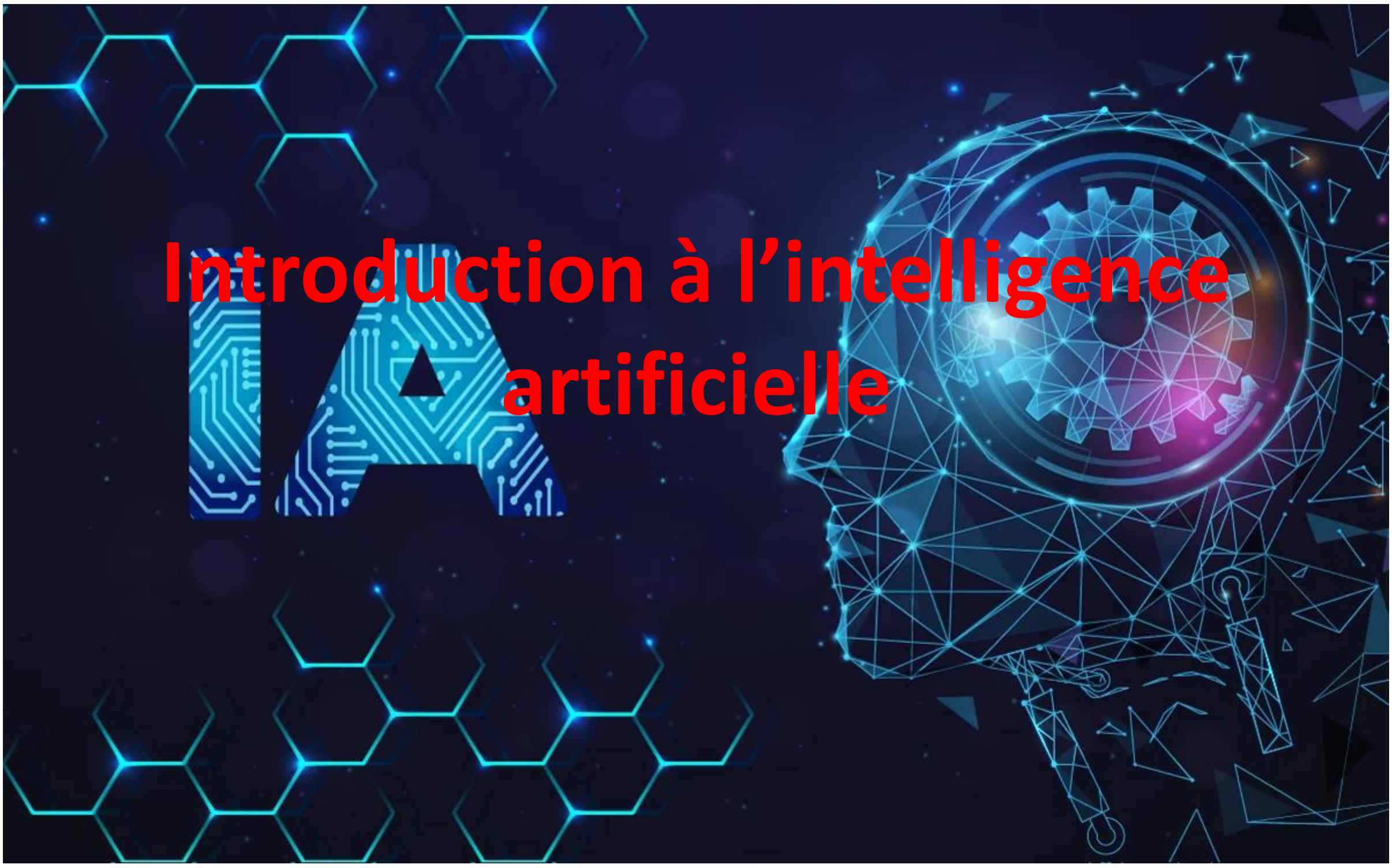


Licence 3SI

Dr Titouna Faiza

2022

GÉNÉRALITÉS

Qu’est-ce que l’intelligence ?

Des réponses générales ont été données par des chercheurs :

- **Turing :** c’est ce qui rend difficile la distinction entre une tâche accomplie par un être humain ou par une machine.
- **Darwin :** c’est ce qui permet la survie de l’individu le plus apte.
- **Edison :** C’est ce qui fait que cela fonctionne.
- **Lorenz :** C’est collectif et cela émerge du comportement collaboratif.

- L’intelligence peut être considérée comme la capacité d’apprendre à partir des expériences passées (acquérir et conserver de nouvelles connaissances) et de pouvoir appliquer ces nouvelles connaissances dans un nouveau contexte (**Jeff Schweitzer)**.
- La capacité de penser, de raisonner et de comprendre au lieu de réagir automatiquement ou par instinct ” (Dictionnaire Collins)

Qu’est-ce que l’intelligence artificielle ?

L’intelligence artificielle est en fait un domaine vaste qui comprend plusieurs axes et diverses définitions selon le point de vue envisagé.

- L’intelligence artificielle est une combinaison de techniques visant à développer les machines dans le but d’accomplir des tâches précises et résoudre les problématiques de l’homme.
- L'Intelligence Artificielle (IA) est la science dont le but est de faire faire par une machine des tâches que l'homme accomplit en utilisant son intelligence.
- Etude des activités intellectuelles de l'homme pour lesquelles aucune méthode n'est a priori connue
- L‘informatique est la science du traitement de l'Information, l'IA s'intéresse à tous les cas où ce traitement ne peut être ramené à une méthode simple, précise ou algorithmique

« La capacité, conférée par les humains aux machines, de mémoriser et d’apprendre sur la base de l’expérience, de penser et de créer, de parler, de juger et de décider »

Test de Turing (1950)

Si l'interrogateur ne peut distinguer d'une manière fiable entre l'être humain et la machine

→ la machine possède une intelligence (artificielle)

Test : A qui avez-vous affaire ?

Le dialogue suivant est la retranscription d'un véritable test de Turing. Essayez de deviner si "l'entité" est un être humain ou une machine.

Examinateur
Aimez-vous le printemps ?
Entité
Ca dépend de mon humeur.
Examinateur
Combien font 11 et 11 ?
Entité
22
Examinateur
Et 512+512?
Entité
Je n'ai jamais été doué en calcul mental.
Examinateur
Ce n'est pas grave, essayez.
Entité
Voyons, 1000 quelque chose. 1024 je pense.
Examinateur
Récitez-moi un poème.
Entité
La servante au grand cœur dont vous étiez jalouse. Et qui dort son sommeil sous une humble pelouse...
Examinateur
Pourquoi ne remplacez-vous pas "au grand cœur" par "au grand air" ? A mon avis ce serait plus joli.
Entité
Je l'aime mieux tel quel.
Examinateur
Pourquoi?
Entité
Vous n'êtes pas sérieux. "Au grand air" et "au grand cœur" ce n'est pas pareil.
Examinateur
Alors remplacez "Et qui dort son sommeil" par "qui tristement sommeille".
Entité
Vraiment, je l'aime mieux tel quel.
Examinateur
Pourquoi ?
Entité
Le rythme du poème est meilleur.
Examinateur
Ce n'est pas mon avis
Entité
Vous aimez Baudelaire?
Examinateur
Oui
Entité
Moi aussi.
Examinateur
Pourquoi
Entité
Je trouve qu'il exprime des sentiments profonds qui correspondent souvent à ce que je peux éprouver. Et puis peut-être aussi que ça me rappelle la période où je l'ai lu pour la première fois quand j'étais ado.
Examinateur
Vous aimez l'art?
Entité
Oui
Examinateur
Vous peignez ?
Entité
Comme un pied. Je suis plus attiré par la musique.
Examinateur
Vous jouez d'un instrument?
Entité
Oui, du piano
Examinateur
Jouez-nous un morceau.
Entité
Je n'ai pas d'instrument.

Histoire de l'IA

- **1936 :** La machine de Turing
- **1943-1950** : Warren McCulloch (1898-1969), et Walter Pitts (1923-1969), neurologues, inventent le premier modèle mathématique du neurone biologique, le neurone formel. En 1943: apparition des premiers véritables ordinateurs. Le premier ordinateur avec programme mémorisé et complété, autrement dit le premier ordinateur qui aura déjà un programme enregistré et prêt à être exécuter. L'EDSAC, réalisé à Cambridge par Maurin Wilkes.
- ***Naissance d'IA (1956)*** : C'est durant cette année qu'un petit groupe d'informaticiens intéressés par l'étude de l'intelligence se réunirent pour une conférence sur ce thème. Cette conférence a permit de poser les fondements de l'intelligence artificielle (nom qui fut choisi à l'issue de cette conférence par John McCarthy) ;
- ***Période active de l'IA (1952-1969)*** : Un grand nombre de programmes furent développés pour résoudre des problèmes d'une grande diversité. LISP fut le premier langage crée pr Mac Carthy 1960). Les premiers programmes d'IA Logic Theorist (par Newell et Simon) et Geometry Theorem Prover (Gelernter) furent en mesure de prouver certains théorèmes mathématiques. Le General Problem Solver (GPS) de Newell et Simon réussissait quant à lui à résoudre des puzzles simples avec un raisonnement semblable au raisonnement humain. Samuel créa un programme jouant (à un niveau moyen) aux dames. Ce fut aussi l'époque du Shakey, le premier robot à être capable de raisonner sur ses propres actions.

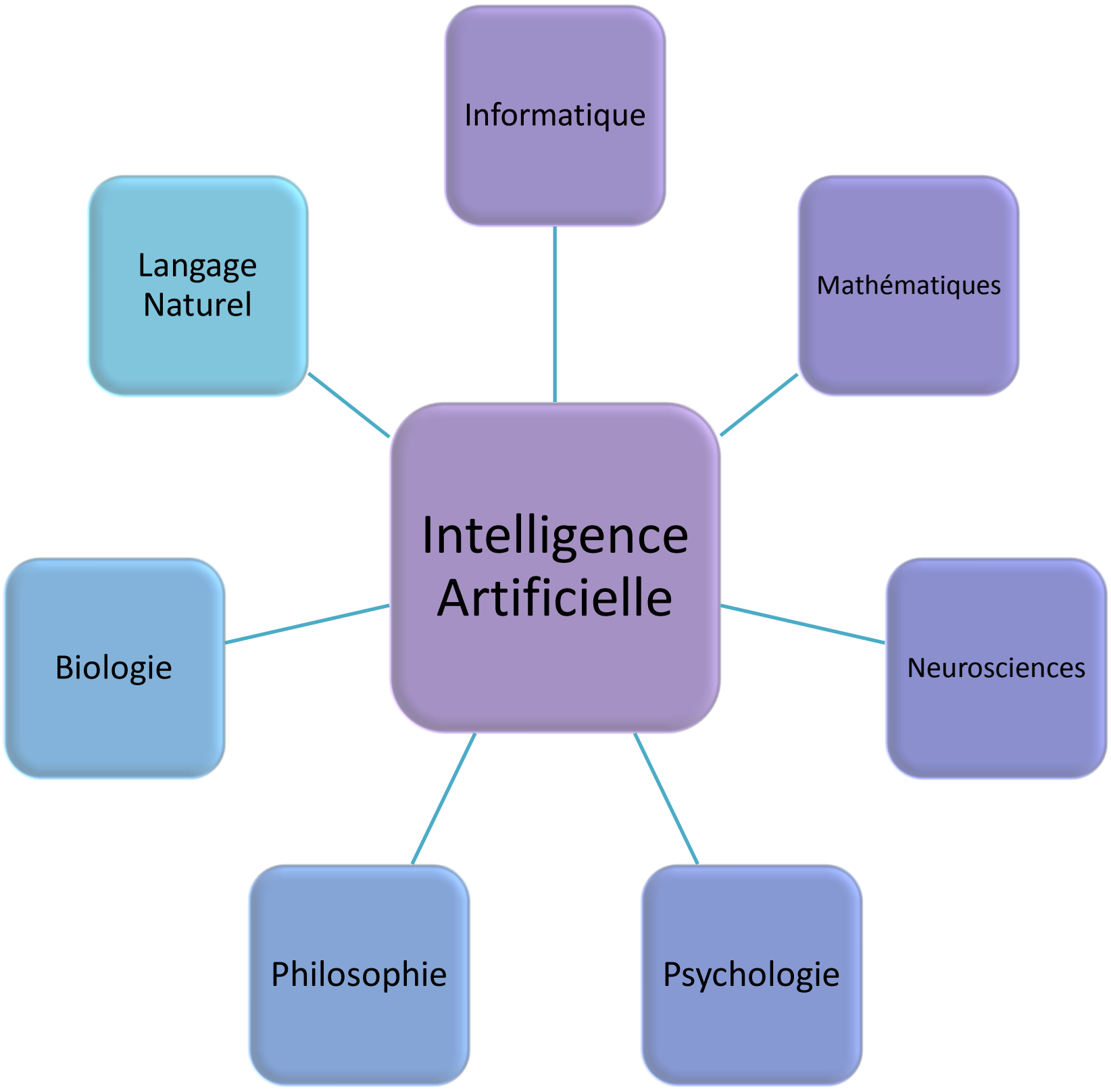
HISTOIRE DE L'IA

- ***Premières Déceptions (1966-1973)*** : De grandes déceptions se produisirent lorsque les chercheurs en IA essayèrent d'appliquer leurs algorithmes aux problèmes de grande taille, et découvrirent alors qu'ils ne fonctionnaient pas, par manque de mémoire et de puissance de calcul. Ce qui provoqua l'arrêt du financement de la quasi-totalité des projets en IA de Grande Bretagne.
- ***Systèmes Experts (1969-1979)*** : PROLOG, qui a la puissance de la logique du premier
- ordre, et a ete concu par A. Colmerauer en 1971.Le premier système expert, appelé DENDRAL, fut créée en 1969 pour la tâche spécialisée consistant à déterminer la structure moléculaire d'une molécule étant donné sa formule. DENDRAL, comme tous les systèmes experts, est basée sur un grand nombre de règles heuristiques élaborées par des experts humains. Après le succès du DENDRAL, d'autres systèmes d'experts furent créés, notamment le système MYCIN, qui réalisait un diagnostic des infections sanguines. Avec 450 règles, MYCIN réussissait à diagnostiquer à un niveau proche des experts humains.
- ***L'IA dans l'Industrie (1980-1987) :*** Les Etats-Unis et le Japon financèrent de gros projets en IA, et la Grande Bretagne relança son programme de financement.

