DÉPARTEMENT SYSTÈMES INFORMATIQUES FORMELS & INTELLIGENTS

SYSTÈMES EXPERTS - Notes de cours

Editions : 2008

Cycle ingénieur - Deuxième annnée

Maria Malek

Table des matières

La notion de systèmes experts est une notion assez ancienne qui a apparu dans les années 70 avec l'apparition du système expert célèbre MYCIN dont le but était d'aider les médecins à effectuer le diagnostic est le soin des maladies infectieuses du sang. La version de base contenait 200 règles ensuite 300 règles concernant les méningites ont été ajoutées.
Aujourd'hui, les systèmes experts constituent une technologie bien définie faisant partie des *systèmes à base de connaissances*. Les systèmes experts ont comme finalité la modélisation de la connaissance et de raisonnement d'un expert (ou d'un ensemble d'experts) dans un domaine donné fixe.
Pour cela, trois acteurs principaux doivent contribuer à l'élaboration d'un système expert à savoir : l'utilisateur final, l'expert du domaine et l'ingénieur de connaissances. L'interaction entre ces trois acteurs amènera à l'élaboration d'une première version de systèmes experts contenant une base de connaissances, une base de faits et un moteur d'inférence *effectuant une forme définie de raisonnement*.

Dans la première partie du poly, l'architecture générale d'un système expert est présentée ainsi que le processus d'acquisition de connaissances. Dans un premier temps, nous étudions les systèmes experts à base de règles de production, en utilisant de notions de la logique de première ordre. Les algorithmes d'inférence appelés chaînage avant et chaînage arrière sont présentés et détaillés. Nous les comparons en terme de complexité, d'efficacité et de transparence de raisonnement. les différentes stratégies de résolution que peut utiliser un moteur d'inférence sont aussi présentées.

Dans un deuxième temps, le raisonnement incertain est présenté. Les approches probabilistiques classiques comme celle de Bayes est étudiée dans le cadre d'un moteur d'inférence incertain. Une approche plus évoluée est l'approche de Dempster-shafer car elle a beaucoup moins de contraintes que l'approche probabilistique. Ensuite, la théorie des facteurs de certitude et celle des ensembles flous qui s'intègrent plus naturellement avec les règles de productions sont présentées.

Dans la troisième partie du poly, d'autre types de représentation de connaissances sont présentés comme les réseaux sémantiques, les graphes conceptuels et les *frames*. Nous montrons les avantages qu'ils peuvent offrir et notamment du côté de la modélisation de la complexité du monde réel.

Dans la dernière partie du poly, le raisonnement à partir de cas est présenté par opposition aux systèmes classiques utilisant les règles. Nous montrons la souplesse que peut avoir un système de raisonnement à partir de cas, sa capacité d'évoluer et d'enrichir sa mémoire ainsi que sa complémentarité naturelle avec un système à base de règles.

Notons bien que la partie de travaux pratiques associés à ce cours concerne l'apprentissage d'un générateur de système expert appelé CLIPS. Ce générateur est fondé sur une inférence utilisant principalement le chaînage avant. Cet apprentissage vient compléter les notions acquises en deuxième année en PROLOG qui fonctionne principalement en chaînage arrière. De plus, CLIPS permet d'illustrer certaines notions du raisonnement incertain et notamment les facteurs de certitudes et la théorie des ensembles flous.

Chapitre 1

Technologie des systèmes experts

1.1 Introduction

Les experts humains sont capables d'effectuer un niveau élevé de raisonnement à cause de leur grande expérience et connaissance sur leurs domaines d'expertise. Un système expert utilise la connaissance correspondante à un domaine spécifique afin de fournir une performance comparable à l'expert humain. En général, les concepteurs de systèmes experts effectuent l'*acquisition de connaissance* grâce à un ou plusieurs interviews avec l'expert ou les experts du domaine. Les humains qui enrichissent le système avec leurs connaissances ne fournissent pas seulement leur connaissance théorique ou *académique* mais aussi des heuristiques qu'ils ont acquises grâce à l'utilisation de leurs connaissances.

Contrairement à la modélisation cognitive, les systèmes experts n'ont pas comme finalité de s'inspirer des théories du fonctionnement du cerveau humain mais ce sont des programmes qui utilisent des stratégies heuristiques pour la résolution des problèmes spécifiques.

Le raisonnement effectué par un système expert doit être objet à l'inspection, et ceci en fournissant d'information sur l'état de la résolution du problème et des explications sur les choix et les décisions du système.
D'un autre côté, la solution fournie par le système doit être évaluée par un expert humain et ceci dans le but de modifier l'information contenue dans la base de connaissances.

Le but de ce chapitre est de définir la technologie *systèmes experts* en présentant l'architecture d'un système expert type ainsi que le processus de son élaboration.

1.2 Structure et fonctionnement d'un système expert

L'architecture d'un système expert typique est constitué de plusieurs modules qui s'interagissent (voir 1.1) :

L'interface utilisateur sert à simplifier la communication, elle peut utiliser la forme question-réponse,

le menu, le langage naturel etc.

La base de connaissances contient les connaissances concernant la résolution du problème.

Le moteur d'inférence applique une stratégie de resolution en utilisant les connaissances et ceci pour en dériver une nouvelle information.

La base de faits contient les données spécifiques liées à l'application traitée. Elle peut contenir aussi les solutions intermédiaires ou les conclusions partielles trouvées lors de l'inférence.

Le module d'explication permet au système expert d'expliquer son raisonnement.

L'éditeur permet l'édition des connaissances dans la base.

