Fondements de l'Intelligence Artificielle

Hatem HADJ KACEM
Maître de conférences, FSEG-Sfax
Institut International de Technologie

2020-2021

Chapitre 1

Émergence de l'intelligence artificielle

Depuis la nuit des temps, l'imaginaire collectif semble peuplé de créatures intelligentes créées de toutes pièce par l'être humain. Ces contes semblent traduire un désir profond de l'homme de reproduire les mécanismes à la base de son intelligence. Outre ces visions fantasmatiques, l'histoire dénombre de multiples tentatives concrétisées, pour construire des machines ou des artefacts ou des instruments capables de se substituer à l'homme pour effectuer à sa place certaines tâches intellectuelles répétitives ou fastidieuses. L'apparition des ordinateurs a permis de cristalliser autour d'un ensemble de paradigmes, la compréhension et la reproduction des mécanismes de l'intelligence. L'intelligence artificielle est une continuité de cette pensé puisqu'il cherche à prolonger nos facultés dans des objets artificiels créés par nous même. Il est évident que l'ordinateur a constitué la première machine qui simule nos facultés de traitement de l'information.

Le but de ce cours est de vous faire appréhender les techniques utilisées pour résoudre des problèmes liés à l'analyse, au traitement et à l'apprentissage de connaissances. Cette problématique s'inscrit dans le cadre général des sciences cognitives, dont le soucis est l'étude de l'intelligence. Ces disciplines regroupent des domaines aussi variées telles que la philosophie, la psychologie, la biologie ainsi que des sciences pour l'ingénieur telles que l'informatique et la robotique. Depuis les premiers ordinateurs, il y a eu en effet une interaction forte entre les sciences qui essayaient de comprendre le fonctionnement de l'intelligence, et celles qui tâchaient d'en reproduire le fonctionnement.

Nous pouvons situer l'intelligence artificielle en tant que domaine qui nous amène à transmettre à nos machines des capacités de plus en plus intelligentes.

1.1 Définition de l'intelligence artificielle

Nous pouvons ranger les différentes définitions données par la plupart des chercheurs en deux approches :

- La première insiste davantage sur le caractère utilitaire de l'intelligence artificielle par le faite de faire faire à la machine ce que l'homme est capable de faire et rendre les ordinateurs plus utiles.
- La deuxième, au contraire, identifie l'intelligence artificielle comme l'étude des facultés mentales et la compréhension de la nature de l'intelligence en construisant des programmes imitant l'intelligence humaine.

Ces deux définitions s'accordent sur le fait que l'objectif de l'IA est de créer des systèmes intelligents, mais elles diffèrent significativement dans leur façon de définir l'intelligence. La première se

focalise sur le comportement du système, tandis que la deuxième considère que c'est le fonctionnement interne (le raisonnement) du système qui importe. Une deuxième distinction peut être faite entre celle qui définit l'intelligence à partir de l'être humain et celle qui ne fait pas référence aux humains mais à un standard de rationalité plus général. On peut donc décliner quatre façons de voir l'intelligence artificielle :

- *créer des systèmes qui se comportent comme les êtres humains* ? cette définition opérationelle de l'IA fut promue par Alan Turing, qui introduisit son fameux "test de Turing" selon lequel une machine est considérée comme intelligente si elle peut converser de telle manière que les interrogateurs (humains) ne peuvent la distinguer d'un être humain.
- *créer des systèmes qui pensent comme des êtres humains* ? si l'on adhère à cette deuxième définition, cela implique que l'IA est une science expérimentale, car il faut comprendre au préalable la façon dont pensent les humains (sinon, comment savoir si une machine pense comme un homme ?) et ensuite évaluer les systèmes par rapport à leurs similarités avec le raisonnement humain.
- *créer des systèmes qui pensent rationnellement* ? selon cette définition, les systèmes doivent raisonner d'une manière rationnelle, c'est à dire en suivant les lois de la logique. Cette approche peut être critiquée car il semble que certaines capacités (la perception, par exemple) ne sont pas facilement exprimables en logique. De plus, ce standard de rationalité ne peut pas être atteint en pratique car la technologie actuelle ne permet pas de réaliser des calculs aussi complexes.
- *créer des systèmes qui possèdent des comportements rationnels* ? cette dernière définition de l'IA concerne le développement des agents qui agissent pour mieux satisfaire leurs objectifs. On remarque que cette définition est plus générale que la précédente car raisonner logiquement peut être une façon d'agir rationnellement mais n'est pas la seule (par exemple, le réflexe de retirer sa main d'un objet brûlant est rationnel mais n'est pas le résultat d'une inférence logique).

Malgré que ces deux approches sont opposées, nous sommes convaincus aujourd'hui de la nécessité de l'une et de l'autre. En effet, pour rendre les ordinateurs plus utiles il faut les doter de comportements intelligents et s'inspirer du fonctionnement de l'intelligence humaine.

Il apparaît de plus en plus indispensable de simuler sur ordinateur le fonctionnement de l'intelligence humaine pour établir et vérifier les modèles et mettre au point les théories qui tentent d'en expliquer le fonctionnement sur le plan cognitif.

1.2 La naissance de l'Intelligence Artificielle

L'année 1956 peut être considère comme l'année de naissance de l'IA puisque c'est pendant cette année que nous avons vue s'exécute sur la machine le premier programme d'intelligence artificielle. Les auteurs de ce programme sont Newell, Shaw et Simon. Ce programme est capable de démontrer des théorèmes de la logique des propositions. Le terme et le domaine d'intelligence artificielle est proposé et accepté pendant cette même année par la communauté scientifique en tant que nouvelle science. A partir de cette date nous avons vue beaucoup de travaux qui se développent autour de ce nouveau thème : Jeux d'échec, démonstration de théorème, traitement du langage naturel, traduction automatique, etc.

Nous avons assisté aussi à la naissance des langages de programmation adaptés. Le langage LISP, le plus utilisé en intelligence artificielle, est créé en 1960 par John McCarthey.

1.3 Les travaux de recherche en IA

Deux grandes approches ont dominé les travaux en IA. La première utilise la ***puissance de calcul*** de l'ordinateur pour examiner le maximum de possibilité jusqu'à ce qu'une solution soit

atteinte. La deuxième s'inspire du comportement humain en utilisant un ensemble de connaissances et de méthodes générales susceptibles de donner souvent de bons résultats.
Au départ, on se sert de la première car la puissance de calcul des ordinateurs se développe rapidement et laisse croire que tout pourra être fait en augmentant simplement le nombre d'opérations possibles. Dès que on commence à s'attaquer à des problèmes où la puissance de l'ordinateur ne suffit plus, alors nous sommes dans une impasse.

- Le meilleur programme d'échec n'arrive pas à gagner les meilleurs joueurs humains.
- Les recherches en traduction automatique ne peuvent réussir uniquement en accumulant plus d'informations de type syntaxique.
- Les recherches en robotique se heurtent au problème de la vision qui nécessitent le traitement d'un nombre gigantesque d'informations.

Face à ce problème, il faut s'orienter vers l'observation plus attentive de l'être humain pour extraire des connaissances nécessaires et imiter leur comportement. On parle plutôt de systèmes experts. Nous avons vu la naissance de MYCIN, système expert dans le domaine de diagnostic médical. Dans ce même courant, on commence à construire des programmes qui parvenaient à comprendre le sens d'une phrase dans le cadre d'un sous-monde très restreint.
Ds 1970, on assiste à de nouvelles perces en :

- traduction automatique et en compréhension du langage naturel,
- développement de nombreux systèmes experts,
- progrès marqués en reconnaissances des formes,
- développement de la programmation logique et des modes de représentation des connaissances.

1.4 Traitement symbolique des connaissances

On peut distinguer trois phases de développement en informatique :

- l'ordinateur traite presque exclusivement l'information numérique,
- traitements alphanumériques tels qu'on les retrouve par exemple dans le fichiers base d'une base de donnes,
- avec l'intelligence artificielle, on franchit une nouvelle tape où les connaissances (faits, énoncés, équations, méthodes, etc) sont représentées par des systèmes de symboles qu'il s'agit de transformer.

méthodes d'intelligence artificielle	**méthodes classiques**
Prs du fonctionnement de l'être humain	Prs du fonctionnement de la machine
Traitement de symboles	Traitement de nombres ou de textes
Utilisent beaucoup d'infrences	Utilisent beaucoup de calculs
Font appels à des heuristiques et à des raisonnements incertains	Suivent des algorithmes rigides et exhaustifs
Sont gnralisables à des domaines compltement diffrents	Ne sont gnralisables qu'à une classe de problèmes semblables

1.5 Les domaines d'application de l'IA

L'IA recouvre donc des domaines fonctionnels déjà connus tels que les systèmes experts, la planification/optimisation ou la robotique. Mais L'IA couvre également de nouveaux domaines que sont le Machine Learning, le Natural Language Processing, la vision (capacité pour une machine d'appréhender son environnement) ou encore le Speech (texte vers parole ou parole vers texte).

