

SORTIR DE PROLOG PUR

Chaînes de caractères, entrées / sorties

Boucles mues par l'échec

Manipuler des programmes

Points de choix, coupure, négation

CHAÎNES DE CARACTÈRES

- Une chaîne de caractères est représentée par une liste de codes ASCII.
- Le prédicat `name` permet de passer d'une version "lisible" de la chaîne à sa représentation sous forme de liste.
- Pour obtenir la liste :

```
?- name("toto",L).  
L = [116, 111, 116, 111].
```
- Pour visualiser la chaîne :

```
?- name(S, [116, 111, 116, 111]).  
S = toto
```
- On peut donc manipuler les chaînes de caractères avec des opérations de listes.

EXEMPLE : LES MUTANTS

```
non_vide([_|_]).

mutant(S) :-
    animal(MotD), animal(MotF),
    name(MotD,D), name(MotF,F),
    append(Debut,Milieu,D),
    non_vide(Debut),
    non_vide(Milieu),
    append(Milieu,_,F),
    append(Debut,F,M),
    name(S,M).

animal("alligator").
animal("lapin").
animal("tortue").
animal("pintade").
animal("cheval").
```

```
?- mutant(X).

X = alligatortue ;

X = lapintade ;

X = chevalligator ;

X = chevalapin ;

false.
```

ENTRÉES-SORTIES

nl

write(Term)

writeln(Term) ou write_In(Term)

read(Term)

BOUCLES MUES PAR ÉCHEC

Deux prédicats prédéfinis :

true : réussit toujours

fail : échoue toujours

?- member(X,[a,b,c]), write_ln(X), fail.

a

b

c

false.

CONJUGUER LES VERBES DU PREMIER GROUPE

?- conjugue("chanter").

je chante

tu chantes

il chante

nous chantons

vous chantez

ils chantent

true.

LE PROGRAMME

```
conjugue(Infinitif) :- name(Infinitif,Liste), radical(Liste,Radical),  
                        termine(Radical).
```

```
radical(Infinitif,Radical) :- append(Radical,[101,114],Infinitif).
```

```
terminaison("je","e").
```

```
terminaison("tu","es").
```

```
terminaison("il","e").
```

```
terminaison("nous","ons").
```

```
terminaison("vous","ez").
```

```
terminaison("ils","ent").
```

```
termine(Racine) :- terminaison(Debut,Fin), write(Debut),  
                    write(' '), name(Fin,Lfin), append(Racine,Lfin,Verbe),  
                    name(V,Verbe), write(V), nl, fail.
```

```
termine(_).
```

TROUVER TOUTES LES SOLUTIONS

findall(Variable, But, Liste)

?- findall(X, member(X,[a,b,c]), R).

R = [a, b, c]

-> compter le nombre de solutions

?- findall(X,solution(X),L), length(L,N).

MANIPULER DES PROGRAMMES (1)

- Ajouter une clause
assert(pere(pierre, paul)).

- Enlever une clause

?- retract(pere(X,Y)).

X = pierre

Y = paul ;

X = paul

Y = jean

?- retractall(pere(_,_)).

MANIPULER DES PROGRAMMES (2)

MANIPULER UNE VARIABLE GLOBALE

```
?- assert(tableau([[marie,5],[jean,3],[arnaud,32],[pierre,6]])).
true.
?- tableau(L).
L = [[marie, 5], [jean, 3], [arnaud, 32], [pierre, 6]].
?- valeur(jean,N).
N = 3 .
?- ajoute_tableau(claire,7).
true.
?- tableau(L).
L = [[claire, 7], [marie, 5], [jean, 3], [arnaud, 32], [pierre, 6]].
?- change_valeur(jean,3,2).
true.
?- tableau(L).
L = [[jean, 2], [claire, 7], [marie, 5], [arnaud, 32], [pierre, 6]].
```

MANIPULER DES PROGRAMMES (3)

MANIPULER UNE VARIABLE GLOBALE

```
valeur(P,N) :-  
    tableau(L), member([P,N],L).  
ajoute_tableau(P,N) :-  
    tableau(L),  
    retract(tableau(L)), assert(tableau([[P,N]|L])).  
supprime_tableau(P,N) :-  
    tableau(L), delete(L,[P,N],NewL),  
    retract(tableau(L)), assert(tableau(NewL)).  
change_valeur(P,N1,N2) :-  
    supprime_tableau(P,N1), ajoute_tableau(P,N2).
```

MANIPULER DES PROGRAMMES (4)

Problème : lorsque le tableau n'a pas été initialisé

```
?- tableau(L).
```

```
ERROR: toplevel: Undefined procedure: tableau/1
```