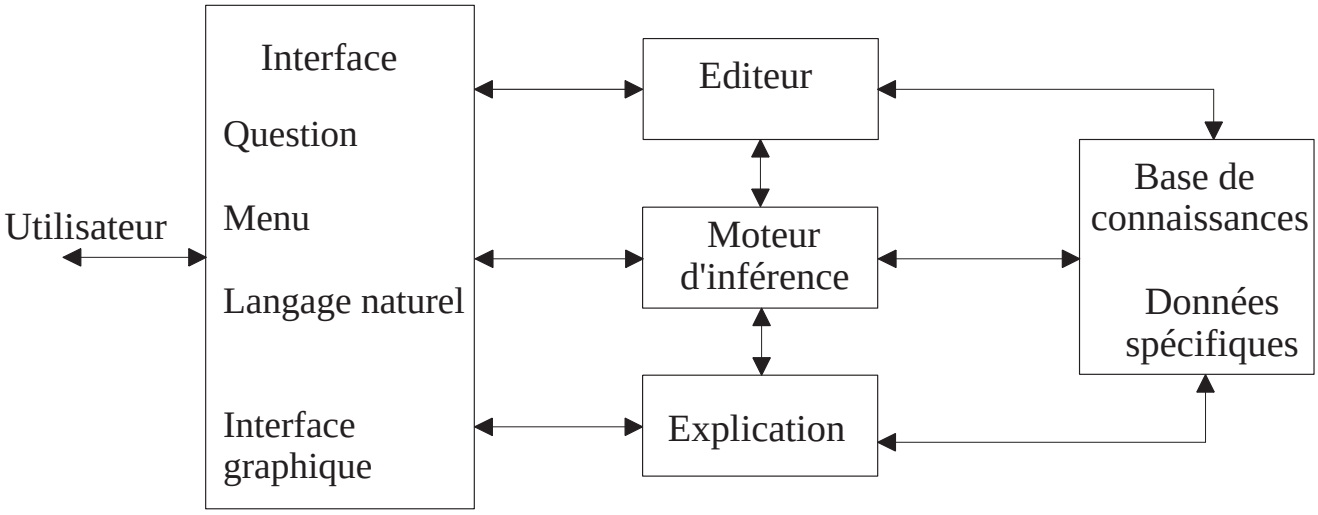

FIG. 1.1 – Structure d'un système expert

Il est très important de remarquer la séparation faite entre les connaissances et l'inférence.

- Cette séparation permet d'utiliser un codage différent, cela nous permet par exemple d'utiliser le langage naturel pour représenter les connaissances (sous forme Si .. ALORS.. par exemple).
- Cette séparation permet au programmeur de se focaliser au codage des connaissances sans se soucier trop de la façon du codage du moteur d'inférence.
- Cette séparation permet aussi de modifier les connaissances sans avoir un effet sur le codage du moteur d'inférence.
- Cette séparation permet également de pouvoir tester plusieurs types d'inférence sur la même base de connaissances.

1.3 Domaines d'application

Les systèmes experts ont été conçus pour résoudre certains types de problèmes comme en médecine, en droit, en chimie, en éducation etc. Les catégories de problèmes abordés par les systèmes experts sont :

- L'interprétation ou la construction d'une description abstraite à partir de données.
- Le prédiction des conséquences à partir de situations données.

- Le diagnostic d'une défaillance à partir d'un ensemble d'observations.
- La conception d'une configuration de composants à partir d'un ensemble de contraintes.
- La planification d'une séquence d'actions pour l'accomplissement d'un ensemble de buts à partir de certaines conditions de départ et en présence de certaines contraintes.
- La réparation d'un dysfonctionnement.
- le contrôle du comportement d'un environnement complexe.

1.4 Problèmes adaptés aux systèmes experts

Les chercheurs ont défini un ensemble informel de critères pour déterminer si un problème est adapté ou non à être résolu par la technologie système expert :

1. Le besoin d'une solution doit justifier le coût et l'effort de la construction d'un système expert.
2. L'expertise humaine n'est pas valable dans toutes les situations dont on a besoin.
3. Le problème peut être résolu en utilisant une technique de raisonnement symbolique.
4. Le domaine est bien structuré.
5. Le problème ne peut pas être résolu en utilisant des méthodes traditionnelles de calcul.
6. La coopération entre experts de domaine existe.
7. Le problème est de taille considérable.

1.5 Processus d'ingénierie de connaissance

Les personnes concernées par le développement d'un système expert sont :

- l'ingénieur de connaissance qui est un expert en langage IA. Son rôle est de trouver les outils et les logiciels nécessaire pour l'accomplissement du projet, d'aider l'expert du domaine à expliciter sa connaissance et d'implanter cette connaissance dans la base de connaissances.
- l'expert du domaine qui fournit les connaissances nécessaires liées au problème.
- l'utilisateur final dont le rôle est de spécifier l'application et de déterminer les contraintes de la conception.

En général, le travail commence par un interview entre l'ingénieur de connaissance et l'expert du domaine. L'ingénieur essaie de comprendre le domaine, d'observer l'expert pendant son travail. Une fois l'expert ait obtenu des informations complètes et précises sur le domaine ainsi que sur la résolution du problème, il pourrait entamer la tâche de la conception du système.

Il choisit la façon de la représentation des connaissances, Il détermine le type du raisonnement et la stratégie utilisée (chaînage avant ou arrière, en profondeur ou en largeur). Il conçoit de même l'interface utilisateur. Le prototype obtenu doit être capable de résoudre correctement un problème typique. Ce prototype doit être testé et affiné par l'ingénieur et l'expert du domaine en même temps.
Le prototype peut être complété au fur et à mesure en ajoutant des nouveaux éléments dans la base de connaissance. Souvent, à la fin de ce travail progressif, l'ingénieur serait amené à refaire une version plus *propre* qui réécrit la connaissance d'une façon plus sommaire.

1.6 Acquisition de connaissance

Dans son rôle, l'ingénieur de connaissance doit traduire l'expertise informelle en un langage formel adapté au mode du raisonnement du système. Plusieurs points doivent être soulevés concernant l'acquisition des connaissances :

1. La compétence humaine n'est pas souvent accessible via la conscience. Avec l'expérience acquise, la compétence et la performance d'un expert s'installe et opère dans l'inconscient. Par conséquence, il est difficile aux experts d'expliciter son savoir-faire.
2. L'expertise humaine prend souvent la forme du *savoir comment* plus que la forme du *savoir quoi*.
3. L'expertise humaine représente un modèle individuel ou un modèle de communauté. Ces modèles sont soumis aux conventions et aux procédés sociaux.
4. L'expertise change et peut subir des reformulations radicales.