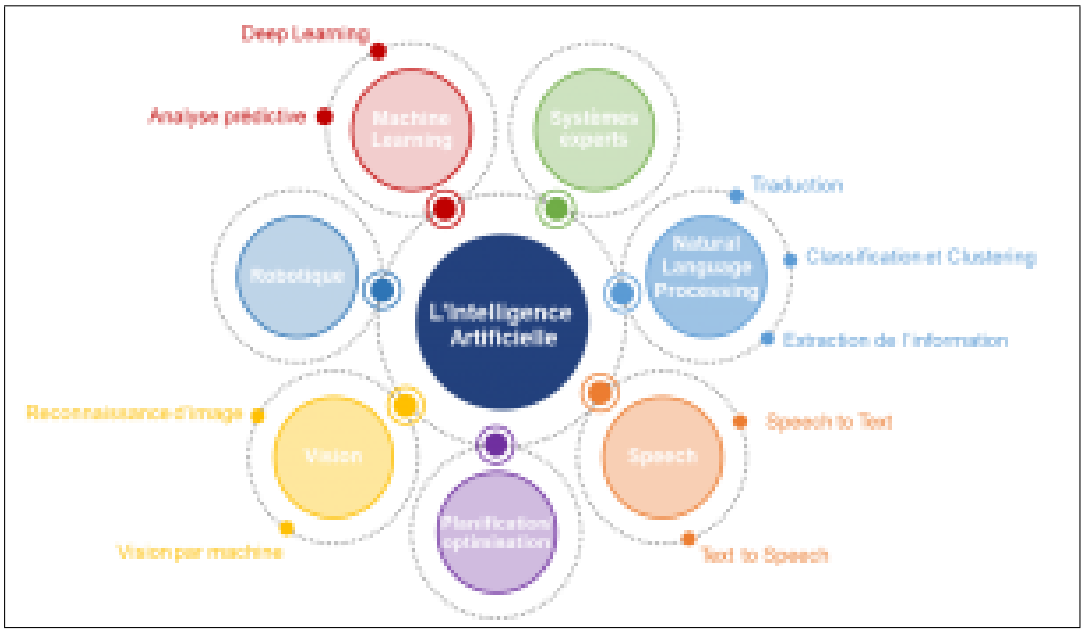


FIGURE 1.1 – Panorama des domaines de l'IA[artik-consulting]

Perception : Il s'agit de doter la machine de facultés de reconnaissance de la parole, de l'écriture ou de scènes visuelles (capteurs : vision par cellule photoélectrique). La perception met à disposition un ensemble de facultés intellectuelles complexes. Concernant la reconnaissance de la parole, IBM a mis en marche une interface auditive qui reconnaît des mots tirs du vocabulaire. Il s'agit de commander l'ordinateur oralement. **Compréhension du langage naturel** : Comprendre un texte est une machine très difficile à concevoir puisque comprendre un langage naturel, compte tenu des liens intimes entre la pense et le langage, c'est presque comprendre comment fonctionne la pense.
Déduction et jugement : La faculté de déduire des conséquences d'un ensemble d'énoncés est maintenant un problème largement résolu en IA, grâce aux travaux en logique mathématique. Il est toutefois à faire pour que l'ordinateur puisse analyser une situation complexe et faire preuve d'un jugement.
Résolution de problèmes, planification : Plusieurs méthodes générales de résolutions de problèmes ont été développées. Souvent on se sert des heuristiques qui sont des méthodes permettant de guider la recherche dans les solutions plausibles les plus favorables pour atteindre une solution.
Apprentissage : C'est faire faire apprendre à la machine des connaissances. D'où l'ide de créer des programmes qui génèrent leurs propres connaissances, en se modifiant à partir de leurs succès et leurs erreurs.
représentation des connaissances : Tous les domaines de l'intelligence artificielle impliquent que l'on construise et transforme des connaissances de divers types. La façon dont ces connaissances seront représentées dans l'ordinateur constitue certainement un point central critique pour l'évolution des systèmes artificiels intelligents.

Chapitre 2

Principe de résolution

le but de ce chapitre est d'introduire des algorithmes de recherche pour la résolution de problèmes. Nous commencerons par une définition formelle d'un problème (et d'une solution), et ensuite nous verrons un nombre d'exemples spécifiques pour illustrer cette définition et pour motiver la suite. Dans la deuxième partie du chapitre, nous détaillerons plusieurs algorithmes de recherche, d'abord les algorithmes de recherche dits non-informés ou aveugles, puis des algorithmes de recherche informés (ou algorithmes de recherche heuristiques).

2.1 La place de la résolution de problèmes en IA

Définition 1 *Un problème est une situation devant laquelle on ne voit pas directement les étapes permettant de dépasser la situation et satisfaire un besoin.*

Résoudre un problème, c'est :

- chercher un chemin qui permet d'aller d'une situation initiale à une situation finale pour atteindre un but.
- Recenser les états d'un système donné et trouver parmi ces états une ou plusieurs solutions.

Comment :

- Le passage d'un état à un autre se fait par application d'une action donnée.
- Développement d'un arbre de recherche à l'aide d'une stratégie de recherche.

2.1.1 Type de problème à étudier

Parmi les problèmes que l'on tente de faire résoudre par une machine sont :

- les jeux,
- la preuve de théorèmes.
- Génération de plan d'action.

Au départ, nous avons cru que les ordinateurs sont capables de résoudre ces problèmes plus vite que l'homme vue leurs capacités de mémorisation et la rapidité de traitement de l'information. Mais puisque le nombre de chemins à explorer pour arriver à une solution est gigantesque, la structure matérielle de la machine reste insuffisante. Dans la plupart des cas, le joueur se base sur des règles et des capacités stratégiques et non sur la puissance de calcul. Il s'agit de doter la machine de telles capacités.

La résolution de problème nécessite des techniques de l'IA fondées sur des démarches de nature intelligente.

2.1.2 Exemples de problèmes

Les exemples d'illustration dans la résolution de problèmes sont les jeux ou la prise de décision.

a) Le fermier, le loup, la chèvre et le chou
Ces quatre se trouvent sur la rive gauche d'une rivière. Un bateau transporte le fermier et un des trois autres. En asbsence du fermier, le loup mange la chèvre et la chèvre mange le chou. Comment faire pour passer de la rive gauche à la rive droite sans se faire manger les uns les autres.

b) Les récipients
Nous avons un récipient de 5L plein d'eau et un récipient de 2L vide. Il s'agit de verser de l'eau d'un récipient à un autre. Comment obtenir 1L dans le récipient de 2L.

c) Le taquin 3x3

Le taquin est une sorte de puzzle. Nous commençons avec une grille N*N de neuf cases où sont placées huit tuiles étiquetées par les nombres 1 à N-1, une des cases restant vide. Une tuile située à côté de la case vide peut être déplacée vers cette case.

1	5	8
3	7	4
⋆	6	2

L'objectif du jeu est d'arriver à obtenir une certaine configuration des tuiles dans la grille. Comment à partir d'une configuration initiale atteindre une autre finale ?

Le taquin est souvent utilisé pour tester les algorithmes de recherche. En augmentant la taille de la grille, les problèmes deviennent de plus en plus complexes. Les algorithmes d'aujourd'hui arrivent à résoudre les taquins 3*3 et 4*4, mais les instances du taquin 5*5 (avec un espace d'états de taille 10^{25}) restent difficiles.

d) Le voyageur de commerce
Un voyageur de commerce part d'une ville et veut se rendre dans un certain nombre de villes sans passer deux fois par la même ville et doit revenir à la ville de départ en fin de voyage. On connaît les distances entre les villes. Quelle est le trajet à faire en minimisant la distance parcourue.

e) Les tours de hanoï
Trois disques grand, moyen et petit enfilés sur une tige selon un ordre. On veut les déplacer sur une tige sachant un nombre d'opérations autorisées.

f) Le singe et les bananes
Un singe affamé veut se saisir d'une banane accessible uniquement en montant sur une caisse. Les opérations autorisées sont : se déplacer seule, se déplacer en poussant la caisse, grimper, attraper la banane.

g) Huit reines

L'objectif de ce jeu est de placer huit reines sur un échiquier (une grille 8*8) telle qu'aucune reine n'attaque une autre reine, c'est à dire qu'il n'y a pas deux reines sur la même colonne, la même ligne, ou sur la même diagonale.

2.1.3 Comment raisonner sur un problème

Il existe différentes méthodes de résolution de problèmes en IA. Les techniques utilisées dépendent de certains facteurs :

- **Optimisation** : trouver une solution facile et rapide sans se poser des questions sur son optimisation (voyageur de commerce).
- **Réversibilité** : c'est le fait d'annuler une action décidée en cours de recherche de la solution une fois qu'elle a été exécutée (taquin).
- **Prévision** : c'est prévoir exactement l'effet d'une action sur l'état courant du problème.
- **Décomposition** : c'est décomposer le problème en sous problèmes plus faciles à résoudre.
- **Connaissances** : c'est toutes les connaissances minimales requises pour résoudre le problème.

2.1.4 Les modes de représentation d'un cheminement de résolution de problème

L'objectif de cette partie est d'introduire les principes de la résolution d'un problème donné en utilisant un espace de recherche. Cet espace de recherche permet de recenser les états d'un système donné et de trouver parmi ces états une ou plusieurs solutions. Le passage d'un état à un autre se fait par l'application d'une action donnée. Nous verrons rapidement que le problème de recherche de l'état solution dans l'espace nous ramènera à développer un arbre de recherche et à définir une stratégie de recherche sur cet arbre. Le but de la recherche dans un tel arbre serait la diminution du temps de recherche en trouvant une stratégie qui converge rapidement vers la solution.

a) Le graphe des états

Définition 2 *L'espace des états successifs d'un problème à résoudre est réduit sous forme d'un graphe dont un nœud est un état du problème et un arc est une transition pour passer d'un état à un autre.*

b) Le graphe ET/OU (graphe de réduction de problèmes)

Définition 3 *Dans une représentation de type ET/OU, le problème est devisé en sous problèmes plus facile.*

2.1.5 Représentation en espaces d'états

La représentation en espace d'états met en œuvre deux types d'entités :
- État : structures de données donnant la spécification d'une situation du problème dans son évolution vers la solution (si elle existe).
- Opérateur : moyen de faire passer le problème d'un état vers un autre. L'ensemble des états initiaux, intermédiaires et finaux d'un problème défini dans une représentation d'espace est appelée *Espace des états.*

Une reprsentation en espace d'états d'un problème est entièrement défini par la donnée de :
- l'ensemble des états initiaux, soit S cet ensemble.
- L'ensemble des états finaux, soit G cet ensemble.
- L'ensemble des opérateurs, soit O cet ensemble.