HISTOIRE DE L'IA

- ***L'IA Moderne (1987-2011) :*** L'intelligence artificielle est devenue au fil du temps une matière scientifique de plus en plus rigoureuse et formelle. La plupart des approches étudiées aujourd'hui sont basées sur des théories mathématiques ou des études expérimentales plutôt que sur l'intuition, et sont appliquées plus souvent aux problèmes issus du monde réel. Arrivée de méthodes probabilistes et vision agent.
  - ***Année 1997 :*** La machine d'IBM Deep Blue bat le champion du monde en jeu d'échecs Gary Kasparov
  - ***Année 2002:*** pour la première fois, l'IA est entrée dans la maison sous forme de Roomba, un robot aspirateur.
  - ***Année 2006:*** L'IA s'est investie dans le monde des affaires jusqu'en 2006. Des entreprises comme Facebook, Twitter et Netflix ont également commencé à utiliser l'IA.
- ***Deep Learning et Big Data (2011- présent)*:** Le concept d'apprentissage profond, le big data et data sciences constituent les travaux de recherche actuels. Aujourd'hui, des entreprises comme Google, Facebook, IBM et Amazon travaillent avec l'IA. L'avenir de l'intelligence artificielle est inspirante.

L’IA et Autres disciplines



Domaines d'application de l'IA

- **Algorithmique**
  - Ecriture et preuve des algorithmes
- **Théorie de la complexité**
  - Complexité des problèmes
- **Vision par ordinateur**
  - Détection d'anomalies (de fraudes)
  - Reconnaissance de formes (image, objet,...)
- **Logique**
  - Logique propositionnelle et logique des prédicats
  - Autres logique (Par défaut, Non monotone, …)
- **Bases de données**
  - Apprentissage
  - Fouille de données
- **Langage naturel**
  - Analyses sémantique automatisée
  - Traduction automatique
  - Reconnaissance vocale
- **Robotique**
  - Planification des taches
- …..

Quelques difficultés rencontrées en IA

- **Difficultés de modélisation** :
  - les problèmes ne sont pas toujours parfaitement définis
  - certaines notions sont difficiles à exprimer : possibilité, probabilité, préférence, . . .
- **Difficultés de résolution :**
  - Difficultés de conception des algorithmes
  - Espaces de recherche très vastes
  - problèmes de temps de réponse
- **Difficultés de généralisation :**
  - Les méthodes sont souvent dédiées à un problème particulier
  - Des problèmes très variés

SYSTÈMES EXPERTS

QU'EST QU'UN SYSTÈME EXPERT?

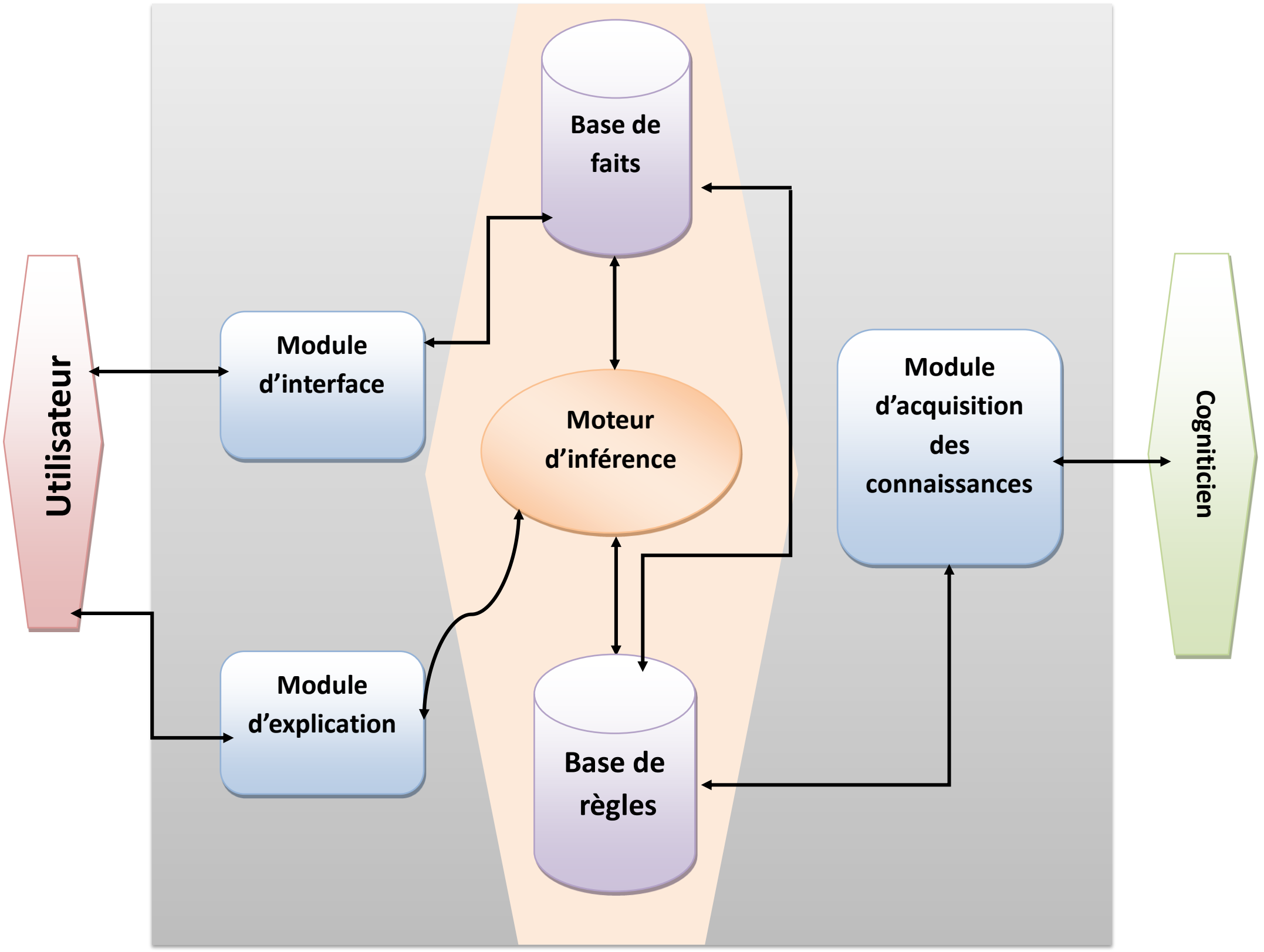
En 1965 une équipe de l'université de Stanford travaillait sur la résolution d'un problème pour lequel on ne connaît pas de solution algorithmique. Le premier système expert est né. L'idée fut d'introduire la connaissance des experts dans les ordinateurs en les rendant intelligents. L'expertise se traduit bien souvent par un ensemble de règles déductives qui reflètent par leurs enchainements le raisonnement des experts. Le programme va lire ces règles et établir un raisonnement en tentant de les appliquer. L'idée qui apparaît ensuite consiste à transformer l'expertise de l'homme à l'humain.

Un système expert (un Système à Base de Règles) est donc un logiciel qui:

- reproduit le comportement d'un expert humain accomplissant une tâche intellectuelle dans un domaine précis**,**
- peut résoudre des problèmes très difficiles aussi bien ou mieux que les experts humains,
- raisonne à l'aide d'heuristiques,
- interagit de façon évoluée (langage naturel),
- manipule des données symboliques,
- peut justifier ses conclusions

ARCHITECTURE D'UN SYSTÈME EXPERT

Un système expert est constitué de plusieurs modules qui s'interagissent.



ARCHITECTURE D'UN SYSTÈME EXPERT

Base de connaissances

C'est la mémoire qui contient l'ensemble de la connaissances acquise dans le domaine d'application.

- **Base de Faits**
  - Mémoire de travail (à court terme)
  - Des faits ou données propres au problème à résoudre
- **Base de Règles**
  - Mémoire à long terme d'un SBC
  - Plusieurs formalismes (Règles de Production, Réseaux Sémantiques, etc.)
  - L'expertise nécessaire pour résoudre un Problème
  - Faits permanents du savoir faire

Autres modules

- **Module d'interface :** Il sert à simplifier la communication entre l'utilisateur et le système. Il peut être sous la forme question-réponse, menu ou en langage naturel.
- **Module d'explication :** Il permet au système d'établir son raisonnement (en donnant la trace du raisonnement)
- **Module d'acquisition des connaissances :** Il permet de traduire la connaissance de l'expert en la représentant sous forme d'un langage compréhensible par le système.

Base de faits

- **Mémoire de travail**

La base de faits BF est la mémoire de travail du système expert. Elle est variable au cours de l'exécution et vidée lorsque l'exécution se termine. Au début de la session, elle contient ce que l'on sait du cas examiné avant toute intervention du moteur d'inférences. Puis elle est complétée par les faits déduits par le moteur ou demandés à l'utilisateur.

Par exemple, dans le domaine médical, la base de faits pourra contenir une liste de symptômes en début de session et un diagnostic lorsque celle-ci se terminera.

- **Type d'un fait**

Les faits peuvent prendre des formes plus ou moins complexes. Nous n'envisagerons que des faits élémentaires dont les valeurs possibles sont : booléennes, symboliques ou réelles

Par exemple : *actif* est un fait booléen, *profession* est un fait symbolique et rémunération est un fait *réel*.

Un système expert qui n'utilise que des faits booléens est dit **d'ordre 0.**

Un système expert qui utilise des faits symboliques ou réels, sans utiliser de variables, est **d'ordre 0+**.

Un système utilisant la logique du premier ordre est **d'ordre 1**.

Base de faits

- **Valeur d'un fait**

Il faut qu'un système expert puisse savoir si une valeur a été attribuée à un fait. Dans la négative, cette valeur pourra être demandée à l'utilisateur.

Mais si l'utilisateur ne peut pas répondre, il faudra que le système puisse le savoir en la déduisant des autres faits. On dira que la valeur d'un fait est :

- ***établie*** ou connue si une valeur lui a été attribuée.

- ***inconnue*** si aucune valeur ne lui a été attribuée et si aucune question à son sujet n'a été posée à l'utilisateur

- ***indéterminée*** si le système ne lui a attribué aucune valeur et si l'utilisateur a répondu "je ne sais pas" à une question concernant sa valeur.

- **Les formules et conditions**

Dans un système expert d'ordre 0, on pourra par exemple écrire des formules de la forme : *actif* **ou** ¬*actif*

Dans un système d'ordre 0+, on pourra trouver les formules :

actif **et** (*profession* ≠ *médecin* **ou** *rémunération* ≤ 20000)

Dans un système d'ordre 1, on pourra trouver :

∀X *maladie*(X) **et** (X ≠ *grippe*) **et** *(symptôme*(X) = *forte_Fievre*)

Ces formules sont appelées conditions lorsqu'elles servent à déclencher des règles. On remarque que les faits booléens peuvent être interprétés comme des formules puisqu'ils possèdent une valeur de vérité.

Base de règles

Les connaissances sont souvent représentées par les ***règles de production*** de la forme :

Si *condition* **alors** *conclusion*, afin d'être proche du langage naturel.

La partie *condition* ou *prémisse* de la règle correspond à une conjonction d'expressions qui doivent être vérifiées pour que la règle se déclenche.

La partie *conclusion* ou *conséquences* correspondent à une conjonction d'actions.

La base de règles n'évolue donc pas au cours d'une session de travail.

En formalisme logique, en notant les conditions $C1,\ldots,Cn$ et les actions $A1,..,An$, une règle est ainsi écrite : $C1 \wedge \ldots \wedge Cn \rightarrow A1,..,An$

En prolog, la règle s'écrit pour chaque action : $Ai \leftarrow C1 \wedge \ldots \wedge Cn$

Exemple :

Ordre 0 : logique des propositions

- **Si** Ferrari **et** Michael **alors** rapide

Ordre 0+: logique des propositions typée (attribut-valeur)

- **Si** voiture = ferrari **et** pilote = michael **alors** vitesse = rapide

Ordre 1 : logique des prédicats

- $\forall$ X, Y voiture(X) et (X = ferrari) $\wedge$ pilote(Y, X) et (Y = michael) $\rightarrow$ rapide(X)
- **Si couleur**(voiture)=*rouge* **et marque(**voiture)=*fer*rari **alors** conducteur = *heureux*

On distingue dans la règle :

- Attributs : voiture, conducteur (en prolog ce sont des variables)
- Valeurs : rouge, ferrari, heureux (en prolog ce sont des constantes) .
- Opérateur : ce qui lie un attribut et une valeur (=).
- Connecteurs : 'et ', il se traduit par la conjonction ',' en prolog).

Le but d'un SE est d'affecter des valeurs à des attributs en déclenchant des règles (exécution de la règle).