A cause de la complexité et de l'ambiguïté posées par le problème de l'acquisition de connaissances, l'ingénieur de connaissances doit avoir un modèle conceptuel lui permettant de faire la liaison entre l'expertise humaine et le langage de programmation, ce modèle constituera ce qu'on appellera *représentation de connaissances*

Chapitre 2

Systèmes à base de règles de productions

Les systèmes à base de règles de production "emmagasinent" la connaissance sous forme de règles : *SI ... ALORS ...* afin d'être le plus proche de la façon naturelle de la représentation des connaissances. La partie entre le *SI* et le *ALORS* s'appelle la partie *prémisses* de la règle. Elle correspond à une conjonction d'expressions qui doivent être vérifiées pour que la règle se déclenche. La partie qui suit le *ALORS* s'appelle la partie *conclusion*. Elle correspond à une conjonction d'actions.

En logique de propositions ou de premier ordre, les règles sont décrites via des clauses contenant des proposition ou des prédicats. Rappelons nous que PROLOG supporte bien ces deux types de modélisation et applique la *stratégie de raisonnement SLD* pour résoudre un but donné, nous allons voir dans la suite que cette stratégie constitue *une inférence en chaînage arrière*.

2.1 Exemple d'une modélisation par règle.

Prenons l'exemple suivant :

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

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

```
;R1
peutAimer(X,bach)<- aime(X,musiqueC), aime(X, maths).
;R2
peutAimer(X,schubert)<- aime(X,musiqueC),romantique(X).
;R3
peutAimer(X,cabrel)<- romantique(X).
```

```
;R4
romantique(X) <- ecritPoeme(X).
```

Remarques Nous avons vu en cours de logique qu'en utilisant la notation en clauses de Horn, nous pouvons construire le modèle minimal de Herbrand ainsi que la résolution SLD. Nous rappelons ces deux mécanismes via l'exemple, en vue de faire la comparaison avec le chaînage avant et arrière ultérieurement.

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

```
aime(toto,muiqueC).
ecritPoeme(toto).
```

Resolution SLD La résolution SLD nous permet de calculer toutes les réponses à une question donnée, en explorant toutes les dérivations possibles, l'arbre obtenu s'appelle l'arbre de dérivation. Les feuilles correspondent à des fins de chemins de parcours correspondant à un échec ou à un succès. Dans le cas d'un succès, la substitution correspondant à la composition des substituions élémentaires trouvées est une solution calculée. Par exemple, une solution possible à la question *peutAimer(toto,Y)* est la substitution $\theta = (Y, schubert)$.

Le modèle minimal d'Herbrand nous rappelons que le modèle permet grâce à l'opérateur conséquence immédiate de retrouver le modèle minimal de Herbrand qui constitue l'ensemble des réponses correctes du programme. Par exemple, pour le programme précédent, nous obtenons le modèle minimal de Herbrand I_E suivant :

T^0= {aime(toto,musiqueC), ecritPoeme(toto)}
T^1= {aime(toto,musiqueC), ecritPoeme(toto), romantique(toto)}
$I_E = T^2$= {aime(toto,musiqueC), ecritPoeme(toto), romantique(toto), peutAimer(toto, schubert), peutAimer(toto,Cabrel)}

2.2 Règles et moteur d'inférence

Dans un système à base de règles, les connaissances sont représentées par des règles. Le moteur d'inférence peut fonctionner en chaînage arrière ou avant. Le chaînage arrière signifie que le raisonnement est guidé par le but tandis que le chaînage avant signifie que le raisonnement est guidé par les données. Nous décrivons dans la suite ces deux mécanismes, et nous les comparons. Notons bien que pour pouvoir décrire ces deux mécanismes nous consultons au fur et à mesure *la mémoire de travail qui nous guide pour le raisonnement*

2.2.1 Le chaînage arrière : raisonnement guidé par le but

En chaînage arrière, le but est placé au début dans la mémoire de travail du système. Le système cherche dans sa base de connaissances les règles dont la conclusion corresponde au but posé. Une des règles est choisie selon une stratégie donnée. Ses prémisses sont empilées dans la mémoire de travail et

deviennent les sous-buts actuels à résoudre. Le système continue à travailler de cette façon jusqu'à ce que tous les sous buts placés en mémoire soient vérifiés.
Les sous-buts peuvent être résolus en posant des questions à l'expert.

Nous donnons l'algorithme permettant d'effectuer le chaînage arrière :

entrée BC la base de connaissances, Mem pile de travail, θ la substitution actuelle.

sortie S un ensemble de substitution

variables locales R, un ensembles vide

- SI Mem est vide ALORS RETOURNER (θ).
  - $elem \leftarrow$ FIRST(Mem).
    - POUR chaque règle $p_1 \wedge ... \wedge p_n \rightarrow C$ tel que $\theta_i \Leftarrow$ UNIFY(C,$elem$) FAIRE
      - $R \leftarrow$ ChaînageArrière(BC, SUBST(θ_i,$[p_1, ..p_n]$),COMPOSE(θ,θ_i)) $\cup R$

Retourner l'union de ChaînageArrière(BC,REST(Mem),θ) pour chaque $\theta \in R$

UNIFY étant une fonction qui calcule l'unificateur le plus général des deux termes et SUBSET est une fonction qui retourne une liste de termes après lui avoir appliqué la fonction θ. Notons bien que l'algorithme Chaînage arrière est un algorithme récursif. Le premier appel est : ChaînageArrière(BC,[but],{}).

Supposons qu'on aimerait bien répondre à la question *peutAimer(toto, X)*.

Au premier appel la mémoire de travail Mem contient *[peutAimer(toto, X)]*, les conclusions des règles R_1, R_2, R_3 peuvent s'unifier avec le but, θ vaut ϵ, R est vide.

Au deuxième appel la mémoire de travail contient *[aime(toto,musiqueC], aime(toto,math)]*, θ vaut {(X,bach)}, cette règle échoue car aime(toto, math) n'est pas dans la base de fait. Par conséquent R reste inchangeable.

Au troisième appel la mémoire de travail contient *[aime(toto, musiqueC], romantique(toto))]*, θ vaut {(X,shubert)}, aime(toto,musiqueC)] étant dans la base, le système cherche à résoudre le but romantique(toto), R reste inchangeable car les prémisses de la règle ne sont pas toutes explorées encore.

Au quatrième appel la mémoire de travail contient *[ecritPoeme(toto))]*, la conclusion de la règle R_4 s'unifie avec le but actuel[1] et la branche aboutit à un succès et donc $R = \{(X, schubert), (X4, toto)\}$.