Trouver une solution c'est donner la liste d'opérateurs qui permettent de passer d'un état initial vers un état final.

Exemple 1 (Jeu de taquin)
Soit l'état S suivant :

1	2	3
8	⋆	4
7	6	5

Soit l'ensemble $G = \{$ensemble des états dont la somme des chiffres des diagonales est égale à 6 avec la case vide est au milieu$\}$. Les solutions possibles sont les suivantes :

1	3	2
8	⋆	6
4	7	5

2	3	1
8	⋆	6
5	7	4

L'ensemble $O = \{H, B, G, D\}$. C'est à dire déplacer la case vide d'une position vers le haut (H), vers le bas (B), à droite (D) ou à gauche (G).

On définit l'espace d'états par :

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

où $x_i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $\forall i \in \{1, 2, \ldots, 9\}$ et $x_i \neq x_j$ $\forall i \neq j$.

Définition 4 *Un espace d'états est l'ensemble des états initiaux + les états finaux + les états générés par application d'une suite d'opérateurs à partir de l'état initial.*

Exemple 2 (Voyageur de commerce)
Étant données un ensemble de 4 villes à visiter A, B, C et D. Il s'agit de trouver un itinéraire débutant en A et se terminant par A après avoir visité toutes les autres villes une seule fois.

	A	B	C	D
A		4	6	10
B			7	10
C				5
D				

Définir la structure de donnée représentant un état. Définir les ensembles S, G et O.
Structure : Une liste (ordonnée) des villes visitées : $(x_1, x_2, \ldots, x_n)/x_i \in \{A, B, C, D\}$

$$
\begin{aligned}
S =& \{(A)\} \\
G =& \{(A, x_1, x_2, \ldots, x_n, A)\}/x_i \in \{B, C, D\}, x_i \neq x_j \forall i \neq j \\
=& \{(A, B, C, D, A), (A, B, D, C), (A, C, B, D, A), (A, C, D, B, A) \\
=& (A, D, B, C, A), (A, D, C, B, A)\} \\
O =& \{O_1\}
\end{aligned}
$$

O_1 *Aller* $(E_D, E_A) : E_D \rightarrow E_A / E_A = E_D + (x_j)$

Remarque 1 *En progressant dans le graphe nous avons une explosion combinatoire du nombre de nœud.*

La représentation d'un problème sous forme d'espace d'états revient à modéliser ce problème par un triplet (S_0, G, O) :

- S_0 : L'état initial.
- G : L'état final.
- O : L'ensemble des actions permettant de passer de l'état initial vers l'état final.

Un arbre de recherche est construit :

- La racine de l'arbre : l'état initial.
- Un état E_{ij} est fils d'un autre état E_i, s'il existe une action qui permet d'obtenir E_{ij} à partir de E_i.
- Si une des feuilles correspond à un état final, la solution est trouvée.

2.2 Algorithme de recherche

La recherche est une partie essentielle de l'intelligence artificielle, et probablement celle la plus fréquemment utilisée par les programmes informatiques en général. Tout programme devant effectuer un choix, devra obligatoirement évaluer chacune des possibilités, afin de déterminer celle qui sera la plus adaptée comme solution au problème. Un problème pourrait être défini selon les caractéristiques suivantes :

— État Initial : État dans lequel le problème doit être résolu.
— État Final : État dans lequel le problème a été résolu.

Les algorithmes de recherche sont un mécanisme de résolution général qui :

— se déroule dans un espace appelé espace d'états.
— Explore systématiquement toutes les alternatives.
— Trouve la séquence d'étapes menant à la solution.

Ces algorithmes suivent à peu près le même schéma : nous commençons toujours dans l'état initial et puis nous exécutons les étapes suivantes en boucle jusqu'à terminaison :

- s'il n'y a plus d'états à traiter, renvoyez échec.
- sinon, choisir un des états à traiter [1]
- si l'état est un état but, renvoyez la solution correspondante.
- sinon, supprimer cet état de l'ensemble des états à traiter, et le remplacer par ses états successeurs.

Dès lors que l'on cherche à résoudre un problème, il est nécessaire de représenter de manière abstraite le problème, de telle sorte que si plusieurs manières de résoudre le problème existent, la même représentation peut être réutilisée pour l'évaluation. La méthode la plus adaptée à ce type de représentation reste la représentation en arbre de recherche. D'une manière simple, l'arbre se constitue d'un nombre variant de nœuds (qui peuvent être nommés) liés les uns aux autres. L'arbre présente un nœud racine à sa base, qui représente l'état initial du problème. Il existe seulement un seul chemin possible pour aller d'un nœud à un autre.

2.2.1 Recherche en largeur

C'est l'un des algorithmes les plus simples pour effectuer une recherche et il se distingue dans sa manière de parcourir les nœuds pour traverser l'arbre. Au final, tous les nœuds seront parcourus quand on utilise cet algorithme qui fonctionne comme suit : La recherche commence au nœud parent, et explore tous les nœuds successeurs. Ensuite, pour chacun de ces nœuds, la recherche parcourt les nœuds enfants inexplorés, et ce jusqu'à ce que l'état final soit atteint.

Il s'agit de développer de façon prioritaire les nœuds situés au même niveau de profondeur.

- Visiter les états en parcourant l'arbre niveau par niveau.
- On utilise une file d'attente.

1. Ce qui va différencier les différents algorithmes est la manière dont on effectue le choix à cette étape

- La recherche s'arrête quand un état final est trouvé ou bien une profondeur maximale est atteint.

L'algorithme de recherche en largeur nécessite l'utilisation d'une file d'attente. La file d'attente est simplement une liste, à laquelle les nœuds sont retirés du début pour être placés à la fin. Initialement, la liste ne contient que le nœud racine (état initial). La recherche se déroule de manière répétitive, en retirant le premier nœud de la file d'attente, puis tente de trouver les successeurs du nœud, en les plaçant enfin à la fin de la file d'attente. Ce traitement continue jusqu'à ce que le premier nœud de la file d'attente soit le nœud de l'état final, dans ce cas l'algorithme signale que la recherche s'est déroulée avec succès, ou bien que la file d'attente soit vide, alors l'algorithme signale que la recherche a échoué.

Algorithme *Recherche en Largeur*

OUVERT : liste des nœuds apparus ou mis en évidence.
FERME : liste des nœuds développés (pour lesquels nous avons mis en évidence les descendants éventuels).

Début
01. Mettre le nœud initial dans OUVERT (initialement vide)
02. **Tant que** (OUVERT $\neq \emptyset$ et BUT non atteint) **Faire**
03. **Si** (la tête de OUVERT est le but) **alors**
04. ne rien faire
05. **Sinon**
06. enlever la tête de OUVERT
07. la mettre dans FERME
08. ajouter ses descendants en QUEUE de OUVERT
09. **Fin_si**
10. **Fin_faire**
11. **Si** (le BUT est atteint) **alors**
12. succès
13. **Sinon**
14. échec
15. **Fin_si**
Fin.

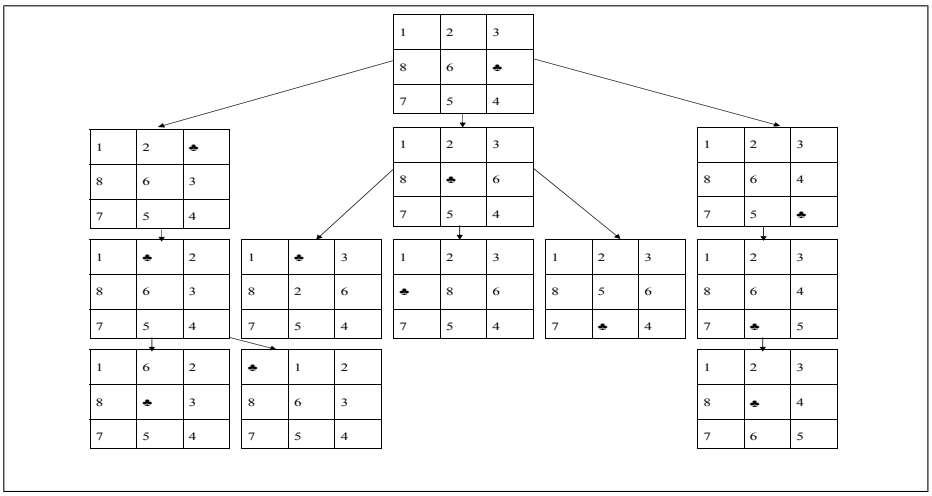
Avantages :

- Trouver une solution si elle existe.
- Si l'arbre possède un nombre fini de branche alors la recherche est dite algorithmique.

Inconvénients :

- Gourmande en espace mémoire.
- Nombre de nœuds à mémoriser par niveau. crois exponontiellement
- Très peu adapter à une solution admettant. plusieurs chemins de solution et la solution se trouve très loin.

