INF7470

Systèmes Tutoriels Intelligents

Introduction: les fondements de l'IA
2- Résolution automatique de problèmes

Roger Nkambou

Contenu

- Solutionnement par recherche
- Recherche avec heuristique
- Représentation de connaissances
- Systèmes à base de connaissances

Intelligence artificielle
La résolution automatique de problèmes

Quelques rudiments

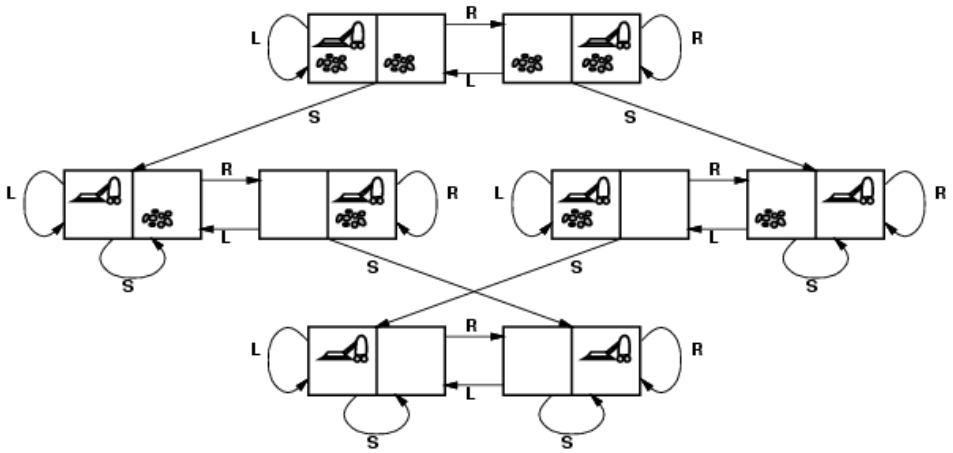
- Représentation de problème
- Résolution par recherche de solution dans un espace d'états
- Améliorations « heuristiques »

Représenter la situation

- Pour raisonner, faire des inférences, il faut une représentation manipulable.
  - État actuel (où se trouve l'agent, ce qui l'entoure, état de progression, ressources disponibles, etc.)
  - But à atteindre : un état du monde désiré
  - Les options de progrès : l'arbre / le réseau des états possibles (l'*espace du problème*)
  - Les moyens disponibles pour progresser (opérations possibles)
  - Le coût de chaque opération

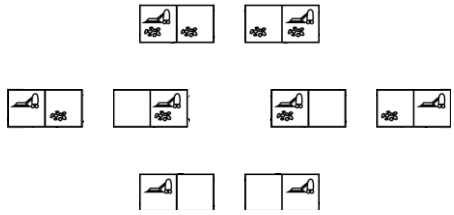
  - Pour Robi, un robot balayeur dans un hôtel de deux chambres:
    - état actuel : deux chambres empoussiérées ← manque-t-il quelque chose ici?
    - But / état souhaité: deux chambres nettoyées
    - Moyens disponibles pour progresser: aller à gauche, aller à droite, aspirer la poussière

Espace des états d'un micro-monde: *Robi l'aspirateur*

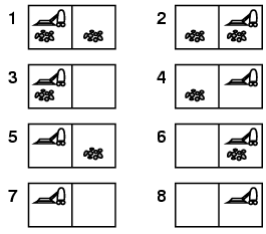


Représentation d'un problème

- Définitions
  - États du problème

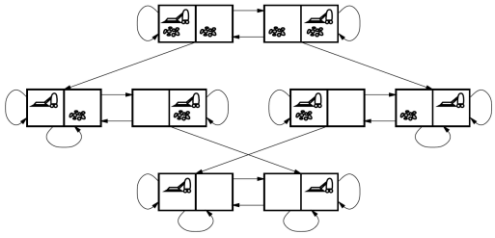


huits états possibles



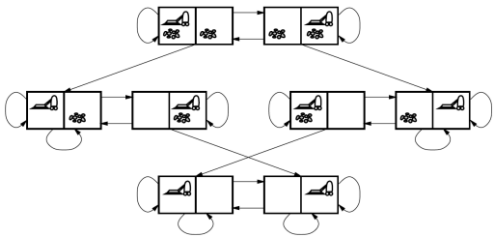
Représentation d'un problème

- Définitions
  - États du problème
  - Transitions (Opérateur de transformation d'état)
  - État initial



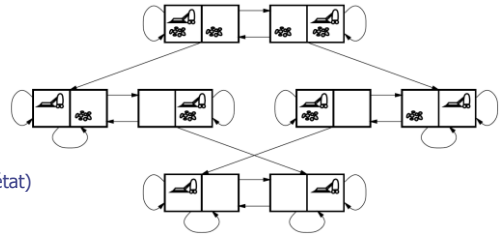
Représentation d'un problème

- Définition
  - États du problème
  - Transitions (opérateur de transformation d'état)
  - État initial
  - État final (But)



Représentation d'un problème

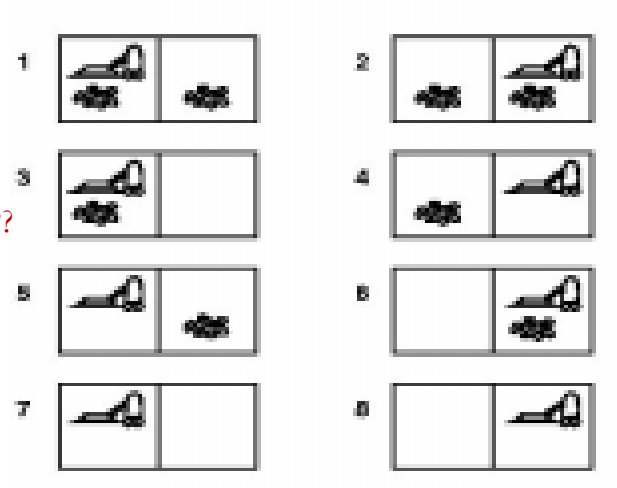
- Définitions
  - États du problème
  - Transitions (opérateur de transformation d'état)
  - État initial
  - État final (but)



- Solution au problème = Résultat de la recherche:
  - Un chemin (une séquence d'états allant de l'état initial à l'état final souhaité (au but)
- Lorsqu'un chemin est trouvé, l'agent l'exécute du début à la fin.
- Question pour vous:
  - Identifiez une limitation importante de ce fonctionnement

Le monde de l'agent aspirateur

Etats??
Etat Initial??
Actions??
Test du but??
Coût du chemin??



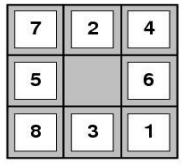
Le monde de l'agent aspirateur

- Etats: L'agent est dans un des deux emplacements, qui peuvent contenir ou non de la poussière (8 états en tout)
- Etat Initial: n'importe quel état
- Actions: Gauche, Droite, Aspirer
- Test du but: Vérifier que tous les emplacements sont propres
- Coût du chemin: somme du nombre d'étapes qui composent le chemin

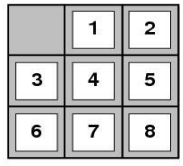
Le jeu de taquin à 8 pièces

- Plateau de 3X3 cases
- 8 cases sont occupées par une pièce numérotée
- 1 case est vide
- But: Atteindre une configuration donnée à partir d'un état initial

- Etats??
- Etat Initial??
- Actions??
- Test du but??
- Cout du chemin??



Start State



Goal State

Le jeu de taquin à 8 pièces

- Etats: L'emplacement des 8 pièces et celui de la case vide
- Etat initial: n'importe quel état
- Fonction successeur: génère les états pouvant résulter de l'exécution de 4 actions (Déplacement vers Gauche, Droite, Haut ou bas)
- Etat final: l'état correspond à la configuration finale
- Coût du chemin: nombre d'étapes qui composent le chemin (coût de chaque étape = 1)

Des exemples de situations nécessitant la recherche de solutions

- Disposition des composantes sur un circuit intégré
  - situation initiale: des millions de composantes non organisées
  - Transitions possibles indiquées par les contraintes de connectivité
    - pas de superpositions de composantes
    - espace disponible pour passer les chemins entre les composantes
  - But : disposition qui minimise l'espace, les délais de transport de l'information, les pertes d'énergie, et les coûts de fabrication
- Assemblage robotique
- Recherche d'un trajet pour aller de Montréal à l'Auberge du Petit Bonheur
- Conception de protéines

Stratégies de *recherche*

- Recherche aveugle
  - Profondeur d'abord
  - Largeur d'abord
  - Profondeur itérative
  - Largeur itérative
- Recherches informées
- Jeux avec adversaire

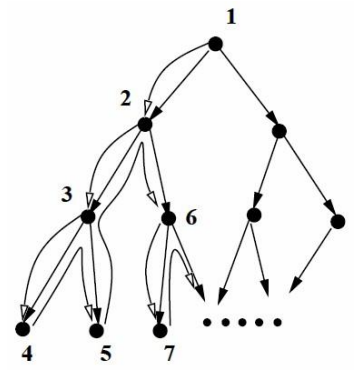
Stratégies de *recherche* : aveugle ou informée

- Recherche aveugle
  - Profondeur d'abord
  - Largeur d'abord
  - Profondeur/largeur itérative
- Recherche informée (avec heuristique)

Stratégie de recherche

- La stratégie indique un ordre de développement des noeuds
- 4 dimensions d'évaluation d'une stratégie:
  - Complétude : permet-elle de toujours trouver une solution s'il en existe une ?
  - Complexité en temps : nombres de noeuds générés pour trouver la solution
    - → combien de temps il pour trouver la solution
  - Complexité en espace: nombre maximal de noeuds mémorisés
    - → combien de mémoire il pour effectuer la recherche
  - Optimalité : permet-elle de toujours trouver le chemin le moins coûteux (ex.: le plus court)?
- La complexité en temps et en espace se mesurent en termes de :
  - *b :* facteur de **b**ranchement maximal dans l'arbre de recherche
  - *d :* profondeur du but le plus "en surface"
    (du plus court chemin vers un but, de la solution la moins coûteuse)
  - *m* : profondeur **m**aximale de l'espace d'état
    (longueur maximale de n'importe quel chemin; peut être ∞)

Stratégie de recherche
En profondeur d'abord



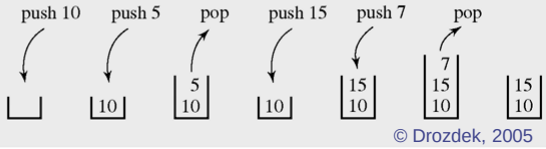
Source: École polytechnique fédérale de Lausanne

- Expansion (et examen) du premier noeud jusqu'à ce qu'il n'y ait plus de successeur
- Retour arrière au niveau précédent
- Modeste en mémoire: ne conserve en mémoire que
  - le chemin direct actuel allant de l'état initial à l'état courant,
  - les enfants (noeuds ouverts) non examinés du parent actuel
  - les noeuds en attente le long du chemin parcouru.

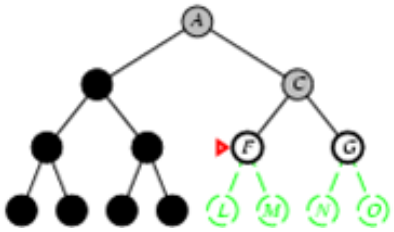
Stratégie de recherche
En profondeur d'abord

- Développer le nœud le plus profond non encore développé
- Implantation: *Exploration_en_arbre* avec une file LIFO=pile => mettre les successeurs à l'avant

FIGURE 4.1 A series of operations executed on a stack.



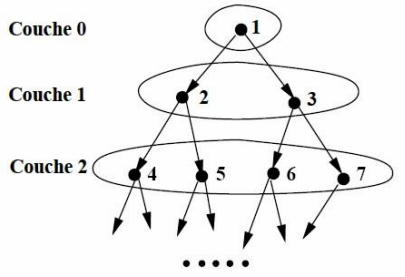
Stratégie de recherche
En profondeur d'abord (suite)



Recherche en profondeur d'abord
Propriétés

- Complète?
- Optimale?
- Temps?
- Espace?

Stratégie de recherche
En largeur d'abord



Source: École polytechnique fédérale de Lausanne

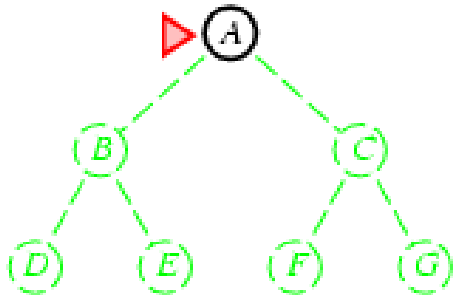
- Expansion et examen par couche
- Trouve toujours le chemin le plus court
- Exige beaucoup de mémoire pour stocker toutes les alternatives à toutes les couches.

Exploration en largeur d'abord

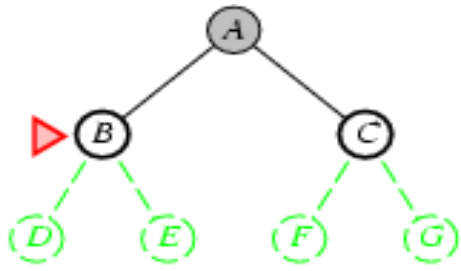
- Principe: Développer les nœuds à un même niveau de l'arbre avant d'aller au niveau suivant.
- La frontière est implantée sous forme de file FIFO (retirer le nœud le plus ancien)
- Développement: mettre les enfants à la fin de la file
- Retirer les nœuds à l'avant de la file

Exploration en largeur d'abord

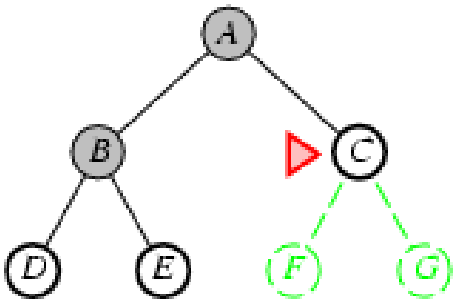
- Développer tous les nœuds d'une profondeur donnée avant de passer au niveau suivant
- Implantation: Utiliser une file FIFO, i.e., les nouveaux successeurs sont rangés à la fin.



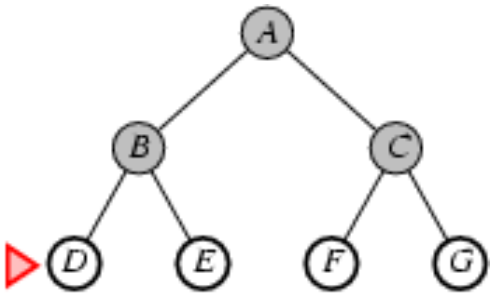
Exploration en largeur d'abord



Exploration en largeur d'abord



Exploration en largeur d'abord



Recherche *en largeur d'abord* *Propriétés*

- Complète?
- Optimale?
- Temps?
- Espace?

Recherche *en largeur d'abord* *Propriétés*

- Complète?
- Optimale?
- Temps?
- Espace?