Métarègles et méta-connaissances

La méta-connaissance s'exprime par des métarègles (c'est-à-dire des règles sur la manière d'utiliser les règles). On trouve par exemple dans MYCIN les métarègles suivantes :

- **Si** le patient est un hôte à risque et **S'**il existe des règles mentionnant des pseudo-monias dans une prémisse et **S'**il existe des règles mentionnant des klebsiellas dans une prémisse **Alors** il est probable qu'il faille utiliser les premières avant les secondes.
- **Si** on recherche une thérapie **Alors** considérer dans cet ordre les règles qui permettent de :
  - acquérir les informations cliniques sur le patient
  - trouver quels organismes, s'il en existe, sont causes de l'infection
  - trouver tous les médicaments potentiellement utiles
  - choisir les plus adaptés en plus petit nombre

Autre exemple :

MR1 : **S'i**l y a des éruptions ou des rougeurs **alors** envisager R1, R2, R3, R4

MR2 : **Si** le patient est une femme adulte et si R1, R2, R3, R4 sont envisagées **alors**

R1 est prioritaire

MR3 : **Si** le patient est un adolescent ou un enfant et si R1, R2, R3, R4 sont envisagées **alors** R4 est peu probable

MR4 : **Si** deux règles sont également probables **alors** prendre en priorité celle qui a le plus de conditions

R1 : **Si** la fièvre est faible, la peau sèche, s'il y a des ganglions, pas de pustules ni de rhume **alors** envisager la rubéole

R2 : **Si** les boutons sont isolés et démangent beaucoup avec une faible fièvre, et si la croûte apparaît vite sur les pustules ou les vésicules **alors** envisager fortement la varicelle

R3 : **S'**il y a rhume, mal aux yeux, taches roses dans la gorge, boutons en taches, fièvre forte **alors** envisager la rougeole

R4 : **S'**il y a amygdales rouges**,** forte fièvre, taches rouge vif **alors** envisager la scarlatine

Moteur d’inférence (MI)

- Le rôle du moteur d’inférences est de simuler la réflexion de l’expert humain.
- Couche logicielle qui correspond à un algorithme de simulation des raisonnements
- Permet aux systèmes experts de conduire des raisonnements logiques et de dériver des conclusions à partir de la base de faits et de la base de connaissances
- C’est un mécanisme qui permet d’inférer de nouvelles connaissances à partir de la BC.
- Exploite la BC en fonction du contenu de la BF afin de mener un raisonnement sur le problème posé
- Indépendant de tout domaine d'application (utilisant même langage d’expression de connaissances et les mêmes heuristiques de résolution de conflit)

→ L’indépendance entre la base de connaissances et le moteur d’inférences est un élément essentiel des systèmes experts

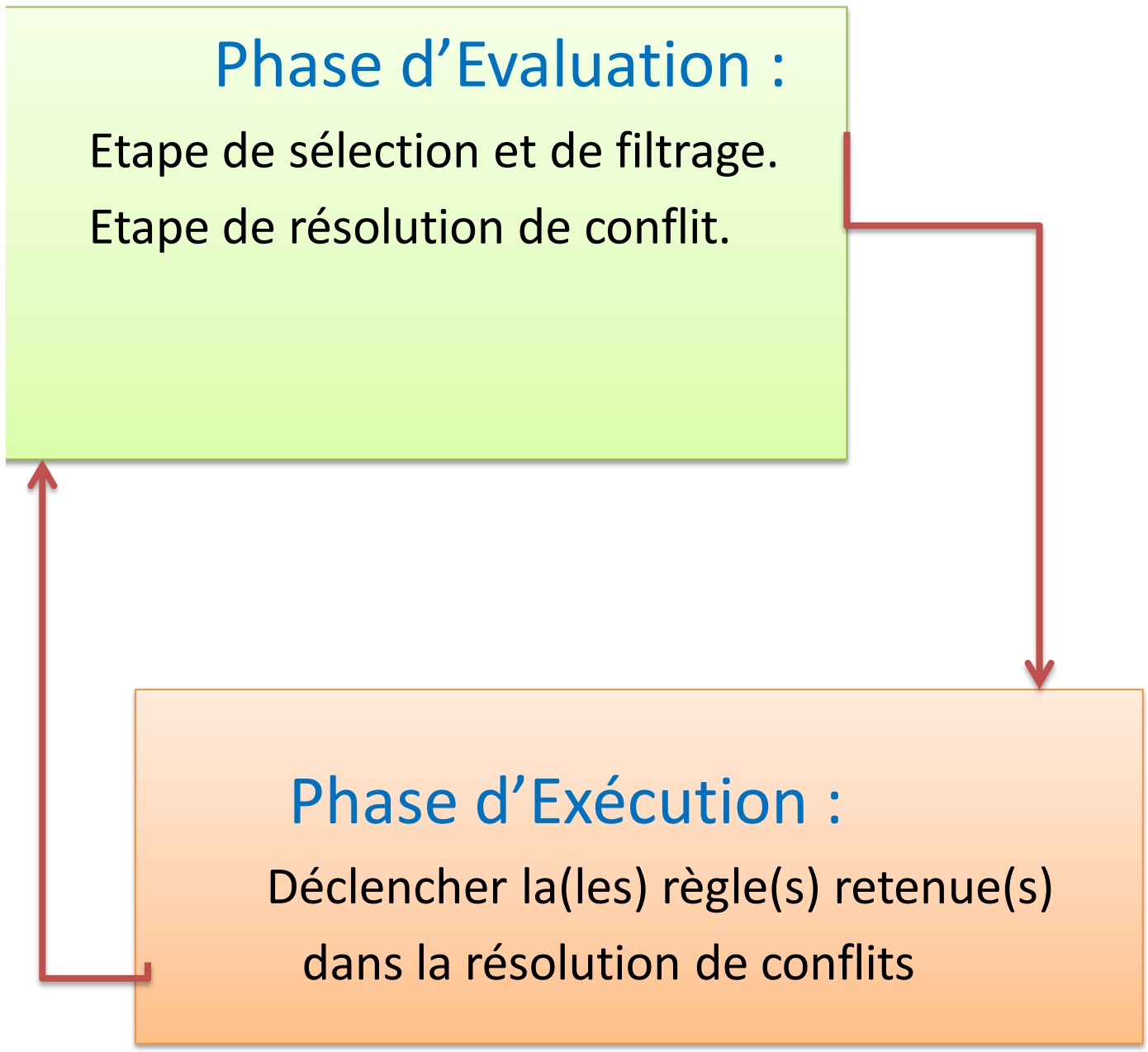
Principe De Fonctionnement (MI)

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 :

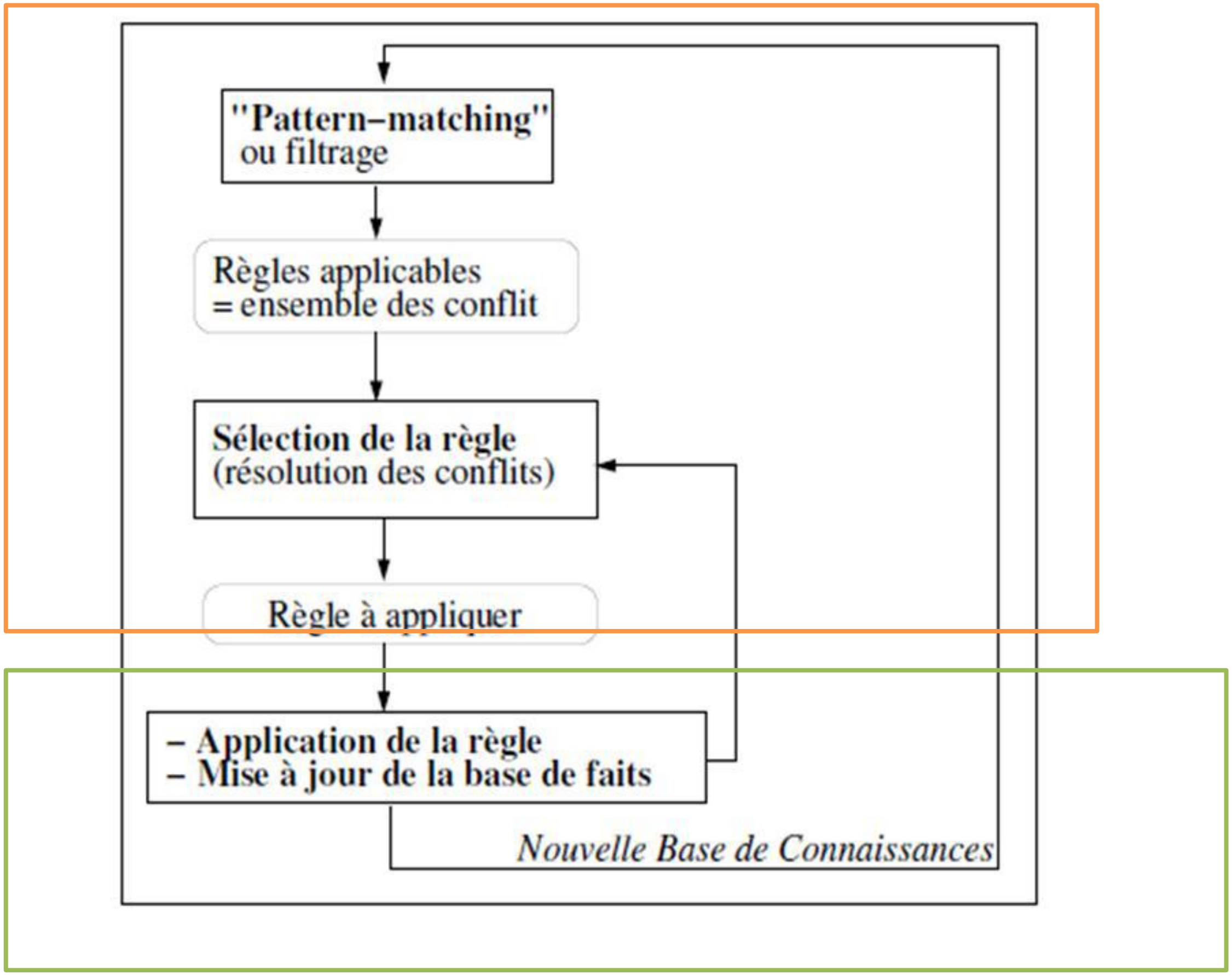
- **Phase d'évaluation :** constitue l'ensemble des règles déclenchables (conflict set)

- **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 fais, 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. L'application d'une règle dans un cycle du MI n'est jamais remise en cause . S'il n'y a plus de règles à appliquer. Le MI s'arrête et signale un échec sans faire retour en arrière

- **Régime par tentative (Backtracking):** Le MI examine la possibilité de déclencher d'autres règles déclenchables. 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.

- **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 moteur d’inférence

- 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

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 développement se présentent:

- Schéma 1: soit on développe toutes les règles d'un même niveau l'une après l'autre avant de passer au niveau suivant (stratégie de développement en largeur d'abord)
- Schéma 2: 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 de développement en profondeur d'abord). Le retour arrière (ou backtracking) est appliqué dans le cas où le développement en profondeur échoue.

Saturation de la base de faits: Effectuer toutes les déductions possibles jusqu'à ce qu'il ne reste plus de règles applicables.

Exemple1 : chainage avant selon schéma 1

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 montre que H se déduit de la base de connaissances.

Chaque étape de l'exécution de l'algorithme correspond à un cycle de déduction,

Etape 1 Règles applicables : 4 et 7. On applique la règle 4 puis la règle7, on ajoute les conclusions à la BF, on obtient:

BF = {B, C, X, D}

Les règles 4 et 7 sont désactivées.

Etape 2 Règles applicables : 5, 8 et 9. On déclenche toutes ces règles l'une après l'autre, on ajoute les conclusions à la BF, on obtient:

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

Les règles 5, 8 et 9 sont est désactivées.

Etape 3 Règles applicables : 1 et 6. On déclenche toutes ces règles l'une après l'autre, on ajoute les conclusions à la BF, on obtient:

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

Les règles 1 et 6 sont est désactivées.

→ Le but H est établi (démontré).

Exemple1 : chainage avant selon schéma 2

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}

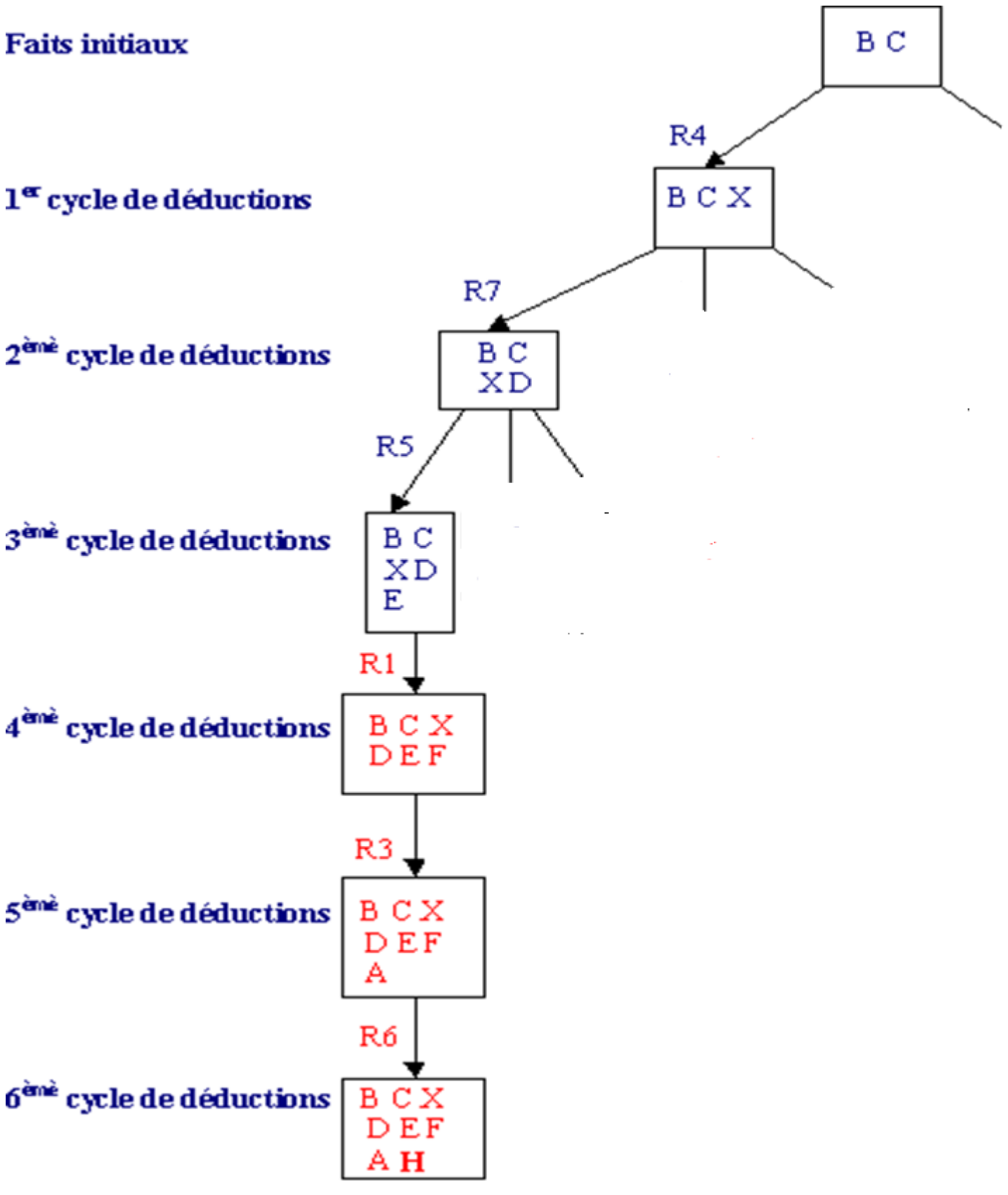
La règle 3 est désactivée et la conclusion de R3 est ajoutée à la BF.

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

H est établi.

Exemple1 : Arbre du chainage avant selon schéma 2

Cet arbre montre que la solution est trouvée en appliquant le chainage avant selon la stratégie du schéma 2. On remarque que le développement de l'arbre est réalisée en profondeur, d'une manière monotone avec un régime irrévocable.