Exemple 3 (Jeu de taquin)



2.2.2 Recherche en profondeur

L'algorithme de recherche en profondeur est relativement similaire et aussi simple que la recherche en largeur, à la différence qu'il se distingue dans la manière dont il itère à travers les nœuds pour effectuer la recherche. La majeure différence est qu'il va utiliser le concept de pile au lieu d'une file d'attente, de telle sorte que les nouveaux nœuds seront ajoutés au début de la liste au lieu de la fin. Exactement comme l'algorithme vu précédemment, la recherche parcourt tous les nœuds jusqu'à ce l'état final soit atteint.
Dans la recherche en profondeur la priorité est donnée aux sommets des niveaux les plus profonds du graphe de recherche. Chaque sommet choisi pour être exploré voit tous ses successeurs générés avant qu'un autre sommet ne soit exploré. Dans la liste OUVERTS le sommet le plus profond est celui le plus récemment généré. Cet ensemble aura donc une structure de pile.

L'algorithme peut s'écrire de la manière suivante :

Algorithme *Recherche en Profondeur*

Entrée :
Sortie :
Début
01. Mettre dans OUVERT le nœud de départ
02. **Tant que** (OUVERT $\neq \emptyset$ et le nœud BUT non atteint) **Faire**
03. **Si** (le premier nœud est un nœud BUT) **alors**
04. ne rien faire
05. **Sinon**
06. enlever le premier nœud de OUVERT
07. le mettre dans FERME
08. ajouter les descendants de nœud en tête de OUVERT
09. **Fin_si**
10. **Fin_faire**
11. **Si** (le BUT est atteint) **alors**
12. succès
13. **Sinon**
14. échec

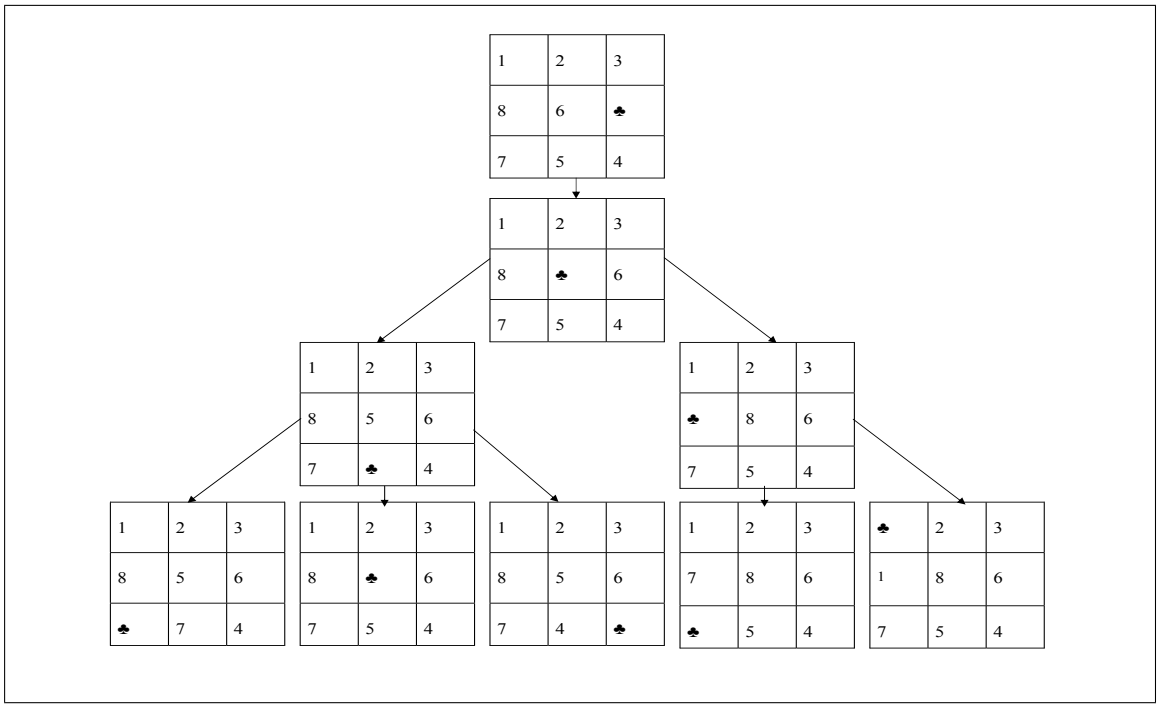
15. **Fin_si**
Fin.

Inconvénients :

— Le parcours en profondeur n'est pas complet parce que l'algorithme peut continuer sur un chemin infini, ignorant complètement un état but qui se trouve sur un autre chemin. Si par contre, nous n'avons qu'un nombre fini de chemins possibles (ce qui n'est pas souvent le cas), le parcours en profondeur sera complet.

— L'algorithme n'est pas optimal : il n'y a rien qui garantie que le premier état but trouvé sera le bon.

Exemple 4 (Jeu de taquin avec une profondeur limitée =3)



2.2.3 Recherche en profondeur limitée

Les graphes de recherche des problèmes qu'on cherche à résoudre peuvent être de très grande taille (voire infinie). Une recherche en profondeur peut se révéler dangereuse : l'algorithme risque de développer une branche infinie stérile sans que le mécanisme de retour en arrière puisse se déclencher. On ajoute dans ce cas une limite de profondeur. On a maintenant deux possibilités pour le retour arrière :

— La limite de profondeur est dépassée.

— Un sommet est reconnu comme une impasse.

La profondeur d'un nœud est définie par :

- profondeur du nœud est égale à zéro
- profondeur du nœud fils est égale au profondeur du nœud père plus un

Le parcours en profondeur limitée est complet si la profondeur maximale est supérieure à la profondeur minimale des solutions. Mais comme nous ne savons pas en général quelle profondeur sera suffisante, nous ne saurons pas si l'algorithme est complet ou non pour une profondeur donnée. Comme c'était le cas pour le parcours en profondeur classique, le parcours en profondeur limitée n'est pas optimal en général.

2.2.4 Recherche en profondeur limitée itérative

Une des difficultés d'utilisation du programme de recherche en profondeur avec limite sur la profondeur de recherche est de déterminer une limite adéquate. Si nous choisissons une limite très petite nous risquons de ne trouver aucune solution et si nous choisissons une limite trop élevée la complexité va devenir inacceptable. Le parcours en profondeur itérée permet de remédier (de façon un peu brutale !) à cet inconvénient : comme nous ne savons pas quelle borne de profondeur choisir, nous allons les essayer les unes après les autres. Nous effectuons donc un parcours en profondeur limitée avec une borne de 1, puis un parcours de profondeur avec une borne de 2, et nous continuons à augmenter la borne jusqu'à ce que l'on trouve une solution.
Comme le parcours en largeur, le parcours en profondeur itérée est complet à condition que le nombre de successeurs soit toujours fini.

2.3 Introduction à la recherche heuristique

Les algorithmes détaillés précédemment retournaient une solution (ou pas) en parcourant la totalité de l'arbre de recherche, mais il est parfois possible d'éviter de parcourir tous les nœuds de l'arbre, surtout quand celui-ci peut atteindre une profondeur et complexité énorme. Par exemple, si l'on tente de construire un arbre avec une partie d'échec, même d'un point de vue traitement informatique, il semble impossible d'évaluer un arbre d'une telle complexité. C'est pourquoi les algorithmes utilisés pour ce type de recherches auront une influence critique sur les performances de résolution du problème. Le but étant alors de ne pas essayer tous les nœuds possibles de l'arbre, mais plutôt de détecter les nœuds qui semblent potentiellement meilleurs que les autres et de se concentrer sur ces nœuds là pour la suite de la recherche. Cela s'appelle la recherche heuristique.

Définition 5 *L'heuristique introduit des stratégies de contrôle de la recherche.*

Une des exigences fondamentales de la recherche heuristique est qu'elle nécessite une fonction d'évaluation f qui détermine l'ordre dans lequel les nœuds sont traités : la liste de nœuds à traiter est organisée en fonction des f-valeurs des nœuds, avec les nœuds de plus petite valeur en tête de liste. Cela implique bien évidemment que la précision de la recherche dépendra considérablement de la puissance et précision de la fonction d'évaluation, et même que certaines erreurs d'évaluation peuvent se produire. Mais, lors de l'utilisation d'un tel algorithme, on part du principe qu'il vaut mieux prendre le risque de faire des erreurs d'évaluation, que de prendre un temps considérable à parcourir tous les nœuds afin de déterminer le meilleur. A priori, il n'y a pas vraiment de restriction sur la nature de la fonction d'évaluation, mais souvent elle a comme composante une fonction heuristique h où :

$$h(n) = \text{coût estimé du chemin de moindre coût reliant n 'a un état but}$$

Notons que la fonction heuristique prend un nœud en entrée mais sa valeur ne dépend que de l'état associé au nœud. Et bien sûr, $h(n) = 0$ si n est un état but. Ce rapport s'intéressera notamment aux types de recherches heuristiques suivants :

— Hill Climbing.
— Best First Search.

— A*.

Il s'agit d'utiliser une fonction d'évaluation pour chiffrer la validité d'un nœud par rapport à l'autre. à chaque étape, les nœuds sont réordonnés au lieu d'être mis dans une pile ou une file. On parle dans ce cas d'une recherche *ordonnée*.

Définition 6 *La recherche ordonnée revient à choisir de développer le nœud ayant les meilleures chances de mener au but.*

2.3.1 Recherche Hill Climbing

L'algorithme de recherche Hill Climbing se base sur le principe de choisir le nœud suivant comme étant le nœud le plus proche de la solution. L'idée est relativement triviale et peut être illustrée par le fait de choisir le chemin le plus court pour graver une colline.