Au cinquième appel la mémoire de travail Mem contient*[romantique(toto))]* et θ vaut {(X,cabrel)}, la branche aboutit à un succès et donc $R = \{(X, cabrel)\}$.

Fin du premier appel On retourne l'union des substitutions réussies :$\{(X, schubert), (X, cabrel)\}$

Notons bien que le système garde aussi la trace de son raisonnement sous forme d'un graphe de type et/ou (voir figure 2.1). Cela permet au système de savoir à n'importe quel moment le cheminement de l'inférence malgré le changement du contenu de la mémoire de travail. Autrement dit, à n'importe quel moment le système est capable de récupérer :

- la décomposition en sous buts d'un but donné.
- le but père d'un sous but donné.

[1] Nous appliquons un renommage aux variables de la règle R1 pour éviter les confusions.

Nous verrons dans la suite que cela peut qualifier une telle inférence d'une capacité de *transparence de raisonnement et d'une capacité d'explication*.
Ce graphe contient deux types de nøeuds à savoir les nøeuds ou et les nøeuds et. **Les nøeuds ou** correspondent aux règles ayant une conclusion qui peut s'unifier avec le but cherché, chacune de ces règles est le début d'un chemin potentiel de résolution. **Les nøeuds et** correspondent aux prémisses d'une règle donnée qui tente de résoudre un but qui s'est unifiée avec sa conclusion. Ces prémisses deviendront à leurs tours des sous-buts à résoudre. Par exemple, dans la figure 2.1), le nøeud *PeutAimer()* est un nøeud ou et le nøeud *R1* est un nøeud et.

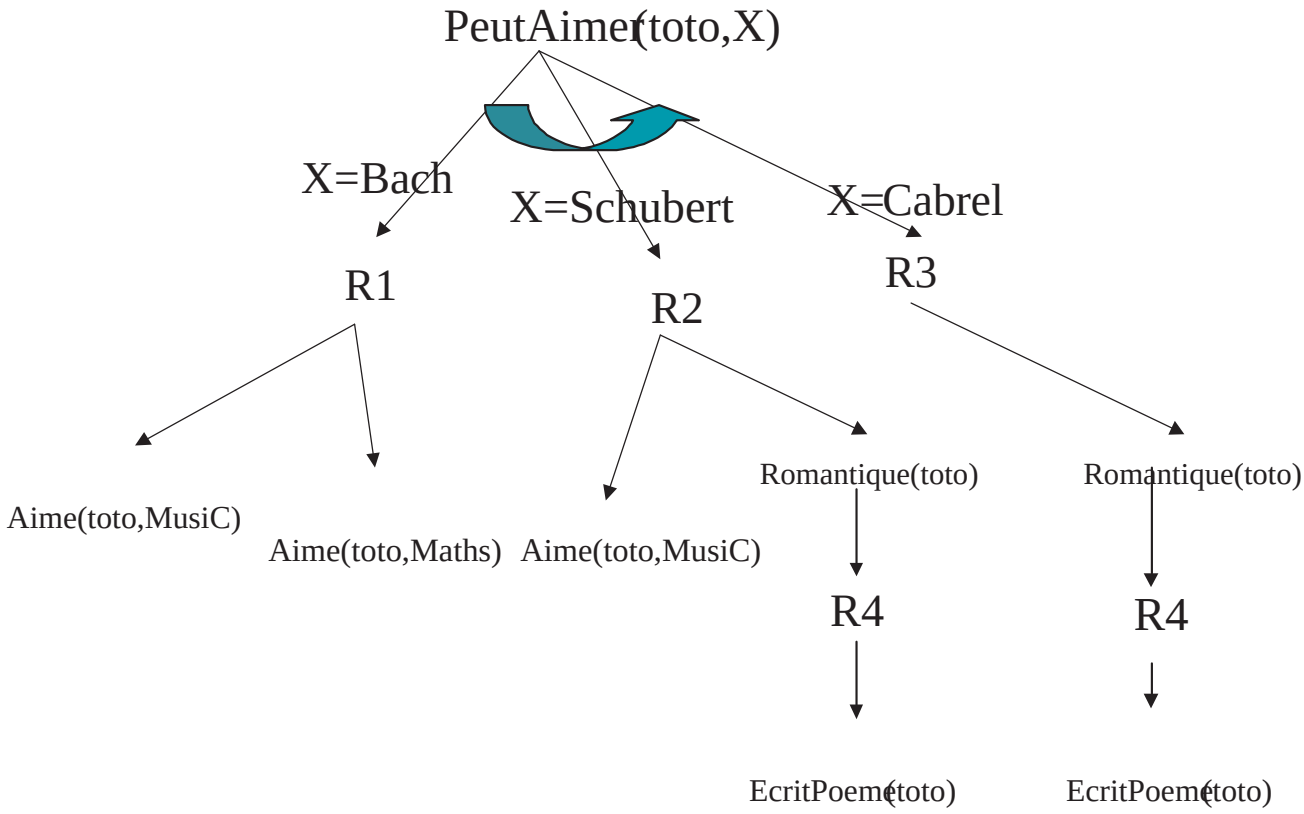

FIG. 2.1 – Le graphe du raisonnement en chaînage arrière

Nous avons montré comment un système basé sur l'inférence par chaînage arrière peut réagir. Pour vérifier le succès ou l'échec d'un chemin de raisonnement, la base de faits est consulté. Une extension de l'algorithme peut être de donner la capacité au système de poser des questions à l'utilisateur quand une réponse positive ne soit pas affirmée par l'existence d'un fait donné dans la base de faits. Nous montrons dans la suite un scénario de questions-réponses, et ceci en se basant toujours sur le exemple cité au dessus, contenant les quatre règles mais en supposant que la base de faits est vide au début. Le but à résoudre étant toujours *peutAimer(toto,X)* :