Exemple2

1. **SI** X aime la musique classique et les maths **ALORS** il pourrait aimer Bach
2. **Si** X aime la musique classique et X est de tempérament-romantique **ALORS** il pourrait aimer Schubert.
3. **SI** X est de tempérament-romantique **ALORS** il pourrait aimer Cabrel.
4. **SI** X écrit des poèmes **ALORS** il est de tempérament-romantique.

Nous pouvons modéliser ces quatre règles avec la logique de prédicats, en utilisant les clauses de Horn :

R1 : peutAimer(X, bach) ← aime(X, musiqueC), aime(X, maths).

R2 : peutAimer(X, schubert) ← aime(X, musiqueC), romantique(X).

R3 : peutAimer(X, cabrel) ← romantique(X).

R4 : romantique(X) ← ecritPoeme(X).

Supposons qu'on ajoute au programme précédent les deux clauses suivantes :

F1 : aime(toto, musiqueC). Et F2 : ecritPoeme(toto).

Remarques

- l'algorithme de chaînage avant peut être très lent, mais il s'arrête toujours.

- si l'on utilise des règles dont les conclusions peuvent être des faits négatifs, pour tout fait F, il peut se produire 4 situations :

- F ∈ BF : le fait est établi.
- F ∉ BF : la négation du fait est établie.
- - ni F ni ¬F ne sont dans BF : le système ne déduit rien à propos du fait.

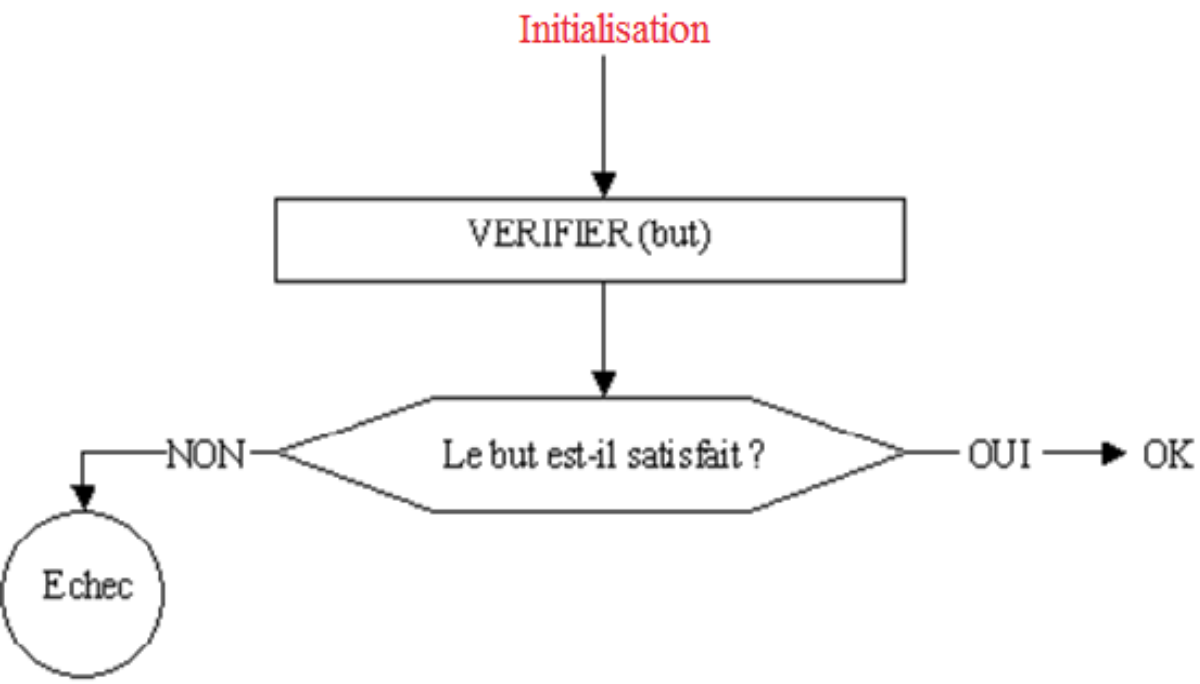
- F et ¬F ∈ BF : la base est incohérente. On peut prévoir une métarègle de la forme : S'il existe un fait qui appartient, ainsi que sa négation, à BF alors "base-incohérente".