Algorithme *Recherche Hill Climbing*

Début
01. Mettre dans État-courant le nœud de départ
02. **Tant que** (État-courant≠ État-final) **Faire**
03. **Si** (le premier nœud est un nœud BUT) **alors**
04. ne rien faire
05. **Sinon**
06. récupérer les successeurs de État-courant
07. assigner une valeur à chaque successeur grâce à la fonction d'évaluation f
08. Mettre dans État-courant le nœud ayant la plus forte valeur selon f
09. **Fin_si**
10. **Fin_faire**
11. **Si** (le BUT est atteint) **alors**
12. succès
13. **Sinon**
14. échec
15. **Fin_si**
Fin.

2.3.2 Recherche Best-first

Cet algorithme est une version hybride entre les recherches en largeur et Hill Climbing. Exactement comme dans la recherche Hill Climbing, il utilise une fonction d'évaluation afin de déterminer le nœud suivant comme étant le nœud avec la meilleure valeur, mais il va également essayer tous les chemins possibles parmi une liste de nœuds à explorer (de manière similaire aux recherches en largeur et en profondeur. la stratégie employée par la recherche best-first consiste à utiliser la fonction heuristique h comme fonction d'évaluation (c'est-à-dire qu'on prend $f(n) = h(n)$).

Algorithme *Recherche Best-first*

Début

01. Mettre dans OUVERT le nœud de départ
02. **Tant que** (OUVERT $\neq \emptyset$ et le nœud BUT non atteint) **Faire**
03. **Si** (le premier nœud est un nœud BUT) **alors**
04. ne rien faire
05. **Sinon**
06. récupérer les successeurs du premier nœud de OUVERT
06. enlever le premier nœud de OUVERT et le mettre dans FERME
07. assigner une valeur à chaque successeur grâce à la fonction d'évaluation f
08. ajouter ces successeurs à OUVERT en ordre croissant selon f
09. **Fin_si**
10. **Fin_faire**
11. **Si** (le BUT est atteint) **alors**
12. succès
13. **Sinon**
14. échec
15. **Fin_si**
Fin.

2.3.3 Algorithme A^*

Les algorithmes décrits précédemment parviennent à retourner une solution de manière relativement raisonnable, mais ne prennent pas en compte si la solution est la meilleure solution ou juste une solution parmi d'autres. La recherche best-first, par exemple, donne la préférence aux nœuds dont les états semblent les plus proche d'un état but, mais il ne prend pas en compte les coûts des chemins reliant l'état initial à ces nœuds. Néanmoins, c'est une information très pertinente car le coût d'un chemin passant par un nœud n est la somme du coût de chemin entre l'état initial et n et le coût du chemin reliant n à un état but. C'est cette idée qui est à la base de la recherche $A^\star$. Si nous appelons $g(n)$ le coût du chemin entre l'état initial et n, la fonction d'évaluation utilisée par la recherche $A^\star$ est donnée par la formule suivante :

$$f(n) = g(n) + h(n)$$

Comme $g(n)$ est le coût réel associé au chemin entre l'état initial et n et que $h(n)$ est une estimation du coût du chemin entre n et un état but, la fonction d'évaluation f donne une estimation du coût de la meilleure solution passant par le nœud n.
L'algorithme $A^\star$ apporte plus de précision à la recherche dans le sens où il accorde une importance à la distance du chemin parcouru au moment de déterminer le nœud suivant à parcourir.

Le principe consiste à définir un ordre total sur tous les choix à un moment de la recherche en utilisant une fonction coût.
Soit $f(n)$ le coût idéal du chemin passant par un nœud n pour arriver au but : $f(n) = g(n) + h(n)$ avec $g(n)$ est la fonction coût réel et $h(n)$ est la fonction heuristique.

Algorithme A^*

Début
01. OUVERT$\leftarrow U_0$
02. FERME$\leftarrow \emptyset$
03. $g(U_0) \leftarrow 0$
04. $U \leftarrow U_0$

05. **Tant que** (OUVERT $\neq \emptyset$ et U $\notin$T) **Faire**
06. Supprimer $\hat{U}$ de OUVERT
07. le placer dans FERME
08. itérer sur $\{ v \in$ Successeur $(\hat{U})\}$
09. **Si** ($v \notin$ {OUVERT ou FERME} ou ($g(v) > g(\hat{U})+k(\hat{U}, v)$)) **alors**
10. $g(v) \leftarrow g(\hat{U})+k(\hat{U},v)$
11. $f(v) \leftarrow g(v) + h(v)$
12. père $(v) \leftarrow \hat{U}$
13. ranger OUVERT dans l'ordre croissant de f et décroissant de g
14. **Fin_si**
15. **Si** (OUVERT $\neq \emptyset$) **alors**
16. $\hat{U}\leftarrow$ tête de OUVERT
17. **Fin_si**
18. **Fin_faire**
19. **Si** (OUVERT $= \emptyset$) **alors**
20. pas de solution
21. **Sinon**
22. la solution est fournie par le chemin créé par père($\hat{U}$)
23. **Fin_si**
Fin.

Exemple 5 (Jeu de taquin)

Appliquons A^* pour le jeu de taquin avec g étant le nombre de jetons déplacés et h le nombre de jeton mal placés.

2.4 La réduction du problème en sous problèmes

2.4.1 La représentation en graphe ET/OU

Définition 7 *Une représentation du problème en sous-problèmes consiste à découper le problème en sous-problèmes élémentaires ou composés.*

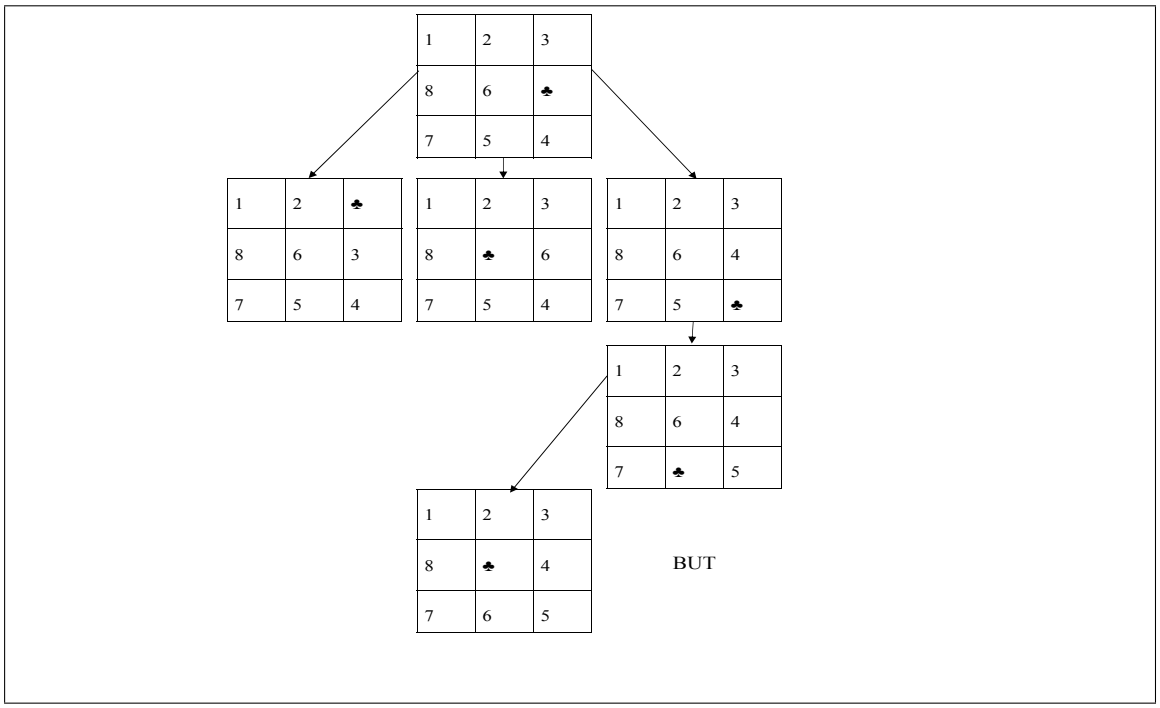
Un problème est dit élémentaire s'il admet une solution immédiate. Les transformations qui permettent de réaliser la décomposition sont appelées des opérateurs.

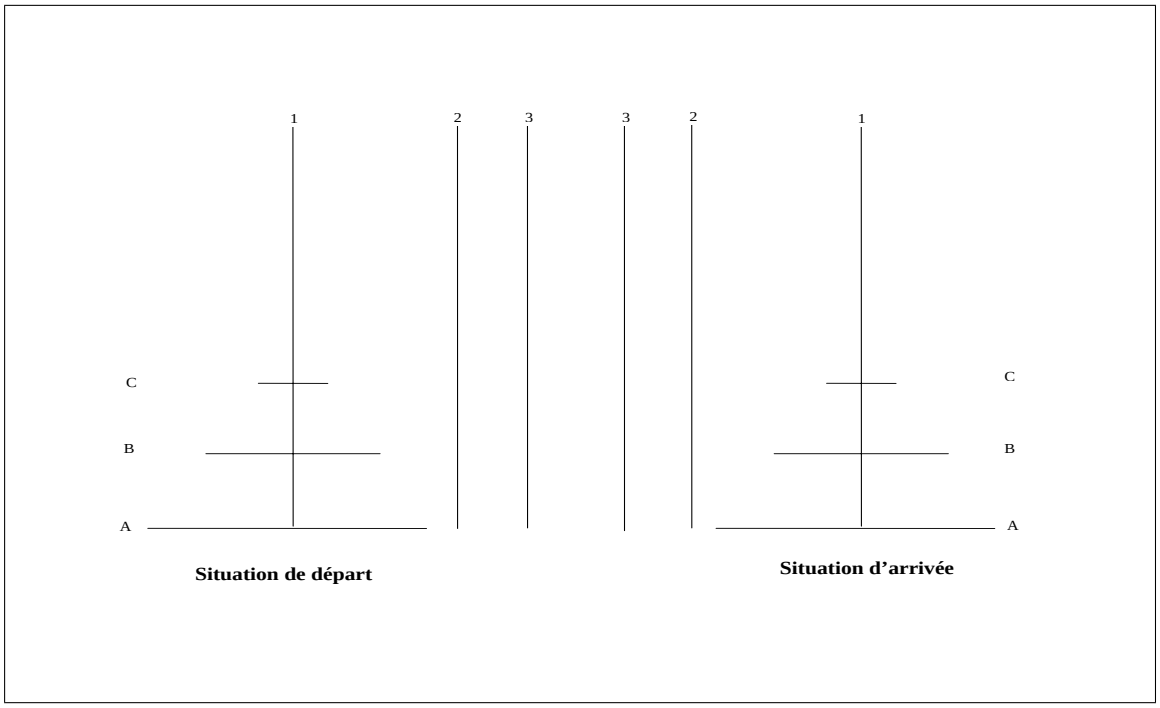
Une représentation par réduction de problèmes est définie par les éléments suivants :

1. une description du problème à résoudre.
2. L'ensemble des opérateurs de réduction.
3. Une description des problèmes primitifs ou élémentaires.