Le meilleur des deux mondes: Exploration Itérative en profondeur

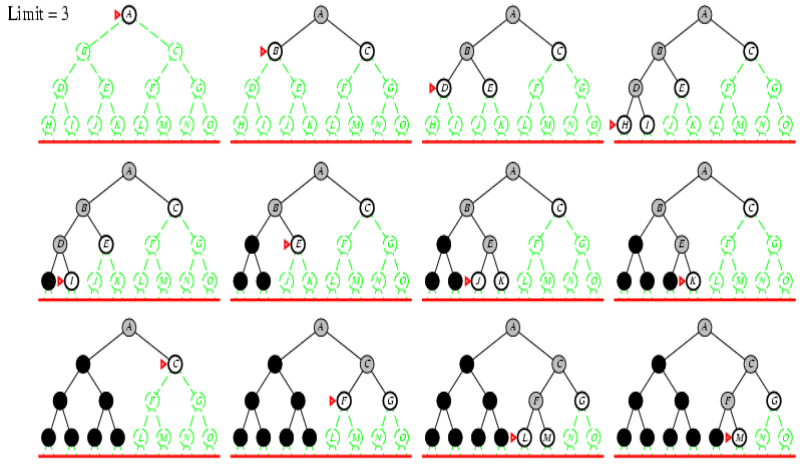
- Profondeur d'abord est efficace en espace mais ne peut garantir un chemin de longueur minimale
- Largeur d'abord trouve le chemin le plus court (en nombre d'étapes) mais requière un espace exponentiel
- Exploration Itérative en profondeur effectue une recherche en profondeur limitée avec un niveau de profondeur croissant jusqu'à ce que le but soit trouvé.

Exploration Itérative en profondeur

Augmentation graduelle de la limite

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or fail-
ure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

L=3



Recherche en profondeur itérative
Propriétés

- Complète?
- Optimale?
- Temps?
- Espace?

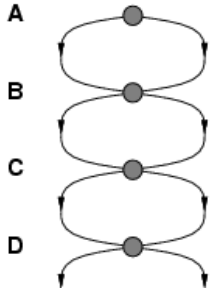
Résumé

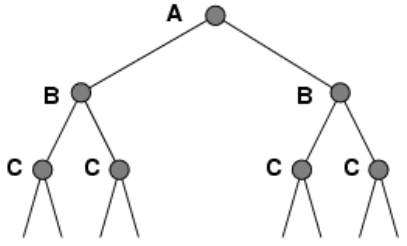
Criterion	Breadth-First	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	No	No	Yes

b: facteur de branchement de l'arbre de recherche
d: profondeur de la solution la moins couteuse
m: prof. max de l'espace

États répétitifs

- Si ces états ne sont pas détectés la recherche peut devenir exponentielle → une recherche théoriquement possible peut devenir techniquement impossible!





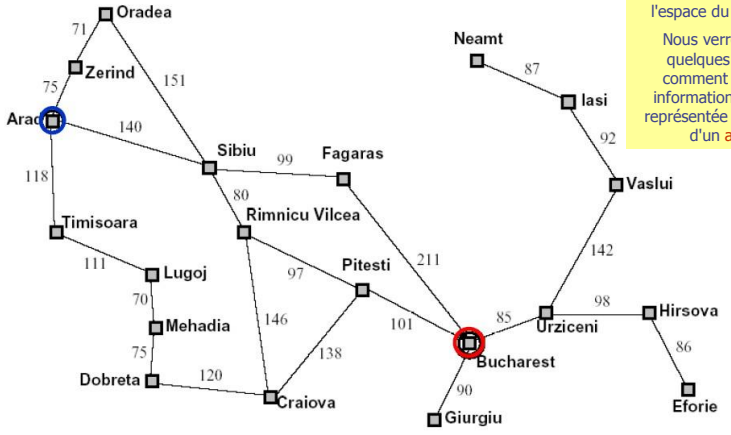
- Solution: Garder les noeuds déjà visités dans une liste appellée Closed

Problème de la recherche aveugle

- Ne garde pas les états du chemin solution au cours de la recherche
- Complexité en espace
  - Dans tous les cas, le nombre de nœuds à mémoriser croît de façon exponentielle)
  - Envisager l'élaguation de l'espace de recherche par les heuristiques (recherches informées)

Planifier un voyage

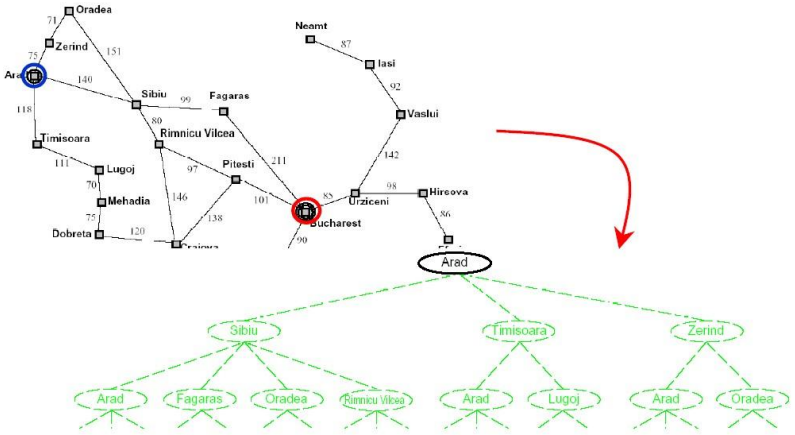
- Pour aller d'Arad à Bucharest, utilisons les multiples stratégies de recherche aveugle présentées jusqu'ici.



Ceci est un graphe de l'espace du problème.

Nous verrons dans quelques instants comment la même information peut être représentée sous forme d'un arbre.

Graphe de l'espace du problème *vs. Arbre*



Source: Alain Boucher, Institut de la Francophonie pour l'Informatique
http://www.ifi.auf.org/personnel/Alain.Boucher/cours/intelligence_artificielle/05-Recherche.pdf.

Rappel : qu'est-ce qu'une heuristique?

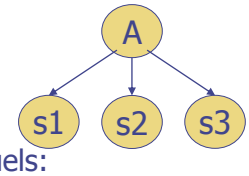
- Fonction qui établit des pistes pour orienter vers la solution, et élaguer les candidats peu intéressants

Méthodes de recherche heuristique

- Les algorithmes de recherche aveugle n'exploitent aucune information concernant la structure de l'arbre de recherche ou la présence potentielle de noeuds-solution pour optimiser la recherche.
- Recherche "rustique" à travers l'espace jusqu'à trouver une solution.
- La plupart des problèmes réels sont susceptibles de provoquer une explosion combinatoire du nombre d'états possibles.
- Un algorithme de recherche heuristique utilise l'information disponible pour rendre le processus de recherche plus efficace.
- Une information heuristique est une règle ou une méthode qui presque toujours améliore le processus de recherche.

Fonction heuristique

- Une fonction heuristique
  $$\boldsymbol{h} : E \rightarrow R$$
  fait correspondre à un état $\boldsymbol{s} \in E$ (espace d'états) un nombre $\boldsymbol{h(s)} \in R$ qui est (généralement) une estimation du rapport coût/bénéfice qu'il y a à étendre le chemin courant en passant par $\boldsymbol{s}$.
- Contrainte: $h(solution) = 0$



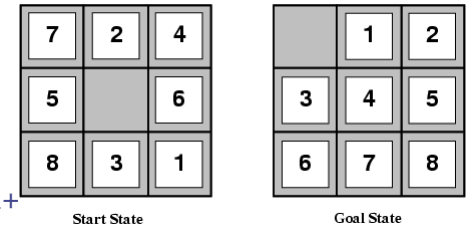
- Le noeud A a 3 successeurs pour lesquels:
  $h(s1) = 0.8$ $h(s2) = 2.0$ $h(s3) = 1.6$
- la poursuite de la recherche par *s1* est heuristiquement la meilleure

Exemple d'heuristique : *Distance de Manhattan*

Taquin à 8 plaquettes
Suggestion de deux heuristiques possibles:

- $h_1(n)$ = nombre de plaquettes mal placées
- $h_2(n)$ = distance de Manhattan
  - déplacements limités aux directions verticales et horizontales
  - somme des distances de chaque plaquette à sa position finale

- $h_1(n) = 8$
- $h_2(n) = 3+1+2+2+$



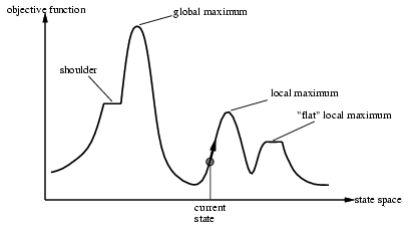
Utilisation d'une heuristique : recherche par *Hill-Climbing*

- Idée:
  - La recherche *par escalade* n'examine que ses voisins immédiats
    - Développer le nœud courant
    - Évaluer ses fils et choisir le meilleur
  - Élimine les frères et les parents du noeud retenu
  - Arrête à l'atteinte d'un état meilleur que tous ses fils.

- Inconvénient:
  - Non complète (ne garantit pas de trouver la solution)
  - Sans mémoire => pas de retour arrière possible => On ne peut pallier à un échec
  - Problème du maximum local : La recherche s'arrête parce qu'il a atteint un état meilleur que tous ses fils mais pas forcément meilleur que tous les noeuds.

Recherche par *Hill-Climbing* *Inconvénients*

- Inconvénient:
  - Non complète (ne garantit pas de trouver la solution)
  - Sans mémoire => pas de retour arrière possible => On ne peut pallier à un échec
  - Problème du maximum local : La recherche s'arrête parce qu'il a atteint un état meilleur que tous ses fils mais pas forcément meilleur que tous les noeuds.



Source: Artificial Intelligence: A Modern Approach (Second Edition) by Stuart Russell and Peter Norvig
http://aima.eecs.berkeley.edu/slides-ppt/m4-heuristics.ppt

Utilisation d'une heuristique dans la stratégie du Meilleur-d'abord

- L'algorithme "Best-First search" permet d'explorer les noeuds dans l'ordre (meilleur-que) de leurs valeurs heuristiques
- fonctions heuristiques classiques
  - distance "Manhattan" (déplacements limités aux directions verticales et horizontales)
  - distance à vol d'oiseau

- On utilise 2 listes pour garder l'historique et permettre les retours-arrières:
  - *Open* = liste ordonnée des noeuds à examiner
  - *Closed* = liste des noeuds déjà examinés

Utilisation d'une heuristique dans la stratégie Meilleur-d'abord (suite)

```
Open ← [start]; Closed ← [];
Tant que Open n'est pas vide
   noeud ← enleverPremier (Open)
 Si noeud = but, retourner le chemin solution
 Sinon
       Générer les enfants de noeud
       Pour chaque enfant
         calculer sa valeur heuristique (h(enfant))
        cas:
         - l'enfant n'est ni dans Open ni dans Closed
               Ajouter dans Open
         - l'enfant est dans Open
              Si h(enfant) meilleur que sa valeur dans Open
                   remplacer la valeur dans Open
         - l'enfant est dans Close
                Si h(enfant) meilleur que sa valeur dans Closed
                   enlever l'enfant de Closed et le mettre dans Open avec la valeur
       h(enfant)
         Fin Cas
       FinPour
     FinSi
     Mettre noeud dans Closed; Reordonner Open;
  FinTantQue
  Afficher ("Échec")
```

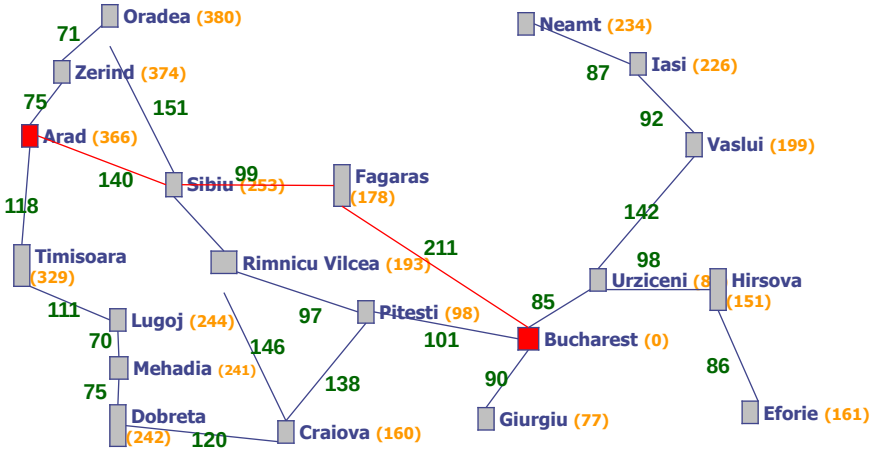
- Cas particuliers de "best-first search":
  - A: avec historique
  - A*: A + évaluation *avare*

Recherche Meilleur-d'abord *avare*

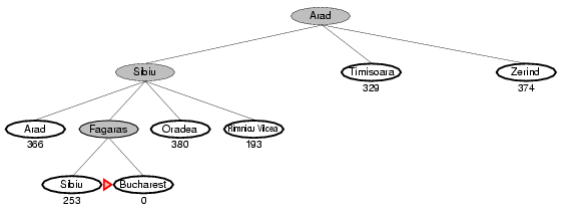
- Stratégie la plus simple des "best-first search"
- fonction heuristique *h(n)* = estimation du coût du noeud n **jusqu'au but**
- recherche avare = minimiser le coût estimé pour atteindre le but
  - le noeud qui semble être le plus proche du but sera étendu en priorité
  - en considérant uniquement l'état actuel, pas l'historique des coûts accumulés

Exemple : Voyage en Roumanie

- État initial : Dans(Arad)
- But : Dans(Bucharest)
- Voyage: **h(n) = distance_en_ligne_droite (n, but)**



Voyage en Roumanie
expansion des noeuds par recherche avare



h = dist. à vol d'oiseau	
Ville (v)	**h(v)**
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Propriétés du meilleur-d'abord avare

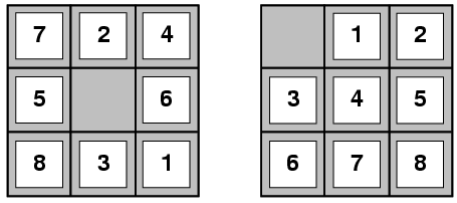
- Complète?
- Temps?
- Espace?
- Admissible?

