La programmation logique pure est *théoriquement* suffisante ; il est facile de vérifier, du point de vue de la théorie de la calculabilité, que Prolog pur est un langage universel. Il est quasi immédiat de traduire une machine de Turing quelconque en un programme Prolog pur, ou encore d'implémenter dans ce langage les mécanismes de construction des fonctions récursives.

En pratique cependant, pour des raisons d'efficacité, le langage Prolog comporte toute une série de mécanismes étrangers à la programmation logique proprement dite. Ces mécanismes altèrent, parfois gravement, la sémantique déclarative de Prolog, et compliquent, parfois grandement, sa sémantique opérationnelle. Utilisés convenablement, ils permettent néanmoins de développer rapidement des prototypes raisonnablement efficaces de programmes intéressants, notamment dans le domaine de l'intelligence artificielle.

Les mécanismes importants sont
- La coupure et la négation
- Les prédicats et opérateurs arithmétiques
- Les prédicats d'analyse de termes
- Les prédicats méta-logiques
- Les prédicats extra-logiques

Analyse de termes I

Reconnaisseurs.

```
integer(X)   X is an integer number
real(X)      X is a floating-point number
number(X)    X is a number (integer or float
atom(X)      X is an atom
atomic(X)    X is an atom or a number
compound(X)  X is a compound term
```


Analyse des termes II

Exemple.

```
flatten(X,[X]) :- atomic(X), X \= [].
flatten([X|Y],Zs) :-
    flatten(X,Xs), flatten(Y,Ys),
    append(Xs,Ys,Zs).


flatten2(X,Zs) :- flat2([X],Zs).

flat2([],[]).
flat2([X|Xs],[X|Zs]) :-
    atomic(X), flat2(Xs,Zs).
flat2([[X|Y]|Xs],Zs) :- flat2([X,Y|Xs],Zs).


flatten3(X,Zs) :- flatten3(X,[],Zs).

flatten3(X,L,[X|L]) :- atomic(X).
flatten3([X|Y],L,R) :-
    flatten3(Y,L,U), flatten3(X,U,R).
```

Analyse des termes III

Exemple, prédicats d'essai.

```
tree(0,X) :- X is random(10).
tree(1,[X|Y]) :- tree(0,X), tree(0,Y).
tree(N,[X|Y]) :- N>1, N1 is N-1, N2 is N-2,
            tree(N1,X), tree(N2,Y).

essai(N,X,U) :- tree(N,X),
                time(flatten(X,U)),
                time(flatten2(X,U)),
                time(flatten3(X,U)).

essai2(N,X,U) :- tree(N,X),
                 time(flatten2(X,U)),
                 time(flatten3(X,U)).
```

Analyse des termes IV

Termes composés et listes ; symboles atomiques.

```
?- [H|T] = [a].
--> H = a , T = []
?- [A1,A2,_,A4|T] = [what,we,call,a,rose].
--> A1 = what, A2 = we, A4 = a, T = [rose]
```

```
?- f(a,b,c) =.. L.
    --> L = [f,a,b,c]
?- f(a,b,c) =.. [F|Args].
    --> F = f , Args = [a,b,c]
?- (a*b+c) =.. [Op|Args].
    --> Op = + , Args = [a*b,c]

?- functor(owns(john,umbrella), F, N).
    --> F = owns , N = 2
?- functor(f(a,b), f, 2).
    --> yes
?- arg(2, owns(john,umbrella), A).
    --> A = umbrella

?- name(blabla,L).
    --> L = [98,108,97,98,108,97]
?- name(bac, [A|_]) , name(X,[A]).
    --> A = 98 , X = b
```


Analyse des termes V

Termes fondamentaux et sous-termes.

```
subterm(Term,Term).
subterm(Sub,Term) :- compound(Term),
                     functor(Term,F,N),
                     subterm(N,Sub,Term).

subterm(N,Sub,Term) :- arg(N,Term,Arg),
                       subterm(Sub,Arg).
subterm(N,Sub,Term) :- N > 1, N1 is N-1,
                       subterm(N1,Sub,Term).
```


Analyse des termes VI

Substitution de termes.

```
substitute(Old,New,Old,New).

substitute(Old,New,Term,Term)
 :- atomic(Term) , Term \= Old.

substitute(Old,New,Term,Term1)
 :- compound(Term),
    functor(Term,F,N),
    functor(Term1,F,N),
    substitute(N,Old,New,Term,Term1).

substitute(N,Old,New,Term,Term1)
 :- N > 0,
    arg(N,Term,Arg),
    substitute(Old,New,Arg,Arg1),
    arg(N,Term1,Arg1),
    N1 is N-1,
    substitute(N1,Old,New,Term,Term1).

substitute(0,Old,New,Term,Term1).
```

