Centre Universitaire Adelhafid Boussouf Mila

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

Intelligence Artificielle

Chapitre 3
Le fonctionnement des systèmes experts

Formalisme de représentation

Pour la représentation des connaissances on utilise des règles :

Une structure IF –THEN qui relie de l'information ou des faits de la partie IF avec l'action dans la partie THEN.

- Ex: IF «lumière est verte» THEN « action est go»
- Ex: IF «lumière est rouge» THEN « action est stop»
- IF «condition est vraie» THEN «faire quelque action».
- Un langage de programmation est une façon de représentation de données (voir cours tp prolog)

Les types de connaissances

Connaissances procédurales:

- «comment».
- Connaissances sur comment accomplir une tâche

Connaissances déclaratives:

- «qu'est-ce que c'est»
- La capacité de formuler ou décrire quelque chose

Les règles de production

Une règle est constituée de deux parties:

1. Antécédent: la partie IF
2. Conséquence: la partie THEN

IF «antécédent» THEN «conséquence»

Une règle est déclenchée lorsque l'antécédent est vrai et que la Partie conséquence est exécutée

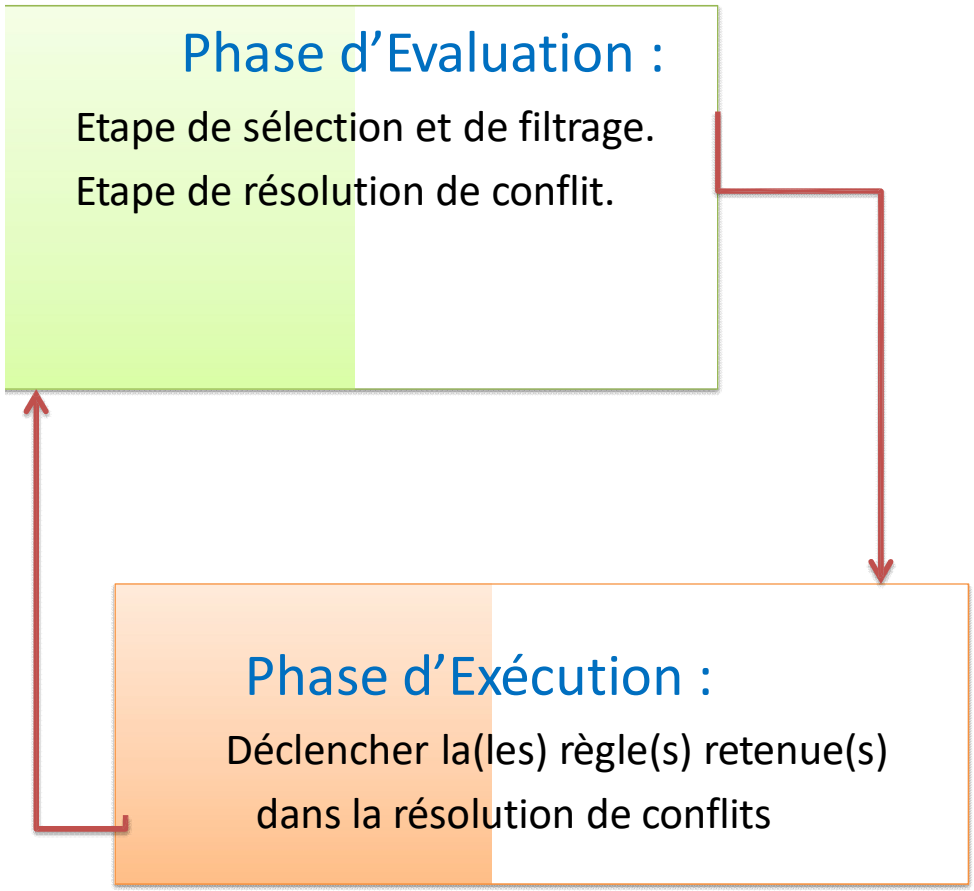
Principe De Fonctionnement (MI)

Un moteur d'inférence est un mécanisme qui permet d'inférer des connaissances nouvelles à partir de la base de connaissances du système.

Le Moteur d'Inférence contrôle le raisonnement, en enchaînant des cycles comportant chacun deux phases :