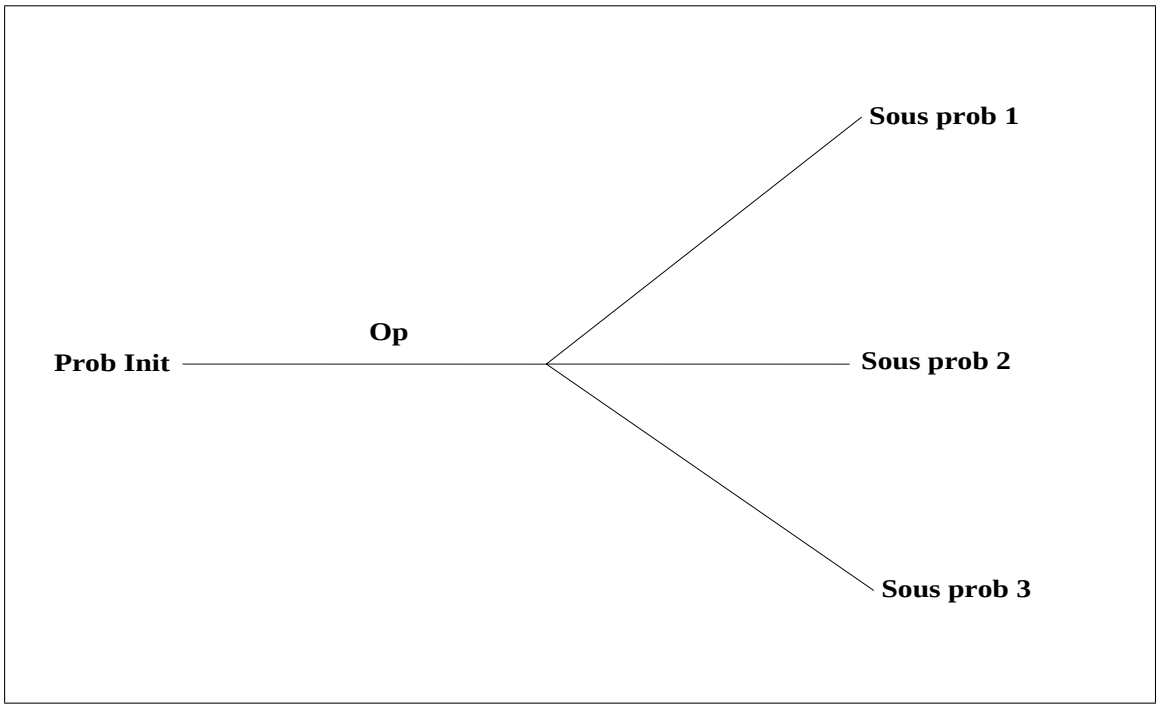
Exemple 6 (Tour de Hanoy à 3 quilles)

Contraintes :
On ne peut déplacer qu'un seul disque à la fois.
On ne peut pas placer un disque sur un disque de taille inférieure.





Nous pouvons représenter la situation de départ : ((A,B,C),(),()) et la situation d'arrivée : ((),(),(A,B,C))



Problème primitif : une description dans laquelle il s'agit de passer d'une situation de départ à une situation désirée moyennant le déplacement d'un seul disque d'une quille i vers la quille j ne contenant pas de disque de taille inférieure.
Un seul opérateur de décomposition suffit et il est défini comme suit : étant donné les quilles i, j, k, le problème de déplacer une pile de la quille i à la quille k peut être remplacé par les trois problèmes suivants :

- déplacer une pile de la quille i vers la quille j.
- Déplacer une pile de la quille i vers la quille k.
- Déplacer une pile de la quille j vers la quille k.

2.4.2 Technique de recherche d'une solution

Il s'agit de parcourir les arbres (graphes ET/OU) en prenant tous les nœuds à l'alternative ET et un seul nœud à l'alternative OU.

En ce qui concerne les techniques de recherche aveugles, ils se rapprochent beaucoup de ceux déjà vue de même pour les techniques heuristiques. Soit AO^* l'algorithme de recherche équivalent à A^* mais dans le cadre d'une représentation en graphe ET/OU.

2.4.3 Lien avec l'espace des états

La différence entre les deux représentations traduit deux modes de raisonnement par rapport à un énoncé. Dans le premier cas, on parle d'opérateurs de transformation d'un état à un autre. Dans le deuxième cas, il s'agit d'opérateurs de décomposition du problème en sous problèmes.

Il faut noter qu'un même problème peut être résolu par les deux méthodes. Le choix de la méthode devient un problème en soit qui dépend de la :

- nature du problème,

- définition des opérateurs,
- fonction de décomposition,
- etc.

Cependant on peut être guidé dans le choix par les considérations suivantes :

- Si la solution du problème se représente de façon pratique comme une structure d'arbre (ou de graphe) alors l'approche réduction de problèmes est plus appropriée.
- Si la solution du problème se représente de façon pratique comme un chemin représentant une suite inconditionnelle d'actions alors l'approche espaces d'états est plus appropriée.

Chapitre 3

Les Systèmes Experts

Les systèmes d'intelligence artificielle ayant pour objectif de traiter les aspects de raisonnement de l'être humain qui reposent sur des connaissances nombreuses et d'atteindre des performances équivalentes à celle des experts dans des domaines limités sont appelés des systèmes à base de connaissances. Parmi ceux ci on parle souvent de systèmes experts. Ce concept est basé sur une dualité *connaissance* et *raisonnement*.
Quelques systèmes experts sont déjà utilisés quotidiennement, principalement aux Etats-Unis. C'est le domaine médical qui est l'objet du plus grand nombre d'applications. L'un des premiers systèmes experts, et peut-être le plus connu, *MYCIN*, développé à l'université Stanford en 1977 par E.H. Shortliffe, fournit une aide au diagnostic et à la décision thérapeutique dans le domaine des infections bactériennes.

3.1 Qu'est ce qu'un Système Expert (SE)

Les systèmes experts sont des outils de l'intelligence artificielle c'est à dire qu'on ne les utilise que lorsque aucune méthode algorithmique exacte n'est disponible ou praticable.
Selon Edward FEIGENBAUM : *... des programmes conçus pour raisonner habillement à propos de tâches dont on pense qu'elles requièrent une expertise humaine considérable ....*

Ce que l'on retrouve aujourd'hui dans tous les SE est l'expression d'un savoir sous forme de connaissances heuristiques, résultant généralement d'une expérience cumulée dans un domaine précis. Ces connaissances traduisent un mode de raisonnement et peuvent s'exprimer par des règles telles que :"SI telle condition est vérifiée ALORS effectuer telle action" (ces règles sont appelées plus précisément "règles de production").

Un SE est un programme qui permet l'exploitation des connaissances dans un domaine précis et rigoureusement limité. Il est utilisé pour effectuer des tâches intellectuelles, c'est-à-dire des travaux exigeant le savoir et l'expérience de l'homme. Un SE est alors capable d'assister l'utilisateur de manière efficace.

3.2 Objectifs des SE

Deux grandes classes de tâches qui requièrent une expertise humaine considérable :
- Comptabilité, calcul numérique, commande de machines-outils, ... souvent des processus intellectuels précisément modélisés.
- Diagnostic médical, prospection minière, prévision météorologique, ... processus intellectuels mal définis.

Nous remarquons que ces deux classes de tâches font intervenir une expertise non suffisamment structurée pour donner lieu à des algorithmes parfaitement identifiés. Il s'agit plutôt d'une expertise représentée comme un ensemble d'unités (ou de règles) relativement indépendantes.

3.3 Les composants de base des SE

Les SE sont des logiciels prenant en charge des tâches intellectuelles dans des domaines où est reconnue une expertise humaine insuffisamment structurée pour constituer une méthode de travail précise, sûre, complète et directement descriptible sous forme d'un programme classique.

Un système expert est composé de trois parties indépendantes :

— **Une base de connaissances** qui contient une base de faits et une base de règles. Elle représente le savoir (les faits permanents) et le savoir-faire (les règles de l'expert). La base de faits intègre deux types de faits : les faits permanents du domaine et les faits déduits par le moteur d'inférences qui sont propres au cas traité.

— **Un moteur d'inférences** capable de raisonner à partir des informations contenues dans la base de connaissance, de faire des déductions, etc.

