Centre Universitaire Adelhafid  Boussouf Mila

Institut des Sciences et de la technologie
Département des mathématiques et de l'informatique
Filière Informatique

# Intelligence Artificielle

# Chapitre 5
# Les listes de Prolog

## I- Définition

- En PROLOG, la liste vide est notée [ ]. La liste [e | L] représente une liste dont e est la tête et L le reste de la liste.

**Exemples :**

?- [A | B] = [1,2,3]
A = 1
B = [2,3]

?- [A,B | C] = [1,5,10,15]
A = 1
B = 5
C = [10, 15]

- Une liste est une structure récursive: la liste [X|L] est composée d'un élément de tête X et d'une queue L qui est elle-même une liste. La totalité d'une liste peut être traitée en agissant sur le premier élément, et ensuite sur le reste de la liste de manière récursive sur la queue.
- Par conséquent, les prédicats Prolog qui manipulent les listes seront généralement définies par :

  - **Une ou plusieurs clauses récursives,** définissant la relation sur la liste [X|L] en fonction de la relation sur la queue de la liste L,
  - **Une ou plusieurs clauses non récursives** assurant la terminaison de la manipulation, et définissant la relation pour une liste particulière comme la liste vide, ou la liste dont l'élément de tête vérifie une certaine condition.

**- Exemples :**

```
afficher([T|Q]) :-  writeln(T) , nl , afficher(Q).
afficher([]).

appartient_à(X,[X|_]).
appartient_à(X,[_|Queue]) :- appartient_à(X,Queue).
```

- En interne, une liste est définie par une construction récursive en utilisant le foncteur '.' d'arité 2, c'est donc un terme composé. Le terme '.'(H, T) est une liste.

- Le premier élément H est appelé la tête, suivi par le reste de la liste, indiqué comme T (la queue).

Exemple :

- La liste [1, 2, 3] est représentée en interne comme .(1, .(2, .(3, [])).

- La liste [a,b|L] correspond à la liste '.'(a, '.'(b, L)) et contient les 2 éléments a et b , suivis de la liste (inconnue) L.

- La liste [X1, X2, X3, ..., Xn|[]] est également notée [X1,X2,X3,..., Xn].

- Les listes peuvent contenir des éléments hétérogènes (toute sorte de termes). En particulier, il est possible de construire des listes qui contiennent d'autres listes.

  Exemple : La liste [a, [a, b, c], [[d, e], f], g(a), [h, i]] est totalement valide.

## II- Opérations sur les listes

## II-1 Extraire un élément d'une liste :

- Le programme suivant ôte un élément d'une liste :

```
extraire(X, [X|L], L).

extraire(X, [T|Q], [T|Q1]) :- extraire(X, Q, Q1).
```

- On peut paraphraser ce programme ainsi :

- si X est en tête de liste alors le résultat est la queue L de la liste.
- Si X est trouvé dans la queue L de la liste [Y|L], alors ôter X de L, ce qui donne L1 et recopier l'élément de tête Y dans le résultat.