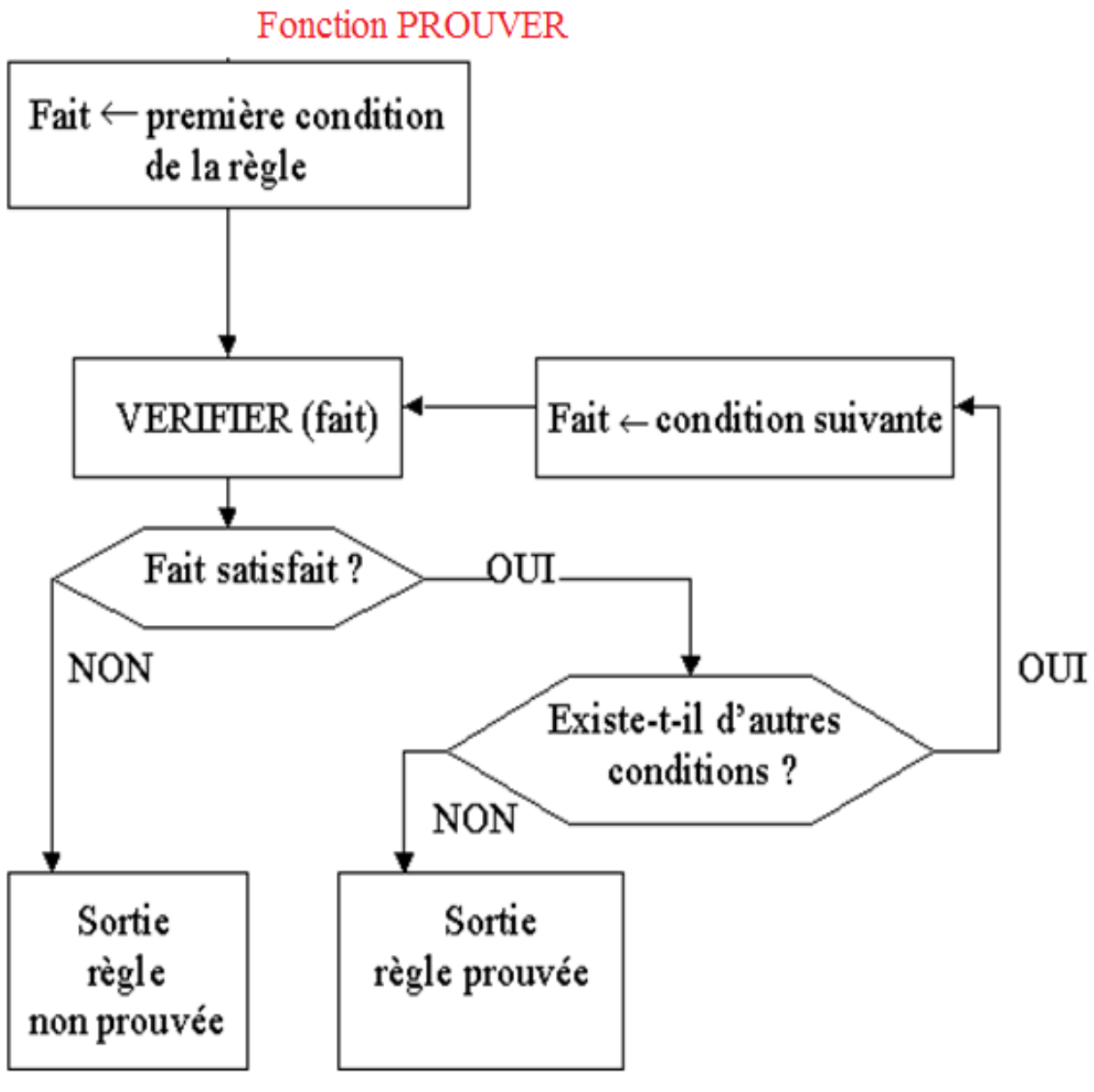
Algorithme du chainage arrière

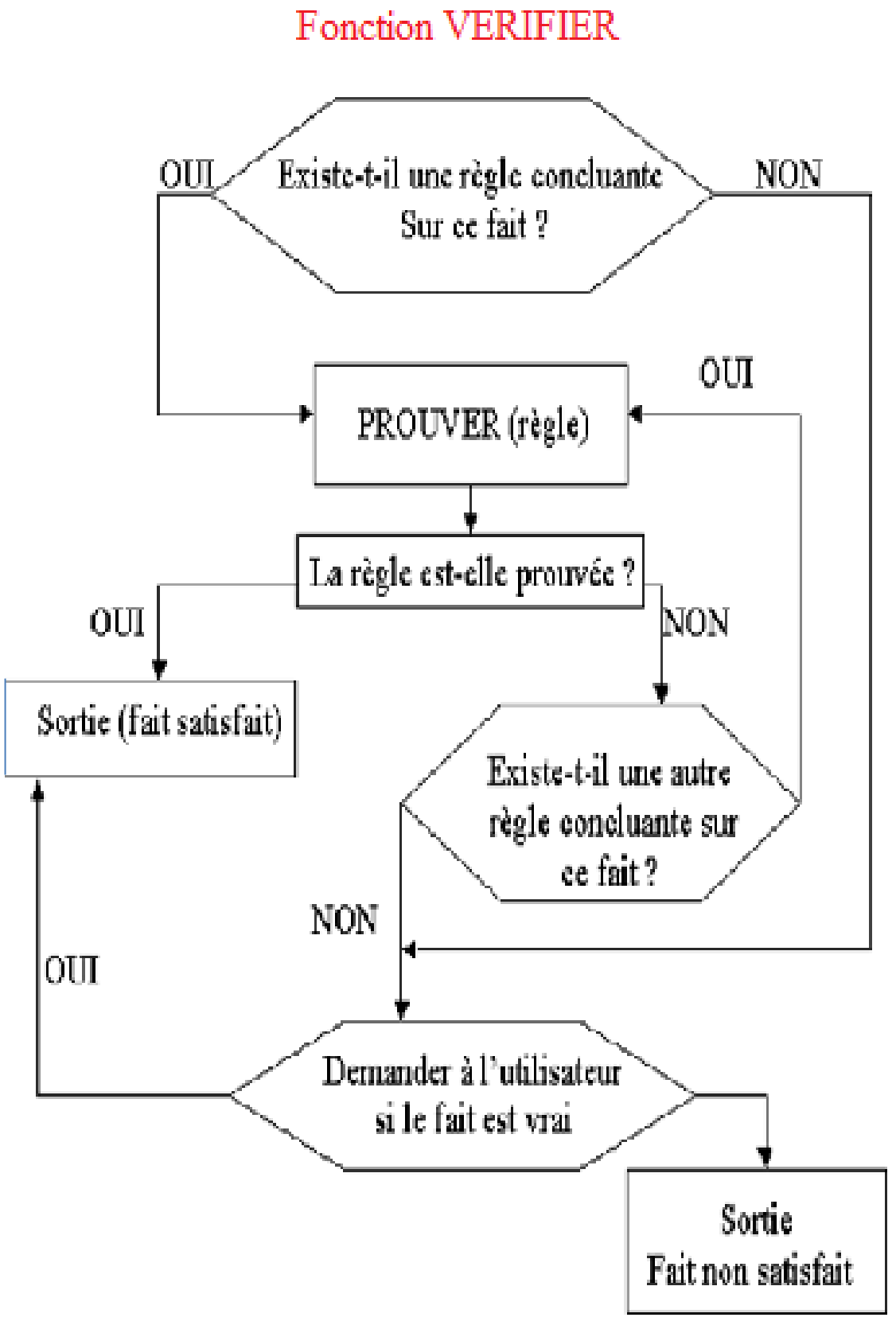
```
Fonction chaînageArriere ( BR, BF,  listeButs).
if   estVide (listeButs)  then
|        res ← SUCCES
else
|    if   demBut ( premier( listeButs ))  then
|    |        res  ← chaînageArriere (suite( listeButs ))
|    else
|    |        res ← ECHEC
|    end if
endif
retourner    res;
```

```
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 du Chainage arrière





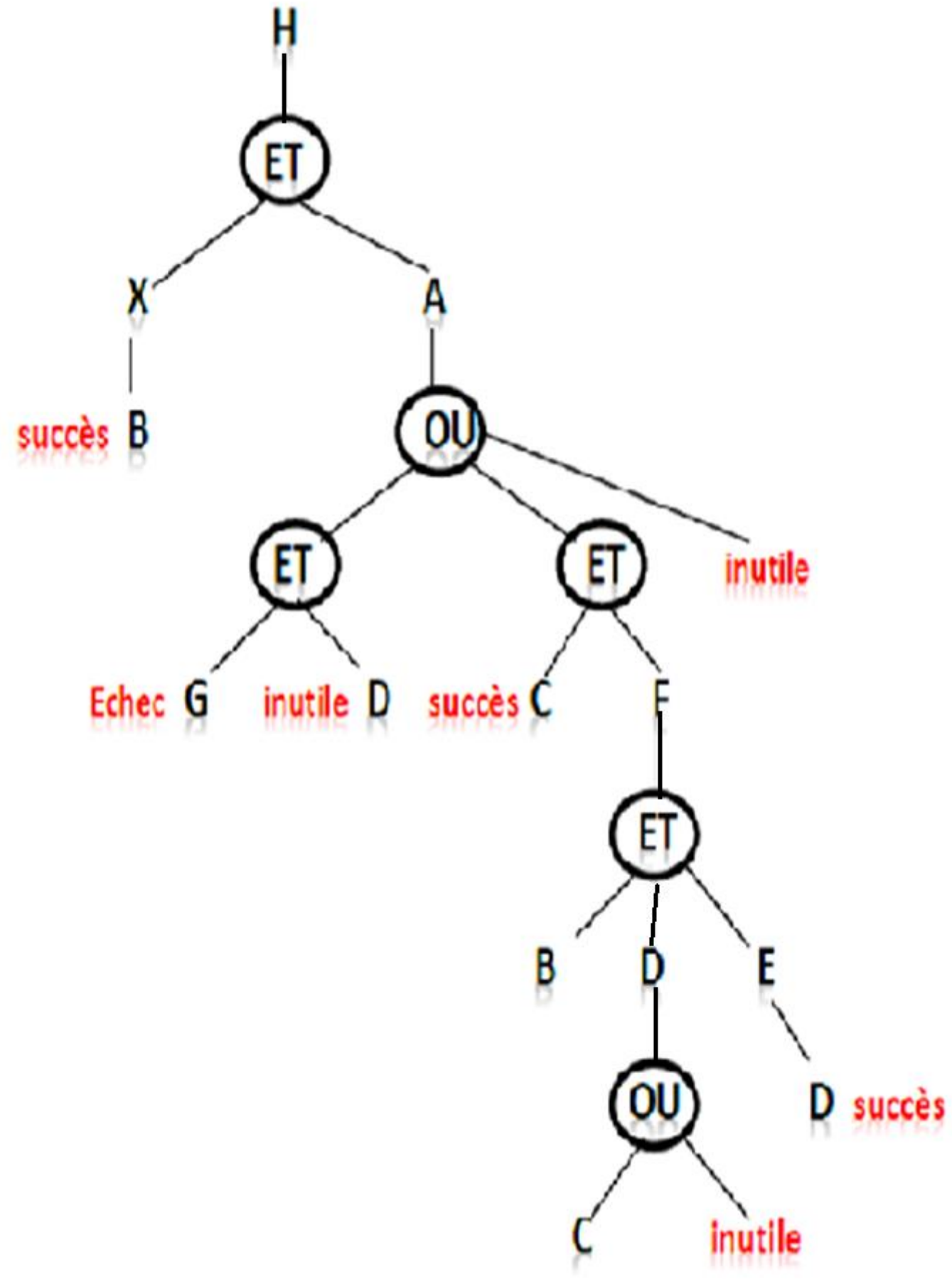


Arbre Et/OU

L'application du l'algorithme du chainage arrière est représenté par un arbre ET/OU.

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



Exercice

Etant donné la base de connaissances BC,

1- Appliquer le chainage avant par saturation de la base de règles?

2- Soit F le but à démontrer,

2.1- Appliquer le chainage avant selon les deux schémas (largeur et profondeur)

2.2 Donner l'arbre ET/OU pour le chainage arrière.

Base de Connaissances BC:

Base de Faits :

BF = {A, D, E, G}

Base de Règles : BR= {

R1: A et B et C alors H

R2: A et U et C alors F

R3: E et G et B alors S

R4: D et G alors C

R5: A et E alors B

R6: U et S et T alors F

R7: G et H alors R

R8: D et E alors T

R9: R et S et H alors F

R10: A et U alors B

}

Chainage arrière amélioré

L'algorithme du chainage arrière est amélioré par l'ajout de faits à caractère demandables. Autrement dit, on donne la possibilité au système de poser des questions pour connaitre la valeur, de ces faits ,initialement inconnu.

Si nous avons un fait à prouver, on favorise les règles qui contiennent le plus de faits demandables. Ceci permet de déterminer l'ensemble des questions pertinentes.

Exemple: Etant donné une base de faits demandables: {B, D, F, K }; une base de faits: { M} et une base de règles. On désire démontrer le fait { A }.

R1: B et C alors A

R2: D et E alors A

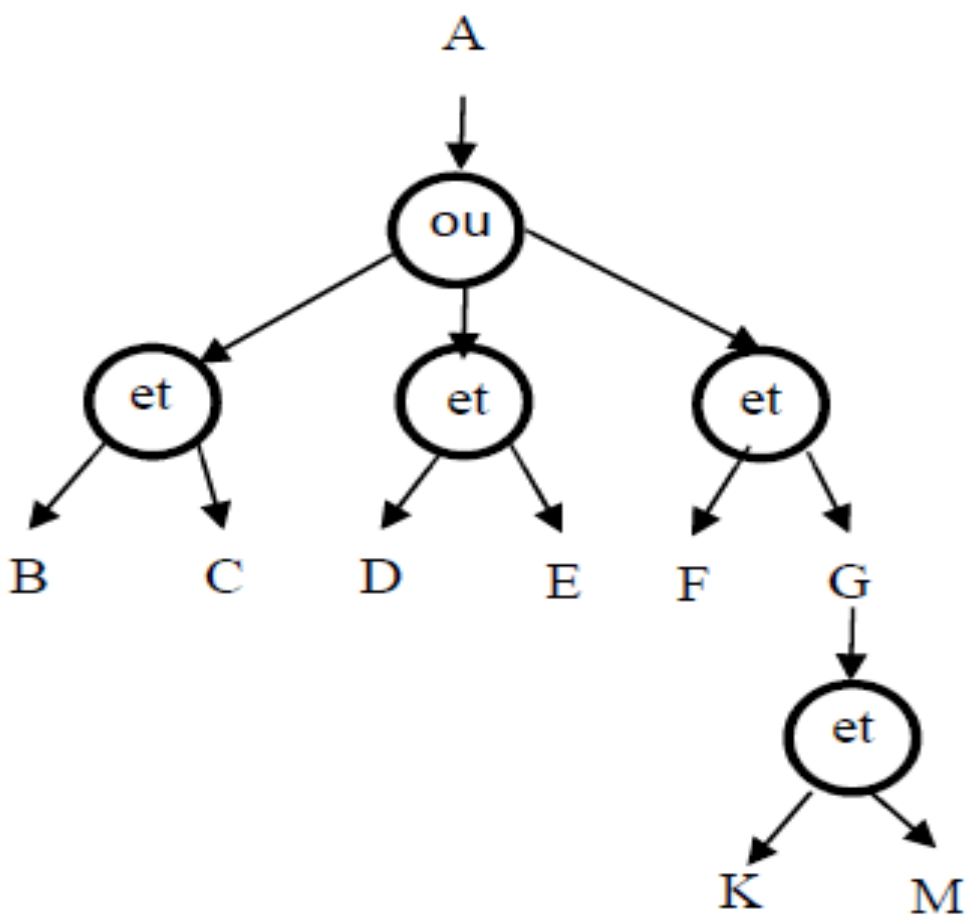
R3: F et G alors A

R4: K et M alors G

R5: M alors ¬ E

Quelles sont les questions pertinentes à poser à l'utilisateur? Pour cela, nous allons d'abord donner l'arbre ET/OU :

Chainage arrière amélioré (suite)



Pour démontrer le but « A » on doit parcourir l'arbre de gauche à droite. Pour chacun des sous-arbres 'ou', on vérifie si le fait appartient à la base de faits sinon on pose une question, s'il est un fait demandable.

- La question « B est-il vrai? » est- elle pertinente, la réponse est NON. Puisque, aucune règle ne conclut sur le fait « C » en plus C n'est pas un fait demandable. D'où la valeur de « B » n'apporte rien pour démontrer « A ».
- La question « D est-il vrai? » est- elle pertinente, la réponse est NON. Puisque, d'après la règle 5, M est vrai (∈ BF) implique que ¬ E est vrai , d'où E est faux.
- La question « F est-il vrai? » est- elle pertinente, la réponse est OUI. Puisque G est encore déductible.
- Mais si la réponse à la question « F est-il vrai? est non (si l'utilisateur a répondu faux pour F), dans ce cas la question « K est-il vrai? » n'est pas une question pertinente puisque la valeur de G ne servira à rien.

Chainage mixte

L'algorithme du chainage mixte combine à la fois le chainage avant et le chainage arrière.

Principe

Entrée: BF , BR , But

Sortie: but démontré ou échec

Tant que le but n'est pas déduit faire

- Saturer la base de faits par chainage avant,
- Chercher les faits encore déductibles par chainage arrière,
- Déterminer les questions pertinentes,
- Ajouter les réponses à la base de faits,

INCERTITUDE DANS LES SYSTÈMES A BASE DE RÈGLES

Pourquoi y a t-il de l'incertitude?

- Raisonnement approximatif,
- Raisonnement inexact,
- Information disponible de l'expert humain
  - Incomplète
  - Incertaine
  - Inconsistante

Incertitude dans les SE

Données inconnues

- si certaines données ne sont pas disponibles, on doit continuer le raisonnement avec valeur ‘inconnue’

Opinions de différents experts

- Les experts n’ont pas toujours la même conclusion,
- Des conclusions contradictoires produiront de règles contradictoires,

Facteur de certitude

- Un SE doit être capable de fonctionner avec de l'incertitude puisque les systèmes réels contiennent toujours des données incomplètes et imprécises. Plusieurs méthodes existent pour travailler avec l'incertitude, on cite la méthode des Facteurs de certitudes,

- Le ***Facteur de certitude fut introduit dans Mycin***

- Le ***Facteur de certitude*** (fc) est un nombre qui mesure la croyance d'un expert. Sa valeur maximale est +1.0 (totalement vrai) et sa valeur minimale est -1.0 (totalement fausse).

Exemple: Si un expert affirme qu'une information est presque certaine, un facteur de certitude de valeur 0.8 est affectée à cette information

Facteur de certitude

Dans les sytèmes experts avec facteurs de certitude, une base de connaissances consiste en un ensemble de règles de la forme suivante:

Si <evidence> alors <hypothèse H> {*fc*}

Où ***cf*** représente la croyance dans l'hypothèse H étant donné que la preuve ***e*** s'est produite.

Le facteur de certitude affecté par la régle est propagé à travers une chaine de raisonnement. Cela consiste à établir la certitude nette de la règle qui en découle lorsque la preuve de l'antécédent de la règle est incertaine :

$$fc\,(H, e) = fc\,(E) \times fc_{règle}$$

Exemple:

Si le ciel est clair alors temps ensoleillé {*fc* 0.8}

où le facteur de certitude de "le ciel est clair" est 0.5, alors

$$fc\,(H,e) = 0.5 \times 0.8 = 0.4$$

Ce résultat peut s'inerpreter : "Il peut être ensoleillé".

Facteur de certitude : Cas de règles conjonctives

Si < evidence *E1*>

⋮

et < evidence *En* >

alors < hypothèse *H* > {fc}

- La certitude de l'hypothèse est établie cmme suit :

$$fc\,(H, E_1 \cap E_2 \cap \ldots \cap E_n) = min\,[fc\,(E_1), fc\,(E_2), \ldots, fc\,(E_n)] \times fc$$

Exemple:

Si ciel est clair

et prévision est ensoleillé

alors l'action est "porter des lunettes solaires' {*fc* 0.8}

En plus, les facteurs de certitude de "le ciel est clair" est **0.9** et prévision du temps est ensoeillé" **est 0.7**, alors

$fc(H, E_1 \cap E_2) = min\,[0.9, 0.7] \times 0.8 = 0.7 \times 0.8 = 0.56$ (il faudrait probablement porter des lunettes)

Facteur de certitude : Cas de règles disjonctives

Si < evidence *E1*>
⋮
ou < evidence *En* >
alors < hypothèse *H* > {fc}

Le facteur de certitude de l'hypothèse *H*, est établi comme suit:

$$cf\,(H, E1 \cup E2 \cup \ldots \cup E_n) = max\,[fc\,(E_1), fc\,(E_2), \ldots, fc\,(E_n)] \times fc$$

Exemple:

Si ciel est nuageux ou prévisions il pleut alors l'action est 'prendre un parapluie' {*fc* 0.9}

les facteurs de certitude de la formule "le ciel est nuageux est **0.6** et celui de " les previsions est qu'il pleut" est **0.8**, alors

$$fc\,(H, E_1 \cup E_2) = max\,[0.6, 0.8] \times 0.9 = 0.8 \times 0.9 = 0.72$$

Facteur de certitude combiné

- Si la meme conséquence est obtenue comme résultat de l'exécution de deux règles ou plus, les facteurs de certitudes associés aux règles sont combinés pour donner le facteur de certitude de l'hypothèse,
- Supposons que la base de connaissance est de la forme suivante:

Règle 1: SI A est X
Alors C est Z {fc0.8}

Règle 2: SI B est Y
Alors C es Z {fc 0.6}

- Quelle est la certitude à affecter à l'objet C de valeur Z si la règles1 et La règle 2 sont déclenchées?

Facteur de certitude combiné

- Si nous avons deux informations évidentes (A est X et B est Y) provenant de différentes sources (règle 1 et la règle 2) soutenant la même hypothèse (C est Z). Ainsi, la croyance de cette hypothèse devrait augmenter et devenir plus forte.
- Le calcul du facteur de certitude combiné est donné comme suit:

$$fc(fc_1, fc_2) = \begin{cases} fc_1 + fc_2 \times (1 - fc_1) \;\; si\; fc_1 > 0\; et\; fc_2 > 0 \\ \dfrac{fc_1 + fc_2}{1 - \min(|fc_1|, |fc_2|)} \;\; si\; fc_1 < 0\; ou\; fc_2 < 0 \\ fc_1 + fc_2 \times (1 + fc_1) \;\; si\; fc_1 < 0\; et\; fc_2 < 0 \end{cases}$$

Où:

fc_1 est la croyance de l'hypothèse H établie par la règle 1;

fc_2 est la croyance de l'hypothèse H établie par la règle 2;

Exemples

Soit un système expert incertain d'ordre 0.

Base de faits = { A: 0.2, B:0.7, C:0.9 }

Base de règles :

R1: (A et B) ou C alors H (0.8)

R2: A ou C alors F (0.3)

R3: B alors F (1.0)

Calculer les facteurs de certitude associés à "H" ensuite à "F".

fc(H) = max(min(0.2, 0.7), 0.9) x 0.8 = 0.72

Par la règle R2:

fc1(F) = max(0.2, 0.9) x 0.3 = 0.27

Par la règle R3:

fc2(F) = 0.2 x 1 = 0.2

Appliquons la définition du calcul de certitude combiné:

fc(F) = fc1(F) + fc2(F) x (1 - fc1(F))

= 0.27 + 0.2 x(1 – 0.27)

= 0.416

Formalisme de représentation de connaissances

LOGIQUE DES PRÉDICATS

Syntaxe

1. Termes
   - **Constantes** : chaînes de caractères commençant par une minuscule ou un nombre
     - E.g. aA, 125, 13B, ali
   - Symboles de **fonctions** ($f(t_1, \ldots t_n)$ où les t_i sont des termes)
     - E.g. pereDe(X), distance_entre(X,Y), ...
   - **Prédicats** (ou atomes) ($p(t_1, \ldots t_n)$ où les t_i sont des termes)
     - E.g. sur(X, Y), sur(Z, table), ...

2. Ensemble de **variables**
   - Chaînes de caractères commençant par une majuscule
     - E.g. X1, B23, Taille, ...

3. **Connecteurs**
   - $\wedge$, $\vee$, $\neg$, =>, $\Leftrightarrow$

4. **Quantificateurs**
   - $\exists$ (existentiel), $\forall$ (universel)

Syntaxe

- **Terme**
  - Toute **variable** est un terme
  - Toute **constante** est un terme
  - Si t_1, ..., t_n sont des termes et f est un symbole de **fonction**, alors $f(t_1, ..., t_n)$
    - E.g. plus_grand(5,2), f(a, b, c, d), ...
- Un terme est **clos** s'il ne contient pas de variable

- **Formules**
  - Un atome ou formule atomique
  - Si F est une formule, alors $\neg F$ est une formule
  - Si F et G sont des formules, alors $F \wedge G$, $F \vee G$, $F \Rightarrow G$ sont des formules
  - La formule vide, {} (ou un carré), définie par $x \wedge \neg x$

Variables libres et variables liées

- Variable **liée** : variable se trouvant **dans la portée d'un quantificateur**
  - $\forall X\ b(X)$: X est liée
  - $\forall X\ b(X, Y) \Rightarrow a(X)$: la porté de $\forall$ est $b(X, Y)$ et X est liée dans $b(X, Y)$, mais libre dans $a(X)$
- Variable **libre** : une variable qui est **non liée**
  - $\forall X\ (b(X, Y) \Rightarrow \exists Y, c(Y))$: X est liée, la 1ère occurrence de Y est libre, mais la 2ème est liée

- La notion de portée et de variable libre est **importante** quand on veut **remplacer des variables** par d'autres variables ou par des termes.

Sémantique

En logique des propositions, à chaque variable (symbole propositionnel) est affecté une valeur de vérité. En logique des prédicats, le sens des formules est récursivement construit en affectant des valeurs aux constantes, aux variables et en définissant les symboles de fonctions et de prédicats. C'est ce qu'on appelle **« interprétation ».**

Exemple: Soient C_1,C_2,C_3 trois constantes et soient « plus/1 » symbole de fonction d'arité 1 (1 argument) et « grand/2 » symbole de prédicat à arité 2. Soit la formule