Fonction heuristique *admissible*

- Théorème:
  Une fonction heuristique est admissible si
  $\forall\ n\ 0 \leq h(n) \leq h^*(n)$ avec $h^*(n)$ = coût optimal réel de n au but
- Autrement dit: ne surestime jamais le coût réel
- Une fonction heuristique admissible est donc toujours optimiste!
- *La recherche avare* minimise le coût estimé $h(n)$ du noeud n au but, réduisant ainsi considérablement le coût de la recherche, mais il n'est pas optimal et pas complet (en général)
- l'algorithme de recherche en coût uniforme minimise le coût $g(n)$ depuis l'état initial au noeud n, il est optimal et complet, mais pas très efficace
- idée: combiner les deux algorithmes et minimiser le coût total $f(n)$ du chemin passant par le noeud n

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

Exemple d'heuristique
Taquin à 8 plaquettes



- h1(n) = nombre de plaquettes mal placées = 8 **est admissible**
- h2(n) = somme des distances de chaque plaquette au but = 18 **est admissible**
- h3(n) = (somme des distances de chaque plaquette au but) + 3 x (somme de la fonction score de chaque plaquette) = 3+1+3+0+2+1+0+3 + 3x(2+2 + 2+2+2+2+2+2) = 66 **n'est pas admissible**
-

fonction score = 2 pour une plaquette non centrale si elle
n'est pas suivie par la bonne plaquette (pas la bonne séquence) et 0 si non

Note:
- La complexité de la fonction heuristique peut nuire à l'efficacité de la recherche
  => Il faut trouver un bon compromis

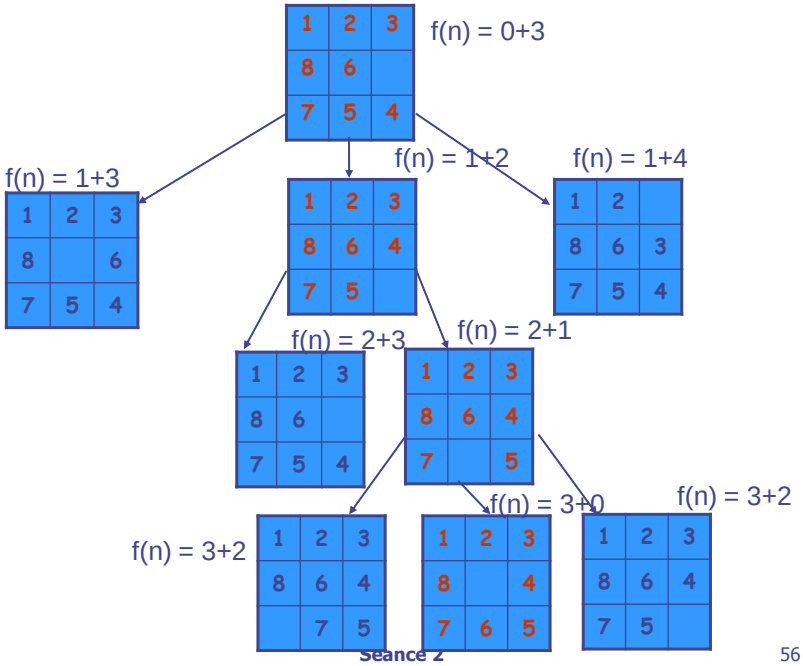
Algorithme A & A*

- Soit la fonction d'évaluation
  f(n) = g(n) + h(n)
- Meilleur-d'abord + f(n) comme fonction d'évaluation = algorithme **A**
- Si en plus f(n) est telle que:
  - g(n) = coût du meilleur chemin jusqu'à n
  - h(n) = une fonction heuristique admissible

  L'algorithme "recherche avare" avec la fonction f(n) est appelé: algorithme **A***

Exemple A* avec Taquin-8

- Taquin-8: f(n) = g(n) + h(n)
  avec h(n) = nombre de plaquettes mal placées

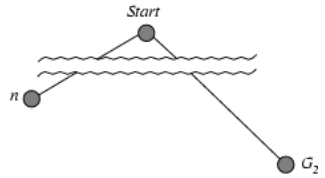


Propriétés de A*

- Complète?
- Temps ?
- Espace ?
- Admissible ?

Démo de l'optimalité (l'admissibilité) de A*

- Supposons l'existence d'un noeud but sous-optimal G_2. Soit n se trouvant sur le plus court chemin menant vers *G le but optimal*.



- $f(G_2) = g(G_2)$ car $h(G_2) = 0$
- $g(G_2) > g(G)$ car G_2 est sous-optimal
- $f(G) = g(G)$ car $h(G) = 0$
- $f(G_2) > f(G)$ de ce qui précède
- $f(G_2) > f(G)$ de ce qui précède
- $h(n) \leq h^*(n)$ car h est admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$

Ainsi $f(G_2) > f(n)$, et A^* ne sélectionnera jamais G_2 comme candidat à l'expansion.

Qualité des fonctions heuristiques

- Facteur de branchement effectif
  - soit N = nombre total d'états produits pour obtenir la solution
  - soit d = profondeur à laquelle la solution a été trouvée
  - alors b* est le facteur de branchement d'un arbre fictif parfaitement équilibré tel que

    $N = 1 + b^* + (b^*)2 + \ldots + (b^*)d$

  exemple: si d = 5 et N = 52 > b* = 1.91
- Une bonne fonction heuristique aura une valeur de b* proche de 1
- (la fonction heuristique idéale aurait b* = 1)

Dominance

- Soient 2 fonctions heuristiques admissibles $h1(n)$ et $h2(n)$
  - Si $h2(n) \geq h1(n)$ $\forall n$ alors on dit que $h2$ domine $h1$ et produira une recherche plus efficace
- Exemples de coûts typiques de recherche (taquin-8)
  - d = 14
    - IDS (approfondissement itératif) = 3'473'941 états
    - A* ($h1$) = 539 états
    - A* ($h2$) = 113 états
  - d = 24
    - IDS = beaucoup trop d'états
    - A* ($h1$) = 39'135 états
    - A* ($h2$) = 1'641 états

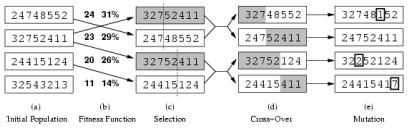
Variantes de A*

- les problèmes réels sont souvent très complexe
  - l'espace de recherche devient très grand
  - même les méthodes de recherche heuristiques deviennent inefficaces
- A* connaît alors des problèmes de place-mémoire

- Y a-t-il des algorithmes de recherche "économes" en place-mémoire?
  2 variantes de A*:
  - IDA* = A* avec approfondissement itératif
  - SMA* = A* avec gestion de mémoire

Un dernier exemple d'algo de recherche: *algorithmes génétiques*

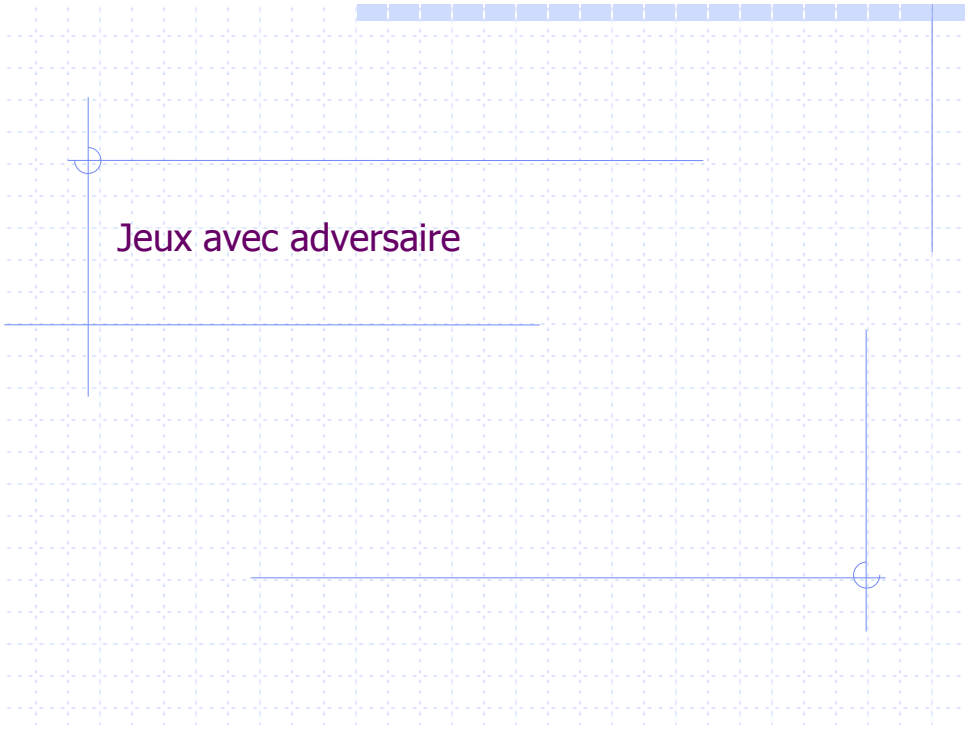
- On débute par la création d'une population de de *k* états aléatoires
- L'état suivant est généré à partir de la combinaison de deux états parents
  - *État* : une chaîne de caractères sélectionnés dans un alphabet fini (délimité) – souvent une chaîne de 0 et 1)
- Une fonction évalue chaque croisement résultant par rapport à un critère (ou plusieurs)
- Les croisements les plus satisfaisants servent à générer la prochaine génération, soit par croisement, soit par mutation.



Source: Artificial Intelligence: A Modern Approach (Second Edition) by Stuart Russell and Peter Norvig
http://aima.eecs.berkeley.edu/slides-ppt/m4-heuristics.ppt

Quand utiliser un algorithme de **recherche**?

- Lorsque l'espace de recherche est de petite taille et
  - qu'il n'y a pas d'autres techniques, ou
  - ce n'est pas rentable de développer une technique plus efficace.
- Lorsque l'espace de recherche est de grande taille et :
  - qu'il n'y a pas d'autre technique, et
  - qu'il n'existe pas de "bonne" heuristique.
- Procéder par recherche comporte des contraintes
  - devoir tout connaître à l'avance, et tout décrire!



Jeux vs. problèmes de recherche

- Adversaire imprévisible → Spécifier un déplacement pour chaque possibilité de replique de l'adversaire
- Contraint dans le temps

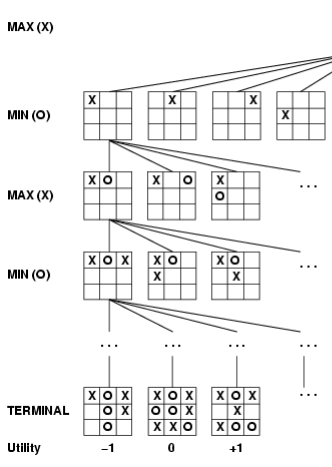
Jeux vs. problèmes de recherche

- Jouer c'est rechercher le mouvement permettant de gagner
- Les jeux de plateau contiennent les notions de:
  - état initial et état gagnant (but)
  - opérateurs de transition (règles du jeu, déplacements de pièces)
- Une fonction heuristique permet d'estimer s'il y a gain, perte ou match nul
- Premiers algorithmes de jeu
  - algorithme pour jeu parfait, J. von Neumann (1944)
  - horizon fini, évaluation/approximation, K. Zuse (1945), C. Shannon (1950), A. Samuel (1952)
  - réduction de coût par élagage, McCarthy (1956)
- Mais il y a d'importantes différences avec un problème standard de recherche

Jeux vs. problèmes de recherche

- Les mouvements de l'adversaire ne sont pas toujours prévisibles
  - il faut donc être capable de prendre en compte toutes les situations
- L'espace de recherche est généralement très vaste:
  exemple: pour le jeu d'échecs
  - nombre de choix par coup: 35 (facteur de branchement)
  - nombre moyen de coups par jeu: 50 (par joueur)
  - nombre total d'états: 35^{100}
  - nombre de noeuds de l'arbre: $\sim 10^{120}$ (pour le jeu de dames $\sim 10^{40}$)
    - Il faudrait 10^{21} siècles pour produire l'arbre complet en générant un nœud en 1/3 de nanoseconde.
- Il est donc impossible d'explorer tout l'espace de recherche pour trouver le meilleur mouvement à effectuer

Arbre de jeu (2-joueurs)



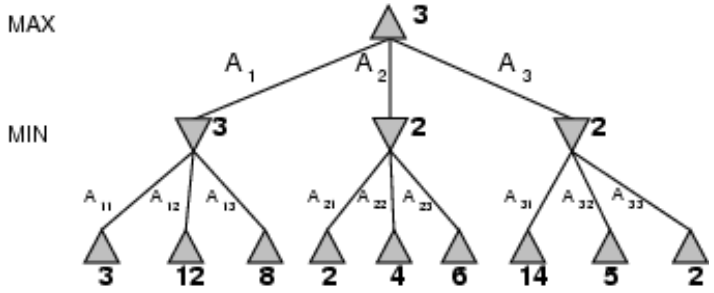
Max et Min sont les 2 joueurs. Un gain pour Max est une perte pour Min et vice-versa.
+1 pour un gain, -1 pour une perte, 0 pour un nul.

La somme des valeurs de la fonction d'évaluation pour les 2 joueurs est nulle à la fin du jeu.

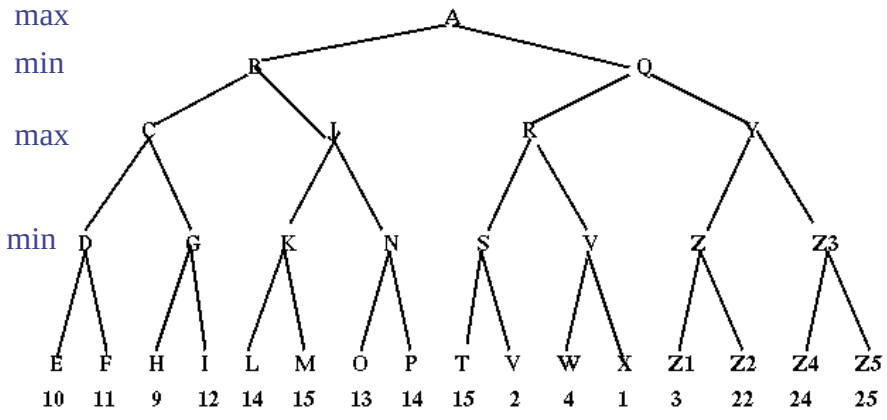
Il faut disposer d'une fonction d'évaluation statique capable de mesurer la qualité d'une configuration par rapport à un joueur (gén. Max)
car il n'est pas possible de produire tout l'arbre de recherche jusqu'à la fin du jeu, càd au moment où la décision gain/perte/nul est claire.

Minimax

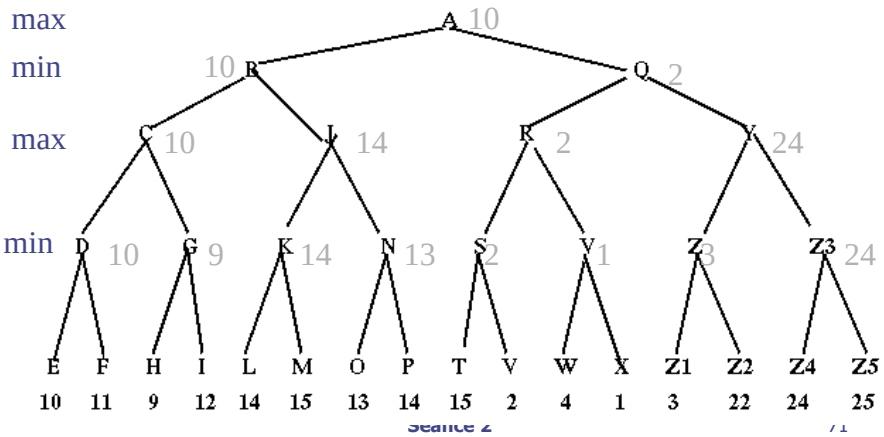
- Principe: maximiser la valeur d'utilité pour Max avec l'hypothèse que Min joue parfaitement pour la minimiser,
  - étendre l'arbre de jeu
  - calculer la valeur de la fonction de gain pour chaque noeud terminal
  - propager ces valeurs aux noeuds non-terminaux
    - la valeur minimum (adversaire) aux noeuds MIN
    - la valeur maximum (joueur) aux noeuds MAX



Minimax



Minimax



Propriétés de minimax

- Complète?
- Optimal?
- Temps ?
- Espace ?

Méthodes de réduction d'espace

Même pour le jeu de Tic-Tac-To la taille de l'espace de recherche est grande:
3^9 = 19'683 noeuds pour un damier entièrement occupé !