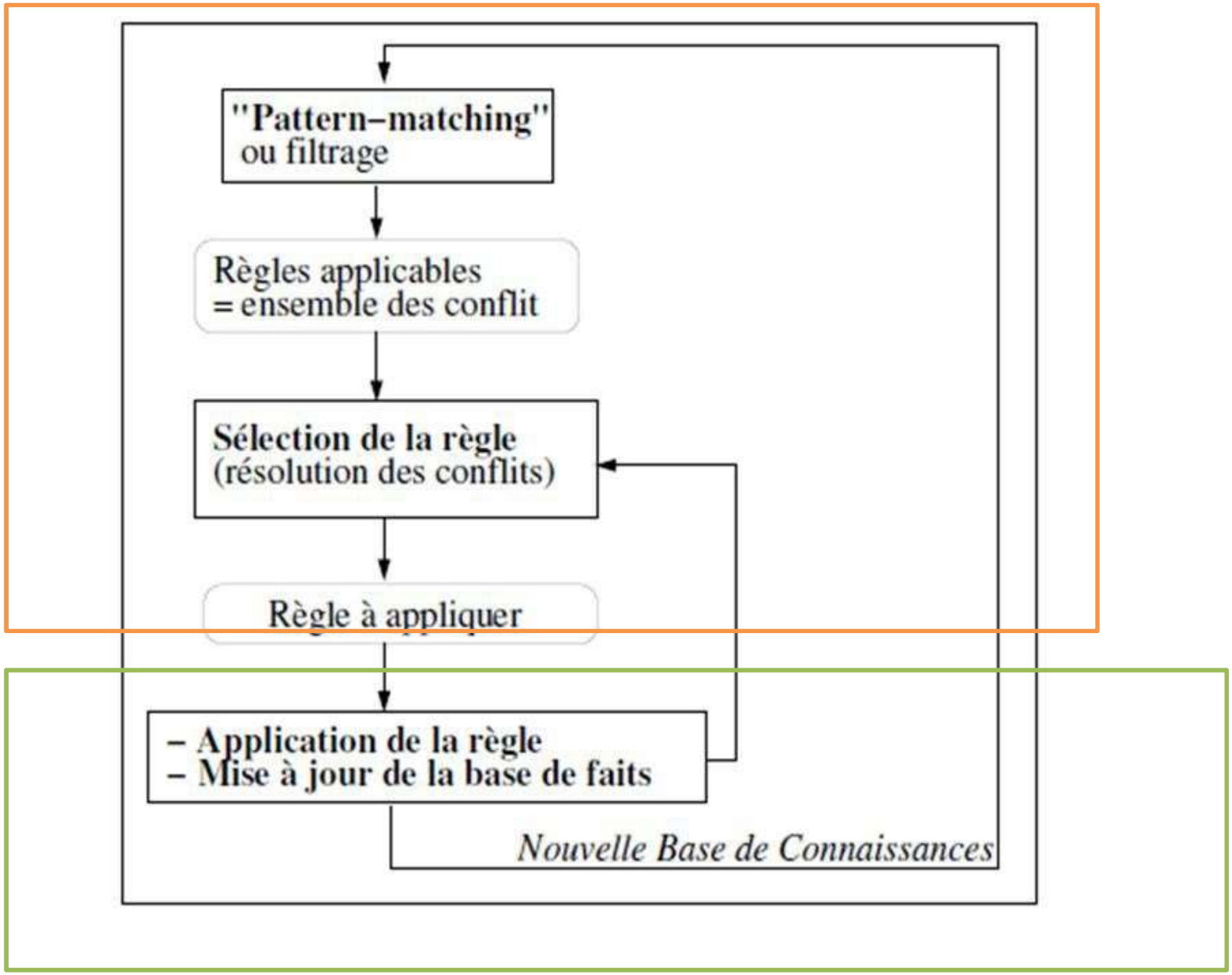
- **Phase d'évaluation :** constitue l'ensemble des règles déclenchables (conflict set)

- **Phase d'exécution :** déclenche la(les) règle(s) retenue(s) dans la résolution de conflits



Cycle de base
d'un moteur d'inférence

Principe De Fonctionnement (Mi)



Phase d ’évaluation

Phase de résolution de conflits : Le Moteur d’Inférences choisit les Règles qui doivent être effectivement déclenchées selon une stratégie : **heuristiques**

Pour résoudre le conflit, entre plusieurs règles à priori applicables suivant une configuration de la BF, le système doit choisir une ou plusieurs Règles suivant certaines heuristiques :

- La première règle qui s’applique au contrôle
- La règle la plus prioritaire suivant un ou plusieurs critères définis par l’expert
- La règle la plus spécifique (la condition la plus détaillée qui s’applique à la BF)
- La règle se référant à un élément le plus récemment ajouté
- Sélection d’un sous-ensemble de Règles résultant de l’application de métarègles
- Déclenchement prioritaire des règles amenant le plus grand nombre de conclusions
- Ne pas choisir, explorer toutes les règles applicables
- Arbitrairement

Phase d ’exécution

- Le Moteur d’Inférence va exécuter les règles obtenues à la fin de la phase d’évaluation, modifiant ainsi la base de connaissance.
- On inclut dans la base des faits, les faits de la partie conclusion de la règle déclenchée.
- Le MI commande la mise en œuvre des actions définies par les Règles prêtes à l’exécution
- Il s’ensuit la mise à jour de la base de faits avec détection d’incohérence
- En logique des prédicats, une règle peut être déclenchée plusieurs fois
- En logique des propositions, une règle n’est déclenchée qu’une seule fois

Régimes de contrôle du MI

- **Régime irrévocable :** Pour des MI très simples, le MI s'arrêtera dès qu'il atteint la Saturation de la Base de Règles Déclenchables,

- **Régime par tentative (Backtracking):** Le MI examine la possibilité de déclencher d'autres règles déclenchables. Le MI opère par un retour arrière et remet en cause les règles déclenchées précédemment,

- **Monotonie:** Le MI ne fait qu'ajouter des faits à la BF,

- **Non Monotonie:** Le MI peut supprimer des faits qui peuvent se révéler contradictoires (Robotique, diagnostic, etc.) .

Raisonnement du MI

- Un moteur d'inférence est un algorithme d'exploration d'un espace d‘états utilisé par un système expert.
  - Un état (ou nœud) de l'espace de recherche est appelé mémoire de travail
- La recherche heuristique est une composante essentielle des systèmes experts.
- Le moteur d'inférence peut être à *chaînage avant* , *chaînage arrière ou chainage mixte.*
- Vu que les règles de la base de connaissances peuvent contenir des variables, le moteur d’inférence utilise un algorithme d’unification.

Chaînage avant

Le mécanisme du chaînage avant est très simple : pour déduire un fait particulier, on déclenche une règle dont les prémisses sont connues dans la BF. Sa conclusion est ajoutée à BF comme fait connu. La règle est ensuite désactivée (une règle n'a besoin d'être déclenchée qu'une fois au plus, puisque ses prémisses restent dans BF). **L'algorithme tourne ainsi jusqu'à ce que le *fait à déduire* soit connu ou qu'aucune règle ne soit plus déclenchable**. Il s'agit d'un algorithme d'essais successifs, programmé en version itérative.