Le système commence à explorer la première branche à gauche en profondeur jusqu'à l'arrivée au sous but **aime(toto,musiqueC)** *qui n'est pas dans la base de faits et qui ne s'unifie pas avec aucune*

conclusion non plus. Au lieu de considérer cette branche comme échec, le système pose la question à l'utilisateur [2] *vérifiant si toto aime la musique classique. Supposons que la réponse est positive, alors le fait* **aime(toto,musiqueC)** *est ajouté à la base de faits. Le système cherche maintenant à continuer la résolution du but peutAimer(toto,bach), il pose donc la question* **aime(toto,maths)**. *En supposant que la réponse est non, le système passe à la vérification de la deuxième règle. La question* **aime(toto,musiqueC)** *n'est pas posée car le système connaît la réponse qui est codée sous forme de fait. Le sous but suivant à vérifier est* **romantique(toto)**, *donc la question* **ecritpoeme(toto)** *est posée ; par conséquent, le système déduit que* **peutAimer(toto, schubert)** *est vrai. Enfin, le système vérifie bien via le chemin de la règle R_3 que toto puisse aimer également cabrel sans poser des questions car il sait déjà que toto écrit des poèmes.* [3]

Remarquer bien que de fait que le système garde le trace du raisonnement sous forme du graphe et/ou permet d'avoir une suite de questions qui ne paraît pas illogique aux yeux de l'utilisateur [4]. De plus la séquence de questions est *optimale* dans le sens ou le système ne pose pas des questions inutiles car son processus est toujours guidé par la résolution du but père immédiat.

Explication et transparence du raisonnement L'utilisateur peut à n'importe quel moment poser deux types de questions : pourquoi et comment ? Par exemple quand le système pose la question *aime(toto,maths)*à l'utilisateur. l'utilisateur peut demander des justification : *Pourquoi poser cette question ?* Puisque le système garde le trace de son raisonnement sous forme du graphe et /ou, il *sait* que la question est le deuxième fils de la règle numéro 1 donc il peut répondre : *Puisqu'il a été établi que aime(toto,musiqueC), alors si aime(toto,maths), cela signifie que peutAimer(toto,bach).*

L'utilisateur peut ensuite poser la question *Comment a-t-il été établi que toto est romantique ?* Le système peut se placer sur le nøeud correspondant du graphe et présenter à l'utilisateur sa trace de raisonnement.

2.2.2 Le chaînage avant : raisonnement guidé par les données

En chaînage avant, le contenu de la mémoire correspond aux données qui peuvent vérifier certaines prémisses des règles ; ces règles deviennent des règles candidates à être utilisées dans le raisonnement. Dans la suite, le système prend en compte les règles candidates selon un ordre donné et teste si le reste des prémisses sont vérifiées (en consultant la base de faits ou bien en posant des questions à l'utilisateur).

Nous présentons dans la suite l'algorithme chaînage-avant :

[2] Elle peut correspondre à une information manquante dans la base de faits.

[3] Remarquer qu'en cas de réponse négative à la question aime(toto, musiqueC), le système ne pose pas la question aime(toto, maths).

[4] Trois questions ont été posé à l'utilisateur suite auxquelles, le système a déduit que toto peut aimer schubert et cabrel. La suite des questions est la suivante : est ce que toto aime la musique classique ? est ce que toto aime les maths ? et est-ce que toto écrit des poèmes ?

entrée BC la base de connaissances, F un certain fait dans la base de faits

- Pour chaque règle $p_1 \wedge ... \wedge p_n \rightarrow C$ tel que $\theta \Leftarrow$ UNIFY(p_i,F) FAIRE
  - trouver-inférer (BF,BC, $[p_1, .., p_{i-1}, p_{i+1}, ..p_n]$),$C$,$\theta$)

La procedure *trouver-inférer* est donnée par :

entrée BF la base de faits, BC la base de connaissances, prémisses, conclusion, θ

- SI prémisses=[] ALORS
  - chaînage-avant(BC, SUBSET(θ,conclusion))
  - SINON POUR chaque F_1 dans la base de faits BF tel que $\theta_2 \Leftarrow$ UNIFY(F_1,SUBSET(θ,FIRST(premises))) FAIRE
    - trouver-inférer (BF,BC, REST(prémisses), conclusion, COMPOSE(θ,θ_2))

Voyons comment le chaînage avant est effectué sur notre programme contenant les quatre règles et les deux faits.

1. Le fait *aime(toto,musiqueC)* s'unifie avec la première prémisse de la règle $R1$.
2. Le système cherche dans la base de faits un autre fait qui peut s'unifier avec la deuxième prémisse de $R1$. Echec.
3. Le fait *aime(toto,musiqueC)* s'unifie avec la première prémisse de la règle $R2$.
4. Le système cherche dans la base de faits un autre fait qui peut s'unifier avec la deuxième prémisse de $R2$. Echec.
5. Le fait *ecritPoeme(toto)* s'unifie avec la prémisse de la règle $R4$ et le fait résultant de la conclusion *romantique(toto)* s'ajoute à la base de faits.
6. Le fait *romantique(toto)* s'unifie avec la deuxième prémisse de la règle R_2.
7. le système cherche à satisfaire la première prémisse de la même règle et réussit avec le fait *aime(toto, musiqueC)*et donc le fait *aime(toto,schubert)* est inféré.
8. Le fait *romantique(toto)* s'unifie avec la prémisse de la règle R_3 et le fait *aime(toto,cabrel)* est inféré.

Nous remarquons que le système a effectué trois parcours de la base de règles pour arriver à inférer que toto puisse aimer schubert et cabrel.

Nous pouvons procéder de la même manière en supposant que la base de faits est vide au début et que le système pose des questions à l'utilisateur[5]. Dans ce cas l'ordre de questions posées à l'utilisateur est le suivant : *aime(toto,musiqueC), aime(toto,maths), ecritPoeme(toto)*.

Remarquer que si la règle R3 n'existe pas et que si la réponse à la question *aime(toto,musiqueC)* est négative, le système posera quand même la question *ecritPoeme(toto)* [6] ce qui n'est pas le cas en chaînage arrière.

[5]Nous supposons que le système stocke quelque part l'information lui permettant de poser une question sur un fait donné. Dans notre exemple, le système sait qu'il ne peut pas poser des questions que concernant les prédicats aime(X,Y), ecritPoete(X).