Pour des jeux non-triviaux les coups doivent pouvoir être déterminés sans pour autant devoir générer tout l'arbre de jeu

Il faut alors pouvoir évaluer des noeuds non-terminaux

2 solutions:

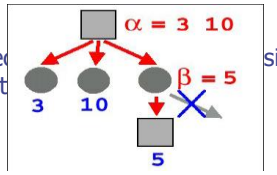
- *MiniMax* avec profondeur limitée
- élagage α-β

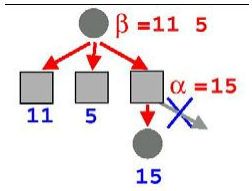
Élagage α-β

- Principe
  - étendre l'arbre de jeu jusqu'à une profondeur h par recherche en profondeur
  - ne plus générer les successeurs d'un noeud dès qu'il est évident que ce noeud ne sera pas choisi (compte tenu des noeuds déjà examinés)
  - chaque noeud Max garde la trace d'une α-valeur = valeur de son meilleur successeur trouvé jusqu'ici
  - chaque noeud Min garde la trace d'une β-valeur = valeur de son plus mauvais successeur trouvé jusqu'ici
  - valeurs initiales: α = - ∞ β = + ∞

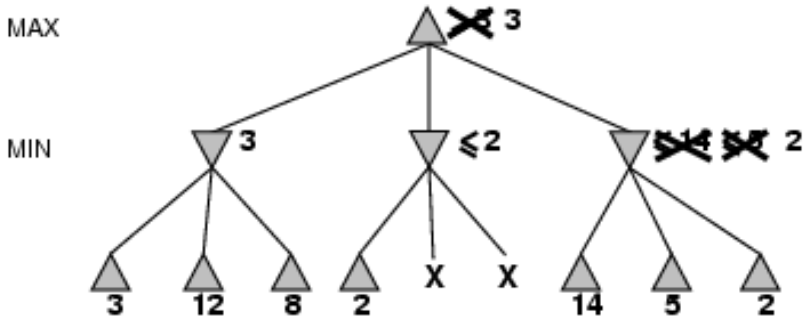
Élagage α-β

- Deux règles:
  1. Interrompre la recherche d'un noeud Max si son α-valeur ≥ β-valeur de son noeud-parent
  2. Interrompre la re si sa β-valeur ≤ α-valeur de son noeud-parent

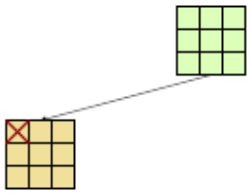




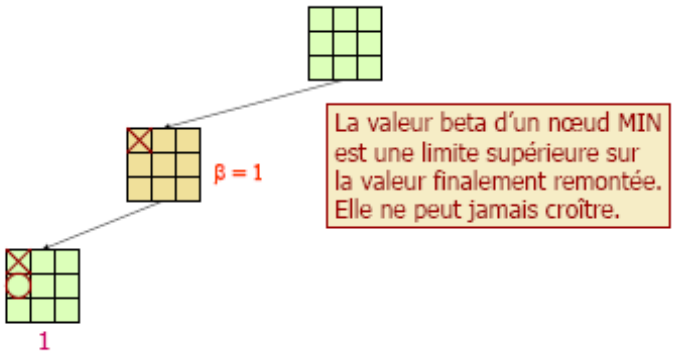
Élagage α-β : exemple



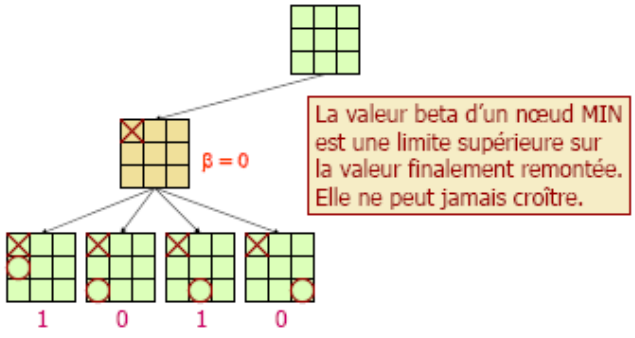
Exemple2



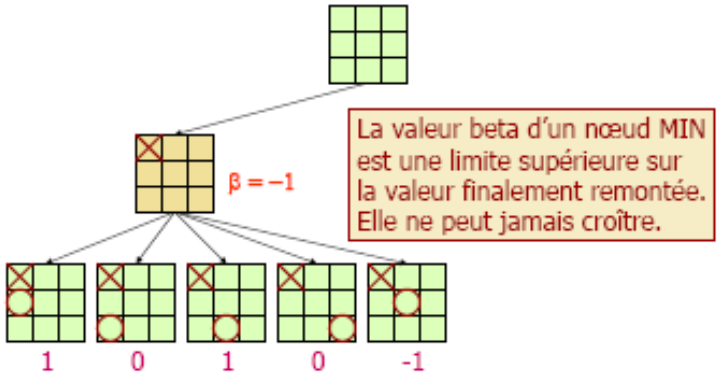
Exemple2



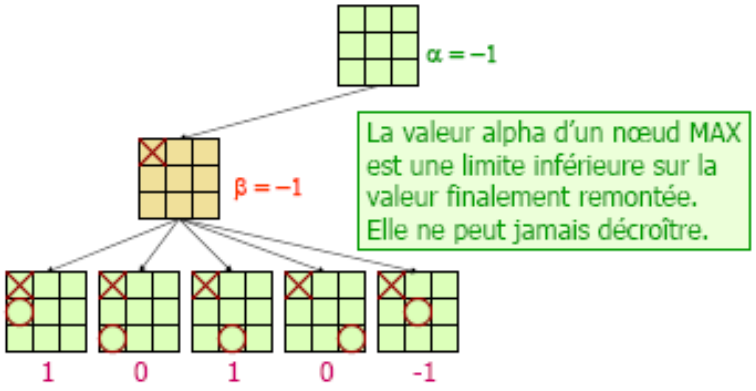
Exemple2



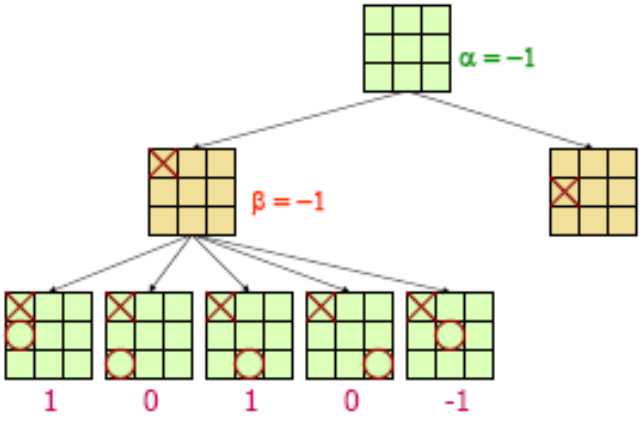
Exemple2



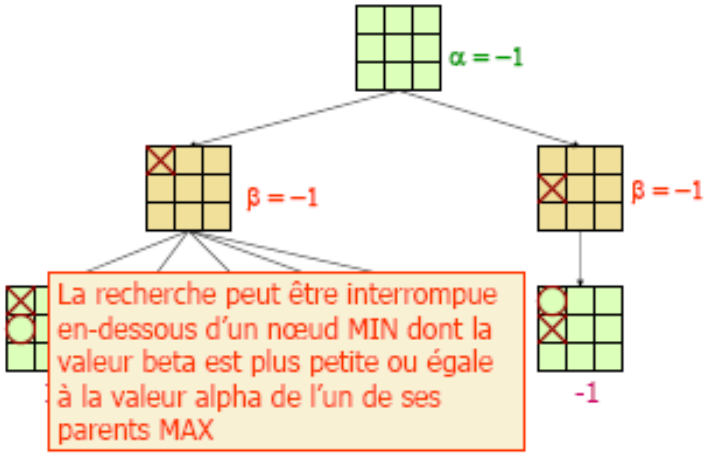
Exemple2



Exemple2



Exemple2

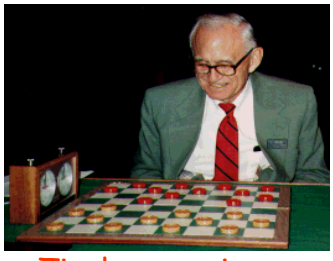


Propriétés de α-β

- L'élagage n'affecte pas le résultat final,
- l'efficacité de l'élagage α - β est fonction de l'ordre d'apparition des noeuds successeurs,
- complexité en temps
  - meilleur des cas: $O(bm/2)$
    - permet de doubler la profondeur de recherche pour atteindre une prédiction sur 8 niveaux et ainsi jouer "dignement" aux échecs
  - pire des cas: identique à MiniMax
  - cas moyen: $O((b/\log b)m)$ [Knuth&Moore 75

État de l'art

Jeu de dames: Tinsley vs. Chinook



Nom: Marion Tinsley
Profession: professeur de mathématiques
Hobby: le jeu de dames
Record: en plus de 42 ans n'a perdu que 3 (!) parties

Mr. Tinsley encaisse sa 4ème et 5ème défaite contre Chinook

Nom: Chinook
Record: premier programme informatique ayant gagné un championnat du monde face au champion en titre en 1994, invaincu depuis! Méthode: base de fins de parties de 8 pièces (ou moins), pour un total de 444 milliards de positions

Jeu d'échecs

Jeu qui a reçu la plus grande attention au début les progrès ont été très lents

➢ 1970: 1er programme à gagner le championnat ACM d'échecs informatiques, utilisant une recherche α - β, des ouvertures classiques et des algorithmes de fins de parties infaillibles

➢1982: "Belle" est le premier ordinateur spécialisé dans le jeu d'échecs capable d'explorer quelques millions de combinaisons par coups

➢1985: "Hitech" se classe parmi les 800 meilleurs joueurs du monde, il est capable d'explorer plus de 10 millions de combinaisons par coups

➢1997: "*Deep Blue*" bat G. Kasparov. Il effectue une recherche sur 14 niveaux et explore plus de 1 milliard de combinaisons par coup à raison de plus de 200 millions de positions par seconde, utilise une fonction d'évaluation très sophistiquée et des méthodes non divulguées pour étendre certaines recherches à plus de 40 niveaux.

Kasparov vs. Deep Blue

Kasparov		Deep Blue
5'10"	Taille	6' 5"
176 lbs	Poids	2,400 lbs
34 ans	Age	4 ans
50 milliards	Ordinateur	512 processeurs
2 pos/sec	Vitesse	200,000,000 pos/sec
Extensive	Connaissance	Primitive
Electrique/chimique	Source d'énergie	Électrique
Démesuré	Ego	Aucun

1997: Deep Blue gagne par 3 victoires, 1 défaite, et 2 nuls

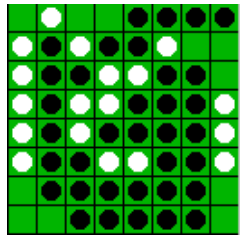
Kasparov vs. Deep Blue Junior

Deep Junior

8 CPU, 8 GB RAM, Win 2000
2,000,000 pos/sec
Disponible à $100

2 Août 2003: Égalité (3-3)!

Othello: Murakami vs. Logistello



Takeshi Murakami
Champion du monde d'Othello

1997: Le logiciel Logistello défait Murakami par 6 jeux à 0

Représentation de Connaissances

Définition, problématique, structures, langages

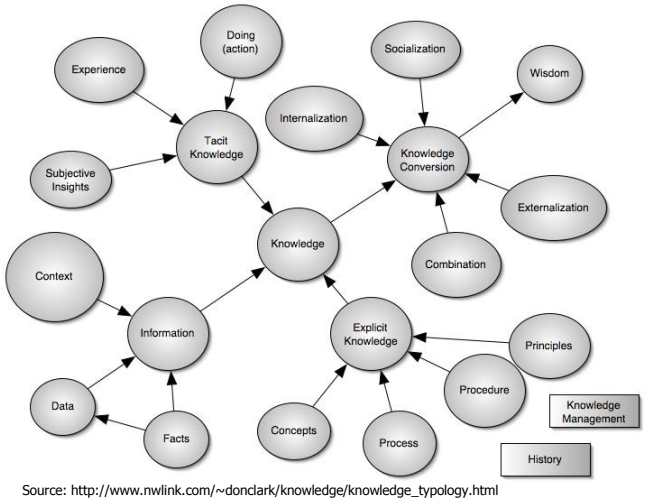
Représentation de connaissances

- Connaissance?
- Notion de représentation de connaissance
- Représentation dans un modèle
  - Langages de représentation (logique)
- Réseaux sémantiques

Définitions : Connaissance

- Connaissance :
  - savoirs, savoir-faire, savoir-être *possédés par une personne*
  - donc, les informations qui ont été assimilées par une personne, c'est-à-dire organisées dans un réseau qui les intègre.
  - savoir-faire : scripts explicites ou habileté implicite
  - script: suite d'opérations génériques propres à une situation.
- Connaissance (en IA) :
  - Toutes les formes de savoir et savoir-faire de l'humain
    - Objets qui forment le monde réel
    - Concepts
    - Relation entre concepts
    - Procédures de raisonnement
    - Métaconnaissance (Vision sur son raisonnement, reflexion sur ses connaissnces. Etc.)
- La connaissance peut être:
  - Spécifique à un domaine ou générale
  - Profonde ou de surface
  - Exacte ou imprécise
  - Explicite ou implicite
  - Incertaine
  - Incomplète

Données/information/connaissance



Source: http://www.nwlink.com/~donclark/knowledge/knowledge_typology.html

Connaissance

- Connaissance => informations organisées dans une structure personnelle (interne à la personne)
- Le passage de INFORMATION à CONNAISSANCE est lié à l'expérience de l'action => pas de frontière parfaitement définie
- Pas de classement universel des différents types de connaissances

Représentation

- Représentation des connaissances = Transcription sous une forme symbolique exploitable par un système de raisonnement
- Formalisme de représentation des K = système de description pour représenter la connaissance + méthode associée pour l'exploitation de cette connaissance
- Plusieurs formalismes: logique du premier ordre, réseau, règles de production, schémas (*frames*), …

Représentation

- Représenter ⇔ Approximer dans le contexte d'une tâche (activité?) particulière
- Représenter ⇔ Structure de symboles pour « décrire » une approximation du « monde » (un modèle du monde) dans le contexte d'une tâche particulière.
- Interpréter une structure (une représentation) ⇔ Composition de l'interprétation des différents symboles la constituant

Représentation
…mise en garde

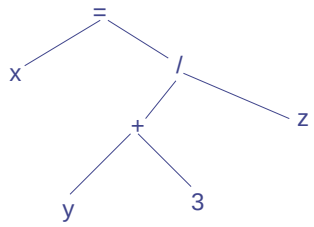
- Dire que A *représente* B
  - Ne suffit pas pour que ce soit *vrai*
    - ex.: S: "Le Soleil tourne autour de la Terre"
  - Il convient de vérifier que si B a un certain effet sur un processus P, A démontre un effet *équivalent* sur un processus équivalent
- A n'est cependant pas *équivalent* à B
  - Une carte n'est pas le territoire (heureusement!)
  - Une carte *représente* le territoire dans le cadre d'un processus de recherche d'un itinéraire (par exemple)

Langage (formalisme) de représentation

- Il s'agit naturellement de langages formels.
  - un alphabet, ensemble de symboles pas nécessairement réduit à des caractères
  - un procédé de formation des expressions, pas nécessairement la concaténation
  - un ensemble d'*axiomes*, c'est-à-dire d'expressions obéissant aux deux premiers points ci-dessus, et dont on décide arbitrairement qu'ils appartiennent au système
  - des règles de dérivation qui, à partir des axiomes, permettent de produire des *théorèmes* (c'est-à-dire des expressions appartenant au système), et peuvent ensuite s'appliquer aux théorèmes pour en produire d'autres

Représentation
Exemple

- Exemple
  - x=(y+3)/z



Cette composition nécessite Int(=), interprétation de l'égalité entre deux termes, dont l'un est Int(x) et l'autre obtenu par l'application de Int(/) à deux autres termes , le premier résultant de l'application de Int(+) à Int(y) et à Int(3) et l'autre étant Int(z).