F = $grand(plus(C_1, C_3), C_2)$, la valeur de vérité de F dépend de l'interprétation I donnée .

- I = {C_1 ←1, C_2 ←2, C_3 ←3 ; plus : '+' ; grand: '>' }, La valeur de vérité de la formule F sous l'interprétation I est I(F) = vraie [(C_1 + C_3) > C_2 ≡ (1 + 3) > 2]
- I1 = {C_1 ←2, C_2 ←3, C_3 ←1 ; plus : ' - ' ; grand: '>' }, d'où I1(F) = fausse

Pour affecter des valeurs aux variables, on a besoin de définir un domaine, ainsi l'interprétation devient ,

- I2 = { D =[1,2,3,4]; C_1 ←1, C_2 ←3 ; plus : ' + ' ; grand: '>' },
  soit G = $\forall x\ grand(plus(C_1, C_2), x)$, d'où I(G) = fausse et pour
  H = $\exists x\ grand(plus(C_1, C_2), x)$; I(H) = vraie
- I3 = { D=[nadia, sonia, leila]; ami = [(nadia, sonia), (sonia, leila), (leila, nadia)]},
  Soit F1 = $\forall x\ ami(nadia, x)$, I(F1)= fausse ; F2 = $\exists x\ ami(nadia, x)$, I(F2)=vraie; F3 = $\forall x\ \exists y\ ami(x, y)$, I(F3)= vraie ; F4 = $\forall x\ \neg ami(x, x)$; I(F4)= vraie; F5 = $\exists x\ \forall y\ ami(x, y)$, I(F5)= fausse

Sémantique (suite)

- Un modèle est une interprétation qui rend vraie une formule.

  Par exemple, I3 est un modèle pour les formules F2, F3 et F4 mais elle ne l'est pas pour F1 et F5.

- On dit qu'une formule G est une conséquence logique d'une formule F si et seulement si tout modèle de F est aussi modèle de G, et on note:

  $\boldsymbol{F \models G}$ cette écriture est équivalente à $\models (\boldsymbol{F \rightarrow G})$ qui signifie que $F \rightarrow G$ est une tautologie (formule valide). Si I(F) = vraie implique que I(G) = vraie pour toute interprétation I, ce qui est équivalent à I($\boldsymbol{F \rightarrow G}$) = vraie pout tout I.

- On dit qu'une formule G est conséquence logique d'un ensemble de formules, et on écrit $\{\boldsymbol{F_1, F_2, \cdots, F_n}\} \models \boldsymbol{G}$, ssi tout modèle de $\boldsymbol{F_1 \wedge F_2 \wedge \cdots \wedge F_n}$ est aussi modèle de $\boldsymbol{G}$.

Equivalences des quantificateurs

- $\forall X\ \forall Y$ est équivalent à $\forall Y\ \forall X$
- $\exists X\ \exists Y$ est équivalent à $\exists Y\ \exists X$
- $\exists X\ \forall Y$ **n'est pas** équivalent à $\forall Y\ \exists X$:
  - $\exists X\ \forall Y$ *aime(X, Y)* : « *Il existe une personne qui aime tout le monde* »
  - $\forall Y\ \exists X$ *aime(X, Y)* : « *Tout le monde est aimé par quelqu'un* »
    (pour toute personne, il existe quelqu'un qui l' aime).

- Soient *G*, *F* et *H* des formules. Soit $Q \in \{\forall, \exists\}$

$QX\ F[X] \vee G \equiv QX\ (F[X] \vee G)$ et $QX\ F[X] \wedge G \equiv QX\ (F[X] \wedge G)$.

$\neg(\forall X\ F[X]) \equiv \exists X\ (\neg F[X])$ et $\neg(\exists X\ F[X]) \equiv \forall X\ (\neg F[X])$.

$\forall X\ F[X] \wedge \forall X\ H[X] \equiv \forall X\ (F[X] \wedge H[X])$ et $\exists X\ F[X] \vee \exists X\ H[X] \equiv \exists X\ (F[X] \vee H[X])$.

Relations d'Equivalences

- On applique autant de fois que nécessaire les équivalences :

1. $\neg(\forall X\ w) \equiv \exists X\ \neg w$
2. $\neg(\exists X\ w) \equiv \forall X\ \neg w$
3. $(\forall X\ w1) \wedge w2 \equiv \forall X(w1 \wedge w2)$ (si X n'apparaît pas dans w2)
4. $(\forall X\ w1) \vee w2 \equiv \forall X(w1 \vee w2)$ (si X n'apparaît pas dans w2)

Forme normale Prenex d'une formule (FNP)

- Une formule *F* est en **forme normale *prenex*** ssi *F* est dans la forme

$$Q_1\, X_1\, Q_2\, X_2 \ldots Q_n\, X_n \quad (M)$$

où chaque $Q_i\, X_i$ est soit $\forall X_i$ soit $\exists X_i$ et *M* est une formule en forme normale *prenex* ne contenant pas de quantificateurs.

- Exemple :

$$\forall X\ \forall Y\ (\exists Z\ (p(X, Z) \wedge p(Y, Z)) \rightarrow \exists U\ (q(X, Y, U)))$$

$$\equiv \forall X\ \forall Y\ (\neg(\exists Z\ (p(X, Z) \wedge p(Y, Z))) \vee \exists U\ (q(X, Y, U)))$$

$$\equiv \forall X\ \forall Y\ (\forall Z\ (\neg(p(X, Z) \wedge p(Y, Z))) \vee \exists U\ (q(X, Y, U)))$$

$$\equiv \forall X\ \forall Y\ (\forall Z\ (\neg p(X, Z) \vee \neg p(Y, Z)) \vee \exists U\ (q(X, Y, U)))$$

$$\equiv \forall X\ \forall Y\ \forall Z\ \exists U\ ((\neg p(X, Z) \vee \neg p(Y, Z) \vee q(X, Y, U)))$$

Algorithme : Mise sous Forme normale de Prenex

Entrée: Formule F en logique des prédicats

Sortie : F sous forme Prenex

1. Eliminer '$\rightarrow$ 'et ' $\leftrightarrow$ ', $(\mathrm{A} \rightarrow B \equiv \neg \mathrm{A} \vee \mathrm{B})$ et $(\mathrm{A} \leftrightarrow B \equiv (\mathrm{A} \rightarrow B) \wedge (B \rightarrow A))$
2. Distribuer la négation sur les formules atomiques en appliquant les lois de DeMorgan,
3. Renommage des variables si nécessaire telle que chaque sous-formule possède ses propres variables :

   Tant qu'il y a dans F une sous-formule Qx G / la variable x apparait en dehors de G dans la formule F **faire**

   - Remplacer Qx G / Qy G {x/y} où y une nouvelle variable n'apparaissant pas dans F,
4. Appliquer autant que possible les relations d'équivalences (pour faire sortir les quantificateurs).

Notons que cette transformation préserve l'équivalence ;

Exemple:

F = $\exists \mathrm{x}\,(\mathrm{p}(\mathrm{x}) \rightarrow \forall x q(x))$

- Eliminer $\rightarrow$, F = $\exists \mathrm{x}\,(\neg \mathrm{p}(\mathrm{x}) \vee \forall x q(x))$
- Renommer la variable x dans la 1ère ou la 2ième sous formule:

  F = $\exists \mathrm{x}\,(\neg \mathrm{p}(\mathrm{x}) \vee \forall y q(y))$
- Sortir $\forall y$, F = $\exists \mathrm{x}\, \forall y\,(\neg \mathrm{p}(\mathrm{x}) \vee q(y))$ est sous forme Prenex

Skolémisation

- Soit $F = Q_1 X_1 Q_2 X_2 \ldots Q_n X_n \ (M)$ une formule en FNP.
- Construction de la forme standard de F :

  Supposons que $Q_r = \exists$, avec $1 \leq r \leq n$.

  - **Si avant Q_r il n'y a pas de $\forall$** alors on **remplace** toutes les occurrences de la variable x_r dans M **par une nouvelle constante** c_r n'apparaissant pas avant dans M et on **élimine** $Q_r x_r$.
  - **Si avant Q_r il y a des $\forall$**, disons $Q_{j1}, Q_{j2}, \ldots, Q_{jt}$, avec $1 \leq j_1 < j_2 < \ldots < j_t < r$ alors on **remplace** toutes les occurrences de x_r dans M **par une nouvelle fonction** t-aire f_r $(x_{j1}, x_{j2}, \ldots, x_{jt})$ dans M et on **élimine** $Q_r x_r$.

- *Exemple* :
  - Soit $F = \exists X \forall Y \forall Z \exists U \forall V \exists W \ (p(X, Y, Z, U, V, W))$.

    F skolemisée : $F = \forall Y \forall Z \forall V \ (p(a, Y, Z, f(Y, Z), V, g(Y, Z, V)))$.

Substitution

- Une substitution est un ensemble $\{X_1/t_1, \ldots, X_n/t_n\}$ où chaque X_i est une **variable**, chaque t_i est un **terme** ≠ de X_i , et pour tout $i \neq j$, $X_i \neq X_j$.
  - *Exemples* : {X/f (z), Z/Y} , {X/a, Y/g(x), Z/f (g(b))}.

- Soient $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ une substitution et *E* une expression. Alors $E\theta$ est une expression obtenue à partir de *E* en **remplaçant de manière simultanée** chaque occurrence de la **variable** X_i dans *E* **par le terme** t_i, $1 \leq i \leq n$.

  $E\theta$ est appelée une instance de *E*.

  - *Exemple* :

    Soient θ = {*X* /*a*, *Y* /*f* (*b*), *Z* /*c* } une substitution et *E* = *p*(*X*, *Y*, *Z*) une expression. Alors, Eθ = p(a, f(b), c).

Unification

- Une substitution θ est appelée un **unificateur** pour un ensemble $\{E_1, E_2, \ldots, E_k\}$ ssi $E_1\theta = E_2\theta = \ldots = E_k\theta$.
- L'ensemble $\{E_1, E_2, \ldots, E_k\}$ est dit unifiable s'il existe un unificateur pour lui.
- Un unificateur σ pour un ensemble d'expressions $\{E_1, \ldots, E_k\}$ est **l'unificateur plus général** ssi pour chaque unificateur θ pour l'ensemble, il y a une substitution λ telle que $\theta = \sigma \circ \lambda$.
  - *Exemple* :

    L'ensemble {p(a, X, f(g(Y))) , p(Z, f(Z), f(W))} est unifiable et a pour u.p.g. la substitution θ = {Z/a, X/f (a), W/g(Y)}.

Algorithme d'unification

algorithme **unification** (t_1, t_2 : des termes)
début
si l'un des termes (t_1 ou t_2) est une variable x, (l'autre est t)
alors si $x = t$ alors *possible* $\leftarrow$ *vrai*, $\sigma \leftarrow \emptyset$
sinon
    si x apparaît dans t alors *possible* $\leftarrow$ *faux*
    sinon *possible* $\leftarrow$ *vrai*, $\sigma \leftarrow (\sigma(x) = t)$ finsi
fin si
sinon ($t_1 = f(x_1, \cdots, x_n)$ et $t_2 = g(y_1, \cdots, y_m)$)
si $f \neq g$ ou $n \neq m$ alors *possible* $\leftarrow$ *faux*
sinon $i \leftarrow 0$, *possible* $\leftarrow$ *vrai*, $\sigma \leftarrow \emptyset$
    tant que $i < n$ et *possible* faire
        $i \leftarrow i + 1$, $(possible, \sigma') \leftarrow unification(\sigma(x_i), \sigma(y_i))$
        si *possible* alors $\sigma \leftarrow \sigma' \circ \sigma$ finsi
    fin tant que
fin si fin si
fin

Exemple

littéraux			substitution
$R(\underline{x}, g(c, z))$	et	$R(\underline{f(y)}, g(y, k(x))$	$(x\|f(y))$
$R(f(y), g(\underline{c}, z))$	et	$R(f(y), g(\underline{y}, k(f(y)))$	$(y\|c)$
$R(f(c), g(c, \underline{z}))$	et	$R(f(c), g(c, \underline{k(f(c))})$	$(z\|k(f(c)))$
$R(f(c), g(c, k(f(c))))$			$(x\|f(c)), (y\|c), (z\|k(f(c)))$
facteur principal			unificateur principal

Autre exemple :

littéraux			substitution
$R(\underline{x}, f(x))$	et	$R(\underline{f(y)}, y)$	$(x\|f(y))$
$R(f(y), \underline{f(f(y))})$	et	$R(\overline{f(y)}, \underline{y})$	échec (test d'occurence)

Autre exemple :

littéraux			
$P(\underline{f(x)})$	et	$P(\underline{c})$	échec (c est une constante)

Forme Clausale

Une clause est une disjonction de formules atomiques positives ou négatives (précédées par la négation).

Exemple: clause 1: $p(\mathrm{x}) \vee q(\mathrm{x})$; clause 2: $p(\mathrm{x}) \vee \neg q\big(\mathrm{x}, f(a)\big)$;
clause 3: $p(a) \vee q(\mathrm{x}) \vee \neg r(\mathrm{y})$; clause 4: $p(x, \mathrm{y})$;

La suite de transformations qui permet de passer d'un ensemble de formules $\sum = \{F_1, F_2, \cdots, F_n\}$ à un ensemble de clauses, est :

Etape 1 Mettre sous forme normale de *Prénexe* chacune des formules de $\sum$. On obtient : $\sum_p = \{F_{p1}, F_{p2}, \cdots, F_{pn}\}$

Etape 2 Mettre sous forme normale de *Skolem* chacune des formules obtenues dans l'étape 1. On obtient : $\sum_s = \{F_{s1}, F_{s2}, \cdots, F_{sn}\}$

Etape 3 Mettre sous forme normale *conjonctive* , chacune des formules obtenues dans l'étape 2, $(\Lambda_i C_{Fi})$ où les formules (C_{Fi}) sont des disjonctions.

Etape 4 Le résultat obtenu est un ensemble de clauses C_{Fi} pour chaque formule $\sum$.

Forme Clausale: Exemple

Considérons la formule suivante :

$$\forall x \forall y \forall z\ [(\forall x\, p(x,x)) \wedge [(\forall x\, p(x,y)) \rightarrow (\exists y\, p(y,x) \wedge \forall y\, p(y,z))]]$$

1-Mise sous forme prénexe : renommage

$$\forall x \forall y \forall z\ [(\forall x'\, p(x',x')) \wedge [(\forall x''\, p(x'',y)) \rightarrow (\exists y'\, p(y',x) \wedge \forall y''\, p(y'',z))]]$$

2-Mise sous forme prénexe : application des règles d'équivalence

$$\forall x \forall y \forall z.[\forall x' \exists x''\ (p(x',x') \wedge (p(x'',y) \rightarrow (\exists y'\, p(y',x) \wedge \forall y''\, p(y'',z))))]$$