Solution : ajouter dans le programme

```
:- dynamic tableau/1, toto/2.
```

→

```
?- tableau(L).
```

```
false.
```

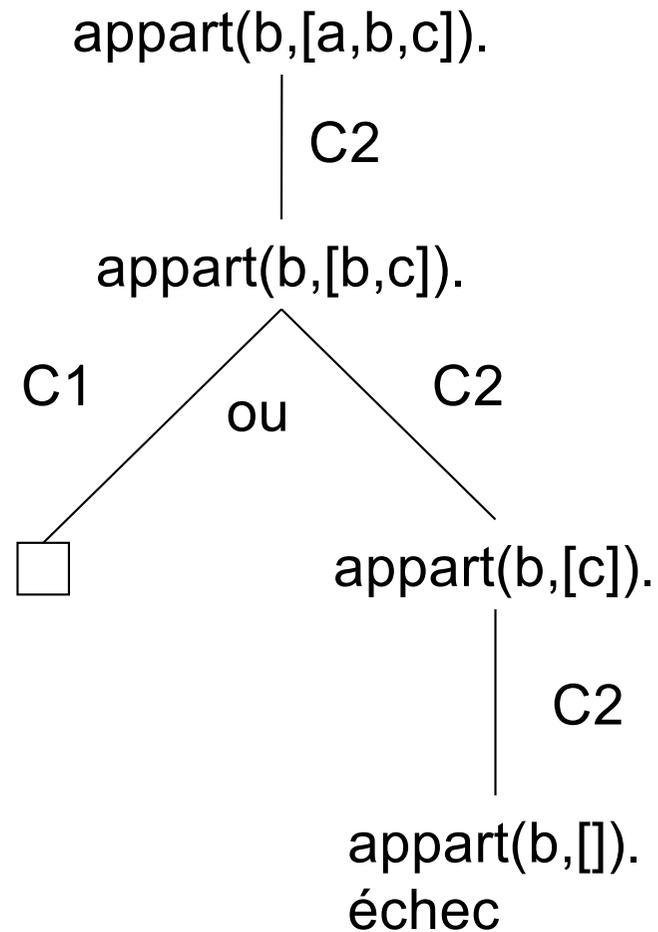
POINTS DE CHOIX

C1 : appart(X,[X|_]).

C2 : appart(X,[_|L]) :- appart(X,L).

```
?- trace, appart(b, [a,b,c]), fail.  
Call: (8) appart(b, [a, b, c]) ? creep  
Call: (9) appart(b, [b, c]) ? creep  
Exit: (9) appart(b, [b, c]) ? creep  
Redo: (9) appart(b, [b, c]) ? creep  
Call: (10) appart(b, [c]) ? creep  
Call: (11) appart(b, []) ? creep  
Fail: (11) appart(b, []) ? creep  
false.
```

REPRÉSENTATION PAR UN GRAPHE ET/OU



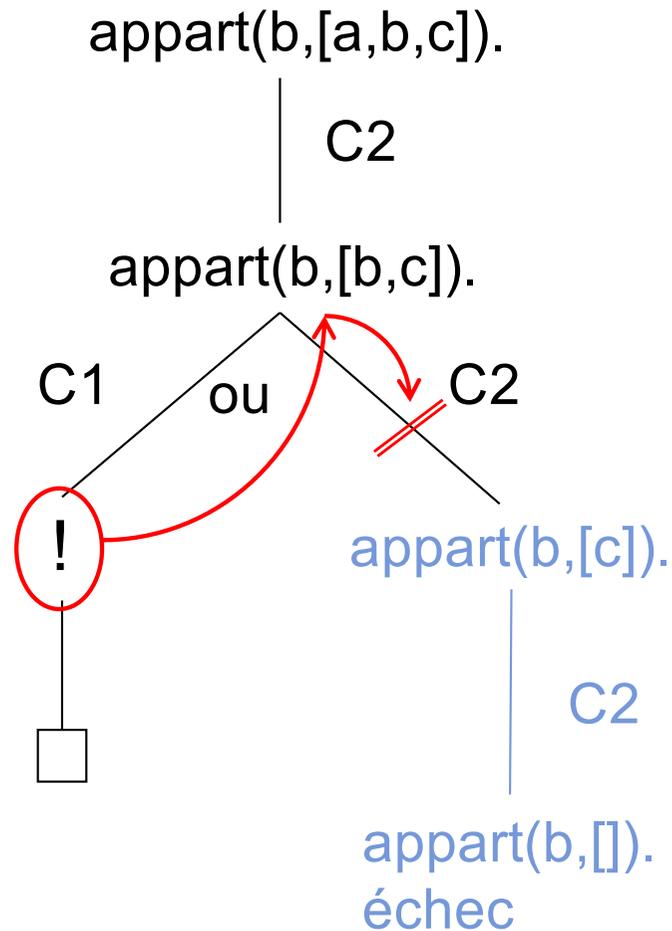
LA COUPURE

La coupure interdit le retour arrière sur des points de choix.