Représentation dans un modèle
Langage: syntaxe et sémantique

- Langage => aspects *syntaxiques* de la représentation (attention : on parle ici de langage **formel**, donc précis, univoque et limité!)
- Système de déduction => aspects *sémantiques* (attention, "déduction" provient d'un calcul et peut être très éloigné d'un « sens » quelconque)
- Règles de valuation => « vrai », « faux »

Types de logique

- Les différentes logiques sont caractérisées par ce qu'elles engagent comme "primitives«
- engagements ontologiques: ce qui existe, faits?, objet?, temps?, croyances?, etc.
- engagements épistémologiques: différents états de la connaissance

Langages	**Ontologies**	**Epistémologie**
Logique propositionnelle	Faits	vrai/faux/inconnu
Logique des prédicats	Faits, objets, relations	vrai/faux/inconnu
Logique temporelle	Faits, objets, relations	Temps vrai/faux/inconnu
Logique floue	Degrés de vérité	Degré de croyance 0 .. 1

Autres logiques

- Logique de croyance
  - Croire(Jean, Père(Zeus, Chronos))
- Logique temporelle
  - permet de raisonner à propos du temps
- Logique non-monotone
  - permet aux valeurs de vérité associées aux faits de changer
- Logique floue
  - accompagne les faits de valeurs de vraisemblance

Inférence

- L'inférence est l'élément de base de toute description de la cognition
- Raisonner ≡ faire des inférences
- Inférer = dériver de nouvelles affirmations à partir des anciennes.
- Composantes de base:
  - axiomes : faits connus du monde
  - règles : transformations syntaxiques qui propagent la valeur sémantique.
- Pour être efficace, l'inférence doit être guidée par la connaissance
- Inférence avec / sans représentation de la connaissance
  - manipulation de connaissances explicites
  - manipulation des connaissances intégrées dans le programme de l'agent, parfois carrément implicites (les suppositions et les croyances non-déclarées/inconscientes du concepteur/programmeur)

Exemples de logique
Logique des propositions

- Une suite de **symboles** (formules atomiques) séparés par des connecteurs logiques:
  - conjonctions (et,), disjonctions (ou,)
  - négations (non, ¬)
  - implication (⇒)
  - Exemples :
    - "Socrate est un homme", représentée par *HommeSocrate*
    - "Platon est un homme", représentée par *HommePlaton*
    - "Platon et Socrate sont des hommes" → HommeSocrate HommePlaton
- De telles assertions élémentaires peuvent être :
  - Niées par l'opérateur non (¬)
  - Associées entre elle grâce aux connecteurs logiques pour former des formule logiques composés (formules *bien formées*)

Logique des propositions

- Les *prémisses* indiquent les faits (axiomes) dont on dispose au début et qui sont utiles à l'induction
  - exemple :
    - P ≡ "Jean aime Marie" (P représente la proposition "Jean aime Marie")
    - Q ≡ "Marie aime Jean"
    - Prémisse 1: ¬P ⇒ ¬Q
    - Prémisse 2: Q
    - Conclusion: alors, P
    - Explication: règle logique *Modus Tollens:*
      - $(\neg\alpha \Rightarrow \neg\beta) \Rightarrow (\beta \Rightarrow \alpha)$; si β, alors α
    - Note: *Modus Ponens*
      - $(\alpha \Rightarrow \beta)$; si α, alors conclusion: β

Logique des propositions

- Proposition = situation particulière susceptible de prendre la valeur VRAI ou FAUX dans un univers donné.
  - (1) : « Le Bordeaux est un vin »
  - (2) : « Le Morgon a une robe rouge »
  - (1) et (2) sont des assertions élémentaires ou formules atomiques.
- De telles assertions élémentaires peuvent être :
  - Niées par l'opérateur
  - Associées entre elle grâce aux connecteurs logiques (∧ (et), ∨ (ou), ¬ (non), ⇒ (implique), ⇔ (équivalent)) pour former des formule logiques composés (formules bien formées)

Modéliser l'inférence en logique des propositions

- prémisses:
  1. "Si Jean n'aime pas Marie, elle ne l'aime pas non plus"
  2. "Marie aime Jean"
- conclusion:
  alors "Jean aime Marie" est vrai
- correspondance en logique propositionnelle:
  soit P ≡ "Jean aime Marie" et soit Q ≡ "Marie aime Jean«
- composantes de base de l'inférence:
  – axiomes: faits connus du monde
  – règles: transformations syntaxiques qui propagent la valeur sémantique
- Inférence
  prémisses en logique propositionnelle
  1. ¬ P ⇒ ¬ Q
  2. Q

  conclusion
  alors P

Modéliser l'inférence en logique des propositions

- axiomes: faits connus du monde
- règles de dérivation: transformations syntaxiques qui propagent la valeur sémantique
  - E1 $\Rightarrow$ E2 est équivalent à $\neg$E1 $\vee$ E2
  - modus ponens
    - Si E1 $\Rightarrow$ E2 et E1 alors on peut déduire E2
  - modus tollens
  - Si E1 $\Rightarrow$ E2 et $\neg$E2 alors on peut déduire $\neg$E1
- prémisses:
  1. "Si Jean n'aime pas Marie, elle ne l'aime pas non plus"
  2. "Marie aime Jean"
- conclusion:

  alors "Jean aime Marie" est vrai

Logique des propositions *Lois d'équivalence* (1 de 2)

- **Lois d'élimination:**
  - $\neg(\neg A) \Leftrightarrow A$
  - $A \Rightarrow B \Leftrightarrow \neg A \vee B$
- **Lois de De Morgan:**
  - $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
  - $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$
- **Lois de distributivité:**
  - $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$
  - $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

Logique des propositions
Lois d'équivalence (2 de 2)

- **Lois de commutativité:**
  - $A \wedge B \Leftrightarrow B \wedge A$
  - $A \vee B \Leftrightarrow B \vee A$
- **Lois d'associativité:**
  - $(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$
  - $(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$
- Loi contrapositive:
  - $A \Rightarrow B \Leftrightarrow \neg B \Rightarrow \neg A$

Exemples de logique
*Logique [des **prédicats**] du premier ordre*

- Une suite de symboles, de **variables** et de relations avec des quantificateurs universels et existentiels
  - exemples:
    - Homme(**Socrate**)
    - $\forall$ x Homme(x) $\Rightarrow$ Mortel(x)
    - Mortel(Socrate)

Quelque règles d'inférence

Modus Ponens $\dfrac{\alpha \rightarrow \beta, \alpha}{\beta}$

Élimination du ET $\dfrac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$

Introduction du ET $\dfrac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$

Introduction du OU $\dfrac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$

Élimination de la double négation $\dfrac{\neg\neg\alpha}{\alpha}$

Résolution simple $\dfrac{\alpha \vee \beta, \neg\beta}{\alpha}$

Résolution $\dfrac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$

Exemple

- 1. Batterie-OK ∧ Ampoules-OK ⇒ Phares-OK
- 2. Batterie-OK ∧ Démarreur-OK ∧ ¬Réservoir-Vide ⇒ Moteur-Démarre
- 3. Moteur-Démarre ∧ ¬Pneu-Plat ⇒ Voiture-OK
- 4. Phares-OK
- 5. Batterie-OK
- 6. Démarreur-OK
- 7. ¬Réservoir-Vide
- 8. ¬Voiture-OK

- 9. Batterie-OK ∧ Démarreur-OK (5+6)
- 10. Batterie-OK ∧ Démarreur-OK ∧ ¬Réservoir-Vide (9+7)
- 11. Moteur-Démarre (2+10) Modus Ponen
- 12. Moteur-Démarre ⏲ Pneu-Plat (3+8)
  - Transformation de 3 en une suite résolution
    - ¬ Moteur-Démarre v Pneu-Plat v Voiture-OK (3), ¬Voiture-OK (8)=> ¬ Moteur-Démarre v Pneu-Plat
- 13. Pneu-Plat (11+12)
    - Moteur-Démarre (11) , Moteur-Démarre => Pneu-Plat (12) => Pneu-Plat (Modus Ponen)

On aurait pu procédér à une preuve par réfutation pour prouver Pneu-Plat:
1) introduire le contraire dans la base (¬ Pneu-Plat)
2) et montrer qu'on arrive à une contradiction en appliquant les règles d'inférences

Résolution par réfutation

- **Résolution par réfutation = Preuve par Contradiction**
  - 1. Convertir les axiomes en Forme clausale (conjonctions de disjonctions)
  - 2. Contredire le But
  - 3. Utiliser la règle d'inférence de résolution autant de fois que nécessaire jusqu'a ce que vous arriviez a une tautologie ($T \wedge \neg T$) ➔ Succès!!!

Exemple

Prouver R, étant donné les fbf suivants:

- P
- $(P \wedge Q) \Rightarrow R$
- $(S \wedge T) \Rightarrow Q$
- T

Limites de la logique des propositions

- La logique propositionnelle est déclarative: les éléments syntaxiques correspondent à des faits invariables.
- La logique propositionnelle permet d'exprimer de l'information sous forme partielle, disjonctive ou négative (contrairement à la plupart des structures ou des bases de données)
- La logique propositionnelle est compositionnelle: la signification de B1,1 ∧ P1,2 est obtenue à partir des significations de B1,1 et P1,2
- La sémantique (signification) en logique propositionnelle est indépendante du contexte (contrairement au langage naturel par exemple)
- La logique propositionnelle a un pouvoir expressif limité (contrairement au langage naturel par exemple)
  ex: on ne peut pas dire "les puits provoquent des courants d'air dans les cases adjacentes" sans écrire une phrase pour chaque case (on ne peut pas généraliser).

La logique des prédicats: *motivation*

- Comment exprimer les phrases suivantes en logique des propositions?
  - Marcus était Pompéien
  - Tous les Pompéiens étaient Romains
  - Tous les Romains étaient soit loyaux envers César soit ils le haïssaient
  - Chacun est loyal envers quelqu'un
  - Les gens tentent d'assassiner uniquement les souverains envers lesquels ils ne sont pas loyaux
  - Marcus a tenté d'assassiner César
  - Par conséquent Marcus haïssait César

La logique des prédicats du 1er ordre *motivation*

- Comment exprimer les phrases suivantes en logique des propositions?
  - Marcus était Pompéien
  - Tous les Pompéiens étaient Romains
  - Tous les Romains étaient soit loyaux envers César soit ils le haïssaient
  - Chacun est loyal envers quelqu'un
  - Les gens tentent d'assassiner uniquement les souverains envers lesquels ils ne sont pas loyaux
  - Marcus a tenté d'assassiner César
  - Par conséquent Marcus haïssait César
- Nécessité d'introduire:
  - Des constantes (marcus, césar…)
  - Des variables (X, Y, …)
  - Des prédicats (romain, loyal, pompeien,…)
  - Des fonctions (plusGrandQue(X, 150))
  - Des connecteurs (∄,∅, ¬, ⇒, ⇔)
  - Des égalités (=)
  - Des quantificateurs (∀, ∃)
- Preuve par le principe de la résolution
  - Transformation sous forme clausale (avec les règles d'équivalence)
  - Introduire la négation du but
  - Chercher à générer une tautologie

La logique des prédicats du 1er ordre *motivation*

- Nécessité d'introduire:
  - Des variables (X, Y, …)
  - Des prédicats (romain, loyal, pompeien,…)
  - Des fonctions (plusGrandQue(X, 150))
  - Des égalités (=)
  - Des quantificateurs (∀, ∃)
- Preuve par le principe de la résolution
  - Transformation sous forme clausale (avec les règles d'équivalence)
  - Introduire la négation du but
  - Chercher à générer une tautologie

La logique des prédicats du 1er ordre

- Voici la forme clausale de la base de connaissance de l'exemple :
  - Personne(Marcus)
  - Pompeien(Marcus)
  - Souverain(Caesar)
  - ¬Pompeien(x1) ∨ Romain(x1)
  - ¬Romain(x2) ∨ Loyal(x2,Caesar) ∨ Haït(x2, Caesar)
  - ¬Personne(x3) ∨ ¬Souverain (x4) ∨ ¬Assassiner(x3, x4) ∨ ¬Loyal(x3,x4)
  - Assasiner(Marcus, Caesar)
- Utiliser la résolution pour montrer que Marcus haïssait Caesar
  - **¬Haït(Marcus, Caesar),** ¬Romain(x2) ∨ Loyal(x2,Caesar) ∨ Haït(x2, Caesar), θ = {x2/Marcus}
    - **¬Romain(Marcus) ∨ Loyal(Marcus,Caesar)**
  - ¬Romain(Marcus) ∨ Loyal(Marcus,Caesar), ¬Pompeien(x1) ∨ Romain(x1), θ = {x1/Marcus}
    - **Loyal(Marcus,Caesar) ∨ ¬Pompeien(Marcus)**
  - Loyal(Marcus,Caesar) ∨ ¬Pompeien(Marcus), Pompeien(Marcus), no substitution
    - **Loyal(Marcus,Caesar)**
  - Loyal(Marcus,Caesar), ¬Personnne(x3) ∨ ¬Souverain(x4) ∨ ¬Assassiner(x3, x4) ∨ ¬Loyal(x3,x4), θ = {x3/Marcus. X4/Caesar}
    - **¬Personne(Marcus) ∨ ¬Souverain(Caesar) ∨ ¬Assassiner(Marcus, Caesar)**
  - ¬Personne(Marcus) ∨ ¬Souverain(Caesar) ∨ ¬Assassiner(Marcus, Caesar), Personne(Marcus), no substitution
    - **¬Souverain(Caesar) ∨ ¬Assassiner(Marcus, Caesar)**
  - ¬Souverain(Caesar) ∨ ¬Assassiner(Marcus, Caesar), Assassiner(Marcus, Caesar), no substitution
    - **¬Souverain(Caesar)**
  - ¬Souverain(Caesar), Souverain(Caesar) , no substitution
    - **Contradiction! Donc tautologie!**

Notions de complétude

- Un système est « complet » si toutes les formules qui sont des tautologies sont des théorèmes.

Les logiques de proposition et des prédicats du 1er ordre sont complet

Les réseaux sémantiques (Argumentation Intelligence Artificielle)

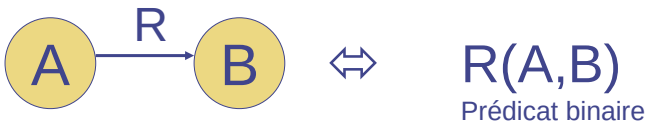
- Difficultés de la représentation à base de modèle logique
  - Système décidable ⇔ logique des propositions, mais …temps de décision exponentiels !
  - Autres logiques ⇔ plus expressives, mais semi-décidables, voire indécidables !
- Comment rendre les inférences efficaces ?
  - Restreindre la logique
  - Abandonner l'exigence de complétude !
- Rendre + facile la « lecture » de la représentation ?

