INFf3
Programmation logique

Cours 3 : Techniques de programmation

Benoît Lemaire

Université Grenoble Alpes
L2 - MIASHS
Grenoble – France

Ces diapos se sont inspirées de celles créées par Jean-Michel Adam

Le prédicat writef/2

- **writef(<chaine à afficher avec des %t pour les parties variables>,<liste des valeurs>)**

  Exemple :

```
writef("Le joueur %t a gagné %t points.",[Nom,Pts])
```

Écriture d’un test (type if)

- **Ecrire le prédicat min/3 qui détermine la plus petite de deux valeurs.**
- **Exemple :**

```
?- min(4,7,R).

R=4.

?- min(3,0,R).

R=0
```

Écriture d'un test (type if)

- **Il y a 2 cas, donc on écrit deux règles.**

```
min(X,Y,Y) :- X>=Y.
```
le minimum entre X et Y est Y si X >= Y.

```
min(X,Y,X) :- X=<Y.
```
le minimum entre X et Y est X si X <= Y.

Si la première échoue, Prolog utilisera la seconde parce qu'il cherche tous les moyens de satisfaire la requête.

Programmation avec accumulateur

- Écrire un prédicat qui inverse l'ordre des éléments d'une liste. Exemple :

```
?- inverse([a,b,c,d],L).

L=[d,c,b,a]
```

Version sans accumulateur

- *Une liste inversée est une liste composée de l'inversion du reste de la liste et du premier élément.*

  On va avoir besoin pour cela de concaténer des listes, parce qu'on ne peut pas utiliser | qui sépare le premier du reste (et non le dernier du reste).

  Prédicat prédéfini : append/3

```
?- append([a,b],[4,5,6,7],L)

L=[a,b,4,5,6,7]
```

Version sans accumulateur

- *Une liste inversée est une liste composée de l'inversion du reste de la liste et du premier élément.*

```
inverse([],[]).

inverse([X|L],R) :-inverse(L,Linv),append(Linv,[X],R).
```

Version avec accumulateur

- On ajoute un argument, initialement une liste vide, et on lui ajoute les éléments de la solution au fur et à mesure. A la fin, cet accumulateur est la solution.
- *Inverser une liste, étant donné un accumulateur, c'est recommencer avec le reste de la liste et l'accumulateur auquel on a ajouté au début le premier élément.*

```
inverse(L,R) :- inverse(L,[],R).

inverse([],Acc,Acc). (ou inverse([],Acc,R):- R=Acc.)

inverse([X|L],Acc,R) :- inverse(L,[X|Acc],R).
```

Coupure (*cut*, coupe-choix)

- L’exploration systématique est une force de Prolog,
- mais elle conduit parfois à une explosion combinatoire...
- Il existe un moyen de restreindre l’exploration en élaguant des branches de l’arbre de recherche.
- Notation : !
- C’est un prédicat qui réussit toujours et qui a pour effet de supprimer les points de choix en attente.

Coupure dans une règle

- Le coupe-choix ne conserve que les unifications faites avant la suppression des points de choix.

```
a(1).
a(2).
b(1).
b(2).
p(X):- a(X), b(X).
```