Plus précisément, soit BF une base de faits, BR une base de règles (on suppose pour le moment qu'elle ne comporte que des faits booléens positifs) et F le fait que l'on cherche à établir ; l'algorithme suivant calcule si F peut être déduit ou non de la base de connaissances.

Remarque :

Dans l'algorithme du chainage avant on n'indique pas comment choisir une règle applicable. C'est à ce niveau que la méta-connaissance du domaine intervient et permet de définir une stratégie de choix.

Algorithme du chainage avant

Un état est un ensemble de faits.

État initial : ensemble de faits initiaux;

État final: État contenant un prédicat unifiable avec le but.

Fonction successeur:

étant donné un état S, ses successeurs sont : Trouver une règle R telle que sa précondition est unifiable avec *S*.

Entrée: BF, BR, F: Fait à établir

Sortie : Mise à jour de la BF.

Tant que F n'est pas dans BF et qu'il existe dans BR une règle applicable **Faire**

- Choisir une règle applicable R par les étapes de filtrage et de résolution de conflits, grâce à l'utilisation de métarègles
- BR ← BR - R (désactivation de R)
- BF ← BF U {conclusion de R}

fintantque

Si F appartient à BF A**lors**

F est établi

sinon

F n'est pas établi

finSi

Exemple1 : chainage avant

Soit BF = {B, C} et soit H le fait à établir, soit BR composée des règles :

1. Si B et D et E Alors F
2. Si G et D Alors A
3. Si C et F Alors A
4. Si B Alors X
5. Si D Alors E
6. Si X et A Alors H
7. Si C Alors D
8. Si X et C Alors A
9. Si X et B Alors D

L'algorithme appliqué à ces paramètres prouve que H se déduit de la base de connaissances. La métarègle utilisée : « choisir la première règle »

Etape 1 Règles applicables : 4 et 7. On choisit 4.

BF = {B, C, X}

La règle 4 est désactivée et la conclusion de R4 est ajoutée à la BF.

Etape 2 Règles applicables : 7, 8 et 9. On choisit 7.

BF = {B, C, X, D, E}

La règle 7 est désactivée et la conclusion de R7 est ajoutée à la BF.

Etape 3 Règles applicables : 5, 8 et 9. On choisit 5.

BF = {B, C, X, D, E}

La règle 5 est désactivée et la conclusion de R5 est ajoutée à la BF.

Etape 4 Règles applicables : 1, 8 et 9. On choisit 1.

BF = {B, C, X, D, E, F}

La règle 1 est désactivée et la conclusion de R1 est ajoutée à la BF.

Etape 5 Règles applicables : 3, 8 et 9. On choisit 3.

BF = {B, C, X, D, E, F, A}

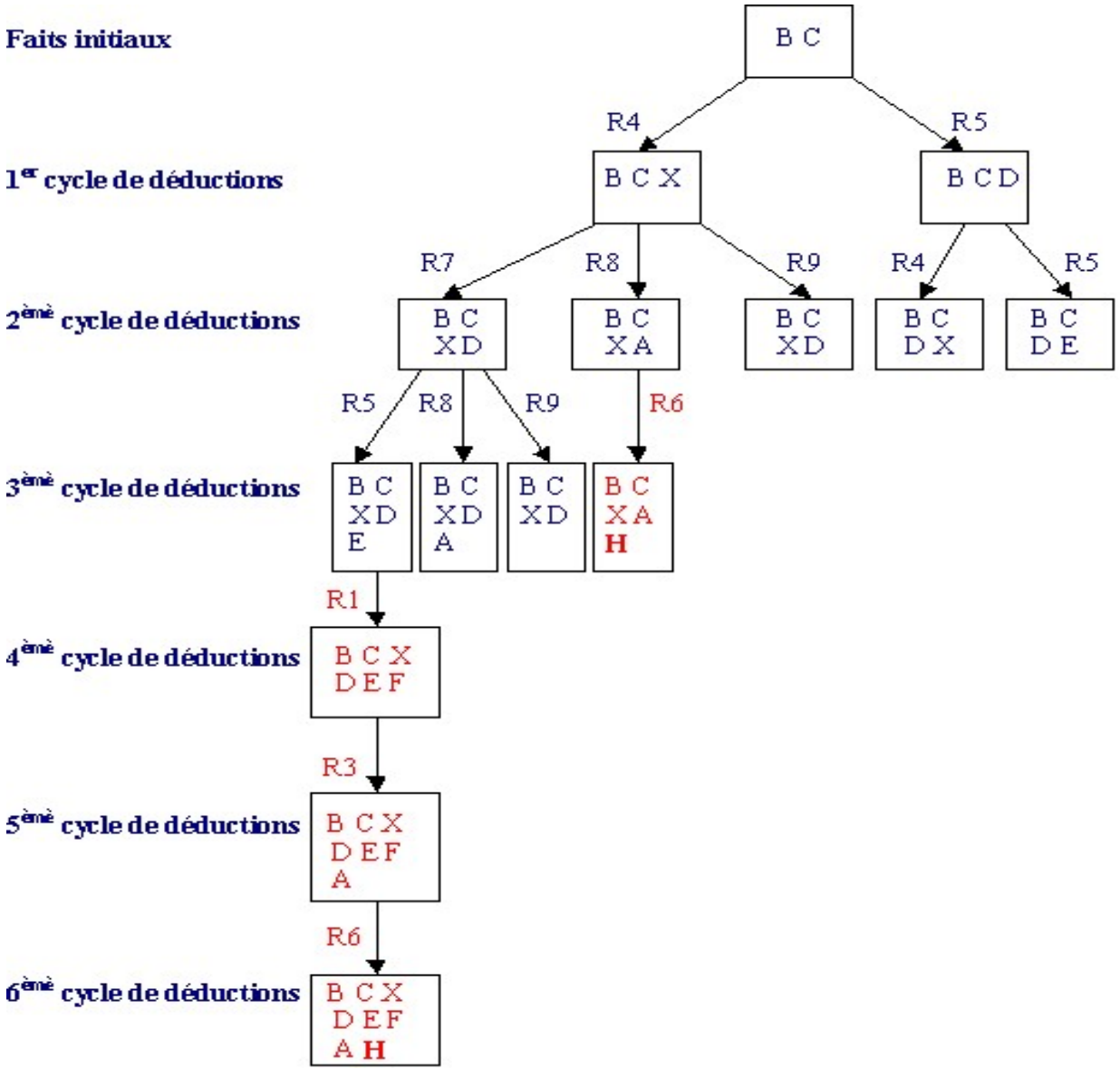
La règle 3 est désactivée et la conclusion de R3 est ajoutée à la BF.