Réseau sémantique (Argumentation Sciences Cognitives)

- Les représentations (humaines) de la connaissance formelle sur des informations factuelles, « dénotées » peuvent se modéliser avec 4 types d'entités
  - Des **concepts** (noms ou propositions nominales)
  - Des **relations** étiquetées entre concepts (verbes ou propositions verbales)
  - Des « **modificateurs** » (ou marqueurs) qui sont attachés aux concepts ou aux relations (pour restreindre ou clarifier leur portée)
  - Des combinaisons de Concept -> Relation -> Concept avec des modificateurs optionnels sont des **instances** de mise en relation
- L'ensemble forme de « larges réseaux d'idées » appelés « réseaux sémantiques »

Réseau sémantique
Définition IA

- Un réseau sémantique est un graphe orienté et étiqueté (un multi-graphe en fait car rien n'empêche deux nœuds d'être reliés par plusieurs arcs).
- Une *sémantique* (au sens de la logique) est associée par le biais des relations.
  - La signification d'un concept vient de ses relations avec d'autres concepts
- Réseau = conjonction de formules logiques associées à chacun des arcs

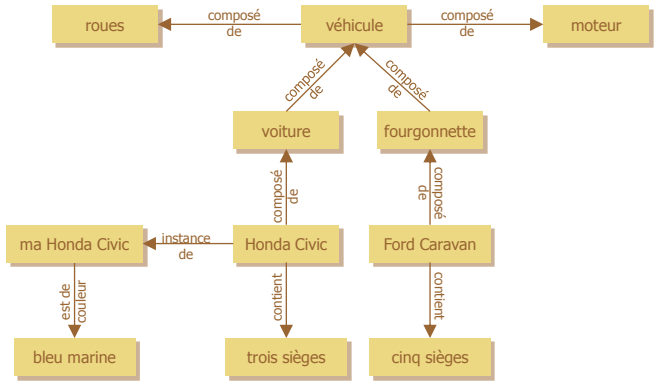


Sémantique de quelques relations

- Appartenance d'individus à une classe d'individus AKO (A kind of) ⇔ appartenance
  - B(A) --- A appartient à la classe B
- Relation de spécialisation (IsA).⇔ *sorte_de*
  - B(A) --- A est une classe *sorte_de* classe B

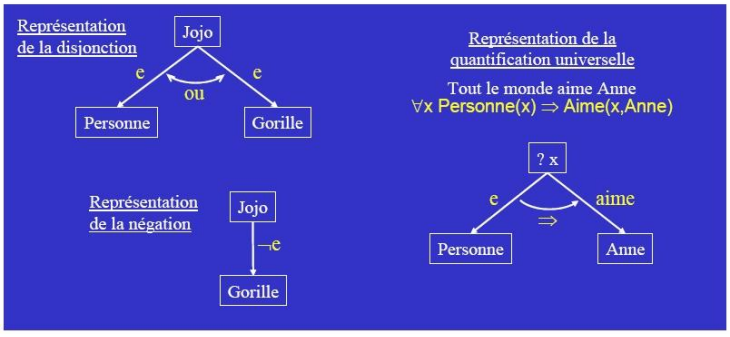
Nécessité de différencier les concepts individus (appartenance) des concepts classe (sorte_de) !

Réseau sémantique
Exemple



Réseau sémantique
Expressivité comparée à la Log. 1er ordre

- Pour exprimer toutes les phrases de la LPO, le formalisme des réseaux sémantiques doit être complété par les nœuds "ou", les arcs "non" et une construction correspondant à ∀



Réseau sémantique - exploitation

- Si on considère un réseau sémantique comme une conjonction de formules logiques, alors mêmes méthodes que pour un modèle logique
- Si on considère un réseau sémantique comme un graphe, alors on peut utiliser les techniques de propagation de marqueurs

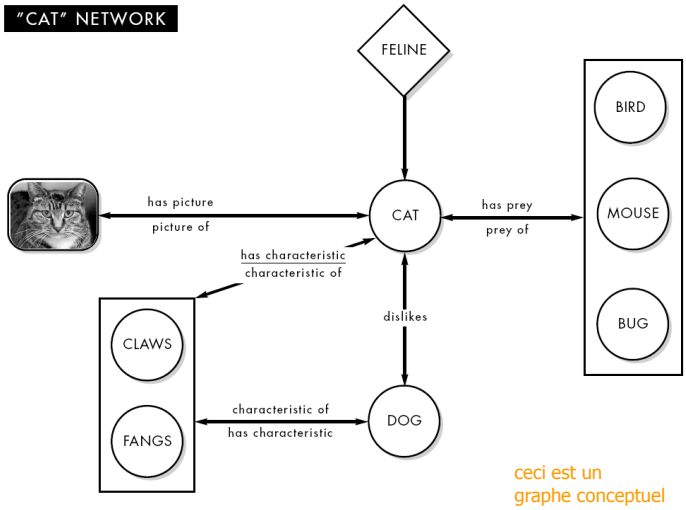
Réseau sémantique et inférences
Lisibilité, simplicité

- Mécanisme pour réaliser les inférences:
  suivre les arcs reliant les noeuds!
- 2 méthodes:
  - par recherche d'intersection
  - par héritage
    - Les relations "est-un" et "partie-de" permettent de suivre les liens d'héritage dans une taxonomie hiérarchique.
    - L'héritage permet aussi de faire du raisonnement par défaut.

Réseau sémantique / phrases ?

- Le chat est un félin
- Le chat a pour proies la souris, l'oiseau, l'insecte
- Le chat n'aime pas le chien
- Le chat est représenté par son image par « id_photo »
- Le chat a des griffes

Exemple de réseau sémantique (dénotationnel)



ceci est un graphe conceptuel

Réseau Sémantique
Propagation de marqueurs

- L'idée est que toutes les unités d'un réseau (arcs et nœuds) possèdent un processeur et une mémoire locale.
- Pour répondre à une question du genre « A est-elle nécessairement une instance de B? »
  - On place un marqueur M1 sur A
    - Tant que (le réseau continue à évoluer)
      - Tout lien AKO ayant un marqueur M1 à son origine propage ce marqueur à son extrémité
      - Si le nœud B est marqué par M1, répondre « toute instance de A est nécessairement une instance de B »
  - Très bonne adéquation au parallélisme ; bonne expressivité en ajoutant des liens « rôles » ; ajout de liens « de négation » ; ajout de liens « exception »
  - Si on propage des valeurs à la place des marqueurs, on se rapproche sensiblement des réseaux connexionnistes ! Mécanismes d'inhibition ; activation sélective de nœuds…

Graphes conceptuels : notions fondamentales

- Sowa-84 : « Conceptual graphs form a knowledge representation language based on linguistics, psychology, and philosophy ».
  - Au niveau conceptuel, c'est donc un langage de communication pour différents spécialistes impliqués dans une tâche cognitive commune.
  - Au niveau de son implantation informatique, ce peut être un outil de représentation commun pour les différentes parties d'un système complexe.

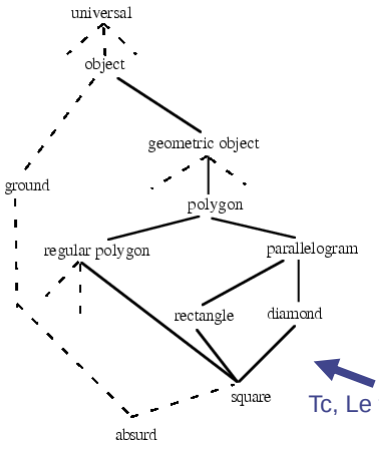
S-graph / Support

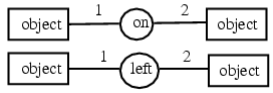
- Un graphe conceptuel est constitué de deux types de nœuds : les nœuds relations et les nœuds concepts.
- Les concepts représentent des entités, des attributs, des états et des événements, et
- Les relations montrent comment les concepts sont interconnectés.
- Certains concepts sommets sont dits « individuels » ⇔ ont trait à des entités particulières
- Certains concepts sommets sont dits « génériques » et représentent des concepts non spécifiés d'un type donné.

S-Graph / Le graphe conforme à un support.

- Un graphe conceptuel est relatif à un « support », qui définit des contraintes syntaxiques permettant décrire un domaine d'application.
  - Cette notion de support regroupe:
    - Un ensemble de « types de concepts », structurés en treillis, représentant une hiérarchie « sorte-de » acceptant l'héritage multiple.
    - Un ensemble de « types de relations »
    - Un ensemble de « graphes étoiles », appelés « bases », montrant pour chaque relation quels types de concepts elle peut connecter (signature de relation).
    - Un ensemble de « marqueurs » pour les « sommets concepts »: un marqueur « générique » et un marqueur « individuel »
    - Une relation de conformité, qui définit les contraintes d'association entre un type de concept et un marqueur (si le type « t » est associé au marqueur « m » ⇔ Il existe un individu m qui « est_un » t.

Support : exemple.

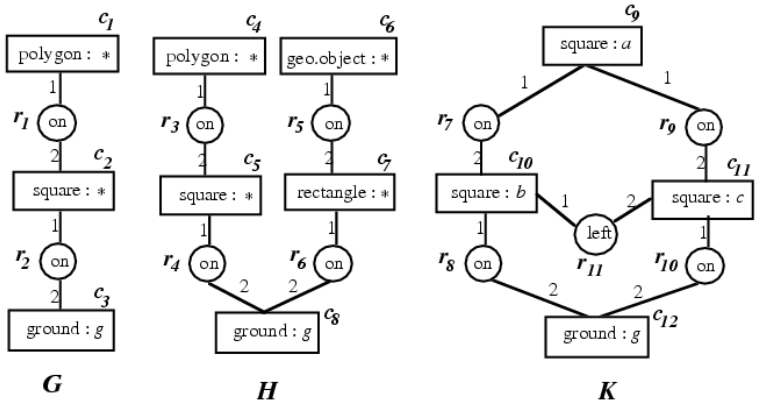




B, Base des relations (graphes étoile)

Tc, Le treillis des concepts

Les S-graphes conformes au support S



Projection et morphisme de S-Graphes

- La projection d'un S-Graphe G sur un S-Graphe G' est une liste ordonnée de « mise-en-correspondance » g de C sur C' (concepts) et f de R sur R' (relations) qui préserve les étiquettes des sommets « r » et peut restreindre les étiquettes des sommets « c ». Restreindre une étiquette d'un sommet « c » revient à restreindre le type de « c » et/ou (si « c » est générique), positionner un marqueur individuel à la place d'un marqueur générique (satisfaisant la conformité de type). Autrement « dit » :

A **projection** from an S-graph $G = (R,C,U,lab)$ to an S-graph $G' = (R',C',U',lab')$ is an ordered pair (f,g) of mappings, f from R to R', and g from C to C', such that:

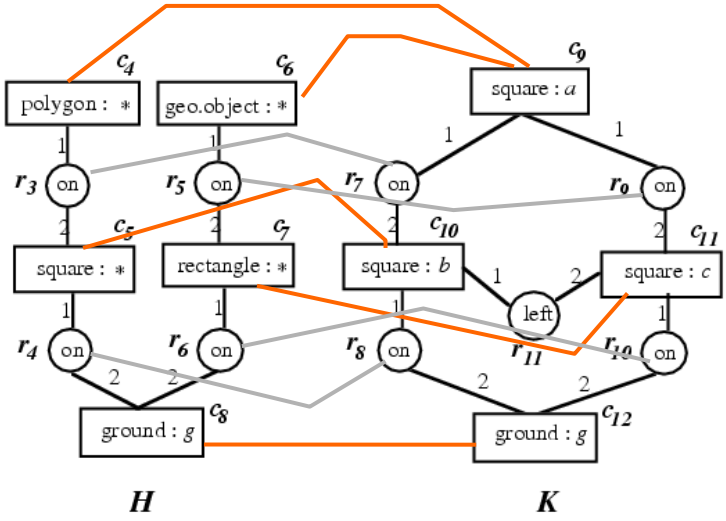
(i) $\forall r \in R$ and $\forall i \in \{1, \ldots, degree(r)\}$, $G_i(r) = c$ implies $g(c) = G'_i(f(r))$,[1]

(ii) $\forall r \in R\ lab'(f(r)) = lab(r)$,

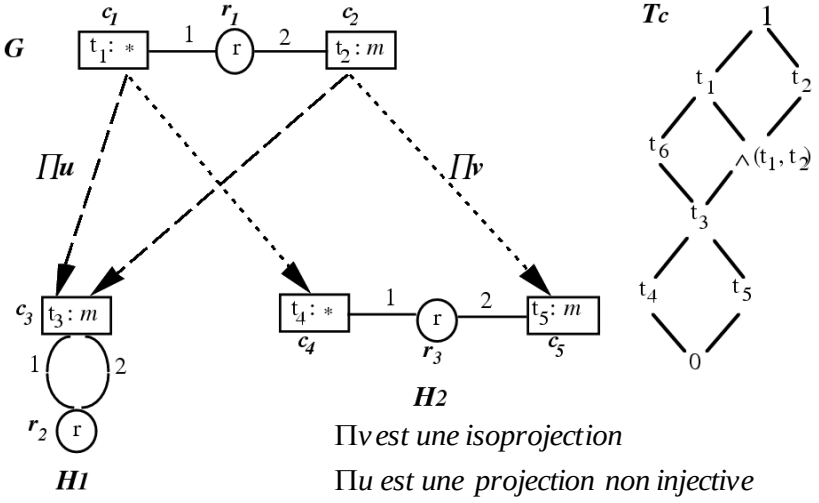
(iii) $\forall c \in C\ lab'(g(c)) \leq lab(c)$.

R={sommets relations r}, C={sommets concepts c}, U={arcs}, lab={étiquettes}.

Projection : exemple



Isomorphisme de S-graphe (iso-projection)



Πv *est une isoprojection*

Πu *est une projection non injective*

Opérations de spécialisation

- Règles de spécialisation
  - Suppression des sommets-r « jumeaux » (simplification). // 2 relations de même type et ayant les mêmes voisins.
  - Restriction élémentaire (restriction) : il s'agit de remplacer une étiquette « e » d'un sommet-c par une autre étiquette « e' » telle que e'≤e et e' conforme au type.
  - Fusion élémentaire : deux sommets-c « c » et « c' » de même étiquette et appartenant à 2 s-graphes sont fusionnés pour donner un nouveau s-graphe.
- G est une spécialisation de H si H appartient à une séquence de spécialisation arrivant à G.
  - La relation de spécialisation s'écrit : ≤

Opérations de généralisation = inverses de la spécialisation

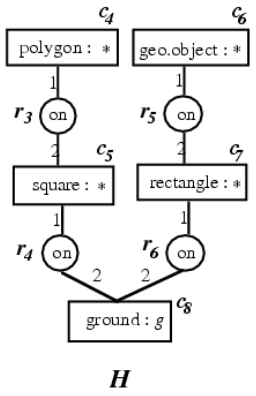
- Règles de généralisation
  - Addition de sommets-r « jumeaux »
  - Extension élémentaire (restriction) : il s'agit de remplacer une étiquette « e » d'un sommet-c par une autre étiquette « e' » de plus haut niveau (pour tout arc r-c avec le label i, type(c) ≤ type du ième voisin du sommet-r du graphe étoile $B_{type(r)}$)
  - Eclatement élémentaire : duplication d'un sommet-c, en deux sommets c1 et c2, avec des étiquettes identiques, et l'ensemble des arcs adjacents à ces nouveaux sommets est une bi-partition de l'ensemble des arcs adjacents à c.
- G est une spécialisation de H si H appartient à une séquence de spécialisation arrivant à G.
  - La relation de spécialisation s'écrit : ≤

Quelques propriétés

- S'il existe une projection de H sur G, alors G≤H
- Si G≤H alors, à toute séquence de spécialisation de H en G, on peut associer une projection de G sur G.
- G≤H si et seulement si il existe une projection de H sur G.
- La relation de spécialisation ≤ est un préordre (la propriété d'antisymétrie n'est pas satisfaite).
- Si G≤H et H≤G alors H et G sont équivalents : H≡G.