[6]qui est une question inutile dans le contexte,

Un autre algorithme basé sur la représentation sous forme de graphe dirigé est proposé dans la littérature (*rete algorithm*). Le principe est de coder dans la base de connaissances *les règles* sous forme de graphe. Les prédicats sont codés par des cercles, les flèches représentent le *et logique entre les différentes prémisses*. Les actions sont codées par des rectangles. Les carrés représentent des contraintes sur les prémisses.
A un cycle donné, un ensemble de faits est injecté dans le graphe. Le résultat de cette inférence serait présenté par des informations qui étiquettent les arcs du graphe. Au cycle suivant, un ensemble de faits resultants seraient injectés dans le graphe, et ainsi de suite.

exemple Par exemple considérons la base de connaissances suivante :

```
a(X),b(X),c(Y) => add d(X)
a(X),b(Y),d(X) => add e(X)
a(X),b(X),e(X) => delete a(X)
```

Et la base de faits suivante : {a(1),a(2),b(2),b(3),b(4),c(5)}.
Le graphe représentant l'activation au premier cycle est donné par la figure 2.2.

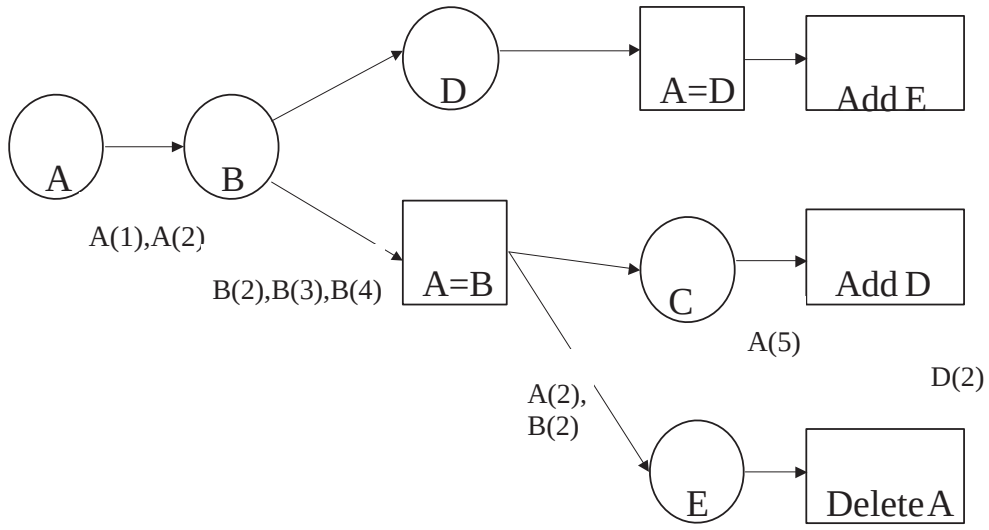

FIG. 2.2 – Le graphe du raisonnement en chaînage avant

L'avantage de cet algorithme par rapport à l'algorithme d'unification est sa capacité de tester en même temps les prémisses communes des règles sans effectuer des unification plusieurs fois. De plus, il donne une représentation graphique de l'ensemble des règles dans la base de connaissances.

2.3 Petite comparaison

Nous remarquons que la capacité d'explication et la transparence est plus développée en chaînage arrière qu'en chaînage avant parce que toutes les questions sont posées pour répondre à des sous buts immédiats selon la trace de raisonnement représenté par le graphe et/ou. Tandis qu'en chaînage avant les

questions sont posées selon l'ordre de présentation des règles dans la base de connaissances et peuvent parfois refléter une incohérence.

En fait, dans l'exemple traité dans ce chapitre, nous avons constaté que l'ordre de questions posées à l'utilisateur est le même. Par contre nous avons montré que le système peut poser des question inutiles en chaînage avant, ce qui n'est pas le cas en chaînage arrière.

D'un autre côté, si nous nous plaçons dans un cadre de modélisation en logique premier ordre, nous remarquons que chaque chemin réussi de résolution en chaînage arrière correspond à une réponse calculée par la résolution SLD. De même, le contenu de la base de faits, après le chaînage avant, correspond bien au modèle minimale de Herbrand.

2.4 Les stratégies de contrôle

En général, les systèmes effectuant les chaînages arrière ou avant appliquent une stratégie de résolution de conflit bien définie. La plus part des systèmes choisissent la première règle déclenchable à partir de l'ordre défini par l'utilisateur. Par exemple, en prolog, la première règle dont la conclusion corresponde au but recherché est activée auparavant. La coupure permet de limiter en effet l'arbre de résolution mais ne change rien à l'ordre des règles.

En Clips, si l'utilisateur ne définit pas de priorité, les règles activées par les mêmes faits, sont également déclenchées d'une façon aléatoire. Pourtant, il existe plusieurs stratégies de résolution de conflit. Par exemple dans certains systèmes les règles les plus spécifiques (ayant les plus de prémisses) sont choisies auparavant. Dans d'autres, les facteurs de certitudes affectés aux règles sont utilisés comme mesures de tri. Certains systèmes privilégient les règles les plus utilisées en classant les règles selon leurs utilités. Dans d'autres, les règles les plus récentes sont prises en priorités.

Nous développons dans la suite quatre stratégies utilisées *en chaînage avant* et plus particulièrement en CLips qui sont : en profondeur d'abord, en largeur d'abord, complexité, simplicité.

La stratégie en profondeur d'abord La base de faits est considérée comme une pile, autrement dit chaque fois qu'un nouveau fait est inféré il est placé au sommet.

Revenons à notre exemple, le dernier fait inséré au premier cycle est *ecritPoeme(toto)*, ce fait déclenche la règle R_4 et le fait *romantique(toto)* est ajouté à la base de fait. Ce fait s'unifie avec la deuxième prémisse de la règle R_2, ainsi qu'avec la prémisse de la règle R_3. La règle R_2 étant à tester, le système cherche un fait qui peut s'unifier avec la première prémisse, et réussit avec *aime(toto,musiqueC)*. Le fait *aime(toto,shubert)* est empilé dans la base de faits, elle ne déclenche aucune règle. Maintenant, la règle R_3 étant en attente de déclenchement, elle est déclenchée et le fait *aime(toto,cabrel)* est empilé dans la base de faits.

La stratégie en largeur d'abord La base de faits est considérée comme une file, autrement dit chaque fois qu'un nouveau fait est inféré il placé à la fin de la file.