Etape 6 Règles applicables : 6, 8 et 9. On choisit 6.

H est établi.

Exemple1 : chainage avant sou s for me d'ar bre

Cet arbre montre toutes les solutions possibles en mode chainage avant en appliquant soit la stratégie recherche en largeur (solution trouvée au 3ième cycle de déduction : soit recherche en profondeur (solution trouvée au 6ième cycle de déduction).



Stratégies de recherche

Au cours des cycles de recherche d'un MI, on développe un arbre de recherche dans lequel chaque niveau correspond à l'ensemble des règles applicables (ensemble de conflits) Chaque règle déclenchée crée une nouvelle situation et de nouvelles règles à invoquer. Deux principales stratégies de recherche se présentent:

• soit on développe toutes les règles d'un même niveau l'un après l'autre avant de passer au niveau suivant (stratégie en largeur d'abord)

• soit d'un niveau à un autre à chaque fois qu'on déclenche une règle et on ne revient aux règles restantes que si on épuise toutes les règles en profondeur (stratégie en profondeur d'abord). Le retour arrière dans le cas où la recherche en profondeur échoue sera appelé: backtracking

Régime de contrôle

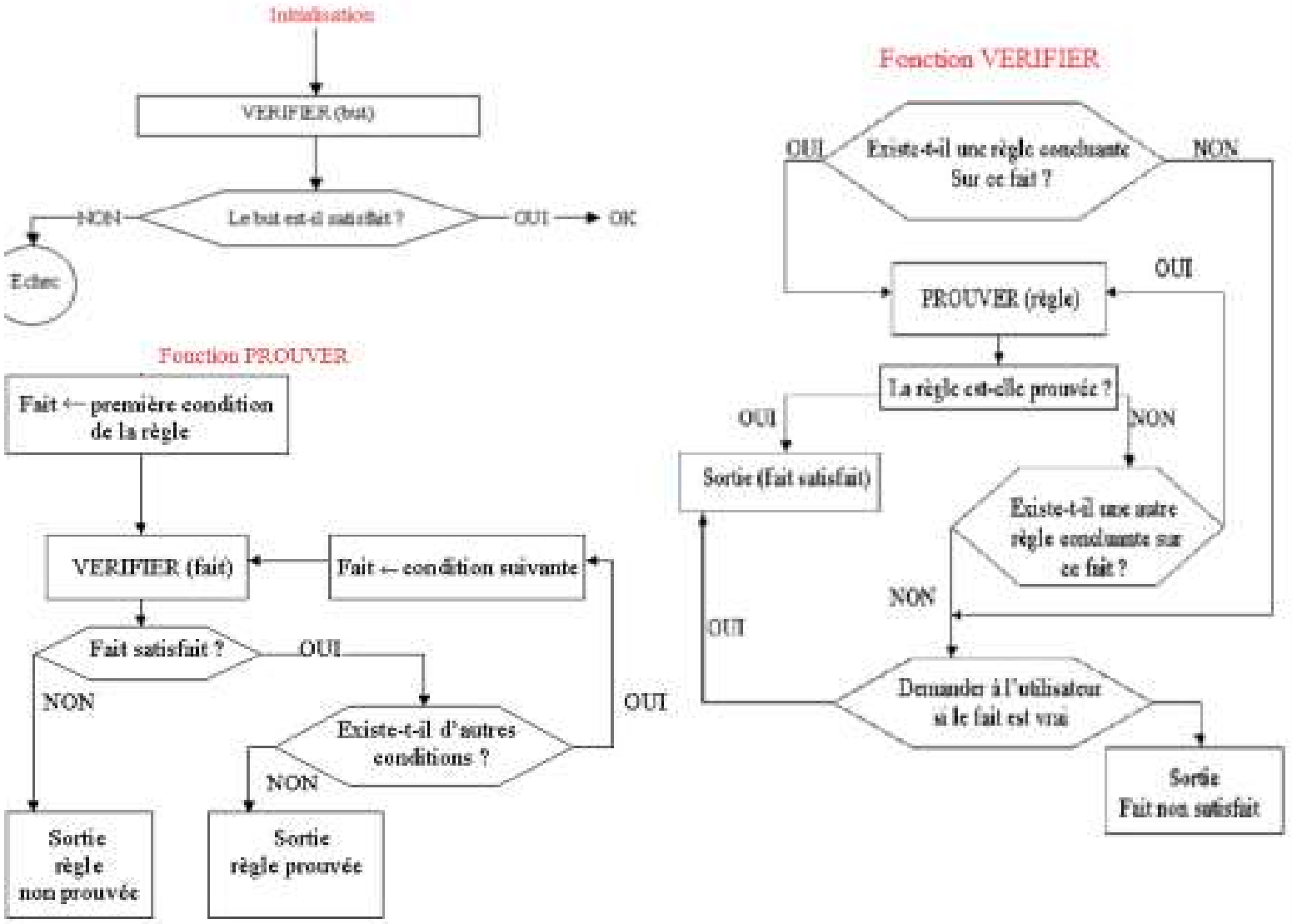
Le régime de contrôle d'un Moteur d'inférence peut être:

