Termes composés

- prédicat etudiant
  - etudiant(jim, white, 17, main, ottawa, ontario, 10, 12, 83, ...) exprimant « l'etudiant jim white habite a l'adresse 17 main a ottawa, ontario et qui est ne le 10/12/83 ...» ;
- Utilisation des termes composés:
  - etudiant(*nom*(john, white), *adresse*(17, main, ottawa, ontario),
    naissance(10, 12, 83), ...)

- On peut alors interroger Prolog de plusieurs manieres:

  ?- etudiant(nom(john,_), X, naissance(_, _, Y)).

  ?- etudiant(X, adresse(_, _, _, quebec), _).

  ?- etudiant(X, adresse(_, _, ottawa, _), naissance(_, _, Y)),
  Y>87.

Base de données et Prolog

- Le Prolog peut s'avérer très performant lorsqu'il s'agit de gérer une base de données
  - Chacune des entités sont représentées à l'aide de faits
  - En utilisant des structures, des listes.

Des livres à la bibliothèque

- Un identificateur, un titre et les auteurs

```
book(cn(qa76, c203, 1984),
        [programming, in, logic],
        [nm(k, clark), nm(f, mcCabe)]).
book(cn(qa76, b123, 1986),
        [prolog, programming, for, artificial, intelligence],
        [nm(ivan, bratko)]).
```

Les lecteurs

- Le nom, l'identificateur, l'adresse et le nombre de livres empruntés

```
reader(nm(ann, blake), 33333,
        addr([100, main], ottawa, k1a2b2),3).
reader(nm(jim, brady), 12345,
        addr([2, second], ottawa, k1n3m3),0).
reader(nm(tony, carp), 666666,
        addr([3, third], ottawa, k1k4p4),0).
```

Un prêt

```
loan(33333,
        cn(qa76, c203, 1984),
        date(nov, 25, 2014)).
loan(765432,
        cn(qa77, r56, 1977),
        date(oct, 20, 2014)).
```

Une requête

```
?- book(cn(X, Y, 1986), Title, Authors),
        \+ loan(_,cn(X, Y, 1986), _).
```

```
X = qa76
Y = b123
Title = [prolog, programming, for, artificial, intelligence]
Authors = [nm(ivan, bratko)]
```

Une relation

```
wrote(Auth, CallN, Title) :-
        book(CallN, Title, Authors),
        member(Auth, Authors).
```

Gérer les prêts – partie statique

```
borrowed(Name, Title) :-
        reader(Name, Id, _Addr, _),
        loan(Id, CallN, _DateDue),
        book(CallN, Title, _Auths).
```

Gérer les prêts – partie dynamique

```
:- dynamic book/3, reader/4, loan/3.

newLoan(Cn, Id, Due) :-
        assert(loan(Id, Cn, Due)).


?- newLoan(cn(qa76, b123, 1986),
           12345,
           date(nov, 15, 1986)).
```

Cachage de l'information

- L'appel à dynamic informe l'interpréteur que la définition d'un prédicat peut changer en cours d'exécution
- Le prédicat *assert* peut être utilisé pour mémoriser des solutions

- Attention: il ne faut pas abuser de *assert;*
  - Avec celui-ci des relations fausses à un moment peuvent devenir vraie plus tard.

?- solve(probleme, solution), assertz(solve(problem,solution)).

Gérer les prêt…

```
returns(Id, Cn) :-
        retract(loan(Id, Cn, _Due)),
        retract(reader(Nm, Id, A, N)),
        N1 is N - 1,
        assert(reader(Nm, Id, A, N1)).
```