C1 : $\text{appart}(X, [X|_]) \text{ :- } ! .$

C2 : $\text{appart}(X, _ | L) \text{ :- } \text{appart}(X, L).$

COUPURE VERTE : ÉLIMINER DES POINTS DE CHOIX INUTILES



?- trace, appart(b,[a,b,c]),fail.

Call: (8) appart(b, [a, b, c]) ?

creep

Call: (9) appart(b, [b, c]) ?

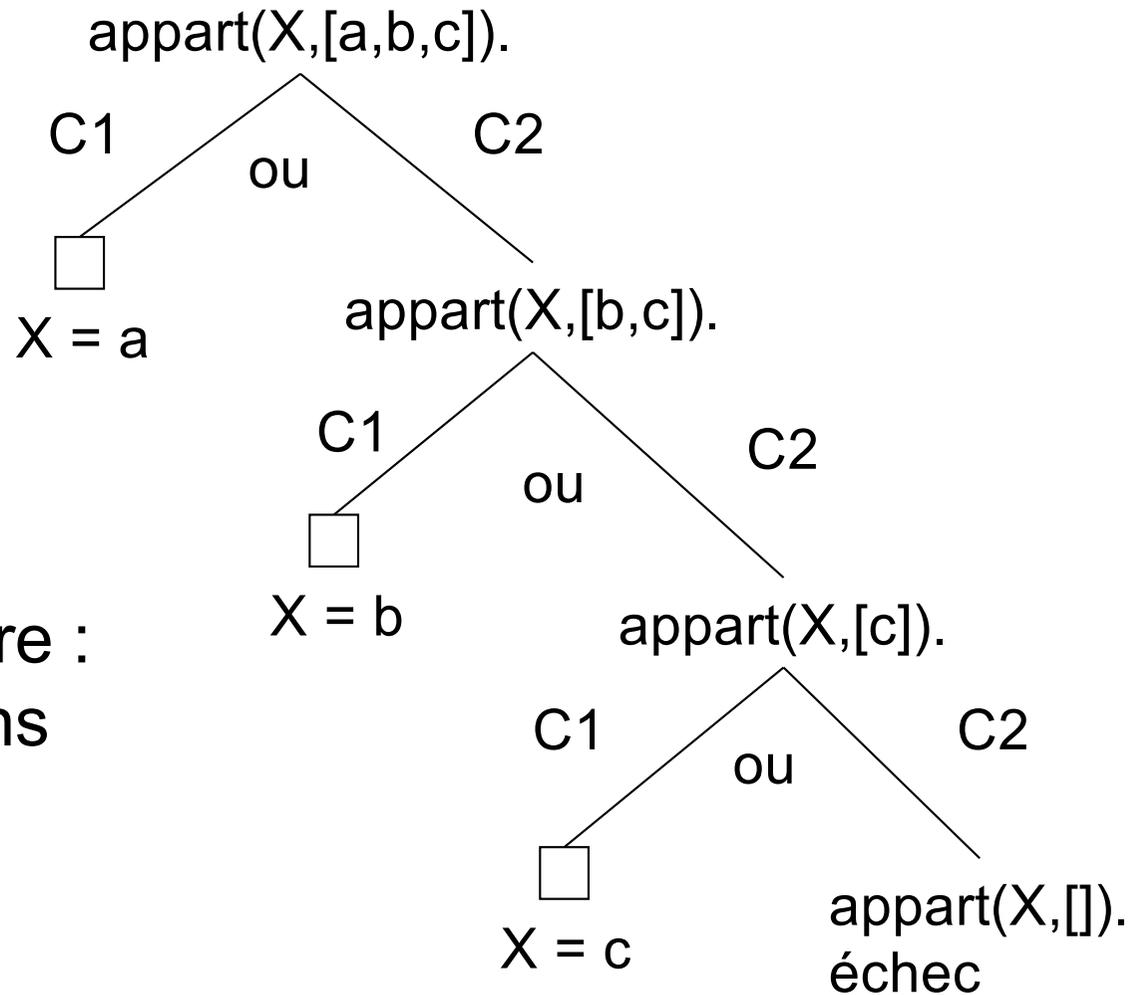
creep

Exit: (9) appart(b, [b, c]) ?

