

- `member(X,Lista)`
- `write(X), write(' '), write(par)`
- `read`
- `append(?L1, ?L2, ?L12)`
- `repeat` – ciclo de entrada de dados até chegar ao fim
- `compound(X)` – identifica um termo composto ex: `pai(X,Y)`
- `integer(X)` – identifica um inteiro
- `atom(X)` – termo atómico ex: `par`
- `constant` – ex: `par, 5`
- `=..` – ex: `Termo =.. [Func|Args]`
- `!` – yes
- `fail` - no
- `ground(X)` – não é composto por variáveis
- `arg(N,Termo,Arg)` – ex: `arg(2,pai(a,b),X) -> X=b`
- `call(Golo)`
- `findall(Formato,Query,Lista)`
- `functor(Nome,F,A)` – ex: `functor(pai(a,b),X,Y) -> X= pai, Y=2`
- `assert` – coloca na bd (`asserta, assertz`)
- `:-dynamic pai` - permite fazer `assert` e `retract` ao predicado `pai`
- `retract` – retira da bd o primeiro que corresponda ao seu argumento
- `retractall` – remove tudo com esse predicado da bd
- `recorded(Key,Value,Reference)`
- `erase(Reference)` – apaga um record da bd
- `abolish(Nome,Aridade)/ abolish(Predicado)` – remove todas as clausulas
- `length(L,N)`
- `listing / listing(pred)` – lista o que está na bd
- `op(prec, assoc, operador)`
- `current_op(p,a,op)`
 - `xf / fx` - a precedencia do arg tem de ser mais baixa do que a do operador
 - `yf / fy` – precedência do arg pode ser igual à do operador
 - `xfx` – ambos os arg tem menor precedência do que o operador
 - `xfy` – direita para a esquerda
 - `yfx` – esquerda para a direita
- `see(File)` – define File como ficheiro de input
- `seeing(File)` – File unifica com o nome do fich de input actual
- `seen` – fecha o fich de input actual
- `tell(File)` – define File como fich de output
- `telling(File)` – unifica com o nome do fich de output actual
- `told` – fecha o fich de output actual
- `clause(Head,Body)` – permite aceder às várias clausulas que constituem um programa em memória, tem como soluções possíveis as substituições
ex: `clause(append(_,_,_), Body) -> Body = true` ou `Body = append(X,Y,Z)`

- bagof – igual ao findall mas pode-se fazer backtracking

ex: bagof(Y, pai(X,Y),L) *X é livre*

Y= _123 , X = joao, L = [luis,rui];

Y= _123, X= luis , L=[ana,ze];

Y= _123 , X=rui, L=[olga,ana];

No

É possível expressar que as var não são livres. A construção Var^Goal diz ao bagof para não criar substituições para Var em Goal

bagof(Y, X^pai(X,Y),L)

Y = _123, X= 456, L = [ana,luis,rui,olga,ze,ana]

O que é equivalente a fazer findall(Y,pai(X,Y),L)

- setof – produz os mesmos resultados que o bagof mas a lista final aparece ordenada e sem elementos repetidos
- minimize(labeling([ff], Tasks), Time). % ou labeling([ff,minimize(Time)], Tasks)
- maximize
- labelling([ff],Vars)
- all_different([A1,A2,A3]) – todas as vars da lista devem ser todas diferentes
- all_distinct – noção de grafo, os nós do grafo são as variáveis
- atom_concat(A1,A2,A) – ex: atom_concat(par,es,N) -> N = pares
- name(AtomoNumero, CodigoAscii)
- phrase