Les expressions arithmétiques

- Prolog connaît les entiers et les nombres à points flottants.
- Les expressions arithmétiques sont construites à partir de nombres, de variables et d'opérateurs arithmétiques.

Les expressions arithmétiques

- Opérations habituelles : addition, soustraction, multiplication, division entière (symbole //), division flottante (symbole /).
- Selon les systèmes Prolog, différentes fonctions mathématiques comme abs(X), ln(X), sqrt(X)

Évaluation arithmétique

- Pour obtenir le résultat d'une expression mathématique, il faut en forcer l'évaluation avec l'opérateur *is*

```
?- write(1+2).
1+2

?- X is 1+2.
X= 3

?- 3 is 1+2.
yes
```

```
?- 1+2 = 2+1.
no

?- X is 1+2, Y is 2+1, X=Y.
X= Y= 3
```

Unification et expressions arithmétiques

- Expressions et unification : attention à certaines tentatives d ’unification
  - la tentative d’unification entre *3+2* et *5* échouera. En effet, l’expression *3+2* est un arbre alors que 5 est un nombre.
  - L'évaluation des expressions ne fait pas partie de l’algorithme d’unification.

Les prédicats de comparaison

- Comparaison des expressions arithmétiques
  - X =:= Y se traduit par X est égal à Y
  - X =\= Y se traduit par X est différent de Y
  - X < Y
  - X =< Y
  - X > Y
  - X >= Y
- Il y a évaluation puis comparaison.

Exemple

```
regne(philippeIV,1285,1314).
regne(philippeIII,1270,1385).

roi(Nom,Annee):- regne(Nom,A,B),
                 Annee>=A, Annee=<B.
```

Exemple

min(X,Y,X) :- X<Y.
min(X,Y,Y) :- X>=Y.

Quelles sont les requêtes valides?

Exemples

```
gcd(U,0,U).
gcd(U,V,W) :- not(V=0), R is U mod V, gcd(V,R,W).

fact(0, 1).
fact(N, F) :-
   N > 0, N1 is N - 1,
   fact(N1, F1), F is F1 * N.

pow( X, 1, X ).
pow( X, Y, Z ) :-
   Y > 1, Y1 is Y - 1,
   pow( X, Y1, Z1 ), Z is X * Z1.
```

Exemples

```
poids(albert,70).
taille(albert,1.90).
imc(Personne, Indice):- poids(Personne,P), taille(Personne,T),
                        Indice is P/(T*T).

fib(0,1).
fib(1,1).
fib(N,F):- N>1, N1 is N-1, N2 is N-2, fib(N1,F1), fib(N2,F2),
           F is F1+F2.

gcd(A,A,A).
gcd(A,B,GCD):- A<B, NB is B-A, gcd(A,NB,GCD).
gcd(A,B,GCD):- A>B, NA is A-B, gcd(NA,B,GCD).
```

Solution d'une équation du second degré

```
resolution2(A,B,C,X):- discriminant(A,B,C,Discriminant),
                                racine(A,B,C,Discriminant,X).
discriminant(A,B,C,Discriminant):-Discriminant is B*B-4*A*C.

racine(A,B,C,0,X):- X is -B/(2*A).
racine(A,B,C,Discriminant,X):- Discriminant>0,
                                X is (-B+sqrt(Discriminant))/(2*A).
racine(A,B,C,Discriminant,X):- Discriminant>0,
                                X is (-B-sqrt(Discriminant))/(2*A).
```

Exemple (récursivité croisée)

```
pair(0).
pair(N):-     N=\=0, M is N-1, impair(M).
impair(N):- N=\=0, M is N-1, pair(M).
```

La condition d'arrêt doit se trouver en première position dans une clause avec récursivité

Prédicat Intervalle

```
intervalle(K,L,H):- K>=L, K=<H.
```

Simple test, pas très utile…

Prédicat Intervalle, générateur

```
intervalle(K,K,H):- K=<H.
intervalle(K,L,H):- L<H, L1 is L+1, intervalle(K,L1,H).

?- intervalle(X,3,6).
```

Notion de coupure

- **Coupure** ou **coupe-choix**
- Introduit un contrôle sur l'exécution de ses programmes
  - en élaguant les branches de l'arbre de recherche
  - rend les programmes plus simples et efficaces
- La notation ! est utilisée.

