) :- 0 is X mod 2.

?- filter(pari,[20,3,12,41,100],Result).
Result = [20, 12, 100].


intersect(S1,S2,S) :-
        findall(X, (member(X,S1),member(X,S2)), S).

figlio(carlo,mario).      figlio(carlo,maria).
figlio(anna,mario).       figlio(anna,maria).

?- findall(X,figlio(_,X),L).
L = [mario, mario, maria, maria].
```

A volte findall è un po' grezzo!

bagof e setof

- **bagof(Term,Goal,L)** : produce una lista L di tutte le istanze di Term che si ottengono soddisfando Goal in tutti i modi possibili.
  Se Goal ha altre variabili oltre a quelle che compaiono anche in Term, bagof fornirà diverse soluzioni.
  Fallisce se Goal fallisce.
- **setof(Term,Goal,L)** : si comporta come **bagof**, ma la lista L è ordinata e senza duplicati

```
figlio(carlo,mario).
figlio(anna,mario).
figlio(carlo,maria).
figlio(anna,maria).

?- bagof(X,figlio(Y,X),L).
Y = anna,
L = [mario, maria] ;
Y = carlo,
L = [mario, maria].
```

bagof e setof

- **bagof(Term,Goal,L)** : produce una lista L di tutte le istanze di Term che si ottengono soddisfando Goal in tutti i modi possibili.
  Se Goal ha altre variabili oltre a quelle che compaiono anche in Term, bagof fornirà diverse soluzioni.
  Fallisce se Goal fallisce.
- **setof(Term,Goal,L)** : si comporta come **bagof**, ma la lista L è ordinata e senza duplicati

```
figlio(carlo,mario).
figlio(anna,mario).
figlio(carlo,maria).
figlio(anna,maria).
```

```
?- bagof(X,figlio(_,X),L).
L = [mario, maria] ;
L = [mario, maria].
```

```
?- bagof(X,figlio(Y,X),L).
Y = anna,
L = [mario, maria] ;
Y = carlo,
L = [mario, maria].
```

bagof e setof

- **bagof(Term,Goal,L)** : produce una lista L di tutte le istanze di Term che si ottengono soddisfando Goal in tutti i modi possibili.
  Se Goal ha altre variabili oltre a quelle che compaiono anche in Term, bagof fornirà diverse soluzioni.
  Fallisce se Goal fallisce.
- **setof(Term,Goal,L)** : si comporta come **bagof**, ma la lista L è ordinata e senza duplicati

```
figlio(carlo,mario).
figlio(anna,mario).
figlio(carlo,maria).
figlio(anna,maria).

?- bagof(X,figlio(Y,X),L).
Y = anna,
L = [mario, maria] ;
Y = carlo,
L = [mario, maria].
```

```
?- bagof(X,figlio(_,X),L).
L = [mario, maria] ;
L = [mario, maria].

?- bagof(X,figlio(X,antonio)
false.
```

bagof e setof

- **bagof(Term,Goal,L)** : produce una lista L di tutte le istanze di Term che si ottengono soddisfando Goal in tutti i modi possibili.
  Se Goal ha altre variabili oltre a quelle che compaiono anche in Term, bagof fornirà diverse soluzioni.
  Fallisce se Goal fallisce.
- **setof(Term,Goal,L)** : si comporta come **bagof**, ma la lista L è ordinata e senza duplicati

```
figlio(carlo,mario).
figlio(anna,mario).
figlio(carlo,maria).
figlio(anna,maria).

?- bagof(X,figlio(Y,X),L).
Y = anna,
L = [mario, maria] ;
Y = carlo,
L = [mario, maria].
```

```
?- bagof(X,figlio(_,X),L).
L = [mario, maria] ;
L = [mario, maria].

?- bagof(X,figlio(X,antonio)
false.

?- setof(X,figlio(_,X),L).
L = [maria, mario] ;
L = [maria, mario].
```

Differenza tra findall e bagof

```
?- findall(Y,(member(X,[2,4]),Y is X*2),Result).
Result = [4, 8].

?- bagof(Y,(member(X,[2,4]),Y is X*2),Result).
X = 2,
Result = [4] ;
X = 4,
Result = [8].
```

Differenza tra bagof e setof

```
figlio(carlo,mario).
figlio(carlo,maria).
figlio(mario,alberto).
figlio(maria,alberto).

discendente(X,Y) :- figlio(X,Y).
discendente(X,Y) :- figlio(X,Z), discendente(Z,Y).

?- bagof(X,discendente(carlo,X),Antenati).
Antenati = [mario, maria, alberto, alberto].

?- setof(X,discendente(carlo,X),Antenati).
Antenati = [alberto, maria, mario].
```