Exemple

```
?- listing([loan, reader]).
loan(33333, cn(qa76, c283, 1984), date(nov, 25, 1986)).
loan(765432, cn(qa77, r56, 1977), date(oct, 20, 1986)).
loan(12345, cn(qa76, b123, 1986), date(nov, 15, 1986)).

reader(nm(ann, blake), 33333, addr([100, main], ottawa, k1a2b2), 3).
reader(nm(jim, brady), 12345, addr([2, second], ottawa, k1n3m3), 0).
reader(nm(tony, carp), 666666, addr([3, third], ottawa, k1k4p4), 0).

Yes
?- returns(33333, cn(qa76, c283, 1984)).

Yes
?- listing([loan, reader]).
loan(765432, cn(qa77, r56, 1977), date(oct, 20, 1986)).
loan(12345, cn(qa76, b123, 1986), date(nov, 15, 1986)).

reader(nm(jim, brady), 12345, addr([2, second], ottawa, k1n3m3), 0).
reader(nm(tony, carp), 666666, addr([3, third], ottawa, k1k4p4), 0).
reader(nm(ann, blake), 33333, addr([100, main], ottawa, k1a2b2), 2).

Yes
```

Exemple

```
:- dynamic lettre/2.
alphabet([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]).
lettre(A, B):-
  alphabet(C),
  lettre(A, C, B),
  asserta(lettre(A,B)).
lettre(A, [A|_], 1).
lettre(A, [B|C], D):-
  \+(A=B),
  lettre(A, C, E),
  D is E+1.
```

Exemple

?- lettre(s,X).
X = 19 .

?- lettre(h,X).
X = 8 .

?- lettre(b,X).
X = 2 .

```
4 ?- listing.

:- dynamic lettre/2.

lettre(A, C) :-
        alphabet(B),
        lettre(A, B, C),
        assert(lettre(A, C)).

lettre(A, [A|_], 1).
lettre(A, [B|C], D) :-
        \+ A=B,
        lettre(A, C, E),
        D is E+1.
alphabet([a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]).
true.
```

Générateur de solutions

- Mémorisation d'une table de multiplication

```
maketable :- L=[0,1,2,3,4,5,6,7,8,9],
             member(X,L),
             member(Y,L),
             Z is X*Y,
             assert(product(X,Y,Z)),
             fail.
```

Machine à états

```
:-dynamic here/1.

room(kitchen).
room(office).
room(hall).
room('dining room').
room(cellar).
location(desk, office).
location(apple, kitchen).
location(flashlight, desk).
location('washing machine', cellar).
location(broccoli, kitchen).
location(crackers, kitchen).
location(computer, office).
here(kitchen).
```

Exemple

```
door(office, hall).
door(kitchen, office).
door(hall, 'dining room').
door(kitchen, cellar).
door('dining room', kitchen).
connect(X,Y) :- door(X,Y).
connect(X,Y) :- door(Y,X).

can_go(Place):-
  here(X),
  connect(X, Place).
```

Exemple

```
goto(Place):-
  can_go(Place),
  move(Place),
  look.

move(Place):-
  retract(here(X)),
  asserta(here(Place)).
```

```
look :-
  here(Place),
  write('You are in the '), write(Place), nl,
  write('You can see:'), nl,
  list_things(Place),
  write('You can go to:'), nl,
  list_connections(Place).
```

Exemple

```
list_connections(Place) :-
  findall(X,connect(Place, X),B),
  tab(2),
  write(B),
  nl.

list_things(Place) :-
  findall(X,location(X, Place),B),
  tab(2),
  write(B),
  nl.
```

Prédicat clause

- Lorsqu'un prédicat est déclaré dynamique, son existence peut être vérifiée:

```
:- dynamic a/2.
a(1,2).
a(3,4).
a(X,Y):- b(X), b(Y).
```

```
?- clause(a(X,Y),B).
X = 1,
Y = 2,
B = true ;
X = 3,
Y = 4,
B = true ;
B = (b(X), b(Y)).
```

Retirer des prédicats

```
p(A) :-
        a(A).
p(A) :-
        a(A),
        b(A).
p(A) :-
        a(A),
        b(A),
        c(A).
```

```
?- retract((p(X):-a(X),b(X))).
true .

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

p(A) :-
        a(A).
p(A) :-
        a(A),
        b(A),
        c(A).
true.
```

Retirer tous les prédicats

?- retractall(p(Y)).
true.

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

true.