Notion de coupure

- Le coupe-choix permet de signifier à Prolog qu'on ne désire pas conserver les points de choix en attente
- Utile lors de la présence de clauses exclusives
  - Ex : les 2 règles suivantes
    - *humain(X) :- homme(X).* s'écrit *humain(X) :- homme(X),!.*
    - *humain(X) :- femme(X).* s'écrit *humain(X) :- femme(X),!.*

Si on a déjà démontré que X est un homme, alors pas la peine de vérifier si X est une femme

Notion de coupure

- Le coupe-choix permet :
  - d'éliminer des points de choix
  - d'éliminer des tests conditionnels que l'on sait inutile

    => plus d'efficacité lors de l'exécution du programme
- Quand Prolog démontre un coupe-choix
  - tous les choix effectués sont figés et lors du retour-arrière, on repart de la clause parente de la coupe.
- Donc:
  - Quand une coupe est démontrée, celle-ci est automatiquement vraie
  - Si il faut retourner en arrière, tous les choix effectués lorsque la coupe a été introduite dans la liste des buts sont figés (incluant la branche où la coupe a été introduite).

Le coupe-choix

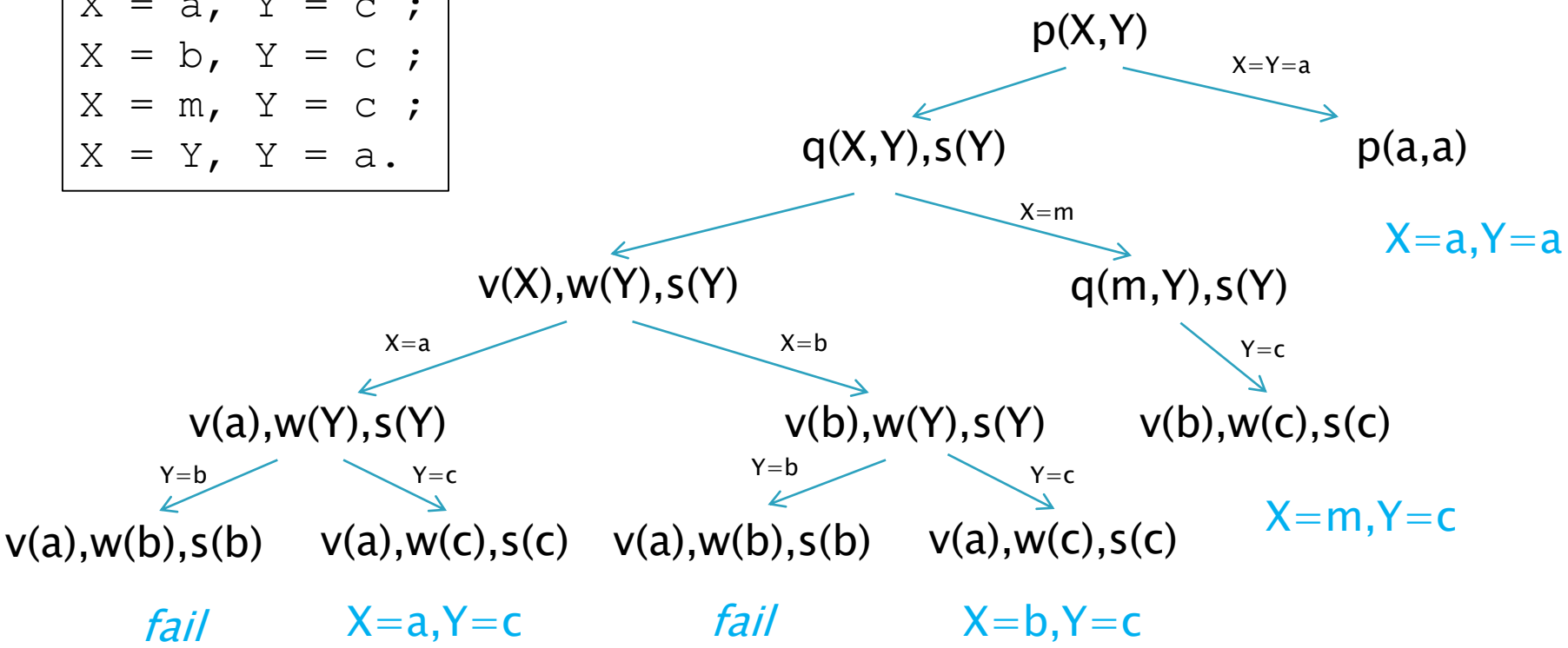
- Le coupe-choix ! permet d'éliminer tous les choix non encore explorés.
- Le coupe-choix devient actif lorsqu'il est démontré
- Son effet remonte jusqu'au nœud qui l'a introduit
- Les choix non-explorés avant l'introduction de la coupe et après sa démonstration demeurent disponibles