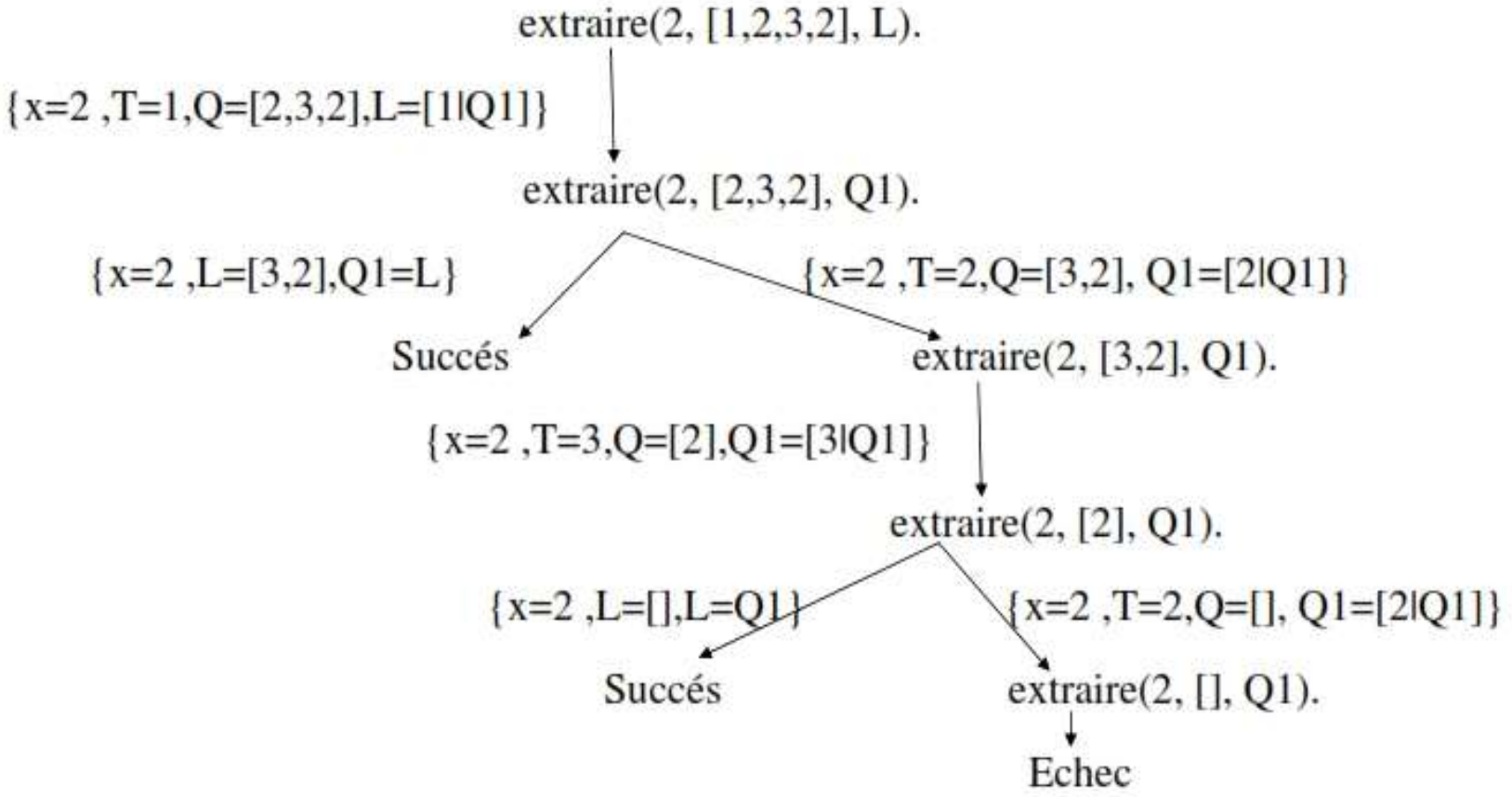
**Exemples :**

?- extraire(3, [3], []).
YES

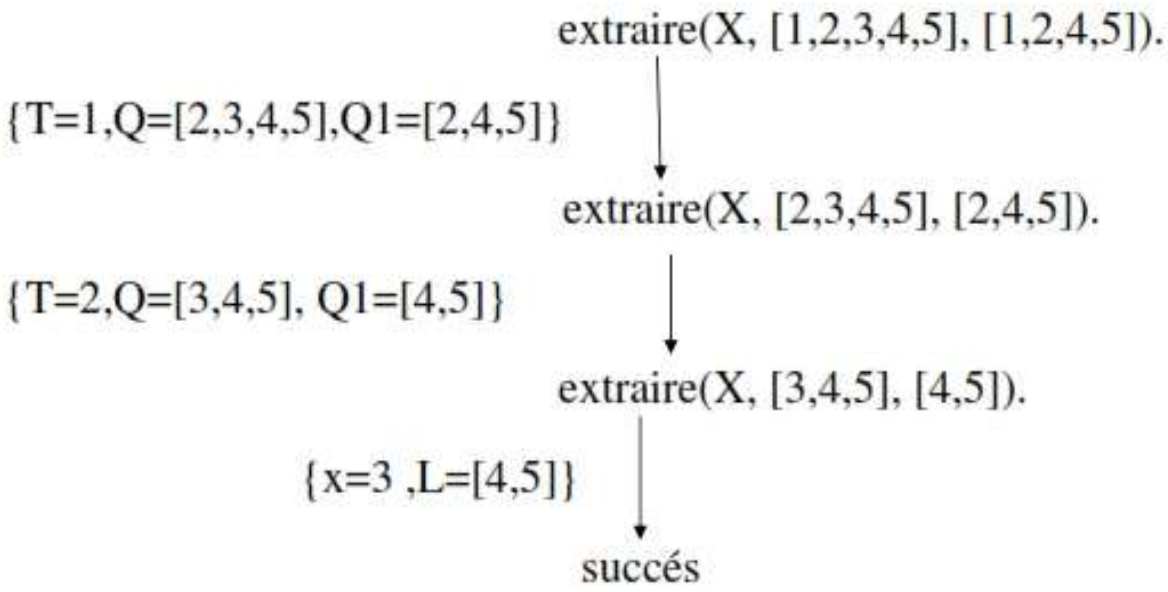
?- extraire(2, [1,3,2,1], L).
L=[1,3,1] ;

?- extraire(2, [1,2,3,2], L).
L=[1,3,2] ;