Tic Tac Toe

```
:- dynamic board/1.
board([A,B,C,D,E,F,G,H,I]).
mark(Player, [X|_],1,1) :- var(X), X=Player.
mark(Player, [_,X|_],2,1) :- var(X), X=Player.
mark(Player, [_,_,X|_],3,1) :- var(X), X=Player.
mark(Player, [_,_,_,X|_],1,2) :- var(X), X=Player.
mark(Player, [_,_,_,_,X|_],2,2) :- var(X), X=Player.
mark(Player, [_,_,_,_,_,X|_],3,2) :- var(X), X=Player.
mark(Player, [_,_,_,_,_,_,X|_],1,3) :- var(X), X=Player.
mark(Player, [_,_,_,_,_,_,_,X|_],2,3) :- var(X), X=Player.
mark(Player, [_,_,_,_,_,_,_,_,X|_],3,3) :- var(X), X=Player.
```

Tic Tac Toe

```
record(Player,X,Y) :-
   board(B),
   mark(Player,B,X,Y),
   retract(board(Bi)),
   assert(board(B)).

?- record(x,1,1), board(B),findall((X,Y),mark(o,B,X,Y),Moves).
```

Tic Tac Toe

```
value(Board,100) :- win(Board,o), !.
value(Board,-100) :- win(Board,x), !.
value(Board,E) :-
   findall(o,open(Board,o),MAX),
   length(MAX,Emax),      % # lines open to o
   findall(x,open(Board,x),MIN),
   length(MIN,Emin),      % # lines open to x
   E is Emax - Emin.
```

Tic Tac Toe

```
win([Z1,Z2,Z3|_],P) :- Z1==P, Z2==P, Z3==P.
win([_,_,_,Z1,Z2,Z3|_],P) :-  Z1==P, Z2==P, Z3==P.
win([_,_,_,_,_,_,Z1,Z2,Z3],P) :-  Z1==P, Z2==P, Z3==P.
win([Z1,_,_,Z2,_,_,Z3,_,_],P) :-  Z1==P, Z2==P, Z3==P.
win([_,Z1,_,_,Z2,_,_,Z3,_],P) :-  Z1==P, Z2==P, Z3==P.
win([_,_,Z1,_,_,Z2,_,_,Z3],P) :-  Z1==P, Z2==P, Z3==P.
win([Z1,_,_,_,Z2,_,_,_,Z3],P) :-  Z1==P, Z2==P, Z3==P.
win([_,_,Z1,_,Z2,_,Z3,_,_],P) :-  Z1==P, Z2==P, Z3==P.
```

Tic Tac Toe

```
open([Z1,Z2,Z3|_],Player) :- (var(Z1) | Z1 == Player),
        (var(Z2) | Z2 == Player), (var(Z3) | Z3 == Player).
open([_,_,_,Z1,Z2,Z3|_],Player) :- (var(Z1) | Z1 == Player),
        (var(Z2) | Z2 == Player), (var(Z3) | Z3 == Player).
open([_,_,_,_,_,_,Z1,Z2,Z3],Player) :- (var(Z1) | Z1 == Player),
        (var(Z2) | Z2 == Player), (var(Z3) | Z3 == Player).
open([Z1,_,_,Z2,_,_,Z3,_,_],Player) :- (var(Z1) | Z1 == Player),
        (var(Z2) | Z2 == Player), (var(Z3) | Z3 == Player).
open([_,Z1,_,_,Z2,_,_,Z3,_],Player) :- (var(Z1) | Z1 == Player),
        (var(Z2) | Z2 == Player), (var(Z3) | Z3 == Player).
open([_,_,Z1,_,_,Z2,_,_,Z3],Player) :- (var(Z1) | Z1 == Player),
        (var(Z2) | Z2 == Player), (var(Z3) | Z3 == Player).
open([Z1,_,_,_,Z2,_,_,_,Z3],Player) :- (var(Z1) | Z1 == Player),
        (var(Z2) | Z2 == Player), (var(Z3) | Z3 == Player).
open([_,_,Z1,_,Z2,_,Z3,_,_],Player) :- (var(Z1) | Z1 == Player),
        (var(Z2) | Z2 == Player), (var(Z3) | Z3 == Player).
```