Exemple 1(sans coupe)

```
p(X,Y):-q(X,Y),s(Y).
p(a,a).
q(X,Y):-v(X),w(Y).
q(m,Y).
v(a).
v(b).
w(b).
w(c).
s(c).
```

Arbre de résolution

```
?- p(X,Y).
X = a, Y = c ;
X = b, Y = c ;
X = m, Y = c ;
X = Y, Y = a.
```

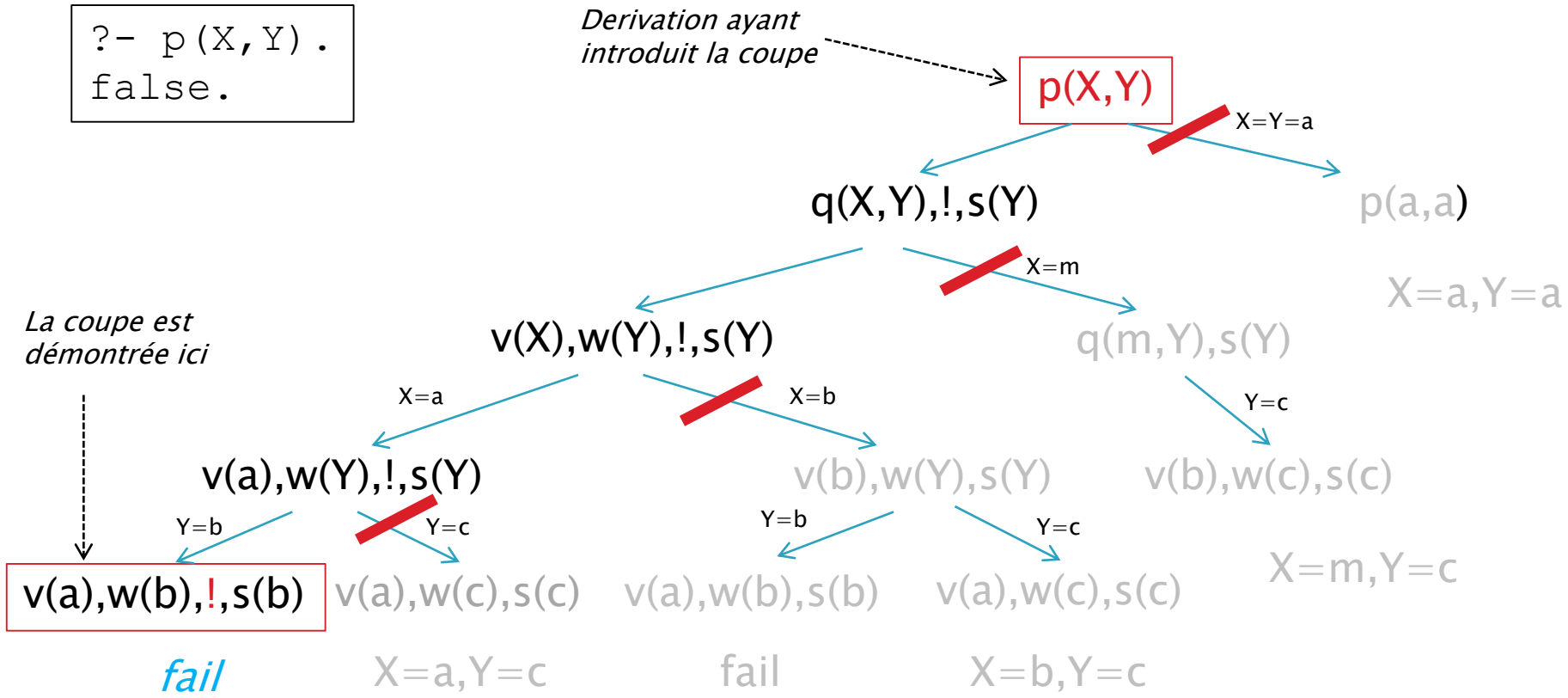


Exemple 2

```
p(X,Y):-q(X,Y),!,s(Y).
p(a,a).
q(X,Y):-v(X),w(Y).