L=[1,2,3]


extraire(2, [1,2,3,2], L).
{x=2 ,T=1,Q=[2,3,2],L=[1|Q1]}
extraire(2, [2,3,2], Q1).
{x=2 ,L=[3,2],Q1=L}
{x=2 ,T=2,Q=[3,2], Q1=[2|Q1]}
Succés
extraire(2, [3,2], Q1).
{x=2 ,T=3,Q=[2],Q1=[3|Q1]}
extraire(2, [2], Q1).
{x=2 ,L=[],L=Q1}
{x=2 ,T=2,Q=[], Q1=[2|Q1]}
Succés
extraire(2, [], Q1).
Echec

?- extraire(X,[1,2,3,4,5],[1,2,4,5]).
X=3

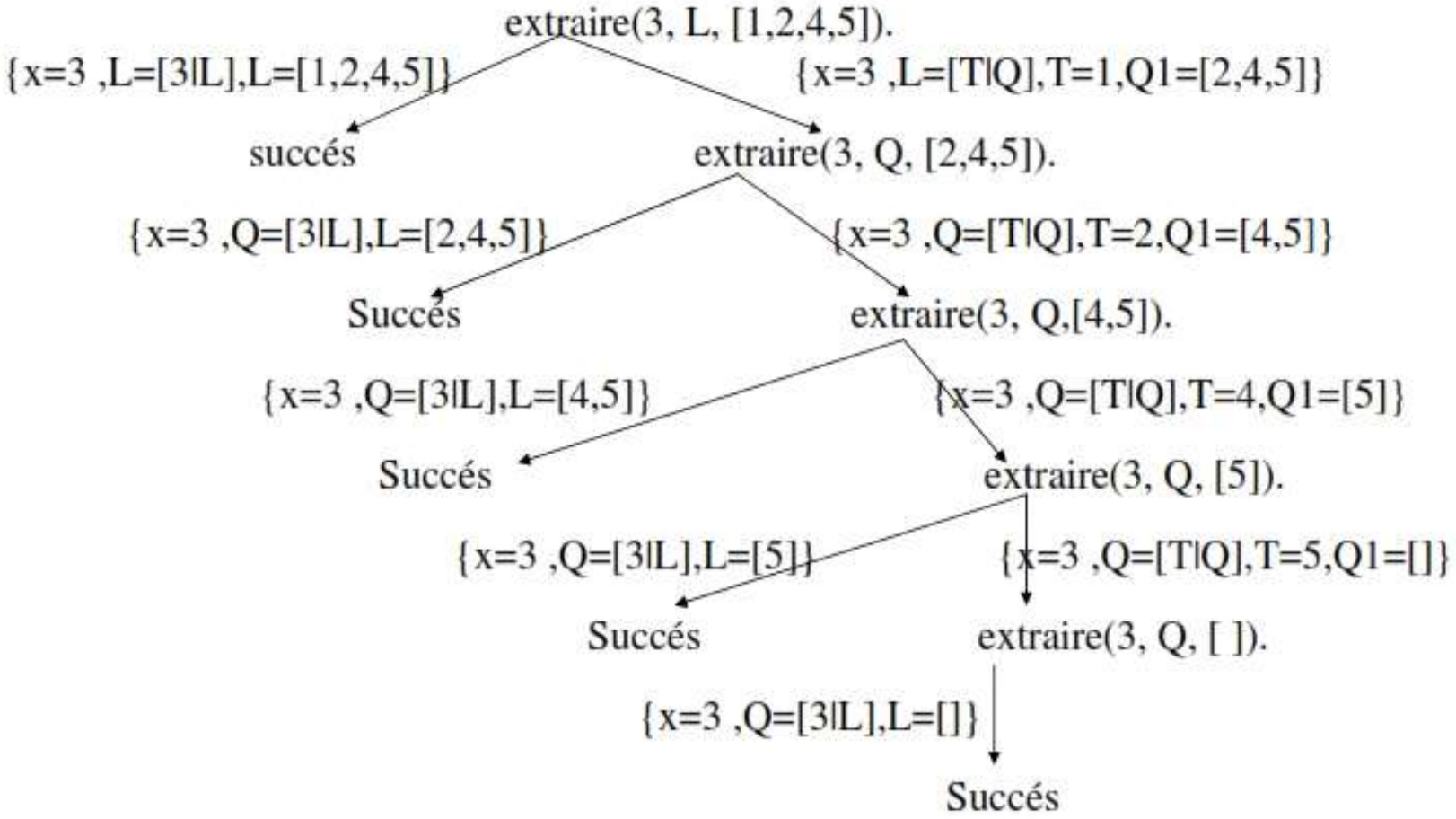


- **La réversibilité** nous permet d'utiliser un même prédicat pour une tache et la tache inverse.