La stratégie de simplicité Tout d'abord il faut savoir que les deux stratégies *simplicité et complexité* sont orthogonales avec les stratégies *en largeur et en profondeur*. Autrement dit, on se place avant dans un contexte donné (profondeur ou largeur) et on peut ensuite appliquer la stratégie simplicité, complexité ou aucune ou bien vice versa [7].

Une règle Rs est dite plus simple qu'une règle Rc ssi la partie prémisse de la règle Rs a moins d'éléments que la règle Rc. Un élément peut correspondre à un prédicat mais aussi à un test donné exprimé sous la forme d'une contrainte sur les données ou les prédicats.

Le choix de la stratégie simplicité signifie que si un fait déclenche plusieurs règles en même temps, la règle la plus simple est choisie avant indépendamment de son ordre. Par exemple le fait *romantique(toto)* déclencherait la règle R_3 avant la règle R_2.

La stratégie de complexité agit exactement à l'inverse de la stratégie de simplicité.

[7] on se place avant dans un contexte donné (simplicité ou complexité) et on peut ensuite appliquer la stratégie profondeur ou largeur.

Chapitre 3

Le raisonnement incertain

Nous avons vu dans le chapitre précédent comment effectuer une inférence en utilisant le chaînage avant ou arrière, et ceci à partir d'une base de faits et d'une base de connaissances. Il se trouve que dans une grande majorité d'applications réelles, l'information codée dans le système expert n'est pas toujours certaine autrement dit elle peut avoir un degré de véracité.

Un fait peut être incertain dans le sens où on n'est pas sûr de sa valeur de vérité. Une règle peut être incertaine dans le sens suivant : si les prémisses sont vrais on n'est pas sur d'aboutir toujours à la conclusion.

Nous présentons dans ce chapitre deux grandes approches du traitement de l'incertitude dans les systèmes experts. La première catégorie comprend les approches statistiques comme *la théorie bayesienne*, *les facteurs de certitudes* et *la théorie de Dempster-Shafer*. La deuxième catégorie comprend les approches liées aux ensembles floues.

3.1 La théorie de Bayes

La théorie de Bayes est fondée sur le principe suivant : *chaque hypothèse a une probabilité à priori* $P(H)$. Cette probabilité est modifiée chaque fois qu'on a une nouvelle preuve E [1] qui arrive. Donc la probabilité à posteriori de l'hypothèse H devient $P(H|E) = \frac{P(H\&E)}{P(E)}$ et $P(E|H) = \frac{P(E\&H)}{P(H)}$. Par conséquent on a

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$$

De plus, sachant que $P(E) = P(E|H) * P(H) + p(E|notH)P(notH)$ On peut écrire la formule suivante :

$$P(H|E) = \frac{p_y * p}{p_y * p + p_n * (1 - p)}$$

avec $p = P(H)$, $p_y = P(E|H)$ et $p_n = P(E|notH)$.

[1] evidence en anglais

Exemple Supposons que la probabilité qu'un patient ait la grippe est à priori $P(H)$. Maintenant supposons qu'on a la preuve E que ce patient ait de la fièvre. Nous cherchons à calculer la probabilité à posteriori que ce patient ait la grippe sachant qu'il a de la fièvre.
Nous devons calculer $P(E)$ la probabilité d'avoir la fièvre en général. Pour connaître $P(E)$, on cherche à estimer les probabilités d'avoir la fièvre sachant qu'on a la grippe $P(E|H)$ et d'avoir la fièvre sachant qu'on n'a pas la grippe $P(E|notH)$. Nous pouvons signaler deux inconvénients liés à la théorie de Bayes :

1. Le calcul de $P(E)$ n'est pas toujours évident, sachant que si on ne connaît pas $P(E|notH)$, il faut arriver à connaître toutes les hypothèses qui peuvent influencer E : $P(E|H_i)$. Dans ce cas, on appliquera la formule $P(E) = \sum_{i=1}^{n} P(E|H_i) * P(Hi)$
2. Une contrainte forte que exige cette approche est l'indépendance des variables utilisées. Souvent, dans les applications réelles, il existe une corrélation cachée entre les différentes variables [2].

Dans la suite, nous fixerons un seuil supérieur $M1$ et un seuil inférieur $M2$ pour chaque hypothèse H_i tels que $M1$ et $M2$ ne dépassent pas les limites théoriques que peut prendre la probabilité à posteriori à savoir $P_{max}(H_i) = P(H|E_1, .., E_n)$,$P_{min}(H_i) = P(H|notE_1, .., notE_n)$ respectivement, où $E_1, .., E_n$ sont toutes les preuves possibles qui peuvent supporter l'hypothèse H.

Remarquons bien que tant qu'on a des réponses concernant certaine preuves, $P_{max}(H_i)$ et $P_{min}(H_i)$ changent de valeurs car ils seront calculés en fonction des preuves non renseignées.
Si à un moment $P_{min}(H_i)$ devient supérieur à $M1$ alors l'hypothèse H_i est confirmée, sans tester les preuves qui restent. De même si $P_{max}(H_i)$ à un moment, devient inférieur à $M2$ alors l'hypothèse H est rejetée, sans tester les preuves qui restent.

Dans la suite, nous montrons comment intégrer cette approche dans un moteur d'inférence donné. Les preuves sont demandées à l'utilisateur une par une et selon sa réponse, une nouvelle probabilité à posteriori est associée à H_i, ainsi qu'une nouvelles $P_{max}(H_i)$ et $P_{min}(H_i)$ sont calculée en fonctions des preuves restantes. Nous pouvons supposer que l'utilisateur donne une réponse plus au moins subjective, avec une certitude donnée. Pour cela, nous pouvons calculer la probabilité à posteriori de H après la réponse R de l'utilisateur par :

$$P(H|R) = P(H|E) * P(E|R) + P(H|notE) * P(notE|R)$$

où $P(E|R)$ est une mesure de la certitude que donne l'utilisateur à la preuve E. De même,$P(notE|R)$ est une mesure de la certitude que donne l'utilisateur à $notE$.