creep

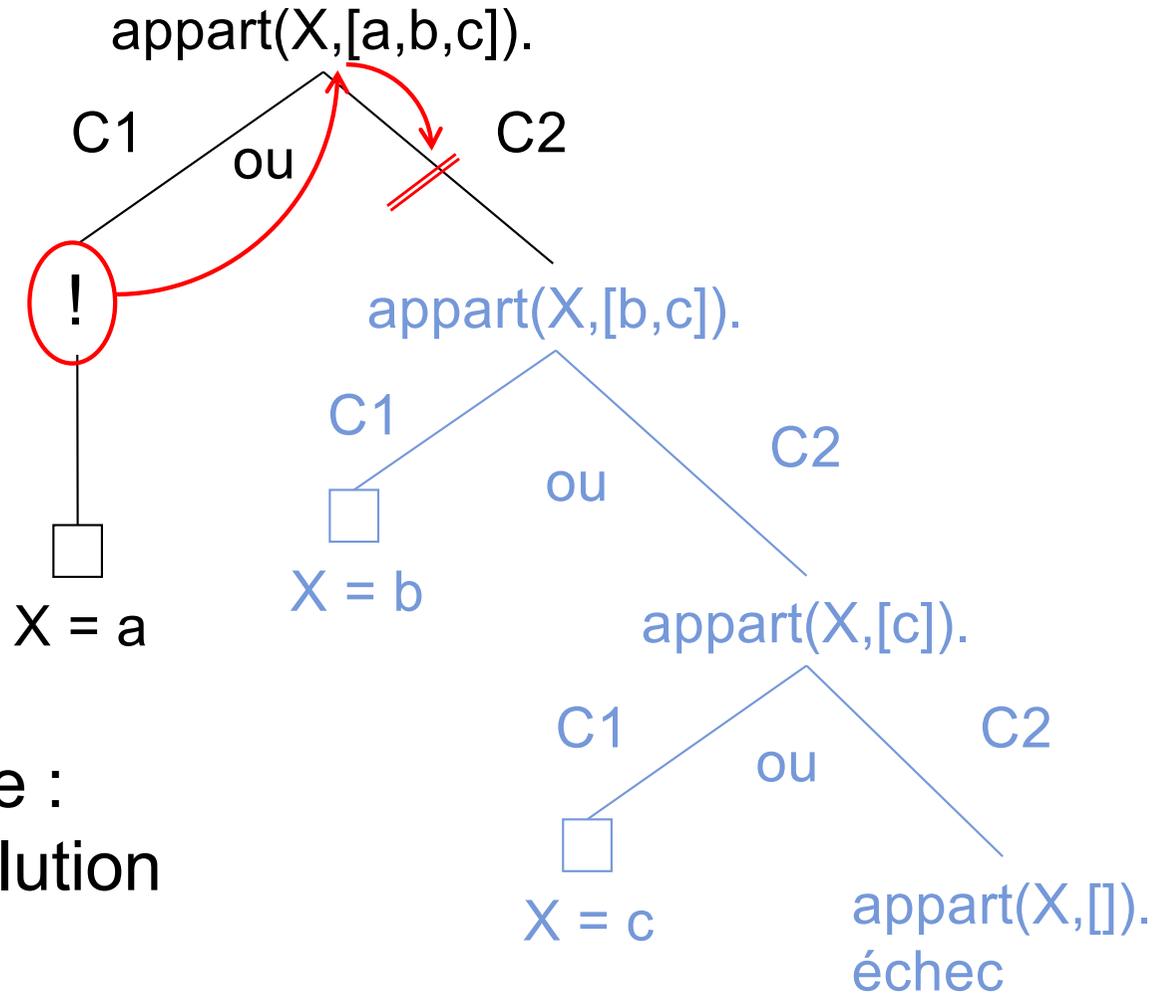
false.

COUPURE ROUGE : MODIFIER LES SOLUTIONS (1)



Sans coupure :
trois solutions

COUPURE ROUGE : MODIFIER LES SOLUTIONS (2)



Avec coupure :
une seule solution

COUPURE : QUELS POINTS DE CHOIX SUPPRIMÉS ?

a :- b, c, d, e.

a :- ...

b.

b :- ...

c.

c :- ...

d :- f, g, **!**, h, j.



(d :- ...)

(d :- ...)

f.

(f :- ...)