**Exemples :**

- Le programme précédent a été conçu pour extraire un élément d'une liste. Pourtant, le même programme peut réaliser l'opération inverse (insérer un élément):

```
?- extraire(3, L, [1,2,4,5]).
L=[3,1,2,4,5] ;
L=[1,3,2,4,5] ;
L=[1,2,3,4,5] ;
L=[1,2,4,3,5] ;
L=[1,2,4,5,3]
```


extraire(3, L, [1,2,4,5]).
{x=3 ,L=[3|L],L=[1,2,4,5]}
{x=3 ,L=[T|Q],T=1,Q1=[2,4,5]}
succés
extraire(3, Q, [2,4,5]).
{x=3 ,Q=[3|L],L=[2,4,5]}
{x=3 ,Q=[T|Q],T=2,Q1=[4,5]}
Succés
extraire(3, Q,[4,5]).
{x=3 ,Q=[3|L],L=[4,5]}
{x=3 ,Q=[T|Q],T=4,Q1=[5]}
Succés
extraire(3, Q, [5]).
{x=3 ,Q=[3|L],L=[5]}
{x=3 ,Q=[T|Q],T=5,Q1=[]}
Succés
extraire(3, Q, [ ]).
{x=3 ,Q=[3|L],L=[]}
Succés

- Le prédicat member vérifie si un élément appartient à une liste, mais il peut être utilisé pour ressortir les éléments de la liste.

```
member(X,[X|T]).
member(X,[H|T]):-member(X,T).
```

```
member1 (X,[X|_]).
member1 (X,[_|T]):-member(X,T).
```

```
?- member(5,[6,7,5,8]).
true .

?- member(X,[6,7,5,8]).
X = 6 ;
```

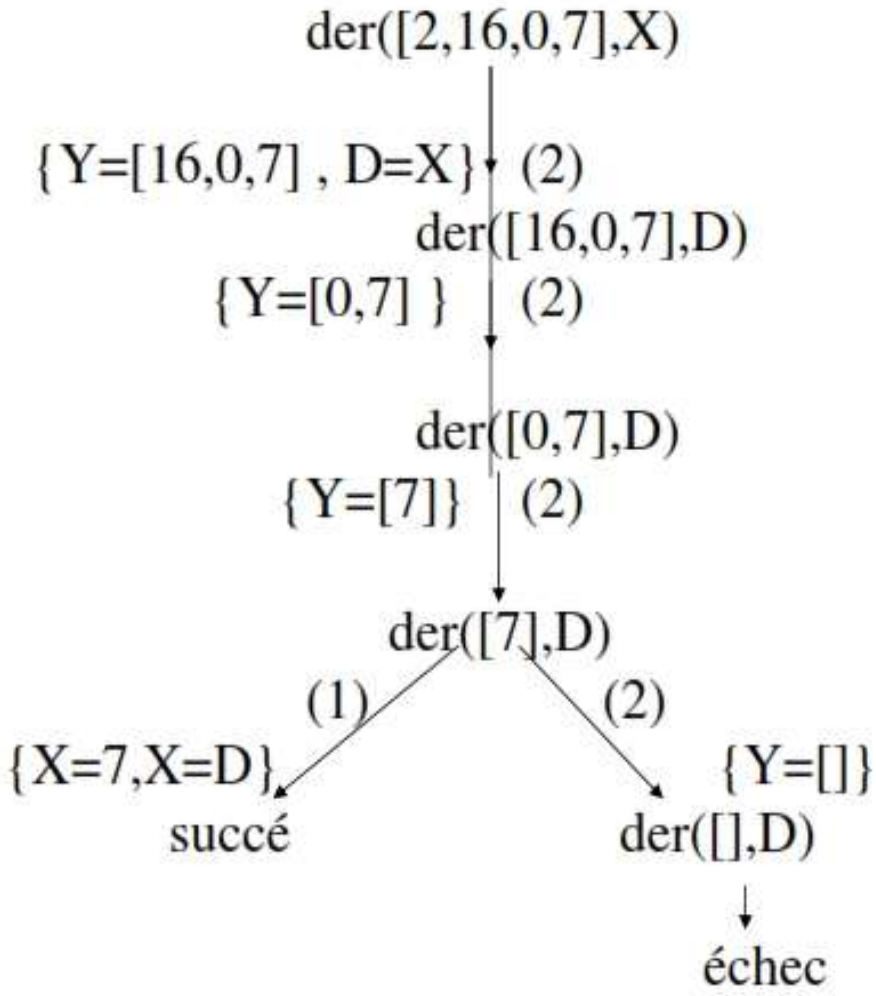
X = 7 ;
X = 5 ;
X = 8 ;
false.