q(m,Y).
v(a).
v(b).
w(b).
w(c).
s(c).
```

Arbre de résolution

```
?- p(X,Y).
false.
```



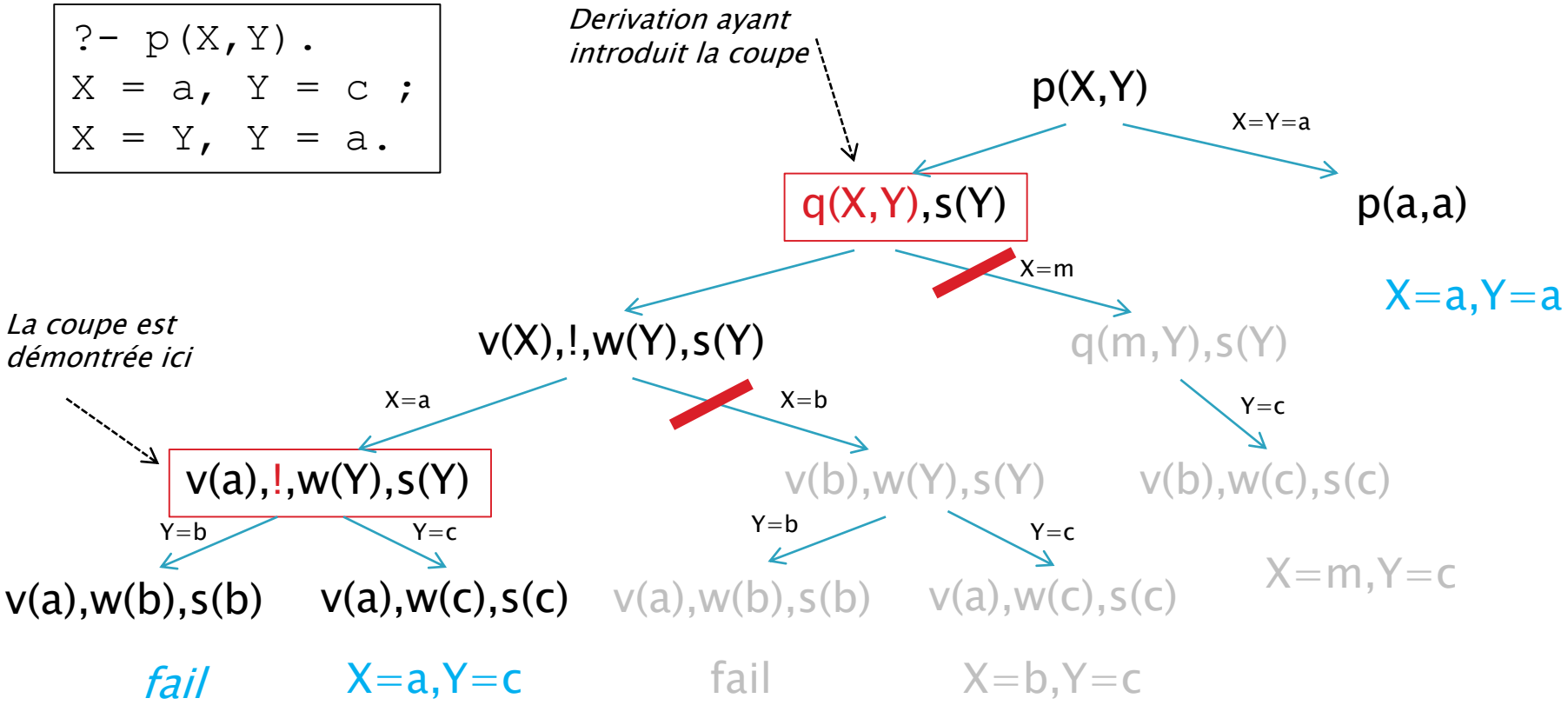
Exemple 3

```
p(X,Y):-q(X,Y),s(Y).
p(a,a).
q(X,Y):-v(X),!,w(Y).
q(m,Y).
v(a).
v(b).
w(b).
w(c).
s(c).
```

```
?- p(X,Y).
X = a, Y = c ;
X = b, Y = c ;
X = m, Y = c ;
X = Y, Y = a.
```