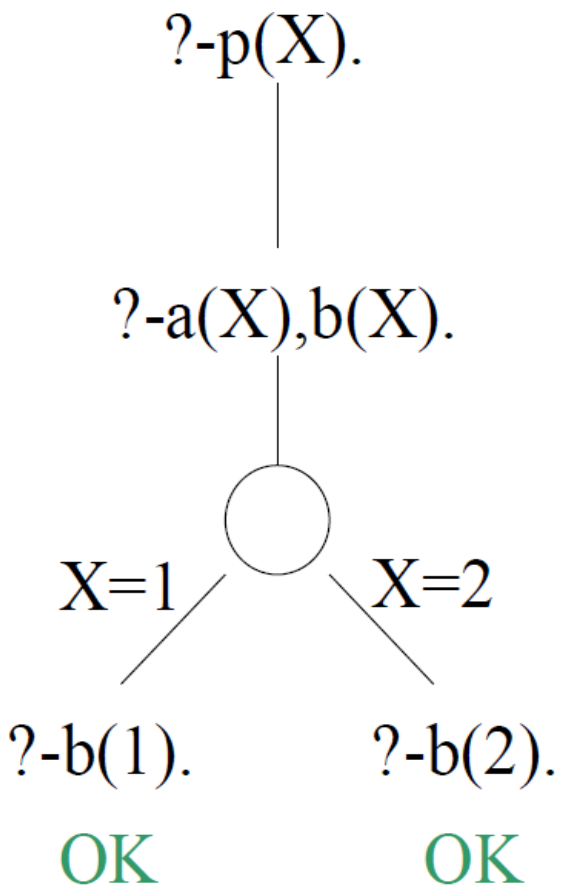
```
?- p(X).
X =1 ;
X=2.
```

```
a(1).
a(2).
b(1).
b(2).
p(X):- a(X), !, b(X).
```