## II-2 Dernier élément d'une liste

```
der([X], X).                    |   der(X,[X]).
der([_|Y], D) :- der(Y, D).     |   der(X, [_|T]):- der (X,T).
```

Exemple:

?- der([2,16,0,7],X).
X=7


der([2,16,0,7],X)
{Y=[16,0,7] , D=X}
(2)
der([16,0,7],D)
{Y=[0,7] }
(2)
der([0,7],D)
{Y=[7]}
(2)
der([7],D)
(1)
(2)
{X=7,X=D}
{Y=[]}
succé
der([],D)
échec

## II-3 Longueur d'une liste

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

## III- Quelques exemples récursifs

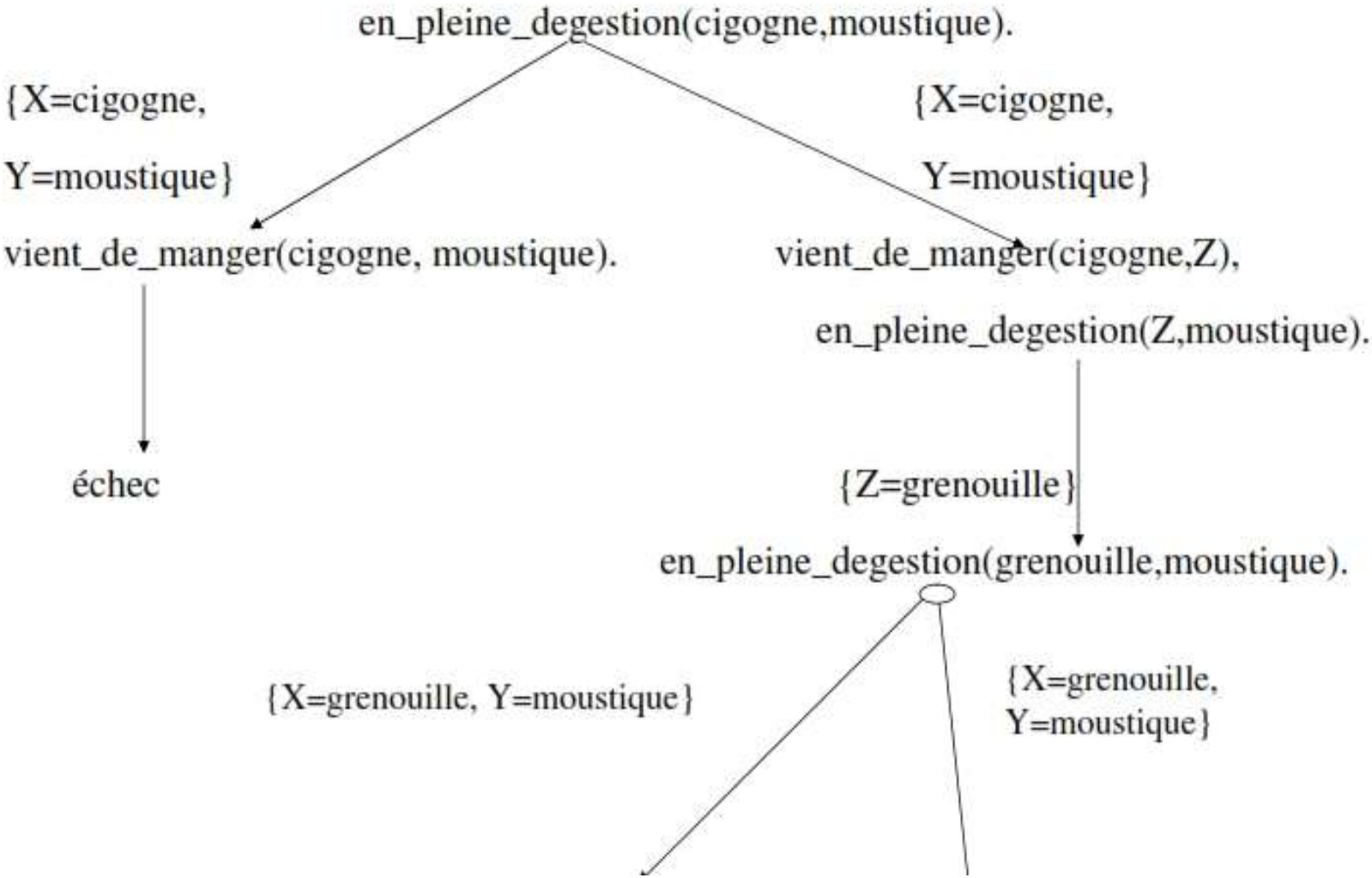
- Un prédicat est récurrent si une ou plusieurs règles de sa définition font référence à ce même prédicat.