Arbre de résolution

```
?- p(X,Y).
X = a, Y = c ;
X = Y, Y = a.
```



Exemple de coupe

- Pour calculer la racine carrée entière d'un nombre, on utilise un générateur de nombres entiers *entier(k)*. L'utilisation du coupe-choix est alors indispensable après le test K*K>N car la génération des entiers n'a pas de fin.

```
entier(0).
entier(N) :- entier(N1), N is N1+1.
racine(N, R) :- entier(K), K*K >N, ! , R is K-1.
```

Coupe rouge – Coupe verte

- Une coupe rouge change l'espace de solution
- Une coupe verte empêche simplement de tester des clauses qui de toute façon aurait été rejetées

Arbre sans coupure

```
x :- h.
h :- d.
h :- e.
d :- a,b.
d :- c.
a.
b.
c.
e.
```

Avec:
?- x.
Les faits visités seront:
a b c e

Arbre avec coupure (élagage)

```
x :- h.
h :- d.
h :- e.
d :- a,!,b.
d :- c.
a.
b.
c.
e.
```

Avec:
?- x.
Les faits visités seront:
a b e

Lorsque la coupe est rencontrée, on ne peut plus prouver ‘d’ autrement.
(*si a alors b sinon c*)

Autre exemple de coupure

```
p(X,Y) :- q(X), r(X,Y).
p(c,c1).

q(a).
q(b).
r(a,a1).
r(a,a2).
r(b,b1).
r(b,b2).
r(b,b3).
```

Autre exemple de coupure

```
p(X,Y) :- q(X), r(X,Y), !.
p(c,c1).
```

1 solution

```
p(X,Y) :- q(X), !, r(X,Y).
p(c,c1).
```

2 solutions

```
p(X,Y) :- !, q(X), r(X,Y).
p(c,c1).
```

5 solutions

Notion de coupure

- En résumé, les utilités du coupe-choix sont :
  * éliminer les points de choix menant à des échecs certains
  * supprimer certains tests d'exclusion mutuelle dans les clauses
  * permettre de n'obtenir que la première solution de la démonstration
  * assurer la terminaison de certains programmes
  * contrôler et diriger la démonstration

Variations sur le coupe-choix

Les faits:

conge(vendredi).
meteo(vendredi,beau).
meteo(samedi,beau).
meteo(dimanche,beau).
weekend(samedi).
weekend(dimanche).

La requête:

?- picnic(J).

Les résultats:

J=samedi;
J=dimanche;
J=vendredi.

Les règles:

picnic(J) :- meteo(J,beau), weekend(J).
picnic(J) :- conge(J).

Variations sur le coupe-choix

Les faits:

conge(vendredi).
meteo(vendredi,beau).
meteo(samedi,beau).
meteo(dimanche,beau).
weekend(samedi).
weekend(dimanche).

La requête:

?- picnic(J).

Les résultats:

No.

Les règles:

picnic(J) :- meteo(J,beau), !, weekend(J).
picnic(J) :- conge(J).