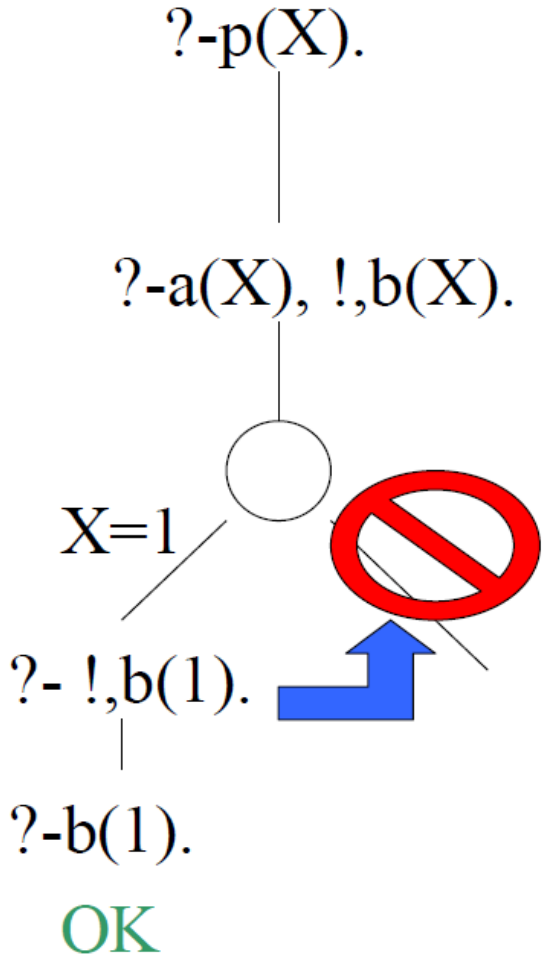
```
?- p(X).
X =1.
```

Arbre de Recherche

Sans Coupure



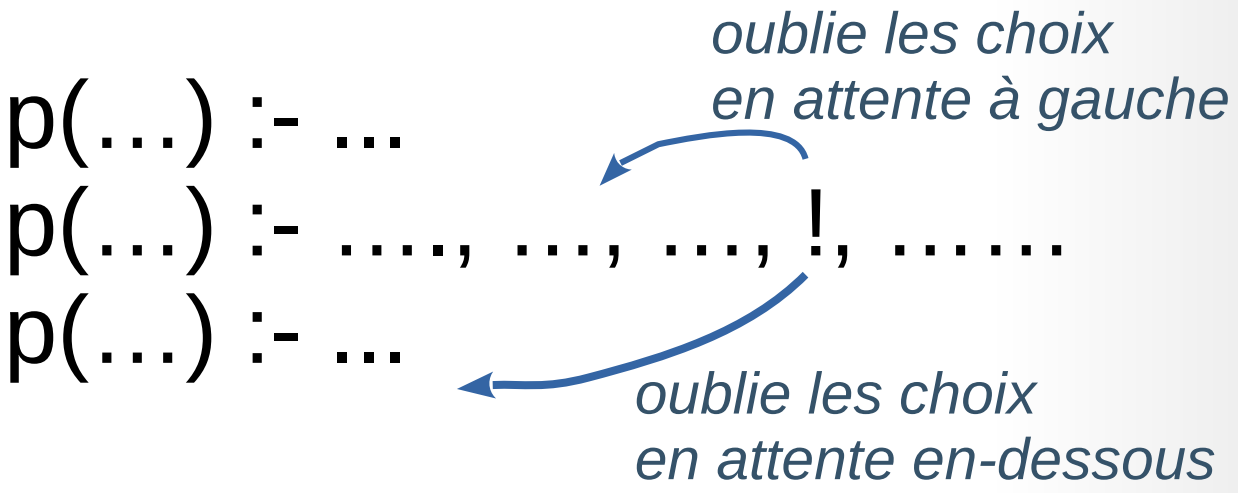
Avec Coupure



Coupure (cut)

- Le coupe-choix permet
  - d'éliminer des points de choix
  - d'éliminer des tests conditionnels que l'on sait inutiles ⇒ plus d'efficacité lors de l'exécution du programme
- Quand Prolog exécute un coupe choix,
  - il ignore toutes les règles du même prédicat **en-dessous**.
  - élague toutes les solutions pour la conjonction de buts qui apparaissent à sa **gauche** dans la règle
  - En revanche il n'affecte nullement ce qui se trouve à sa droite

Coupure (cut)


p(…) :- …
p(…) :- …., …, …, !, ……
p(…) :- …
oublie les choix
en attente à gauche
oublie les choix
en attente en-dessous

Retour sur l'exemple du minimum

```
min(X,Y,Y) :- X>=Y.
```
le minimum entre X et Y est Y si X >= Y.

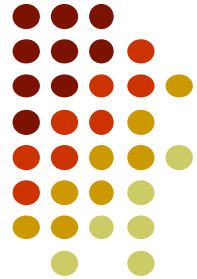
```
min(X,Y,X) :- X=<Y.
```
le minimum entre X et Y est X si X <= Y.

```
min(X,Y,Y) :- X>=Y,!.
```
le minimum entre X et Y est Y si X >= Y.

```
min(X,Y,X).
```
sinon, c'est X.

Autre exemple, suite de Syracuse

Référence Wikipedia



```
syracuse(1,1).
syracuse(N,L):- N mod 2 =:= 1, N1 is N*3+1, syracuse(N1,L1), L is L1+1.
syracuse(N,L):- N mod 2 =:= 0, N1 is N//2, syracuse(N1,L1), L is L1+1.
```

Ce programme produit des solutions fausses !

Autre exemple, suite de Syracuse

Référence Wikipedia

```
syracuse(1,1).
syracuse(N,L):- N mod 2 =:= 1, N1 is N*3+1, syracuse(N1,L1), L is L1+1.
syracuse(N,L):- N mod 2 =:= 0, N1 is N//2, syracuse(N1,L1), L is L1+1.
```

```
syracuse(1,1):- !.
```

évite d'autres solutions fausses

```
syracuse(N,L):- N mod 2 =:= 1, !, N1 is N*3+1, syracuse(N1,L1), L is L1+1.
syracuse(N,L):- N1 is N//2, syracuse(N1,L1), L is L1+1.
```

évite la condition inverse dans la règle suivante

Le prédicat fail

- Ce prédicat est toujours faux. Il oblige Prolog à un retour arrière jusqu'au dernier point de choix.
- Il permet de faire des itérations :

```
afficheChiffres :- between(1,9,X), write(X), nl, fail.
afficheChiffres.
```

Combinaison cut-fail

Permet d’exprimer une exception ou la négation d’une condition. Exemple :

```
premier(N):-N1 is N-1,between(2,N1,X),N mod X=:=0,!,fail.
premier(_).
```