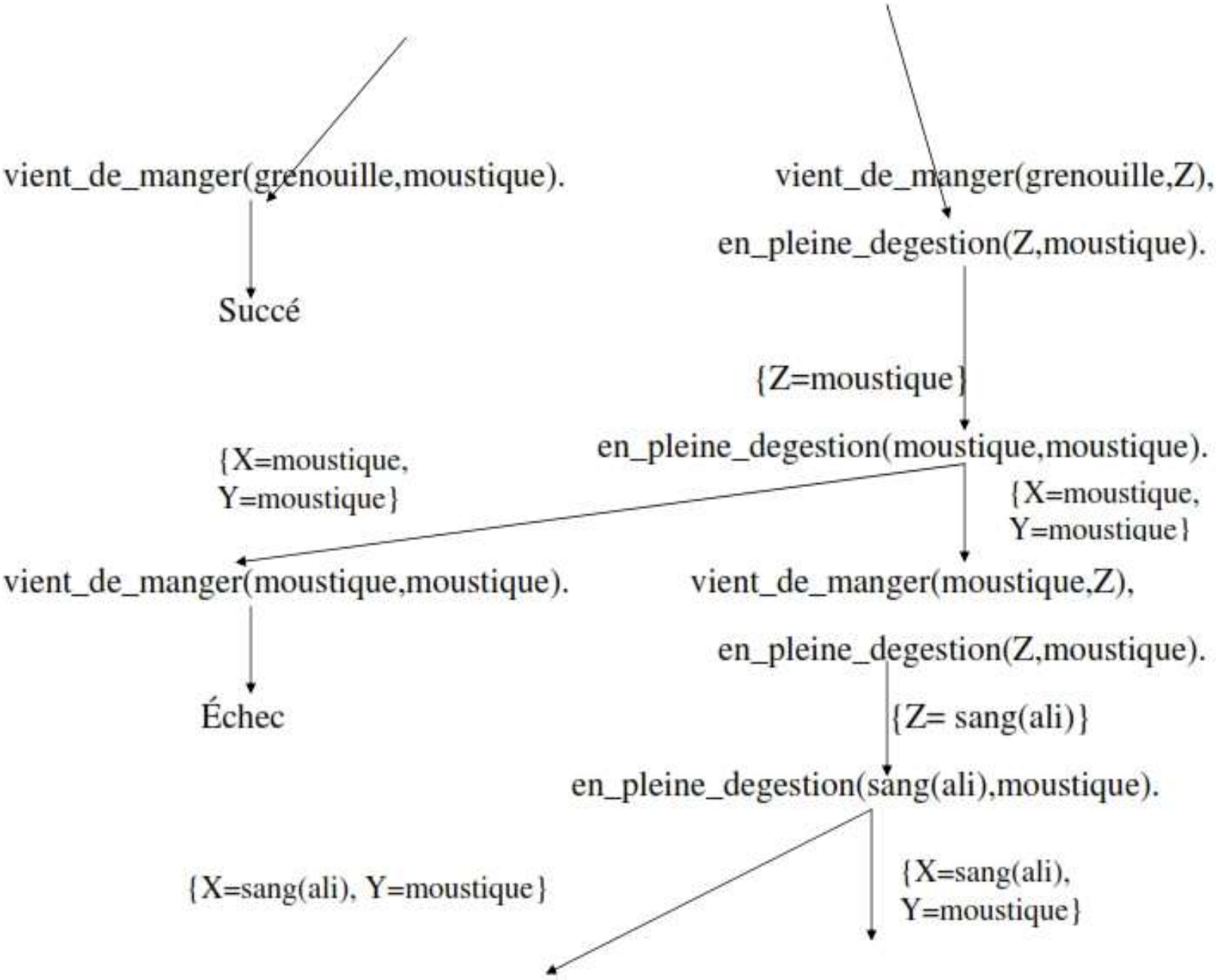
Exemple1 :

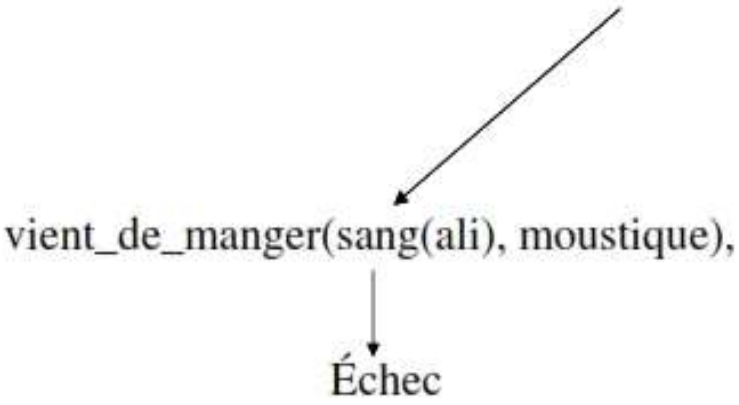
```
en_pleine_degestion(X,Y) :- vient_de_manger(X,Y).

en_pleine_degestion(X,Y) :- vient_de_manger(X,Z) ,
en_pleine_degestion(Z,Y).
```