- Irrévocable : L'application d'une règle dans un cycle du MI n'est jamais remise en cause et on n'opère pas de backtracking. S'il n'y a plus de règles à appliquer. Le MI s'arrête et signale un échec sans faire retour en arrière
- Par tentatives : Ce régime peut remettre en cause des règles déjà appliquées si elles n'ont pas abouti, et faire un backtracking en retirant aussi les faits qui en étaient déduits

Algorithme chainage arrière

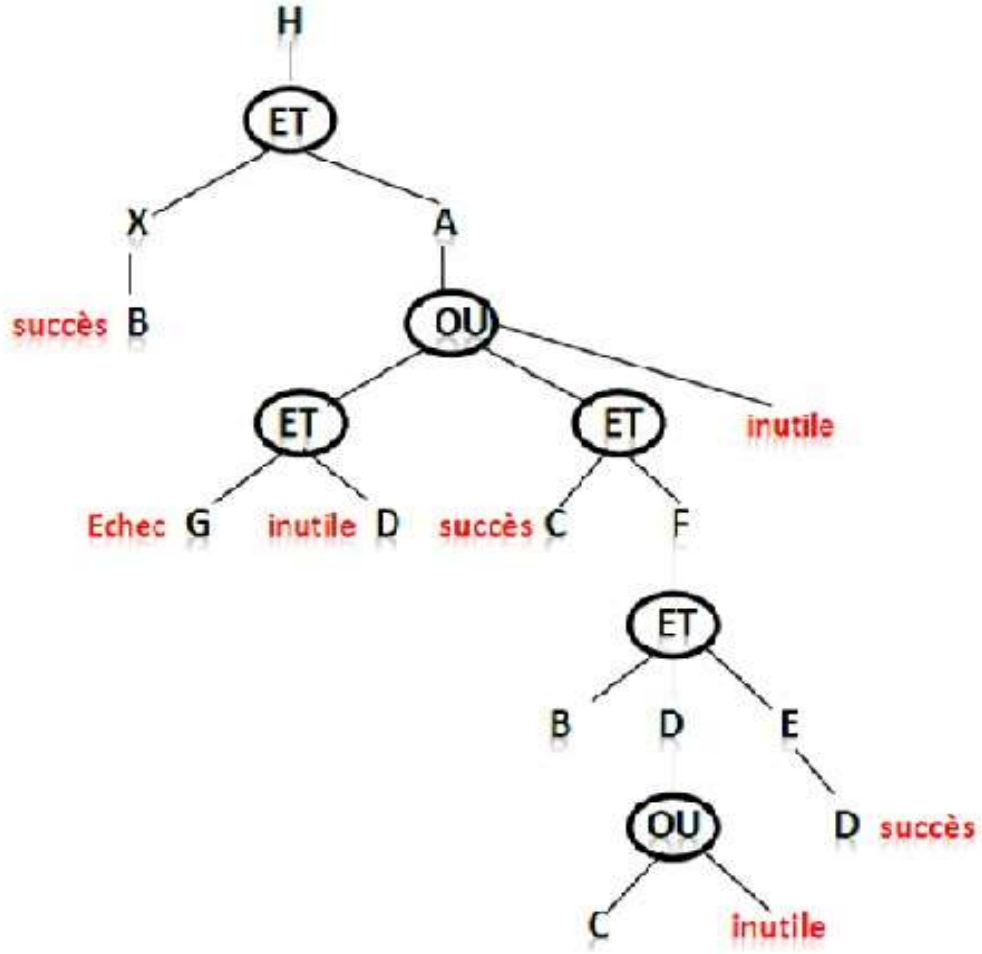
```
Fonction chainageArriere ( BR, BF,  listeButs).
if  estVide (listeButs)  then
|       res ← SUCCES
else
|    if  demBut ( premier( listeButs )) then
|    |       res  ← chaînageArriere (suite( listeButs ))
|    else
|    |       res ← ECHEC
|    end if
endif
retourner    res;

Fonction demBut( BR, BF, but).
if   but  in  BF   then
|      res  ← SUCCES
else
|     regles ← BR;   res ← ECHEC
|     while   regles non vide   et   res \= SUCCES    do
|     |     r ← choix(regles);    regles ← regles - {r}
|     |     if    conclusion(r) = but    then
|     |     |       res ← chaînageArrière (BR, BF, premisse(r))
|     |     end if
|     end
|    retourner res
|
end if
```

Organigramme chainage arrière



Exemple : Arbre Et/OU

Appliquons le chainage arrière sur l'exemple vu précédemment,



Exemple

Hypothèses : {A, D, E, G}

Chainage avant

But: F?