Analyse des termes VII

Substitution de termes : trace

```
substitute(cat,dog,owns(jane,cat),X)   X=owns(jane,
   atomic(owns(jane,cat))           f         cat)
substitute(cat,dog,owns(jane,cat),X)
   compound(owns(jane,cat))
   functor(owns(jane,cat),F,N)         F=owns, N=2
   functor(X,owns,2)                   X=owns(X1,X2)
   substitute(2,cat,dog,
            owns(jane,cate),owns(X1,X2))
      2 > 0
      arg(2,owns(jane,cat),Arg)        Arg=cat
      substitute(cat,dog,cat,Arg1)     Arg1=dog
      arg(2,owns(X1,X2),dog)           X2=dog
      N1 is 2-1                        N1=1
      substitute(1,cat,dog,
               owns(jane,cat),owns(X1,dog))
         1 > 0
         arg(1,owns(jane,cat),Arg2)    Arg2=jane
         substitute(cat,dog,jane,Arg3) Arg3=jane
            atomic(jane)
            jane \= cat
         arg(1,owns(X1,dog),jane)      X1=jane
         N2 is 1-1                     N2=0
         substitute(0,cat,dog,
                  owns(jane,cat),owns(jane,dog))
            0 > 0                  f
         substitute(0,cat,dog,
                  owns(jane,cat),owns(jane,dog))
            true
               Output: (X=owns(jane,dog))
```

Prédicats métalogiques I

Utiliser les opérateurs efficaces.

```
plus(X,Y,Z) :- nonvar(X), nonvar(Y),
               Z is X+Y.

plus(X,Y,Z) :- nonvar(X), nonvar(Z),
               Y is Z-X.

plus(X,Y,Z) :- nonvar(Y), nonvar(Z),
               X is Z-Y.
```

Optimiser l'ordre des clauses.

```
grandparent(X,Z) :- nonvar(X),
               parent(X,Y), parent(Y,Z).

grandparent(X,Z) :- nonvar(Z),
               parent(Y,Z), parent(X,Y).
```


Prédicats métalogiques II

Termes fondamentaux.

```
ground(Term) :- nonvar(Term),
                atomic(Term).

ground(Term) :- nonvar(Term),
                compound(Term),
                functor(Term,F,N),
                ground(N,Term).

ground(0,Term).

ground(N,Term) :- N > 0,
                  arg(N,Term,Arg),
                  ground(Arg),
                  N1 is N-1,
                  ground(N1,Term).
```


Prédicats métalogiques III

Unification I.

```
unify(X,X).
```

Unification II.

```
unify(X,Y) :- var(X), var(Y), X=Y.

unify(X,Y) :- var(X), nonvar(Y), X=Y.

unify(X,Y) :- var(Y), nonvar(X), Y=X.

unify(X,Y) :- nonvar(X), nonvar(Y),
              atomic(X), atomic(Y), X=Y.

unify(X,Y) :- nonvar(X), nonvar(Y),
              compound(X), compound(Y),
              term_unify(X,Y).
```

Prédicats métalogiques IV

Unification II (suite).

```
term_unify(X,Y) :- functor(X,F,N),
                   functor(Y,F,N),
                   unify_args(N,X,Y).

unify_args(N,X,Y) :- N > 0,
                     unify_arg(N,X,Y),
                     N1 is N-1,
                     unify_args(N1,X,Y).

unify_args(0,X,Y).

unify_arg(N,X,Y) :- arg(N,X,ArgX),
                    arg(N,Y,ArgY),
                    unify(ArgX,ArgY).
```

Prédicats métalogiques V

Unification III.

```
unify(X,Y) :- var(X), var(Y), X=Y.
unify(X,Y) :- var(X), nonvar(Y),
              no_oc(X,Y), X=Y.
unify(X,Y) :- var(Y), nonvar(X),
              no_oc(Y,X), Y=X.
unify(X,Y) :- nonvar(X), nonvar(Y),
              atomic(X), atomic(Y), X=Y.
unify(X,Y) :- nonvar(X), nonvar(Y),
              compound(X), compound(Y),
              term_unify(X,Y).

term_unify(X,Y) :- functor(X,F,N),
                   functor(Y,F,N),
                   unify_args(N,X,Y).
unify_args(N,X,Y) :- N>0, unify_arg(N,X,Y),
                     N1 is N-1,
                     unify_args(N1,X,Y).
unify_args(0,X,Y).
unify_arg(N,X,Y) :- arg(N,X,ArgX),
                    arg(N,Y,ArgY),
                    unify(ArgX,ArgY).
```


Prédicats métalogiques VI

Unification III (suite).

```
no_oc(X,Y) :- var(Y), X \== Y.
no_oc(X,Y) :- nonvar(Y), atomic(Y).
no_oc(X,Y) :- nonvar(Y), compound(Y),
              functor(Y,F,N), no_oc(N,X,Y).

no_oc(N,X,Y) :- N > 0, N1 is N-1,
                arg(N,Y,Arg),
                no_oc(X,Arg), no_oc(N1,X,Y).

no_oc(0,X,Y).
```


Prédicats extralogiques I

Modification “on-line” d’un programme logique.