Interprétation logique

$\exists x \exists y \exists z \exists t \; \text{polygon}(x) \wedge \text{square}(y) \wedge \text{geo.object}(z) \wedge \text{rectangle}(t)$
$\wedge \text{on}(x, y) \wedge \text{on}(y, g) \wedge \text{on}(z, t) \wedge \text{on}(t, g)$



H

Complexité de la projection

- Le problème de la projection est NP-Complet
- Le problème de la recherche de sous-graphe (projection injective) est NP-Complet
- Le problème de l'isoprojection est NP-Complet (mêm si on a un ordre total sur les étiquettes des sommets-c!).
- Le problème de l'équivalence est NP-Complet.
  - => on peut facilement vérifier qu'un graphe est une projection d'un autre graphe (par exemple), mais le temps pour établir les projections possibles d'un graphe est exponentiel à sa dimension…

  (NP = Non-déterministe Polynomial)

Introduction aux logiques de descriptions

- Langage de représentation de connaissances
- Connaissances du domaine représentées par des entités qui ont une description syntaxique à laquelle est associée une « sémantique ».
- Logiques de descriptions ⇔ Logiques terminologiques ⇔ famille de langages

Logique de descriptions

- Un concept permet de représenter un ensemble d'individus
- Un rôle représente une relation binaire entre individus.
- Concept ⇔ entité générique d'un domaine
- Individu ⇔ une entité singulière, une instance d'un concept.

Principes des LD

- Un concept et un rôle possèdent une *description structurée* élaborée à partir de *constructeurs*
- Une sémantique est associée à chaque description de concept et de rôle par l'intermédiaire d'une *interprétation*.
- Représentation des concepts et des rôles relèvent du niveau *terminologique* ⇔ *TBox*
- Description et manipulation des individus relèvent du niveau *factuel* ou niveau des *assertions ABox*

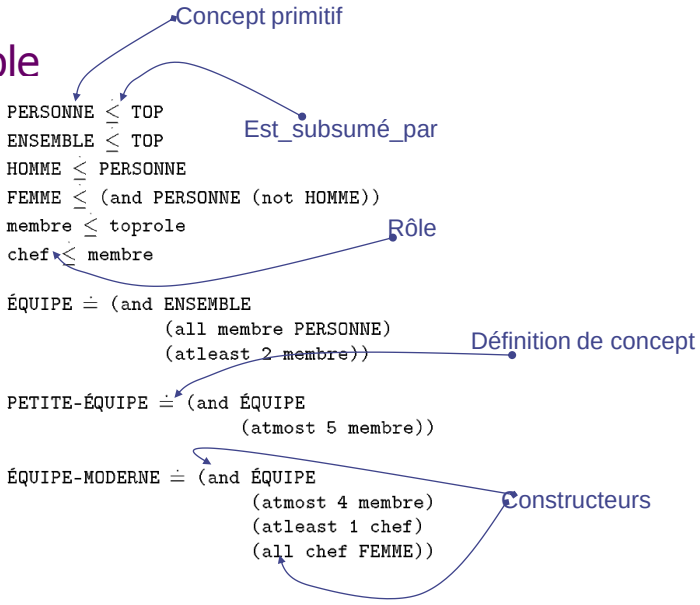
Principes des LD (suite)

- La relation de *subsomption*
  - organise concepts et rôles par niveau de généralité.
  - C subsume D si C est plus général que D au sens que l'ensemble d'individus représenté par C contient l'ensemble d'individus représenté par D
    - ⇔ hiérarchie de concepts et (parfois) hiérarchie de rôles.
- Opérations de base : classification et instanciation.
  - Classification de concepts (ou rôles) : détermine la position d'un concept (d'un rôle) dans une hiérarchie.
    - Construction et maintenance de la hiérarchie est assistée par le processus de classification.
  - L'instanciation permet de retrouver les concepts dont UN individu est susceptible d'être une instance (*sens différent dans les langages à objet*).

Définitions

- Un concept dénote un ensemble d'individus (l'extension du concept)
- Un rôle dénote une relation binaire entre individus.
- La description structurée d'un concept est faite par des constructeurs introduisant les rôles associés au concept et les restrictions attachées à ces rôles:
  - Restrictions en général sur le co-domaine du rôle (le concept avec lequel le rôle établit une relation) et la cardinalité du rôle (nombre minimal et maximal de valeurs élémentaires que peut prendre le rôle : les valeurs élémentaires sont des instances de concept ou des types de base –entier, réel, chaînes de caractères-).
  - Concepts primitifs ⇔ atomes servant de base à la construction des concepts définis.
  - Concept défini, concept possédant une définition.

Exemple



La famille de langage $\mathcal{AL}$

C,D ⟶	A |	
	Top |	$\top$
	Bottom |	$\bot$
	(and C D) |	$C \sqcap D$
	(not A) |	$\neg A$
	(all r C) |	$\forall r.C$
	(some r)	$\exists r$
	syntaxe lispienne	syntaxe allemande

Éléments syntaxiques

Notion d'interprétation

Définition 1 ***(Interprétation)***

Une interprétation $\mathcal{I} = (\Delta_\mathcal{I}, .^\mathcal{I})$ *est la donnée d'un ensemble* $\Delta_\mathcal{I}$ *appelé* domaine de l'interprétation *et d'une fonction d'interprétation* $.^\mathcal{I}$ *qui fait correspondre à un concept un sous-ensemble de* $\Delta_\mathcal{I}$ *et à un rôle un sous-ensemble de* $\Delta_\mathcal{I} \times \Delta_\mathcal{I}$*, de telle sorte que les équations suivantes soient satisfaites :*

$$\top^\mathcal{I} = \Delta_\mathcal{I}$$
$$\bot^\mathcal{I} = \emptyset$$
$$(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$$
$$(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$$
$$(\neg C)^\mathcal{I} = \Delta_\mathcal{I} - C^\mathcal{I}$$
$$(\forall r.C)^\mathcal{I} = \{x \in \Delta_\mathcal{I} / \forall y : (x,y) \in r^\mathcal{I} \rightarrow y \in C^\mathcal{I}\}$$
$$(\exists r.C)^\mathcal{I} = \{x \in \Delta_\mathcal{I} / \exists y : (x,y) \in r^\mathcal{I} \wedge y \in C^\mathcal{I}\}$$
$$(\geq n\, r)^\mathcal{I} = \{x \in \Delta_\mathcal{I} / |\{y \in \Delta_\mathcal{I} / (x,y) \in r^\mathcal{I}\}| \geq n\}$$
$$(\leq n\, r)^\mathcal{I} = \{x \in \Delta_\mathcal{I} / |\{y \in \Delta_\mathcal{I} / (x,y) \in r^\mathcal{I}\}| \leq n\}$$
$$(r_1 \sqcap \cdots \sqcap r_n)^\mathcal{I} = {r_1}^\mathcal{I} \cap \cdots \cap {r_n}^\mathcal{I}$$

Satisfiabilité, équivalence, incompatibilité de concepts

- *Un concept* C *est* satisfiable *ou* cohérent *si et seulement s'il existe une interprétation* $\mathcal{I}$ *telle que* $C^{\mathcal{I}} \neq \emptyset$; C *est* non satisfiable *ou* incohérent *sinon.*
- *Deux concepts* C *et* D *sont dits* équivalents, *ce qui se note* $C \equiv D$, *si et seulement si* $C^{\mathcal{I}} = D^{\mathcal{I}}$ *pour toute interprétation* $\mathcal{I}$.
- *Deux concepts* C *et* D *sont* incompatibles *ou* disjoints *si et seulement si* $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ *pour toute interprétation* $\mathcal{I}$.

Exercice : satisfiabilité des exemples de concepts suivant ?

(1) (and HOMME (some enfant MUSICIEN))
(2) (and FEMME (all enfant HOMME))
(3) (and FEMME (all enfant (and MUSICIEN HOMME)))
(4) (and (atmost 0 r) (atleast 1 r))
(5) (some r (and A (not A)))

Relation de subsomption

Définition 3 ***(Subsomption)***

Un concept D *est* subsumé *par un concept* C *(respectivement* C subsume D*), ce qui se note* $D \sqsubseteq C$ *(respectivement* $C \sqsupseteq D$*) si et seulement si* $D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ *pour toute interprétation* $\mathcal{I}$*. Le concept* C *est appelé le* subsumant *et* D *le* subsumé*.*

Niveau terminologique

- Deux déclarations terminologiques :
  - introduction de concepts primitifs
  - Introduction de définitions
- Déclaration = équation terminologique
  - Nom de concept utilisé une seule fois en partie gauche (pas de circuit terminologique).
  - ⇔ possibilitéde substituer tout nom de concept par sa définition dans n'importe quelle expression conceptuelle (développement des définitions).

Test de subsomption

- Méthodes de type « normalisation-comparaison » (algorithmes NC)
- Méthode dérivée de la méthode des tableaux sémantiques

Normalisation-Comparaison

- Développement et factorisation des définitions
- Production des « formes normales » de description.
- Comparaison

Méthodes des tableaux sémantiques

- La question « Est-ce que C subsume D » est remplacée par « Est-ce que D⊓¬C est non satisfiable ? ».
- La méthode est la réfutation.
- Il est possible de donner à la démonstration une forme d'arbre fini étiqueté qui est appelé « tableau sémantique » où chaque branche mémorise une série d'évaluations possibles pour les énoncés testés.

Base de connaissances terminologique

```
(1) (not BACHELIER) ⊑ (not DOCTEUR)
(2) (all enfant DOCTEUR) ⊑ (all fils BACHELIER)
(3) (all ami (not BACHELIER)) ⊑ (all ami (not DOCTEUR))
(4) (atleast 2 fille) ⊑ (atleast 1 enfant)
(5) (atmost 1 enfant) ⊑ (atmost 2 fille)
```

TBox

```
ÉQUIPE-MODERNE(Trio-54)
HOMME(Antoine)
PERSONNE(Colette)
membre(Trio-54,Antoine)
membre(Trio-54,Patrick)
chef(Trio-54,Colette)
(atmost 3 membre)(Trio-54)
```

ABox

Exemple

$$
\begin{aligned}
\text{(t1)}\ \mathcal{T} = \{&(\texttt{some enseignant COURS}) \sqsubseteq \\
&\quad (\texttt{or PROFESSEUR (and ÉTUDIANT (some diplôme DEA))}),\\
\text{(t2)}\quad &\texttt{PROFESSEUR} \sqsubseteq (\texttt{some diplôme DOCTORAT}),\\
\text{(t3)}\quad &(\texttt{some diplôme DOCTORAT}) \sqsubseteq (\texttt{some diplôme DEA}),\\
\text{(t4)}\quad &(\texttt{and DOCTORAT DEA}) \sqsubseteq \bot\}
\end{aligned}
$$

$$
\begin{aligned}
\text{(a1)}\ \mathcal{A} = \{&\texttt{enseignant}(\texttt{Jean}, \texttt{Cours-ia}),\\
\text{(a2)}\quad &(\leq 1\,\texttt{diplôme})(\texttt{Jean}),\\
\text{(a3)}\quad &\texttt{COURS}(\texttt{Cours-ia})\}
\end{aligned}
$$

Exemple (explication)

(t1) quelqu'un enseignant un cours est soit un professeur soit un étudiant ayant comme diplôme un DEA,

(t2) un professeur est quelqu'un ayant comme diplôme un doctorat,

(t3) quelqu'un ayant comme diplôme un doctorat a forcément comme diplôme un DEA,

(t4) les diplômes de doctorat et de DEA sont différents ; s'il existe une personne P ayant comme diplôme un doctorat, alors il doit exister une instance de DEA qui est associée à P *via* le rôle diplôme.

Exemple - Interprétation

$\Delta_{\mathcal{I}}$ = {Jean,Cours-ia,Dea-de-Jean},
$\text{Jean}^{\mathcal{I}}$ = Jean,
$\text{Cours-ia}^{\mathcal{I}}$ = Cours-ia,
$\text{ÉTUDIANT}^{\mathcal{I}}$ = {Jean},
$\text{PROFESSEUR}^{\mathcal{I}}$ = $\emptyset$,
$\text{COURS}^{\mathcal{I}}$ = {Cours-ia},
$\text{DEA}^{\mathcal{I}}$ = {Dea-de-Jean},
$\text{DOCTORAT}^{\mathcal{I}}$ = $\emptyset$,
$\text{enseignant}^{\mathcal{I}}$ = {(Jean,Cours-ia)},
$\text{diplôme}^{\mathcal{I}}$ = {(Jean,Dea-de-Jean)}

Complexité de la subsomption?

- Si le langage est pauvre (pas de and ni restrict par exemple) alors Complet et Polynomial
- Si le langage est expressif, alors NP-Complet, voire Incomplet et exponentiel…

Représentation des connaissances
Par les schémas (frames)

- Formalismes proposé par Minsky, Winograd, Schank
- Schéma:
  - Paquet d'information représentation des entités et leurs instances
  - Représentation des éléments essentiels d'une entité structurelle
    - Objet
    - événement
    - place
    - tâche
  - Utiles pour classifier; utiles pour représenter les attributs

Schéma

- Les parties de l'entités sont contenues dans des "cavités" (*slots*)
  - en LPO, les cavités seraient des fonctions
- Les cavités intègrent des contraintes sur leur contenu (type et valeur par défaut)
- Chaque cavité comporte une ou plusieurs *facettes*
  - différents points de vue sur l'attribut
    - ex: valeur par défaut, exceptions, type de données
  - comment utiliser l'information représentée par l'attribut; quel comportement est possible
    - ex: si-besoin: méthode de calcul de la valeur de l'attribut
    - ex: si-ajout: que faire si la valeur est ajoutée
- On relie des schémas par leurs cavités
- Mécanismes d'inférence: subsomption, classification

Schéma
Exemple

- TWEETY Est-un CANARI avec
  - couleur JAUNE
  - sexe MALE
  - age 2
  - sante EXCELLENT
  - propriétaire ANNE-SOPHIE
- Formalisé en Logique du 1er ordre
  - CANARI (TWEETY)
  - couleur (TWEETY) = JAUNE
  - sexe (TWEETY) = MALE
  - …
- Forme générale:
  - [<nomDuSchéma> Est-un <type>
    - <nomDeLaCavité> <valeur-cavité>
    - <nomDeLaCavité> <valeur-cavité>

    …]

Source : Christian Pellegrini, Université de Genève
http://cui.unige.ch/DI/cours/1815/slides/12-representationStructuree.pdf

Schéma
Expressivité comparée à la Log. 1er ordre

- Similairement aux réseaux sémantiques:
  - plus efficaces (utilisation de procédures spéciales)
  - plus faciles à interpréter
  - plus faciles à implanter
  - sémantique parfois "floue"
  - moins expressifs (problèmes de la négation, disjonction et quantification)
  - plus expressifs (héritage avec exceptions, attachements procéduraux)

Conclusion sur la représentation de la connaissance

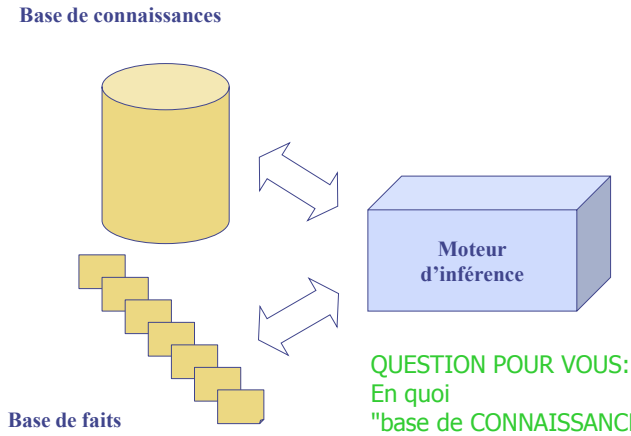
- Richesse expressive ⇔ complexité
- Lecture toujours difficile
- Mécanismes de « calcul » (projection, subsomption, etc.) ne permettent pas des expressions de requêtes toujours « simples » (il faut reformuler).
- Ne règle pas le « frame problem »
  (les difficultés liées au caractère "inexhaustible" d'une description de l'environnement.)