Variations sur le coupe-choix

Les faits:

```
conge(vendredi).
meteo(vendredi,beau).
meteo(samedi,beau).
meteo(dimanche,beau).
weekend(samedi).
weekend(dimanche).
```

La requête:

```
?- picnic(J).
```

Les résultats:

```
J=samedi.
```

Les règles:

```
picnic(J) :- meteo(J,beau), weekend(J), !.
picnic(J) :- conge(J).
```

Variations sur le coupe-choix

Les faits:

conge(vendredi).
meteo(vendredi,beau).
meteo(samedi,beau).
meteo(dimanche,beau).
weekend(samedi).
weekend(dimanche).

La requête:

?- picnic(J).

Les résultats:

J=samedi;
J=dimanche.

Les règles:

picnic(J) :- !, meteo(J,beau), weekend(J).
picnic(J) :- conge(J).

If-then-else avec coupe-choix

```
max(X,Y,Y) :- X <= Y.
max(X,Y,X) :- X > Y.
```

```
max(X,Y,Y) :- X <= Y, !.
max(X,Y,X) :- X > Y.
```

```
max(X,Y,Y) :- X <= Y, !.
max(X,Y,X).
```

```
max(X,Y,Z) :- X <= Y, !, Y=Z.
max(X,Y,X).
```

fail

- Le prédicat *fail* est un prédicat qui n'est jamais démontrable, il provoque donc un échec de la démonstration où il figure.

La négation: not ou \+

- Négation par l'échec
  - Si *F* est une formule, sa négation est notée *not(F)* ou *not F*. L'opérateur *not* est préfixé associatif.
  - Prolog pratique la **négation par l'échec**, c'est-à-dire que pour démontrer *not(F)* Prolog va tenter de démontrer *F*. Si la démonstration de *F* échoue, Prolog considère que *not(F)* est démontrée.
  - Pour Prolog, *not(F)* signifie que la formule F n'est pas démontrable, et non que c'est une formule fausse. C'est ce que l'on appelle **l'hypothèse du monde clos**.

Exemples simples avec *not*

```
?- not(fail).
yes
?- not(X=1).
no
?- X=0, not(X=1).
X= 0
?- not(X=1), X=0.
no
```

Démontrer la négation

- Elle est utilisée pour vérifier qu'une formule n'est pas vraie.
- La négation par l'échec ne doit être utilisée que sur des prédicats dont les arguments sont déterminés et à des fins de vérification.
  - Son utilisation ne devrait jamais déterminer la valeur d'une variable

Négation et variables

étudiantCelibataire(X):-
    not(marrié(X)), étudiant (X).

étudiant(joe).
marrié(jean).

?- étudiantCelibataire(joe).
yes
?- étudiantCelibataire(jean).
no
?- étudiantCelibataire(X).
no

Négation et coupe-choix

not P :- P, !, fail.

not P.

- Pour démontrer *not P*, Prolog essaie de démontrer *P*
  - S'il réussit, un coupe-choix élimine les points de choix éventuellement créés durant cette démonstration puis échoue.
  - Si la démonstration de *P* échoue, Prolog utilise la deuxième règle qui réussit.

Le cut-fail

```
peutEntrerDansLaPolice(P) :-
        aUnDossierCriminel(P), !, fail.
```

Utilisé lorsque la démonstration d'une condition invalide toutes les autres

La différence

- La **différence** est définie comme le contraire de l'unification
  - Elle est notée : X \= Y. Elle signifie que X et Y ne sont pas unifiables, et non qu'ils sont différents.
    - Ex : Z \= 3. sera faux car Z et 3 sont unifiables.
  - Elle peut être définie comme:

    X \= Y :- not X = Y.

    ou

    X \= X :- !, fail.

    X \= Y.

Opérateurs =

- Comparaison arithmétique
  - =:= versus =\=
  - Les variables doivent être instanciées
- Comparaison de variables
  - == versus \==
  - Vrai seulement si les deux opérandes ont été unifiées
- Unification
  - = versus \=

Prédicat Intervalle revisité

```
intervalle(K,L,H):- number(K), number(L), number(H),
                    !, K>=L, K=<H.
intervalle(K,K,H):- number(K),number(H), K=<H.
intervalle(K,L,H):- number(L), number(H), L<H, L1 is L+1,
                    intervalle(K,L1,H).
```