Chainage arrière

Hypothèses : {A, D, E, G}

Arbre Et/OU de recherche

Base de Connaissances

Base de Faits :

BF = {A, D, E, G**}**

Base de Règles : BR= {

R1: A, B, C-> H

R2: A, U, C ->F

R3: E, G, B -> S

R4: D, G -> C

R5: A, E -> B

R6: U, S, T -> F

R7: G, H -> R

R8: D, E -> T

R9: R, S, H -> F

R10: A, U-> B

}

LE MECANISME DE RESOLUTION EN PROLOG

UN LANGAGE DE HAUT NIVEAU

- **Programmation Procédurale**
  Instruction = *Ordre*
  → *Ordre pour la machine*
  → *Ordre dans l'énoncé*

Spécification d'une solution en terme de *comportement de la machine*

- **Programmation Fonctionnelle**
  Instruction = *Fonction*

Spécification d'une solution en terme de *valeurs calculées*

- **Programmation en Logique**
  Instruction = *Relation*

Spécification d'une solution en terme de *relations* entre entités (ou classes d'entités)

Programme en logique ≅ spécification exécutable

UN LANGAGE DÉCLARATIF

- **Programmer en logique = Décrire l'univers** du problème

- **Programme Prolog = Ensemble de propriétés et relations entre les objets de l'univers**

Un programme Prolog *ne décrit pas une solution* : c'est une suite *d'affirmations*

- **Problème = Ensemble de questions concernant certains objets**

- **Exécution = Déduction de nouvelles relations** à partir des affirmations du programme

CONSTITUANTS ÉLÉMENTAIRES ET TERMES

Constituants élémentaires :

- **Variable :** ***objet inconnu de l'univers du problème***
  - **chaîne commençant par une majuscule ou par _ (variable anonyme : _)**
  - **exemples : *X*, *Y1*, *_ObjetInconnu*, ...**
- **Constante :** *objet particulier de l'univers du problème*
  - **nombre : *12*, *1.056*, ...**
  - **chaine commençant par une minuscule : *toto*, *a*, *jean_paul_II*, ...**
  - **chaine entre "" : *"Constante chaine"*, *"123"*,**

Un terme est soit une constante, soit une variable, soit un terme fonctionnel

Un *terme fonctionnel* est de la forme *f(t1,…,tn)* avec:

- ***f* un symbole fonctionnel,**
- ***t1,…,tn* : suite de termes**

Exemples : *succ(zero)*, *f(X,12)*, *adresse(2,"rue des mimosas", valbonne)*, ...
les constantes sont des fonctions d'arité nulle)

LES
ATOMES

- **atome logique :** *propriété, relation entre termes*

  Syntaxe : **symbole_de_prédicat(term$_{e_1}$,…,term$_{e_n}$)**
  n : arité du prédicat

  Exemples:
  est_pere_de(pierre,paul), temps(ensoleillé)
  est_mere_de(X,paul), atome_sans_termes

- **atome clos :** *atome sans variables*

  Exemples:
  est_pere_de(pierre,paul), temps(ensoleillé)

LES CLAUSES

- **clause :** *relation (certaine ou conditionelle)*

  T :- Q1,…,Qn.

  T : littéral *positif*, appelé *Tête de Clause*
  Q1,...,Qn : suite de littéraux *négatifs* appelée *Corps de clause.*

  Si { Q1,...,Qn } ≠ ∅ et T ≠ ∅, la clause est une
  affirmation conditionnelle *(règle)*
  Exemple *même_pere(X,Y) :-*
  pere_de(P,X), pere_de(P,Y).

  Si { Q1,...,Qn } = ∅, la clause est une **affirmation inconditionnelle** *(fait)*
  exemple : *homme(pierre).*

  Si T = ∅, la clause est une question ***(dénégation)***
  exemple : *?-homme(pierre).*

- **Sémantique informelle**

  Si tous les atomes du corps sont vrais, alors l'atome de tête est vrai

  ':-' *:* ***Implication l****ogique*
  ',' *:* ***ET logique***

PROGRAMME ET PAQUETS

- **Un programme Prolog :** suite de clauses regroupées en paquets
- **Paquet = ensemble de clauses qui ont :**
  - le même *symbole de prédicat* en tête de clause
  - la même *arité*.

Deux clauses d'un même paquet sont liées par un *ou* logique.

```
parent(X,Y)  :- est_pere_de(X,Y).
parent(X,Y)  :- est_mere_de(X,Y).
```

Remarque :
Un prédicat est défini par une conjonction de clauses.
Soit le prédicat p défini par le programme :

```
:-  a1,  a2,  a3.
:-  b1, b2.
```

LA SYNTAXE PROLOG : RECAPITULATIF

Programme = ensemble de *Paquets*

Paquet = ensemble de *Clauses* qui ont le même prédicat (i.e., même symbole de prédicat et même arité de prédicat) comme tête de clause

Clause =

Atome_logique '.'
| Atome_logique ':-' Atome_logique ',' ... ','
Atome_logique '.'

Atome_logique =

Symbole_de_prédicat
| Symbole_de_prédicat '(' Terme ',' ... ',' Terme ')'

Terme =

Constante
| Variable
| Symbole_de_fonction '(' Terme ',' ... ',' Terme ')'

Constante = Entier | Réel | "" Caractère* "" |
Minuscule (Car_alphanum | '_')*

Variable = Majuscule (Car_alphanum | '_')* | '_'

Symbole_de_prédicat =
Minuscule (Car_alphanum | '_')*

Symbole_de_fonction =
Minuscule (Car_alphanum | '_')*

VISION PROCÉDURALE DE PROLOG

Question ≈ Appel de procédure
Unification ≈ Transmission de paramètres
Paquet ≈ Procédure
Clauses d'un paquet ≈ Définition de la procédure

Exemple :

```
add(zero,X,X).
add(suc(X),Y,suc(Z))  :- add(X,Y,Z).
        ≈
procédure add(arg1, arg2, arg3) :
  if arg1 = zero then unify(arg2,arg3)
  elseif  unify(arg1,suc(X))  and   unify(arg3,suc(Z))
  then add(X, arg2, Z)
  endif
end add
```

Le mécanisme de résolution en prolog

I- L'unification

✓ Le fonctionnement de Prolog repose en partie sur le mécanisme d'unification qui permet d'apparier une question avec la tête d'une clause.