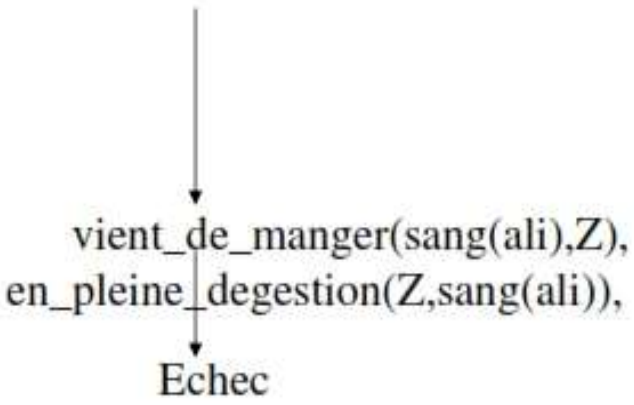
```
vient_de_manger(moustique,sang(ali)).
vient_de_manger(grenouille,moustique).
vient_de_manger(cigogne,grenouille).
```

?- en_pleine_degestion(cigogne,moustique).




vient_de_manger(grenouille,moustique).
Succé
vient_de_manger(grenouille,Z),
en_pleine_degestion(Z,moustique).
{Z=moustique}
en_pleine_degestion(moustique,moustique).
{X=moustique,
Y=moustique}
{X=moustique,
Y=moustique}
vient_de_manger(moustique,moustique).
Échec
vient_de_manger(moustique,Z),
en_pleine_degestion(Z,moustique).
{Z= sang(ali)}
en_pleine_degestion(sang(ali),moustique).
{X=sang(ali), Y=moustique}
{X=sang(ali),
Y=moustique}




Exemple2 :

enfant_de (ali , mohamed).

enfant_de (mohamed , adam).

enfant_de (adam , khalil).

descendant_de (X,Y) :- enfant(X,Y).

descendant_de (X,Y) :- enfant(X,Z) , enfant_de (Z,Y).

?- descendant_de (ali , khalil).

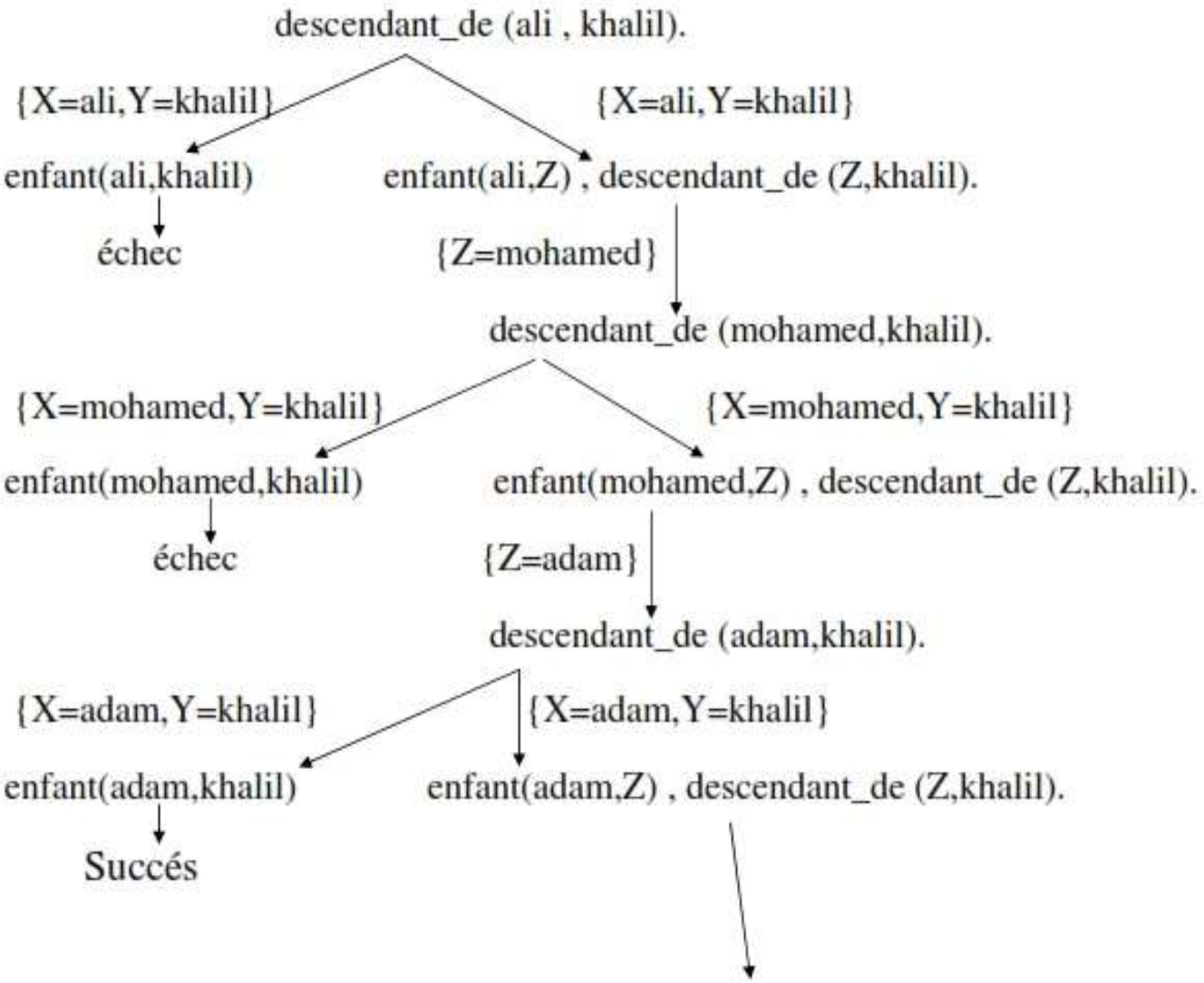
False.