$$\forall x \forall y \forall z.[\forall x' \exists x'' \exists y' \forall y''\ (p(x',x') \wedge (p(x'',y) \rightarrow (p(y',x) \wedge p(y'',z))))]$$
$$\forall x \forall y \forall z \forall x' \exists x'' \exists y' \forall y''\ [p(x',x') \wedge (p(x'',y) \rightarrow (p(y',x) \wedge p(y'',z)))]$$

Forme Clausale: Exemple

Ensuite il faut appliquer la Skolémisation :

$$\forall x \forall y \forall z \forall x' \exists x'' \exists y' \forall y''.[p(x',x') \wedge (p(x'',y) \rightarrow (p(y',x) \wedge p(y'',z)))]$$
$$\forall x \forall y \forall z \forall x' \exists y' \forall y''.[p(x',x') \wedge (p(f(x,y,z,x'),y) \rightarrow (p(y',x) \wedge p(y'',z)))]$$
$$\forall x \forall y \forall z \forall x' \forall y''.[p(x',x') \wedge (p(f(x,y,z,x'),y) \rightarrow (p(g(x,y,z,x'),x) \wedge p(y'',z)))]$$

Enfin, nous mettons sous forme normale conjonctive et nous obtenons l'ensemble de clauses :

$$\{p(x',x'), \quad \neg p(f(x,y,z,x'),y) \vee p(g(x,y,z,x'),x), \quad \neg p(f(x,y,z,x'),y) \vee p(y'',z)\}$$

Remarque : chaque ensemble de clauses est équivalent à la conjonction de leurs clôtures universelles, on obtient donc un ensemble de clauses équivalent en renommant chaque variable de clause. Par exemple, l'ensemble ci-dessus est équivalent à :

$$\{p(x_0,x_0), \quad \neg p(f(x_1,y,z,x'),y) \vee p(g(x_1,y,z,x'),x_1), \quad \neg p(f(x_2,y_3,z',x''),y_3) \vee p(y'',z')\}$$

Dans la suite, on suppose que **les clauses ne partagent aucune variable**.

Résolution: Preuve par réfutation

- Pour montrer qu'une formule est valide, on montre que ¬F est inconsistante (contradictoire).

- Pour montrer qu'une formule G est conséquence logique d'un ensemble de formules $\sum = \{F_1, F_2, \cdots, F_n\}$, on note, $\{F_2, \cdots, F_n\} \vDash G$, on montre que $\{F_1, F_2, \cdots, F_n\} \cup \{\neg G\}$ est inconsistante.

  – Pour cela, on transforme l'ensemble des formules $\{F_1, F_2, \cdots, F_n, \neg G\}$ sous forme clausale,

  – **Répéter**

    Appliquer la règle de coupure autant que possible sur l'ensemble des clauses obtenues

    jusqu'à obtention clause vide.

  – **Si** non clause vide **alors** $\{F_2, \cdots, F_n\} \nvDash G$

Règle de résolution (coupure)

- Soient deux clauses A et B **sans variables communes** (sinon, on renomme les variables communes dans l'une des clauses) telles que $A = C_1 \vee L_1$ et $B = C_2 \vee \neg L_2$.

- **S'il existe un unificateur plus général** θ pour L_1 et L_2 (c-a-d, $L_1\theta = L_2\theta$) alors, la **résolution** entre A et B peut s'exprimer par :

$$C_1 \vee L_1, C_2 \vee \neg L_2 \text{ donne } (C_1 \vee C_2)\theta$$

  - *Exemple* :

    Soient $A = p(X) \vee q(X)$ et $B = \neg p(a) \vee r(X)$. En renommant d'abord X par Y dans B, le résolvant de A et B est $q(a) \vee r(Y)$ sous la substitution $\theta = \{X/a\}$.

Exemple de résolution

- Soit la **formule(**F1 ∧ F2 ∧ F3**)**→ C où :
  - F1 = ∀X ((e(X) ∧ ¬v(X)) → ∃Y(s(X, Y) ∧ c(Y))),
  - F2 = ∃X (p(X) ∧ e(X) ∧ ∀Y (s(X, Y) → p(Y))),
  - F3 = ∀X (p(X) → ¬v(X)),
  - C = ∃X (p(X) ∧ c(X)).

- **Les formules sous forme Prenex (FNP) + Skolem sont :**
  - S1 =(¬e(X) ∨ v(X) ∨ s(X,f(X))) ∧ (¬e(X) ∨ v(X) ∨ c(f(X))),
  - S2 = p(a) ∧ e(a) ∧ (¬s(a, Y) ∨ p(Y)),
  - S3 = (¬p(X) ∨ ¬v(X))
  - ¬C = (¬p(X) ∨ ¬c(X)).

Exemple de résolution

1. ¬e(X) v v(X) v s(X, f (X))
2. ¬e(X) v v(X) v c(f (X))
3. p(a)
4. e(a)
5. ¬s(a, Y) v p(Y)
6. ¬p(X) v ¬v(X)
7. ¬p(X) v ¬c(X)

8. ¬v(a) (3 et 6 par {X/a})
9. v(a) v c(f (a)) (2 et 4 par {X/a})
10. c(f (a)) (8 et 9)
11. v(a) v s(a, f (a)) (1 et 4 par {X/a})
12. s(a, f (a)) (8 et 11)
13. p(f (a)) (12 et 5 par {Y/f (a)})
14. ¬c(f (a)) (13 et 7 par {X/f (a)})
15. □ (10 et 14).

Terminaison

- Les procédures de preuve en logique des prédicats ne peuvent être que *semi-décidables* :
  - elles terminent en un **temps fini** si **une preuve existe** à la requête
  - elles peuvent **ne pas terminer** si la requête n'est **pas prouvable**

Prolog : La résolution SLD

- La programmation logique utilise une stratégie spécifique de résolution : la Sélection Linéaire Définie (SLD).
  - À chaque étape, une clause de l'ensemble de départ est utilisée

- La **SLD-resolution** (Sélectionné Linéaire Défini) est une procédure servant à prouver une formule de logique à partir d'un ensemble de clauses de Horn.

- Clause définie : clause de Horn contenant exactement un littéral positif

  ¬a v ¬b v c

  (a ∧ b) => c Trois manières de l'exprimer

  c :- a, b

  - **c** est la conclusion et **a** et **b** sont les prémisses ou antécédents

Clauses de Horn

- Une ***clause de Horn*** est une clause contenant *au plus un littéral positif*
  - p
  - ¬p
  - ¬p v ¬q v r ≡ (q ∧ r => p)

- Trois types de clauses de Horn
  - « **Faits** » : un atome (e.g. P) (en Prolog : p.)
  - « **Règles** » :
    - antécédent = conjonction de littéraux positifs
    - conclusion = un littéral positif
      - (e.g. (q ∧ r => p) ≡ p V ¬q V ¬r) (en Prolog : p :- q, r.)
  - « **Buts** » : ensemble de littéraux négatifs (en Prolog : :- q.)

Programme logique

- Programme = ensemble de clauses de Horn
- Requête = conjonction de littéraux positifs

1. Construction de la **négation de la requête** = clause de Horn négative (liste de buts)
2. Unification du **1er élément de la négation** avec **une clause** du programme
   - 1ère clause du programme dont la tête (conclusion) est la contrepartie positive de l'élément considéré
     - Si contradiction => succès et passage à l'élément suivant
     - Si fini par échouer, on cherche la clause suivante éligible

Exemple: Résolution-SLD

- Soit le **programme** :

  $q \vee \neg p$

  p

- Et la **requête** : $\{q\}$
- Liste de buts = négation de la requête : $\{\neg q\}$

  1. $\neg q$ peut s'unifier avec $\{q \vee \neg p\}$ donnant : $\neg p$
  2. qui peut s'unifier avec la $2^{ème}$ clause : $\{p\}$, donnant contradiction
  3. Donc la requête est prouvée

- ... est complète

  – Si BC $\models$ F, alors BC U $\neg$F possède une réfutation linéaire.

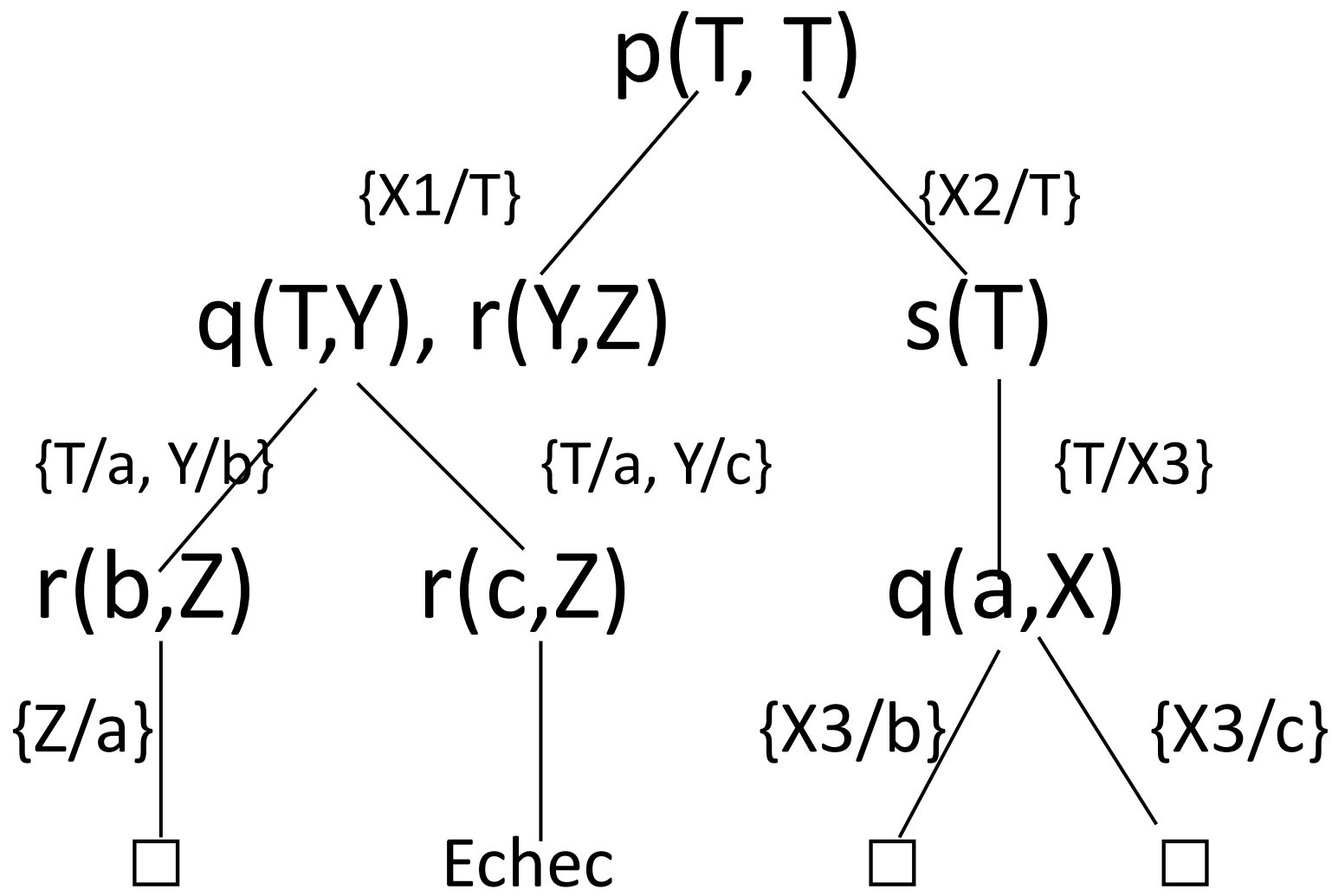
Exemple

1. p(X, X):- q(X,Y), r(X,Z).
2. p(X, X):- s(X).
3. q(a, b).
4. q(a, c).
5. r(b, a).
6. s(X):- q(a, X).
7. :- p(T, T).

Il est préférable de renommer les variables au niveau de chaque clause, d'où:

1. p(X1, X1):- q(X1,Y), r(X1,Z).
2. p(X2, X2):- s(X2).
3. q(a, b).
4. q(a, c).
5. r(b, a).
6. s(X3):- q(a, X3).
7. :- p(T, T).

Arbre SLD de la requête p(T,T)



Les solutions trouvées sont données (de gauche à droite):

T= a;

T= b;

T = c;

no

Opérateurs en Prolog

Dans Prolog, un utilisateur peut définir son propre opérateur, en utilisant le prédicat prdéfini op/3 :

op(Précédence, Associativité, Nom)

Déclare l'opérateur comme étant un Nom ayant un type d'associativité avec une priorité. L'opérateur peut également être une liste de noms, auquel cas tous les éléments de la liste sont déclarés comme étant des opérateurs identiques.

La priorité est un entier compris entre 0 et 1200. L'opérateur le plus prioritaire est celui possédant la plus petite valeur.

L'associativité correspond à l'un des types suivants: {xf , yf , xfx , xfy , yfx , fy , fx} où f indique la position du foncteur (opérateur) et x et y indiquent les positions des opérandes.

Préfixé : fx (unaire non associatif), fy(associatif)

Infixé : xfx (binaire non associatif), xfy (associatif à droite), yfx (associatif à gauche)

Postfixé: xf (unaire non associatif), yf(associatif).

Le tableau suivant résume des opérateurs prédéfinis.

Exemple: ?- op(150, xfy, et). True

« Et » est un nouvel opérateur binaire infixé de priorité 150 et associatif à droite.

Priorité	Type	Les opérateurs)	Utilisation
1200	xfx	:- -->	
1200	fx	:- ?-	Directive, requête
1100	xfy	;	
1050	xfy	->	
1000	xfy	','	
900	fy	\+	
700	xfx	= \\=	Unification du terme
700	xfx	== \\== @< @= < @> @>=	Comparaison de terme
700	xfx	=..	
700	xfx	is =:= =\= < > = < >=	Évaluation arithmétique et comparaison
600	xfy	:	Qualification de module
500	yfx	+ - /\ \/	
400	yfx	* / div mod // rem << >>	
200	xfx	**	Pouvoir flottant
200	xfy	^	Quantification variable, puissance entière
200	fy	+ - \	Identité arithmétique, négation; complément binaire

Opérateurs en Prolog (suite)

- Une expression arithmétique est évaluée en utilisant le prédicat prédéfini « is ».
- **Le prédicat « is /2» :** c'est un opérateur infixée non associatif.

  la requête « R is E » réussit si l'expression E est évaluée puis unifiée avec R. Ce prédicat sert généralement à affecter une valeur à une variable. Ainsi, la requête « X is 3+4*2 » réussit pour X = 11. Alors que la requête « X is Y+1 » produit une erreur puisque la valeur de Y est inconnue et par conséquent l'expression Y+1 ne peut être évaluée.

  La requête « 2+3 is 1+4 » échoue puisque l'évaluation de l'expression 1+4 donne 5, mais l'expression 2+3 ne peut s'unifier avec la constante 5.
- **Le prédicat d'unification « = » :** la requête « E1 = E2 » réussit si E1 et E2 sont deux expressions arithmétiques unifiables. Ce prédicat ne permet pas l'évaluation. Par exemple 2*3 = X réussit si X = 2*3, alors que 2*3=6 échoue parce que les deux termes ne sont pas unifiables. Le terme 2*3 est un terme composé (en fait il correspond au terme *(2,3)) alors que 6 est un entier. L'opération inverse est défini par le prédicat « \= » qui permet de tester si deux termes ne sont pas unifiables.

Exemples: Quelles sont les réponses prolog aux requêtes suivantes :

?- X = 2, Y = 5, Z = 9, T is X * X, Z = T + Y.

No

?- X = 2, Y = 5, Z = 9, T is X * X, Z = := T + Y.

X = 2, Y = 5, Z = 9, T = 4

yes

?- X = 2, Y = 5, Z = 9, T is X * X, Z is T + Y.

X = 2, Y = 5, Z = 9, T = 4.

yes

?- X +Y = 2+5, Z = 9, T = X * X, Z is T + Y.

X = 2, Y = 5, Z = 9, T = 2*2. yes

Coupure en Prolog « Cut »

La coupure est un prédicat prédéfini agissant sur le comportement de prolog lors du retour arrière (backtracking). On le nomme également le coupe-choix,

La syntaxe de la coupure en Prolog est définie par le prédicat prédéfini ‘ ! ’ qui réussit toujours. On introduit la coupure dans un programme logique aussi bien au niveau des règles qu’au niveau des requêtes (questions).

La sémantique de la coupure : elle permet de couper des branches de l’arbre SLD qui qui par exemple engendrent des solutions inutiles ou redondantes (répétées).

Soit le programme suivant:

(1) C :- C1, !, C2.

(2) C :- C3.

Si C1 est résolu (but C1 réussit), le cut coupe l’alternative restante pour C (dans la règle2), ainsi bien sûr que tous les autres choix possible pour la résolution de C1.

Si C1 n’est pas résolu (but C1 echec), le cut n’est pas atteint, le backtracking est possible sur la deuxième règle de C (règle2).

Le coupe-choix n’affecte pas les résolutions englobant la clause le contenant.

C :- C1, C2.

C1 :- D1.

C1 :- D2.

C2 :- C3, !, C4.

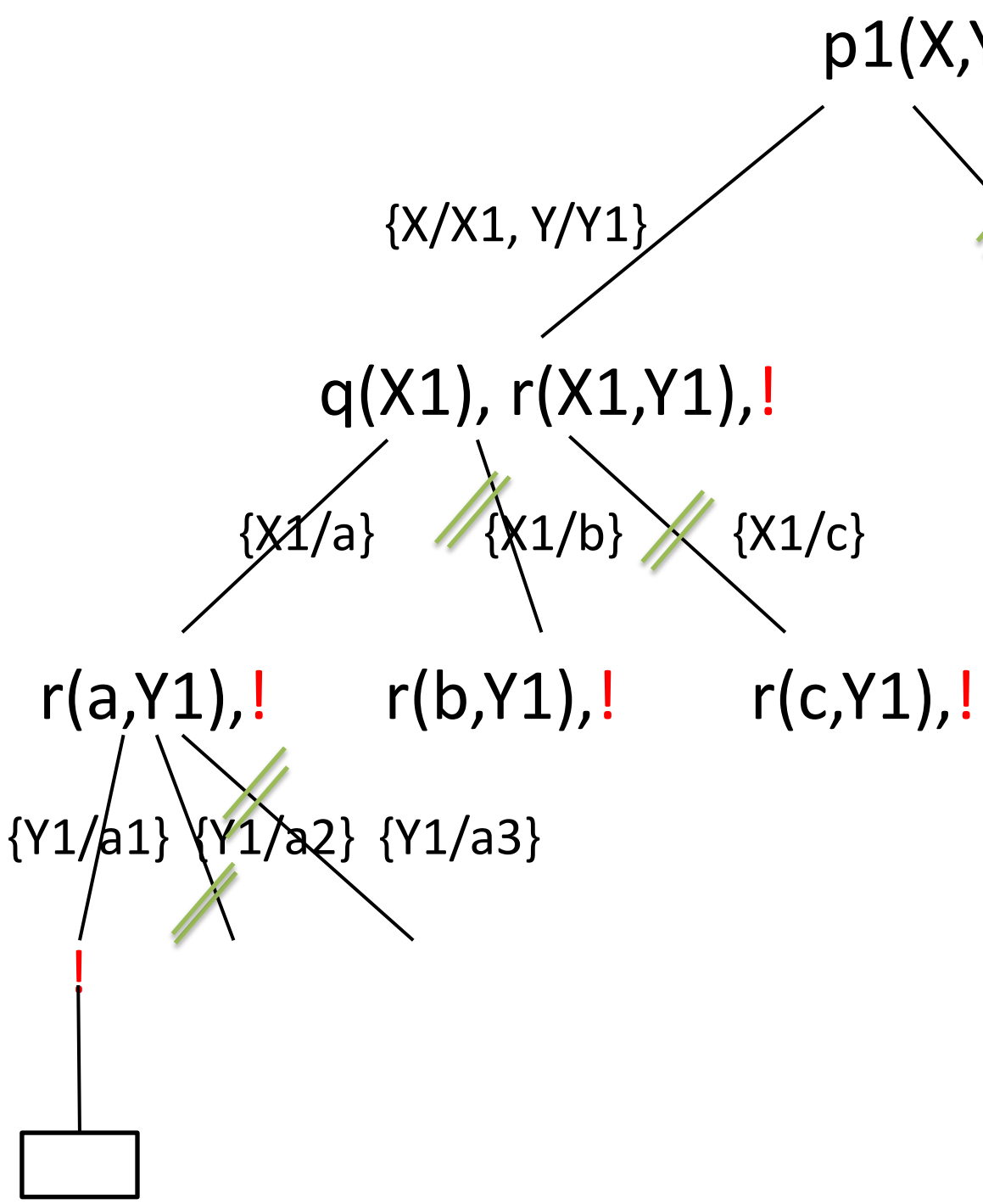
C2 :- D3.

le coupe choix empêche un backtracking sur C3 et la 2eme règle de C2 mais pas sur C1.

Coupure en Prolog : Exemple

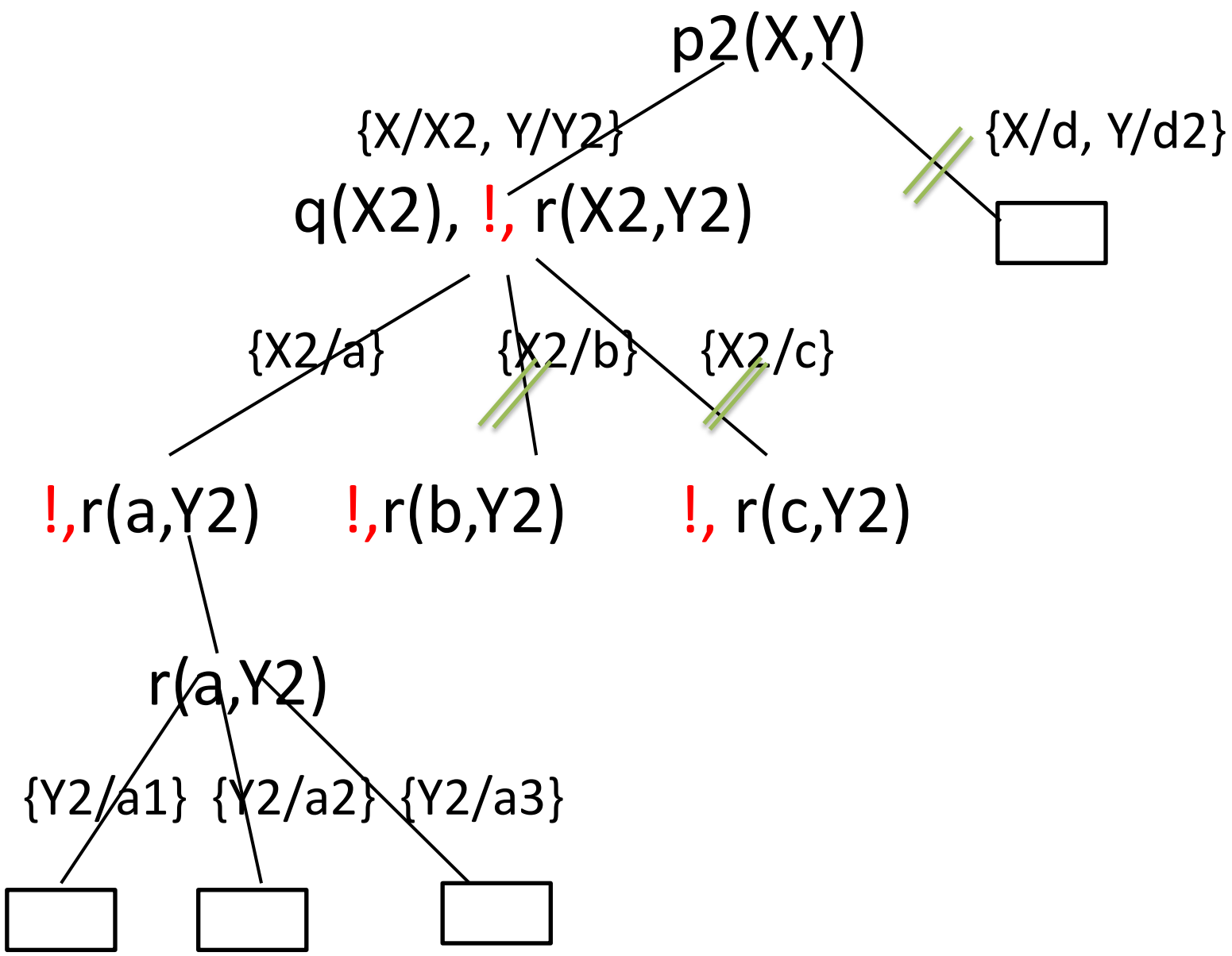
```
q(a).  q(b).  q(c).  r(a,a1).  r(a,a2).  r(a,a3).  r(b,b1).  r(c,c1).