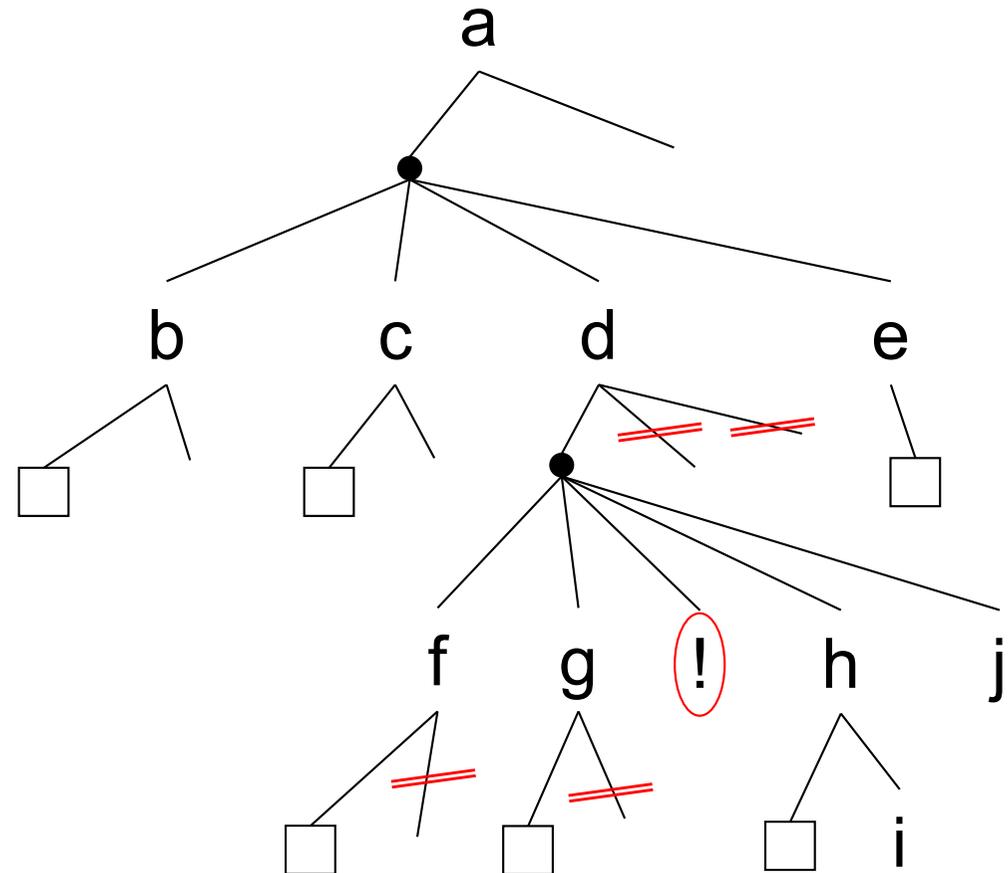
g.

(g :- ...)

h.

h :- i.

e.



La coupure supprime les points de choix sur les sommets aînés et sur le sommet père

EXERCICE

- On définit le programme suivant :
b(1). b(2). c(3).c(4). d(5). d(6).
a(X,Y,Z) :- ⁽¹⁾ b(X), ⁽²⁾ c(Y), ⁽³⁾ d(Z) ⁽⁴⁾.
- Donner toutes les réponses à la requête a(X,Y,Z) dans l'ordre où Prolog les fournit, selon qu'une coupure est placée en (1), (2), (3) ou (4).

COUPURE ROUGE : OMETTRE DES CONDITIONS

```
delete([],_,[]).
```

```
delete([X|L],X,R) :- !, delete(L,X,R).
```

```
delete([Y|L],X,[Y|R]) :-  
    Y\==X, delete(L,X,R).
```

```
delete([],_,[]).
```

```
delete([X|L],X,R) :- !, delete(L,X,R).
```

```
delete([Y|L],X,[Y|R]) :- delete(L,X,R).
```

RETOUR SUR GÉNÉRATION/TEST

- Version sans coupure pour le prédicat en génération
- Version avec coupure pour le prédicat en test (ou calcul d'une seule solution)
- Exemple : appartenance à une liste

IF-THEN-ELSE

`if_then_else(P,Q,R) :- P, !, Q.`

`if_then_else(P,Q,R) :- R.`

`(P -> Q ; R).`

NÉGATION PAR ÉCHEC

not(But) est vrai si But échoue

not(B) :- B, !, fail.

not(_).

EXERCICE

- Définir le prédicat $\text{occ}(X,L,N)$ qui calcule le nombre N d'occurrences de l'élément X dans la liste L .
- Construire l'arbre de résolution de la requête suivante : $\text{occ}(a,[z,a,r,a,t],N)$.
- Que donne cette requête si on définit le prédicat occ :
 - avec une coupure verte ?
 - une coupure rouge ?
 - et si on enlève la coupure rouge ?