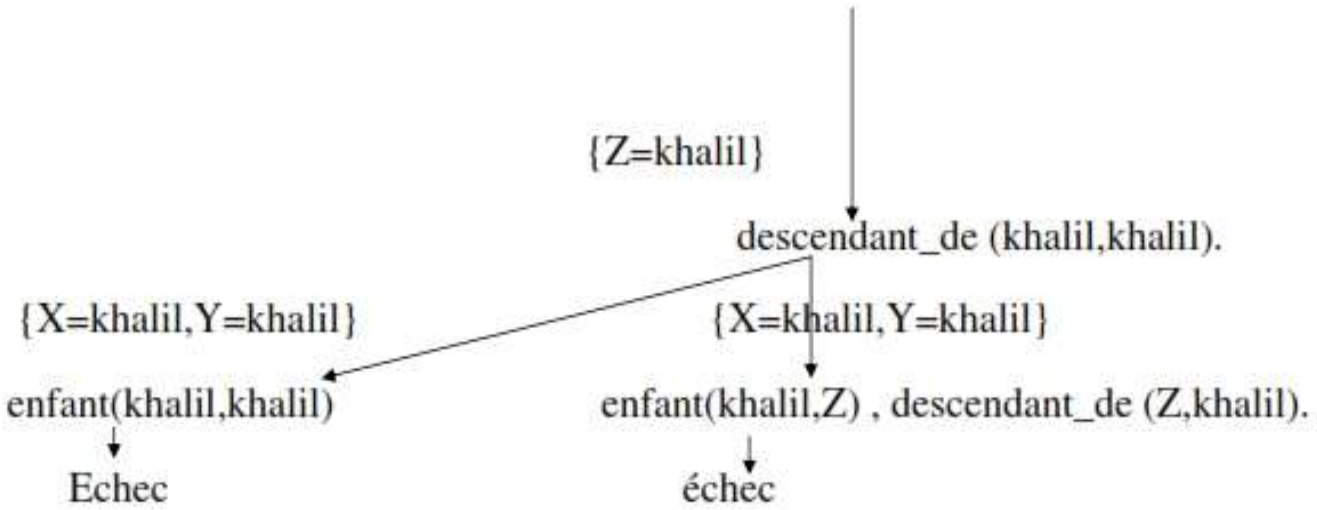
descendant_de (X,Y) :- enfant(X,Y).

descendant_de (X,Y) :- enfant(X,Z) , descendant_de (Z,Y).

?- descendant_de (ali , khalil).

true.


descendant_de (ali , khalil).
{X=ali,Y=khalil}
{X=ali,Y=khalil}
enfant(ali,khalil)
enfant(ali,Z) , descendant_de (Z,khalil).
échec
{Z=mohamed}
descendant_de (mohamed,khalil).
{X=mohamed,Y=khalil}
{X=mohamed,Y=khalil}
enfant(mohamed,khalil)
enfant(mohamed,Z) , descendant_de (Z,khalil).
échec
{Z=adam}
descendant_de (adam,khalil).
{X=adam,Y=khalil}
{X=adam,Y=khalil}
enfant(adam,khalil)
enfant(adam,Z) , descendant_de (Z,khalil).
Succés


{Z=khalil}
descendant_de (khalil,khalil).
{X=khalil,Y=khalil}
{X=khalil,Y=khalil}
enfant(khalil,khalil)
enfant(khalil,Z) , descendant_de (Z,khalil).
Echec
échec

Dans vos définitions d'un prédicat récursif :

- Donnez le cas de base. Puis,
- Dans la règle récurrente, placez les buts qui déclenchent les appels récursifs aussi loin que possible vers la droite du corps de la règle.