— **Un module de dialogue avec l'usager.**

L'indépendance entre la base de connaissances et le moteur d'inférences est un élément essentiel des systèmes experts. Elle permet une représentation des connaissances sous forme purement déclarative c'est à dire sans lien avec la manière dont ces connaissances sont utilisées. L'avantage de ce type d'architecture est qu'il est possible de faire évoluer les connaissances du système sans avoir à agir sur le mécanisme de raisonnement. Il en est de même pour nous un accroissement ou une modification de nos connaissances n'entraîne pas nécessairement une restructuration en profondeur de nos mécanismes de fonctionnement.

3.3.1 Base de connaissances

Une base de connaissances est composée d'une base de règles qui modélise la connaissance du domaine considéré et d'une base de faits qui contient les informations concernant le cas que l'on est en train de traiter

1. La base de faits

Les faits se sont les connaissances *assertionnelles*.
La base de faits 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.

2. La base de règles

les règles se sont les connaissances opératoires. La base de règles rassemble la connaissance et le savoir-faire de l'expert. Elle n'évolue donc pas au cours d'une session de travail.

Le format de règle :

- La partie gauche est le déclencheur ou le corps
- La partie droite est composée des conclusions ou actions

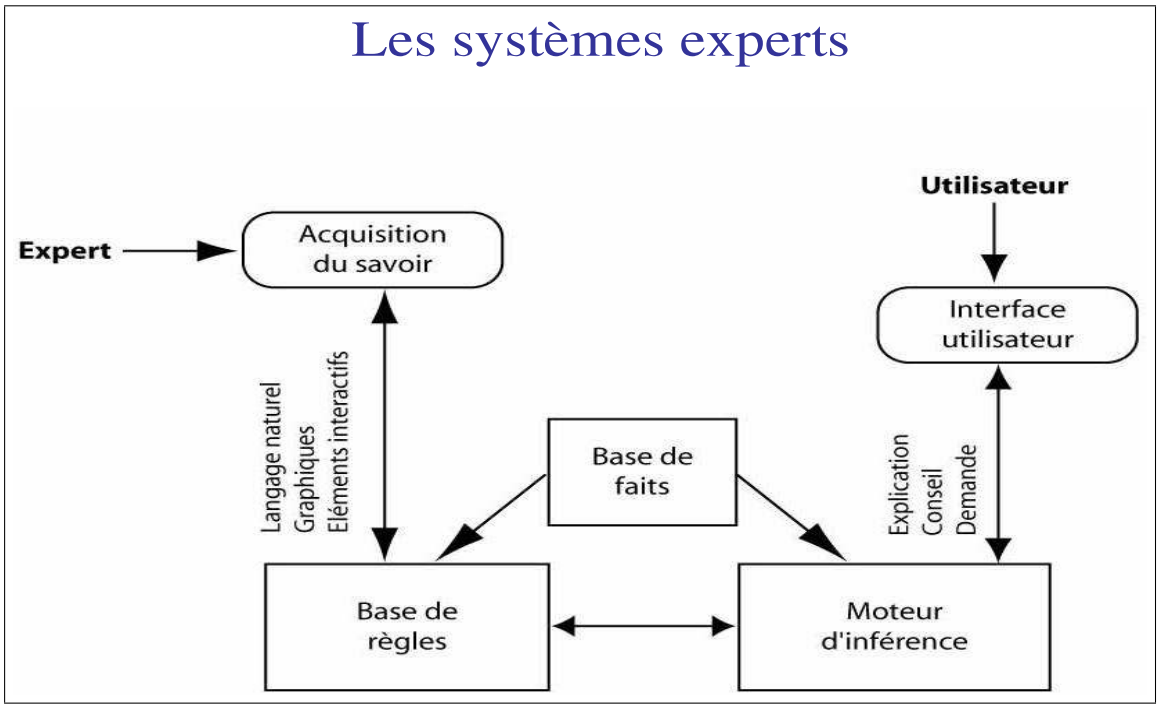


FIGURE 3.1 – Les composants d'un système Expert

Une règle est de la forme

Si conjonction de conditions **alors** conclusion

Une base de règles est un ensemble de règles et sa signification logique est la conjonction de la signification logique de chacune des règles.

Exemple 7
fait : il pleut.
règle : s'il pleut alors il fait mauvais.

3.3.2 Le moteur d'inférence

C'est un programme chargé d'exploiter la BC pour mener un raisonnement sur le problème posé en fonction du contenu de la base de faits. Pour cela, il contient un algorithme qui examine les conditions de règles et vérifie si elles sont vraies ou fausses. Une règle dont la prémisse (ou partie condition) est vraie est dite "applicable". Une prémisse peut contenir une ou plusieurs conditions. Chaque condition correspond à un fait ; elle est vraie si le fait est présent dans la base, fausse si le fait contraire est présent et inconnue si le fait est absent. Chaque règle possède également une partie "conclusion". Ce peut être une action à effectuer ou un fait à déduire et à rajouter dans la base de faits.

Le moteur d'inférence est le programme qui construit les raisonnements à partir de la base de connaissances. Face à une situation donnée, il détecte les connaissances intéressantes, les utilise, les enchaîne, construit un plan de résolution. Indépendamment du domaine, il rassemble les mécanismes de raisonnement qui vont exploiter la base de connaissances.

Le moteur fonctionne, cycliquement, de la manière suivante :

- évalue l'état courant de la base de connaissance c'est-à-dire détermine les règles à déclencher.
- Exécute les actions décrites par les règles déclenchées.

3.3.3 Module de dialogue avec l'usager

Ce module regroupe des structures et des conventions de représentation des connaissances.

C'est le module d'interaction avec l'utilisateur (Relation Homme-Machine). Ce module ne doit pas être mis au second plan. Pour que le SE soit utilisé son aspect interface doit être soigné. Sans cela, le système risque de rebuter les personnes destinées à l'utiliser. En effet, l'introduction d'un SE dans un service modifie les méthodes de travail, ainsi que l'organisation du groupe, et elle représente parfois une première approche de l'informatique pour certaines personnes. Autant faire en sorte que le changement soit agréable et apporte effectivement une amélioration.

3.4 Fonctionnement des moteurs d'inférence

3.4.1 Chaînage Avant

Un moteur d'inférences fonctionne en chaînage avant quand il lit les règles "à l'endroit", c'est-à-dire qu'il utilise les règles des conditions vers les conclusions. Le raisonnement est alors guidé par les faits. Pour le moteur, cela consiste à essayer d'activer les règles en examinant leur condition (ou partie à gauche du ALORS) et à appliquer celles-ci chaque fois que c'est possible. L'application d'une règle permet de déduire de nouveaux faits qui viennent enrichir la base. Le moteur s'arrête dès qu'il ne trouve plus de règles activables.

Le mécanisme du chaînage avant est très simple : pour déduire un fait particulier on déclenche les règles dont les prémisses sont connues jusqu'à ce que le fait à déduire soit également connu ou qu'aucune règle ne soit plus déclenchable. Le chaînage avant consiste à consigner les faits élémentaires qui sont vérifiés. C'est un mécanisme d'exploitation des règles guidé par les faits. C'est la traduction du modus ponens :

$$\text{Si } f_1 \text{ est vrai et } f_1 \rightarrow f_2 \text{ alors } f_2 \text{ est vrai.}$$

Le chaînage avant traduit un raisonnement déductif :

— de f1 sont déduits f2 et f3
— de f2 sont déduits f4 et f5
— etc ...

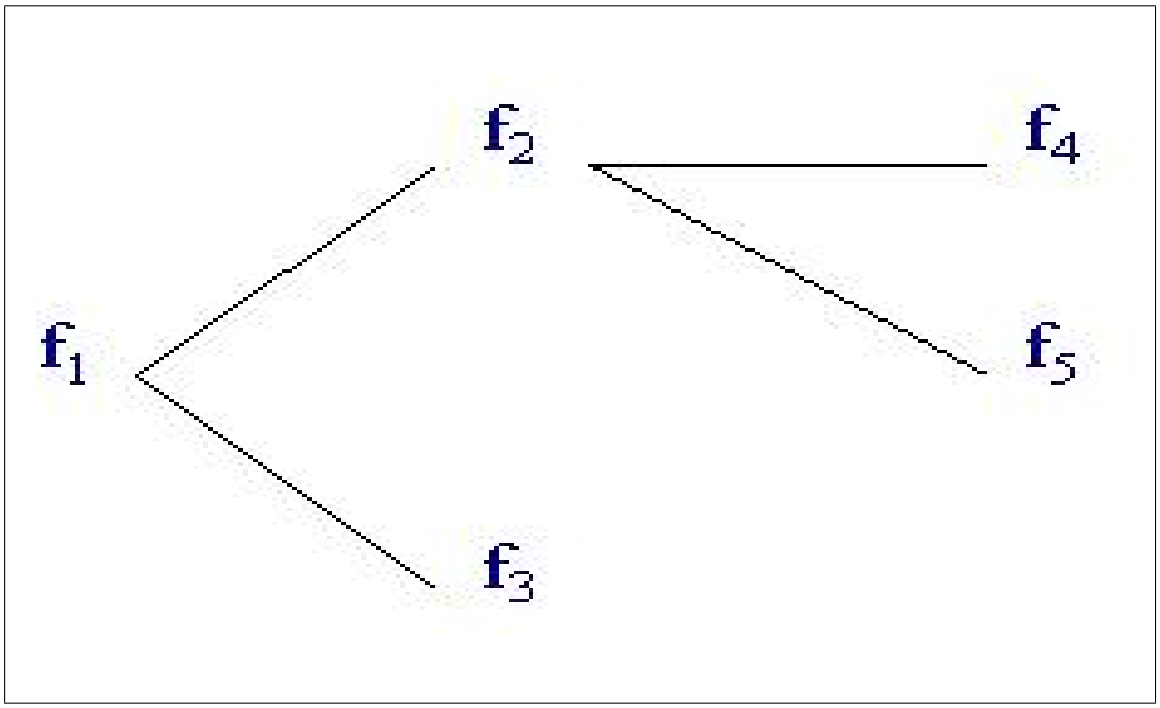
A chaque étape le moteur d'inférence parcours un cycle de trois phases :

- Recherche des règles applicables : la partie condition est vérifiée à partir de la base de faits
- Sélection d'une règle à appliquer parmi les candidates
- Application de la règle sélectionnée : les éléments de la conclusion vont s'ajouter à la base de faits

Lorsque le but est ajouté à la liste (cas de succès), ou lorsque nous ne pouvons plus rien ajouter (cas d'échec), nous arrêtons.