Adjonction de la clause <clause>

en début ou en fin de programme.

```
asserta((<clause>)).
assertz((<clause>)).
```

Rétraction de la première clause s’unifiant avec <C>.

```
retract((<C>)).
```

Attention : n’utiliser ceci qu’avec prudence !

Correspond à une évolution d’une base de connaissance.

Prédicats extralogiques II

Mémo-fonctions : un exemple.

```
hanoi_1(0,A,B,C,0) :- !.
hanoi_1(N,A,B,C,NMoves) :-
   N1 is N-1,
   hanoi_1(N1,A,C,B,NM1),
   hanoi_1(N1,C,B,A,NM2),
   NMoves is NM1+1+NM2.

hanoi_2(0,A,B,C,0) :- !.
hanoi_2(N,A,B,C,NMoves) :-
   N1 is N-1,
   lemma(hanoi_2(N1,A,C,B,NM1)),
   hanoi_2(N1,C,B,A,NM2),
   NMoves is NM1+1+NM2.

lemma(P) :- P, asserta((P :- !)).

test_hanoi_2(N,LPegs,NMoves) :-
   hanoi_2(N,A,B,C,NMoves),
   LPegs = [A,B,C].
```

Prédicats extralogiques III

Mémo-fonctions (suite)

```
| ?- listing.
append('[]', A, A).
append([B|C], A, [B|D]) :-
        append(C, A, D).
hanoi_1(0, _, _, _, 0) :- !.
hanoi_1(A, B, C, D, E) :-
        F is A-1,
        hanoi_1(F, B, D, C, G),
        hanoi_1(F, D, C, B, H),
        E is G+1+H.
hanoi_2(0, _, _, _, 0) :- !.
hanoi_2(A, B, C, D, E) :-
        F is A-1,
        lemma(hanoi_2(F,B,D,C,G)),
        hanoi_2(F, D, C, B, H),
        E is G+1+H.
lemma(A) :-
        A,
        asserta((A:-'!')).
test_hanoi_2(A, B, C) :-
        hanoi_2(A, D, E, F, C),
        B=[D,E,F].
```


Prédicats extralogiques IV

Mémo-fonctions : deux essais

```
| ?- hanoi_1(12,a,b,c,LM).

LM = 4095?
yes        /* TRES LENT !! */

| ?- test_hanoi_2(12,[a,b,c],LM).

LM = 4095?
yes        /* TRES RAPIDE !! */
```


Prédicats extralogiques V

Mémo-fonctions : avant . . .

```
| ?- listing.
append('[]', A, A).
append([B|C], A, [B|D]) :-
        append(C, A, D).
hanoi_1(0, _, _, _, 0) :- !.
hanoi_1(A, B, C, D, E) :-
        F is A-1,
        hanoi_1(F, B, D, C, G),
        hanoi_1(F, D, C, B, H),
        E is G+1+H.
```

Prédicats extralogiques VI

Mémo-fonctions : . . . et après.

```
hanoi_2(11, _, _, _, 2047) :- !.
hanoi_2(10, _, _, _, 1023) :- !.
hanoi_2(9, _, _, _, 511) :- !.
hanoi_2(8, _, _, _, 255) :- !.
hanoi_2(7, _, _, _, 127) :- !.
hanoi_2(6, _, _, _, 63) :- !.
hanoi_2(5, _, _, _, 31) :- !.
hanoi_2(4, _, _, _, 15) :- !.
hanoi_2(3, _, _, _, 7) :- !.
hanoi_2(2, _, _, _, 3) :- !.
hanoi_2(1, _, _, _, 1) :- !.
hanoi_2(0, _, _, _, 0) :- !.
hanoi_2(A, B, C, D, E) :-
        F is A-1,
        lemma(hanoi_2(F,B,D,C,G)),
        hanoi_2(F, D, C, B, H),
        E is G+1+H.
lemma(A) :- A, asserta((A:-'!')).
test_hanoi_2(A, B, C) :-
        hanoi_2(A, D, E, F, C),
        B=[D,E,F].
```