✓ L'unification est l'opération élémentaire que réalise Prolog pour rendre deux termes identiques.

✓ Trois cas sont possible pour que l'unification réussie :

1- Deux termes atomiques s'unifient s'ils sont identiques (comme les constantes).

2- Une variable s'unifie avec tout type de terme.

3- Un terme composé s'unifie avec un terme composé de même foncteur et même nombre d'arguments, à condition que ces arguments s'unifient.

✓ En voici un exemple, les deux prédicats suivants sont appariables (l'opérateur d'unification est "=") :

?- p(X, b(Z,a), X) = p(Y, Y, b(V,a)).

X = b(V, a),
Z = V,
Y = b(V, a).

✓ Les atomes logiques P(X,a,Y) et P(c,a,Z), où a et c sont des constantes, sont unifiables {X=c, Y=Z}. Par contre P(X,a,Y) et P(c,b,Z) ne sont pas unifiables car les constantes ne peuvent être remplacées.

✓ Prévoyez les réponses fournies par PROLOG pour ces requêtes :

?- k(s(g),Y)=k(X,t(f)).
Y = t(f),
X = s(g).

?- p(f(Y),W,g(Z)) = p(U,U,V).

W = f(Y),

U = f(Y),

V = g(Z).

?- p(f(Y),W,g(Z)) = p(V,U,V) .

false.

?- p(a,X,f(g(Y))) = p(Z,h(Z,W),f(W)).

X = h(a, g(Y)),

Z = a,

W = g(Y).

II- Principe de résolution

- Différentes stratégies ont été développées pour guider le processus de résolution.
-
- Le mécanisme de résolution de Prolog consiste à parcourir un arbre de gauche à droite et en profondeur d'abord. Chaque appel de prédicat constitue un nœud de cet arbre, et les branches qui en partent correspondent aux possibilités d'appariement avec des têtes de clauses.
-
- L'ensemble des solutions d'un programme prolog peut être calculé par une approche ascendante, dite en chaînage avant: on part des faits, et on applique itérativement toutes les règles pour déduire de nouveaux faits ... jusqu'à ce qu'on ait tout déduit.
-
-
- Considérons par exemple le programme Prolog suivant:

```
parent(ali,said).
parent(said,amina).
parent(amina,marwa).

homme(ali).
homme(said).

pere(X,Y) :- parent(X,Y), homme(X).

grand_pere(X,Y) :- pere(X,Z), parent(Z,Y).
```

✓ A partir de l'ensemble des faits E_0:

E_0 = { parent(ali,said), parent(said,amina), parent(amina,marwa), homme(ali), homme(said) }

✓ Et en appliquant les règles du programme, on obtient l'ensemble E_1:

E_1 = { pere(ali,said), pere(said,amina) }

✓ Et à partir de E_0, E_1 et des règles, on déduit l'ensemble E_2:

E_2 = { grand_pere(ali,amina), grand_pere(said,marwa) }

✓ À partir de E_0, E_1, E_2 et des règles, on ne peut plus rien déduire de nouveau.

✓ L'union de E_0, E_1 et E_2 constitue l'ensemble des conséquences logiques du programme (La dénotation du programme).

✓ D'une façon générale, on ne peut pas calculer l'ensemble des conséquences logiques d'un programme par l'approche ascendante: ce calcul serait trop coûteux, voir infini.

✓ En revanche, on peut démontrer qu'un but composé d'une suite d'atomes logiques est une conséquence logique du programme, en utilisant l'approche descendante, dite en chaînage arrière:

✓ Pour prouver un but composé d'une suite d'atomes logiques :

?- A1, A2, .., An.

Prolog commence par prouver le premier atome logique A1. Pour cela, il cherche une clause dans le programme dont l'atome de tête s'unifie avec cet atome logique. Par exemple la clause :

A1 :- A11,A12,…,A1n.

✓ Puis Prolog remplace l'atome logique A1 dans le but par les atomes logiques du corps de la clause, et on aura un nouveau but à prouver :

A11, A12, …, A1n, A2, …, An.

✓ Prolog recommence alors ce processus, avec toutes les clauses qui s'unifient avec les atomes logiques du but, jusqu'à ce qu'il n'y ait plus rien à prouver. A ce moment, Prolog affiche yes or no ou bien les valeurs de variables.