p(X,Y) :- q(X), r(X,Y).
p(d,d1).
p1(X,Y) :- q(X), r(X,Y), !.
p1(d,d1).
p2(X,Y) :- q(X), !, r(X,Y).
p2(d,d1).
p3(X,Y) :- !, q(X), r(X,Y).
p3(d,d1).

?- p(X,Y).
X = a, Y = a1 ;  X = a, Y = a2;   X = a, Y = a3 ;   X = b,  Y = b1 ; X = c, Y = c1; X = d, Y = d1 ; no
?- p(d,Y).
Y = d1
?- p1(X,Y).
X = a, Y = a1 ; no
?- p1(d,Y).
Y = d1
?- p2(X,Y).
X = a, Y = a1; X = a, Y = a2; X = a, Y = a3; no
?- p2(d,Y).
 Y = d1
?- p3(X,Y).
X = a, Y = a1; X = a, Y = a2; X = a ,Y = a3; X = b, Y = b1; X = c, Y = c1; no
?- p3(d,Y).
 no
```

Coupure en Prolog : Arbre SLD



La solution est trouvée est : X = a, Y = a1

Coupure en Prolog : Arbre SLD



Les solutions trouvées sont :

X = a, Y = a1;

X = a, Y = a2;

X = a, Y = a3;

no