Le chaînage avant traduit un raisonnement déductif : Plus précisément, soit BF une base de

faits, BR une base de règles (ne comportant que des faits booléens positifs) et Fait le fait que l'on cherche à établir l'algorithme suivant calcule si Fait peut être déduit ou non de la base de connaissances.



Algorithme *Chainage Avant*

Entrée : BF : base de faits,
BR : base de règles,
F : Fait

Début

01. **Tant que** (F n'est pas dans BF et il existe dans BR une règle applicable) **Faire**
02. Choisir une règle applicable R
03. BR = BR -R (désactivation de R)
04. BF = BF union conclusion(R) (déclenchement de la règle R, sa conclusion est rajoutée à la base de faits)
05. **Fin_faire**
06. **Si** (F appartient à BF) **alors**
07. F est établi
08. **Sinon**
09. F n'est pas établi
10. **Fin_si**

Fin.

3.4.2 Chaînage Arrière

Un moteur d'inférences fonctionne en chaînage arrière quand il lit les règles "à l'envers", c'est-à-dire quand il utilise les règles des conclusions vers les conditions. Le raisonnement est alors guidé par les buts. Un but est un fait à démontrer. Pour le moteur, cela consiste donc à se "fixer un but", puis à examiner les règles permettant de l'établir. Cela l'amène à vérifier de nouveaux buts (sous-buts du but initial ou buts intermédiaires) et ainsi de suite jusqu'à atteindre des faits connus

(appartenant à la base). Il échoue chaque fois qu'un but intermédiaire (ou fait) nécessaire n'est pas prouvable.

Le chaînage arrière consiste donc à se fixer le but à réaliser et à utiliser les règles à l'inverse, c'est-à-dire en examinant les propositions dont les conséquents sont le but et en essayant de prouver leurs antécédents.

Le moteur fonctionne selon un cycle en trois phases :

1. Détection des règles qui concluent sur le but courant
2. Sélection d'une règle
3. Application de cette règle : la partie condition de la règle devient un nouveau but à atteindre. Si cette condition est déjà dans la BF (rien à faire), sinon elle est empilée dans la pile des buts.

Le cycle reprend jusqu'à ce que la pile soit vide (succès) ou aucune règle n'est plus envisageable (échec).

Algorithme *Chainage Arrière*

Entrée : BF : base de faits, BR : base de règles, F : Fait
Début
01. **Si** (F ∈ BF)
02. **Alors**
03. renvoyer VRAI
04. **Fin_Si**
05. **POUR TOUT** R dans BR **Faire**
06. activable (R) ← VRAI
07. **Fin_faire**
08. R-Sélectionnée ← {}
09. **POUR TOUT** R dans BR **Faire**
10. **Si** activable (R) = VRAI et F ∈ Conclusions (R) et ∀ p ∈ Prémisses (R), p ∈ BF)
11. **Alors**
12. Ajouter R à R-Sélectionnée
13. **Fin_faire**
14. **SI** R-Sélectionnée = {}
15. **Alors**
16. renvoyer FAUX
17. **Sinon**
18. R ← CHOISIR(R-Sélectionnée)
19. activable (R) ← FAUX
20. Soit $p_1, \cdots, p_n$ les Prémisses(R)
21. CHAINAGE-ARRIERE(BF, BR, p_1) $\cdots$ CHAINAGE-ARRIERE(BF, BR, p_n)
22. **Fin_Si**
Fin.

3.4.3 Chaînage mixte

L'homme met en jeu une démarche dirigée par les données (tirer des conclusions à partir des données) et une démarche dirigée par les buts (tenter de prouver une idée). Ceci se traduit par un chaînage mixte.

Une partie des faits de la BF est considérée comme étant à établir (des buts à démontrer) alors que d'autre comme établies (les faits proprement dit). L'utilisation du chaînage mixte demande la mise en place d'heuristiques particulières capables de décider face à une situation donnée quel est le type de stratégie le plus adapté pour poursuivre la résolution.

Algorithme *Chainage Mixte*

Entrée : F : Fait
Début
01. **Tant que** (F n'est pas déduit mais peut encore l'être) **Faire**
02. Saturer la base de faits par chaînage AVANT.
C'est à dire déduire tout ce qui peut être déduit.
03. Chercher quels sont les faits encore éventuellement déductibles
04. **Fin_faire**
05. **Si** (F appartient à BF) **alors**
06. F est établi
07. **Sinon**
08. F n'est pas établi
09. **Fin_si**
Fin.

3.4.4 Chaînage avant ou chaînage arrière ?

Aucune de ces deux techniques n'est reconnue supérieure à l'autre, cependant il y a des types de problèmes pour lesquels on choisira plus facilement l'une ou l'autre. Si un but précis doit être atteint par exemple : "Le moteur électrique est-il en régime établi ?", il est conseillé de disposer d'un moteur à chaînage arrière. En revanche, lorsque le programme a pour tâche de recueillir des informations pour déterminer des états successifs et prendre une décision ou déclencher une action, le chaînage avant est alors mieux adapté. C'est le cas par exemple du diagnostic où l'on part de l'état observé sur la machine en panne pour remonter à l'élément qui a causé la défaillance.

3.5 Les caractéristiques des moteurs d'inférence

3.5.1 Monotonie ou non monotonie

Un moteur d'inférence est qualifié de monotone si et seulement si :
- Aucune connaissance ne peut être retirée de la BC
- Aucune connaissance ajoutée à la BC n'introduit de contradiction.

Un moteur d'inférence est qualifié de non monotone si et seulement si il permet de supprimer provisoirement des connaissances (faits, règles).

3.5.2 Régime de contrôle d'un moteur d'inférence

Deux régimes de contrôle sont possibles :
- *Irrévocable* : dans ce cas le moteur d'inférence s'arrête dès que l'ensemble des règles selectionnables sera vide.
- *Tentative* : dans ce cas le moteur d'inférence réalise un retour en arrière et remet en cause les règles déclenchées précédemment.

3.5.3 Ordre d'un moteur d'inférence

Un moteur d'inférence d'ordre 0 est un moteur fondé sur la logique des propositions et respectant les principes suivants :

- Déclencheur (prémisse ou conclusion) est une fbf de la forme : F, $\neg F$, $F \vee G$, etc.
- Le nombre de connecteurs acceptés est limité.
- On accepte uniquement la forme normale conjonctive ou la forme normale disjonctive.

Un moteur d'inférence d'ordre 0^+ ou 1 est un moteur fondé sur la logique des prédicats du premier ordre et respectant les principes tel que le déclencheur (prémisse ou conclusion) est une fbf selon cette logique.

3.6 Implémentation du moteur d'inférence

Soit le moteur en chaînage avant avec régime irrévocable et monotone.

Algorithme
Début

1. Si $F \in BF$ alors renvoyer "succès"
2. Appeler $un - cycle(BR)$
   Procédure $un - cycle(les - reg, les - f, une - r)$
   (a) Si $les - reg = \emptyset$ alors renvoyer "échec"
   (b) $une - r \leftarrow elementdeles - reg$ (i.e. :première rencontrée)
   (c) $les - reg \leftarrow les - reg$ diminué de $une - r$
   (d) si toutes les prémisses de $une - r \in BF$ alors
   4.1. début
   4.2. si la conclusion de $une - r \equiv F$ alors renvoyé "succès"
   4.3. rajouter la conclusion de $une - r$ à la BF
   4.4. Appeler $un - cycle(BR)$
   4.5. fin
3. Appeler $un - cycle(les - reg)$

Fin

Soit le moteur en chaînage arrière avec régime par tentative et monotone.

Algorithme
Procédure Arrière (F)

1. Si $F \in BF$ alors renvoyer "succès"
2. Appeler Etape_1 (BR)

Procédure Etape_1 ($les - reg$)

1. Si $les - reg = \emptyset$ alors renvoyer "échec"
2. $une - r \leftarrow$ element de les $-reg$ (i.e. première rencontrée)

3. $les - reg \leftarrow les - reg$ diminué de $une - r$
4. si $une - r \subset F$ alors si $Etape2(une - r) = succes$ alors renvoyer $succes$
5. Appeler Etape_1 $(les - reg)$

Procédure Etape_2 $(la - r, le - f)$

1. $des - f \leftarrow$ tous les premisses de la $-r$
2. Appeler $conjonction(des - f)$

Procédure conjonction $(des - f)$

1. si $les - f = \emptyset$ alors renvoyer succés
2. $un - f \leftarrow$ un element de les $-f$
3. $les - f$ diminue de un $-f$
4. Si $Arriere(un - f) =$ échec alors renvoyer échec
5. Appeler conjonction $(des - f)$