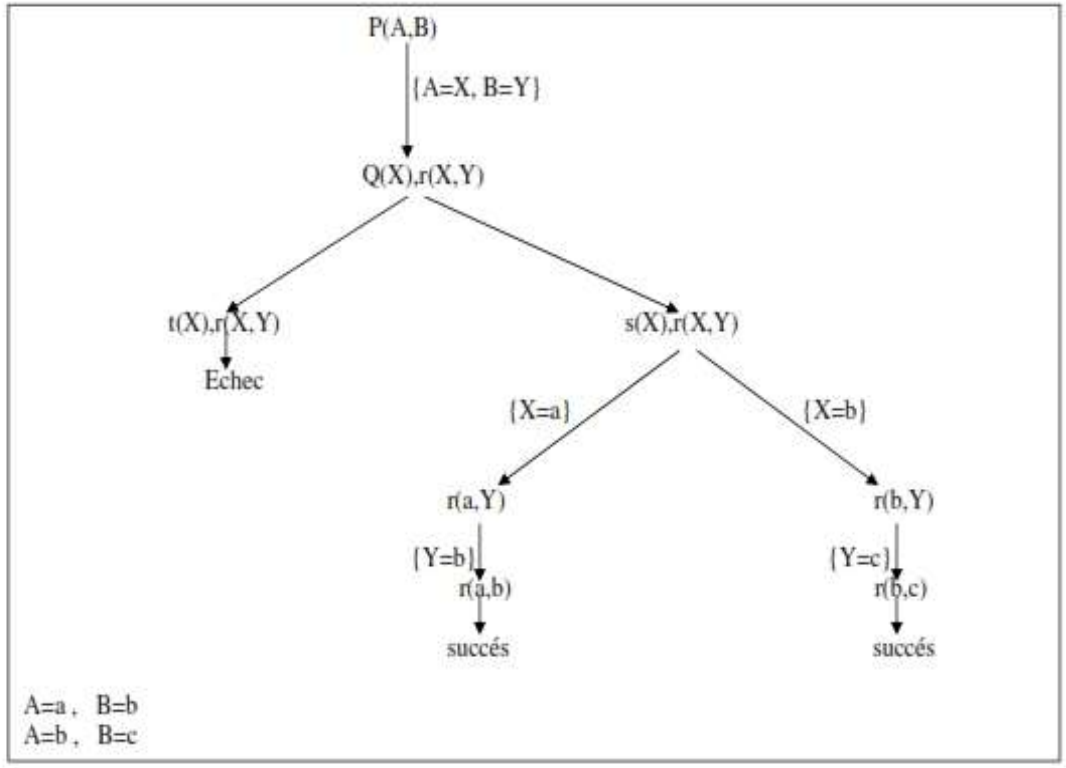
✓ Exemple : Soit le programme prolog suivant :

s(a).
s(b).
r(a,b).
r(b,c).
r(c,b).

p(X,Y) :- q(X),r(X,Y).
q(X) :- t(X).
q(X) :- s(X).

✓ Soit la requête suivante : ?- p(A,B).

✓ L'arbre de recherche du but p(A,B) est le suivant :


P(A,B)
{A=X, B=Y}
Q(X),r(X,Y)
t(X),r(X,Y)
Echec
s(X),r(X,Y)
{X=a}
{X=b}
r(a,Y)
r(b,Y)
{Y=b}
{Y=c}
r(a,b)
r(b,c)
succés
succés
A=a , B=b
A=b , B=c

Retrouvez La dénotation de ce programme (l'ensemble des solutions) en utilisant l'approche ascendante.

E_0= { s(a), s(b), r(a,b), r(b,c), r(c,b) }
E_1= { p(a,b), p(b,c)}
E_2= {q(a),q(b)}
E={E_0+E_1+E_2}

✓Considérons le programme prolog suivant :

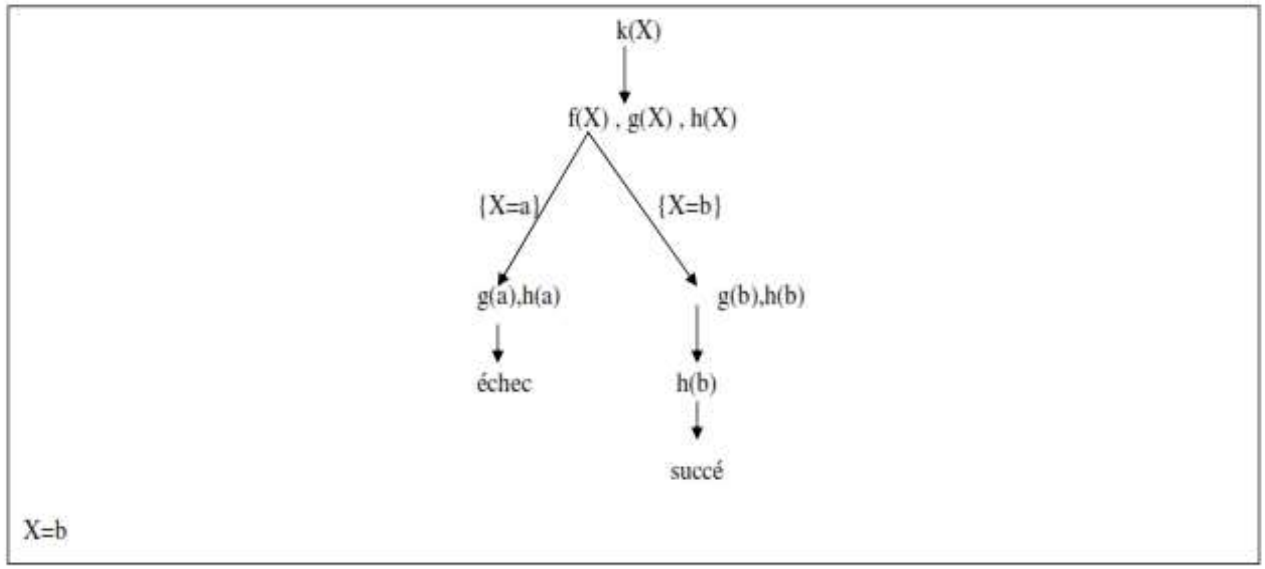
```
f(a).
f(b).
g(b).
h(b).
k(X) :- f(X), g(X), h(X).
```

Et soit la requête suivante :

```
?- k(X).
```

La seule réponse de prolog est : X=b.

Comment prolog a-t-il trouvé cette réponse ? Dessinez l'arbre de résolution (l'arbre de recherche).



E={f(a),f(b),g(b),h(b).k(b)}