Systèmes à base de connaissances

Objectifs, attentes d'un SBC

- Modéliser des connaissances
  - Inscrire les connaissances en tant que connaissances (pas seulement en tant qu'information) dans un système :
  - Pour « conserver » des savoirs, des savoir-faire
  - QUESTION POUR VOUS:
    - Quelle est la différence entre *information* et *connaissances*?
- Modéliser le raisonnement
  - pour générer des explications, pour assister un expert, pour remplacer un expert
  - Disposer d'un « moteur » permettant d'enchaîner des inférences sur les connaissances ainsi inscrites :
    - Pour « exploiter » les savoirs et savoir-faire ainsi « conservés »

Architecture d'un SBC



QUESTION POUR VOUS:
En quoi
"base de CONNAISSANCES" diffère de
"base de FAITS"

-connaissances ⊂ conn. procéd.
-BdeC est générique au domaine; "faits" = spécifiques à la situation"

Un SBC …

- Inscrit des connaissances issues de l'expertise ou/et de la pratique
  - reproduit notre compréhension du raisonnement suivi par un expert humain lorsqu'il résout le problème.
  - on dit que ses connaissances sont « représentées » dans le système informatique --- difficulté de traduire l'expertise
- Est orienté « résolution de problème »
  - Est spécialisé sur une expertise ou une pratique donnée
- Intègre une représentation symbolique
- Autorise parfois une certaine prise en compte de l'incertitude
  - peut permettre des règles "floues" (avec degré d'appartenance, de véracité)
  - Parce que se fonde sur l'existence d'heuristiques (connaissances spécifiques au domaine qui guident la recherche de solutions)

Rappel: Divers modes de représentation

- Triplets <objet, attribut, valeur>
- Règles
- Réseaux sémantiques
- Schémas (*frames*)
- Logique
  - logique des propositions (d'ordre 0, sans variable)
  - logique des prédicats (d'ordre 1, qui utilise des variables)
  - logiques multivaluées, floues, temporelles, etc.

SBC à partir de règles

- Il s'agit des SBC historiques que l'on appelait initialement « systèmes experts »
  - Les connaissances expertes sont représentées par des règles de la forme
    - Si (prémisses) Alors (conclusions)
      - Prémisses = conditions de déclenchement de la règle
      - Conclusions = effets du tirage de la règle
  - Les connaissances sont déclaratives (révisables en principe)
  - La banque de "**connaissances**" : L'ensemble des **règles** disponibles dans le système – indépendantes d'une situation spécifique
  - La banque de **faits** : Les **faits** décrivent ce qui est vrai dans la situation d'exploitation (faits stables ou faits transitoires)

Règle

- Connaissance servant à **faire le lien** entre des informations connues et d 'autres informations que l'on peut inférer.
  - PoidsPièce < PoidsMin ⇒ PorositéÉlevée (*Interprétation*, *Classification*)
  - PoidsPièce < PoidsMin ⇒ RejeterPièce (*Décision d'action*)
  - peut viser diverses manipulations: relation entre des faits (classification), inférence, directive, stratégie
    - TempsAttente > OpératiionTempsMax ⇒ Offrir de l'assistance
    - TempsAttente > OpératiionTempsMax & AssistanceOfferte ⇒ PrésenterExplication

Base de **règles**

- **R1 :** Si (distance.<.2km)
  Alors (aller.à.pied)
- **R2 :** Si ((non distance.<.2km) ^ distance.<.300km)
  Alors (prendre.le.train)
- **R3 :** Si (non distance.<.300km)
  Alors (prendre.l'avion)
- **R4 :** Si (acheter.un.billet ^ avoir.le.téléphone)
  Alors (téléphoner.à.l'agence)
- **R5 :** Si (acheter.un.billet ^ (non avoir.le.téléphone))
  Alors (aller.à.l'agence)
- **R6 :** Si (prendre.l'avion)
  Alors (acheter.un.billet)
- **R7 :** Si (durée.>.2.jours ^ être.fonctionnaire)
  Alors (non prendre.l'avion)

Banque de **faits**

- F1 : (non distance.<.300km)
- F2 : (avoir.le.téléphone)

Moteur d'inférence (1)

```
ça marche
tant que ça marche
  ça ne marche pas
  boucle sur les Ri
    boucle sur les Fj
      si Ri est de la forme «  Si Fj Alors Fk"
        ajouter Fk à la BdF
        ça marche
      finsi
    finboucle
  finboucle
fintant
```

Ce moteur ne sait pas gérer les conjonctions de faits...

Moteur d'inférence (2)

```
ça marche
tant que ça marche
  ça ne marche pas
  boucle sur les Ri
    boucle sur les Fj
      si Ri est de la forme «  Si Fj Alors Fk"
        ajouter Fk à la BdF
        ça marche
        sinon
           boucle sur les Fl
                 si Ri est de la forme « Si Fj ^ Fl Alors … »
                   ajouter Fm = (Fj ^ Fl) à la BdF
                   ça marche
                 finsi
           finboucle
      finsi
    finboucle
  finboucle
fintant
```

Ce programme boucle !!

Moteur d'inférence (3)

```
ça marche
tant que ça marche
  ça ne marche pas
  boucle sur les Ri
    boucle sur les Fj non marqués
      si Ri est de la forme «  Si Fj Alors Fk"
        ajouter Fk à la BdF
        marquer Fj
        ça marche
        sinon
           boucle sur les Fl
                 si Ri est de la forme « Si Fj ^ Fl Alors … »
                   ajouter Fm = (Fj ^ Fl) à la BdF
                   marquer Fj
                   ça marche
                 finsi
           finboucle
      finsi
    finboucle
  finboucle
fintant
```

Ce programme est correct, donne la réponse en un temps fini et quel que soit l'ordre des Ri et Fj

Gérer les contradictions

- R1 : Si (distance.<.2km) Alors (aller.à.pied)
- R2 : Si ((non distance.<.2km) ^ distance.<.300km) Alors (prendre.le.train)
- R3 : Si (non distance.<.300km) Alors (prendre.l'avion)
- R4 : Si (acheter.un.billet ^ avoir.le.téléphone) Alors (téléphoner.à.l'agence)
- R5 : Si (acheter.un.billet ^ (non avoir.le.téléphone)) Alors (aller.à.l'agence)
- R6 : Si (prendre.l'avion) Alors (acheter.un.billet)
- R7 : Si (durée.>.2.jours ^ être.fonctionnaire) Alors (non prendre.l'avion)

Considérons une nouvelle base de faits :
F1 : ¬distance<300km
F2 : avoir_le_téléphone
F3 : durée>2jours
F4 : être_fonctionnaire

R3 et F1 → F5 : prendre_l'avion
R7 et F3 et F4 → F6 : ¬prendre_l'avion

Solution : tester la présence d'un fait contradictoire dans la BdF avant d'y ajouter un nouveau fait -> signaler le problème à l'utilisateur.

Fournir des « explications »

- Le SBC peut « expliquer » chaque fait produit par la trace de son exécution.
- Les règles et les faits étant exprimés à un haut niveau d'abstraction (symbolique), ces explications sont réputées « lisibles » par les opérateurs humains.
- Certains systèmes, Mycin par exemple, donnent la possibilité d'accéder aux « documents » justifiant l'inscription de telle ou telle connaissance.

Schéma général de fonctionnement d'un SBC par règles

- *Constituer l'ensemble des règles déclenchables*
  - Sélectionner les règles pertinentes au problème
    - En comparant les prémisses de chaque règle avec les faits de la BdF
  - ⇒ "Conflict Set"
- *Choisir la règle à déclencher* selon une stratégie
  - ex.: choisir celle qui demande le moins d'effort, ou le moins de temps, ou est le moins complexe
- *Déclencher les règles*
  - = mise à jour de la BdF avec détection des contradictions,
  - effectuer une action
  - == passer le contrôle à une autre entité.
  - Recommencer….

Multiples règles exécutables
Résolution des conflits

- Si plusieurs règles peuvent correspondre à la situations (selon les faits actuels), le SBD doit disposer de règles de résolution des conflits
  - Première règle qui s'apparie avec les faits avérés
  - Règle ayant la plus haute priorité
  - Règle la plus spécifique
  - Règle faisant référence aux faits les plus récemment introduits dans la base de faits
  - Ne pas déclencher à nouveau une règle l'ayant déjà été

Raisonnement *monotone* ou *non monotone*

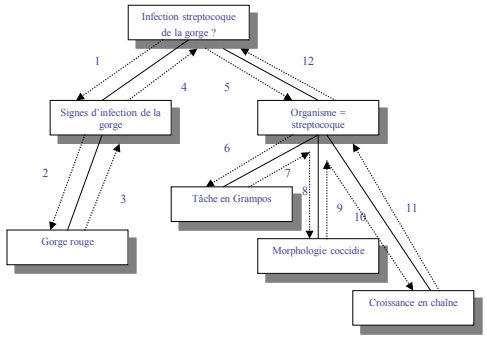
- Monotone:
  - information statique (l'état de véracité de l 'information ne change pas).
  - ex.: Si il pleut, j'ouvre mon parapluie
- Non monotone
  - information dynamique (l'état de véracité de l 'information change).

Raisonnement par chaînage avant, sans but, irrévocable et monotone

- Pas de but ⇔ déclenchement des règles jusqu'à épuisement des faits possibles à produire (ou arrêt).
  - Par intégration immédiate des conclusions des règles
  - En largeur d'abord
- Irrévocable ⇔ déclenchement d'une règle non remis en cause
- Monotone ⇔ les faits produits ne sont pas remis en cause.
- *(avec un but ⇔ une « distance » au but est calculée pour choisir la règle à appliquer)*

Raisonnement par chaînage arrière

- Pour prouver une hypothèse en recherchant les informations pouvant la supporter.
- On sélectionne alors les règles ayant ce but comme conclusion, et vérifie si les prémisses de ces règles font partie de la base des faits. Les prémisses n'appartenant pas à la base de faits deviennent à leur tour des buts à prouver de la même façon.
- Le raisonnement se fait des solutions vers les faits initiaux.



Source : Kaddour Boukerche, GLIC
www.crim.ca/files/documents/evenements/duCRIM/tiDinners/ti-dinnerbis.ppt

Chaînage arrière, par tentatives + monotone

- Un but est assigné au système
- La partie « conclusions » des règles est unifiée avec ce but. En cas de succès, les prémisses de la règle unifiée sont les nouveaux buts assignés.
- Il s'agit donc d'un arbre ET/OU de buts dont les feuilles sont vérifiées ou non par les faits
- Par tentatives ⇔ les buts sont substitués lorsque le chaînage arrière est infructueux.
- Stratégies :
  - Production de buts en profondeur
  - Profondeur d'abord sauf si une règle conclue immédiatement
- Cas de feuilles non vérifiables
  - Poser la question à l'utilisateur
  - Formuler une hypothèse jusqu'à vérification ou contradiction.

Comparaison

	Chaînage avant	Chaînage arrière
Points forts	√ Fonctionne bien lorsque le problème se présente « naturellement » avec des faits initiaux ; √ Produit une grande quantité de faits à partir de faits initiaux très peu nombreux ; √ Adapté à la planification, le contrôle, l'interprétation.	√ Fonctionne parfaitement lors-que le problème consiste à prouver une hypothèse ; √ Il est focalisé sur le but à prouver et pose donc des questions pertinentes, qui ne déroutent pas l'utilisateur ; √ Contrairement au chaînage avant, il recherche dans la base de connaissances les infor-mations intéressantes pour le problème courant ; √ Adapté au diagnostic et à la prescription.
Points faibles	√ Souvent ne perçoit pas certaines évidences ; √ Le système peut poser de nombreuses questions, qui parfois s'avèrent non pertinentes.	√ Poursuit une ligne de raisonnement même s'il s'avère qu'il devrait l'abandonner pour une autres. Les facteurs de croyance et les méta-règles peuvent aider à résoudre ce problème.

Chaînage mixte
Tentatives + monotone

- Tant que des règles sont déclenchables ⇔ chaînage avant
- Puis, on choisit une règle « presque » déclenchable et on essaie de vérifier les prémisses inconnues par chaînage arrière
- En cas de succès, on repart en chaînage avant.

Chaînage mixte
Tentatives, non monotonie

- La partie déclencheur de règles = un but B et des prémisses
  - Ex: Pour prouver B quand F est vérifié, il suffit d'exécuter l'action A et de prouver B'
  - Action = ajouter ou retirer un fait ⇔non monotonie
- Déclenchement en profondeur avec empilement et dépilement des buts
- En cas d'échec, retour arrière avec restauration du contexte initial

Exemple de moteur avec des variables : PROLOG (standard)

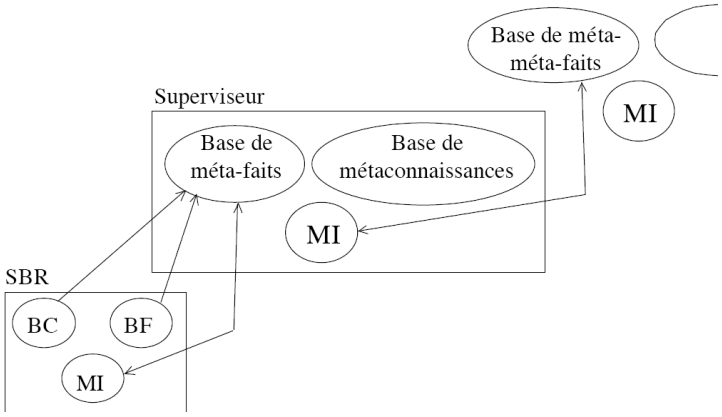
- R1 : papy(X,Y) :- pere(X,Z), pere(Z,Y).
- F1 : pere(pierre,jean).
- F2 : pere(jean,rené).
- B : papy(U,V).

  - Chaînage arrière
  - Tentatives
  - Monotonie

Meta-connaissances

- Connaissances sur les connaissances
  - Mécanisme de « conscience » du raisonnement en cours et de sa validité
  - Possibilité d'organiser progressivement les connaissances par un mécanisme de supervision
  - Mécanisme « empilable » ⇔ Meta-connaissances sur des meta-connaissances
  - Exploitation pour « contextualiser » l'usage d'un SBC – à l'individu, à la situation, au lieu, à l'instant, à la nécessité de précision ou non, etc…

Supervision / Meta-connaissances



Grandes familles d'inférence pour le raisonnement

- Déduction
  - Si ((A -> B est vrai) et (A est vrai) Alors B est vrai.
- Induction
  - Si (P est vraie pour a,b,c de {a,b,c,…,x} Alors (P est vraie pour tout élément de l'ensemble)
- Abduction
  - Si (B est vrai) et que (A->B est vrai) Alors A est vrai.
- Analogie
  - Les A' sont à B' ce que les A sont à B (A' est similaire à A ⇔ B' est similaire à B)
- « Sens commun »
  - Introduction des heuristiques (voir le problème de Send + More)

Quelques coquilles de SE

- Les Coquilles de système expert :
  - EMYCIN
  - CLIPS (C Language Integrated Production System)
  - JESS
  - TMYCIN (http://www.cs.utexas.edu/users/novak/tmycinb.html)
  - FUZZYShell http://rvl.www.ecn.purdue.edu/RVL/Projects/Fuzzy/)
  - FuzzyCLIPS
  - Exshell (en prolog)