3.1.1 Le moteur d'inférence

Nous proposons une méthode qui permet de choisir la question à poser à l'utilisateur [1] [3]. Cette méthode permet de choisir une question concernant une certaine preuve. Nous associons à chaque

[2] Les symptômes que peut avoir un malade ne sont pas forcément indépendants.
[3] L'auteur appelle cette inférence Sideways Chaining.

preuve dans le système [4] une valeur selon

$$Val(E) = \sum_{i=1}^{n} \|P(H_i|E) - P(H_i|notE)\|$$

Ces valeurs ne sont pas inchangeables car ils modifient leurs valeurs selon la probabilité à posteriori de l'Hypothèse H. La preuve ayant la valeur maximale obtenue va constituer la première question à poser à l'utilisateur.

L'algorithme Nous présentons dans ce chapitre l'algorithme nous permettant d'activer l'inférence qu'on appelle *chaînage lateral* [5].

1. Pour chaque hypothèse, affecter une probabilité à priori $p(H_i)$.
2. Pour chaque preuve calculer la valeur val(E_j).
3. Trouver la preuve ayant la valeur maximale : E_{max}.
4. Interroger l'utilisateur sur cette preuve en lui donnant un intervalle de confiance [-5,+5], la réponse étant R.
5. Étant donné R, recalculer $P(H_i|R)$ pour toutes les hypothèses dépendantes de E_{max}.
6. Recalculer aussi les valeurs associées aux preuves différentes restantes.
7. Pour chaque hypothèse Recalculer le min et le max que peut avoir la probabilité à posteriori de H_i.
8. Trouver l'hypothèse ayant le minimum le plus élevé : H_{maxmin}
9. Chercher une hypothèse dont $P_{max}(H_k)$ dépasse la valeur trouvée dans l'étape suivante.
10. En cas d'existence de cette hypothèse, retourner à l'étape 3 sinon arrêter et l'hypothèse H_{maxmin} est confirmée.

3.2 Les facteurs de certitude

Le principe de la théorie des facteurs de certitude est d'affecter à un fait donné ou à une règle donnée un facteur de certitude. Ce facteur sera pris en compte dans l'inférence effectuée. La théorie est fondée sur certaines hypothèses :

1. La somme de la confiance associée à une hypothèse et celle associée à son inverse est égale à 1.
2. Les mesures de confiance correspondent à l'évaluation informelle que donnent les humains à l'information.

Deux mesures doivent être affectées à une hypothèse donnée étant donnée une preuve : la mesure de croyance $MB(H|E)$ et la mesure de non-croyance $MD(H|E)$. Le facteur de certitude associé à cette même hypothèse étant donné la preuve E : est $CF(H|E) = MB(H|E) - MD(H|E)$.

[4] Selon notre terminologie : les preuves correspondent aux faits ; les conclusions correspondent aux hypothèses qu'on essaie de déduire.

[5] l'auteur appelle ce type de chaînage sideways chaining.

Les mesures de croyance varient dans l'intervalle [0,1] et par conséquence : $-1 \leq CF(H|E) \leq 1$.

Nous pouvons associer un facteur de certitude à un fait donné ou à une règle donnée. Nous donnons dans la suite les règles qui permettent la propagation des facteurs de certitudes pendant l'inférence. Supposons que nous associons aux deux faits F_1 et F_2 les mesures de certitudes suivantes : $CF(F_1)$ et $CF(F_2)$. Supposons maintenant que nous avons les deux règles suivantes :

R1 : SI P1 ET P2 ALORS C1

R2 : SI P1 OU P2 ALORS C2

En supposant que le fait F_1 s'unifie avec P1 et F_2 avec P2, la certitude associée à la partie prémisse de R1 est donnée par

$$CF(P1ETP2) = \min(CF(F_1), CF(F_2))$$

de même, la certitude associée à la partir prémisse de R2 est donnée par

$$CF(P1OUP2) = \max(CF(F_1), CF(F_2))$$

Maintenant, en supposant qu'un facteur de certitude est associé à la règle $R : CF(R)$, nous calculons le facteur de certitude associé à la conclusion résultante par $CF(C) = CF(R) * CF(P)$, où CF(P) est le facteur de certitude associé à la partie prémisses de la règle R.

Considérons maintenant le cas où deux règles $R1$ et $R2$ (ou plus) aboutissent à la même conclusion C. Nous appelons $CF_{R1}(C)$, le facteur de certitude calculé par la première règle et $CF_{R2}(C)$, le facteur de certitude calculé par la deuxième règle ; dans ce cas le facteur de certitude final est donné par :

$CF(C) = CF_{R1}(C) + CF_{R2}(C) - CF_{R1}(C) * CF_{R2}(C)$ si les deux facteurs de certitudes initiaux sont positives et par : $CF(C) = CF_{R1}(C) + CF_{R2}(C) + CF_{R1}(C) * CF_{R2}(C)$ si les deux facteurs de certitude sont négatifs

et par : $CF(C) = \dfrac{CF_{R1}(C) + CF_{R2}(C)}{1 - \min(\|CF_{R1}(C)\|, \|CF_{R2}(C)\|)}$ sinon

Remarquer bien que grâce à ces formules, les facteurs de certitude résultants sont toujours dans l'intervalle [-1,1].

Remarquer également que les affectations des facteurs de certitude aux règles et aux faits sont subjectives et dépendent de l'humain. Par contre les calculs qui permettent la propagation de ces facteurs de certitude obéissent aux règles précédentes.

3.3 Les ensembles flous

Une *donnée floue* est une pièce d'information qui peut prêter à une ambiguïté dans le sens où elle correspond à un intervalle de données fixe dans un univers de discours. Par exemple les concepts *jeune, grand* sont des concepts flous. Par exemple avoir un age de 35 ans ne signifie pas qu'on est jeune à 100% ni qu'on n'est pas jeune à 100%.

Mathématiquement parlant un ensemble flou *A* défini sur un univers de discours U est caractérisé par la fonction d'appartenance :

$$\mu_A : U \rightarrow [0, 1]$$

Par exemple, le terme flou jeune est caractérisé par :

$$\{(25, 1.00), (30, 0.8), (35, 0.6), (40, 0.4), (45, 0.2), (50, 0.00)\}$$

Les variables linguistiques Une variable linguistique est décrite par l'ensemble de termes flous qu'elle pourrait prendre. Par exemple la variable age est décrite par l'ensemble des termes flous : {jeune,vieux}, chaque terme étant un ensemble flou. D'un autre côté, nous appelons fait flou un fait contenant le couple : {variable linguistique, valeur floue} (voir figures 3.1 et 3.2).

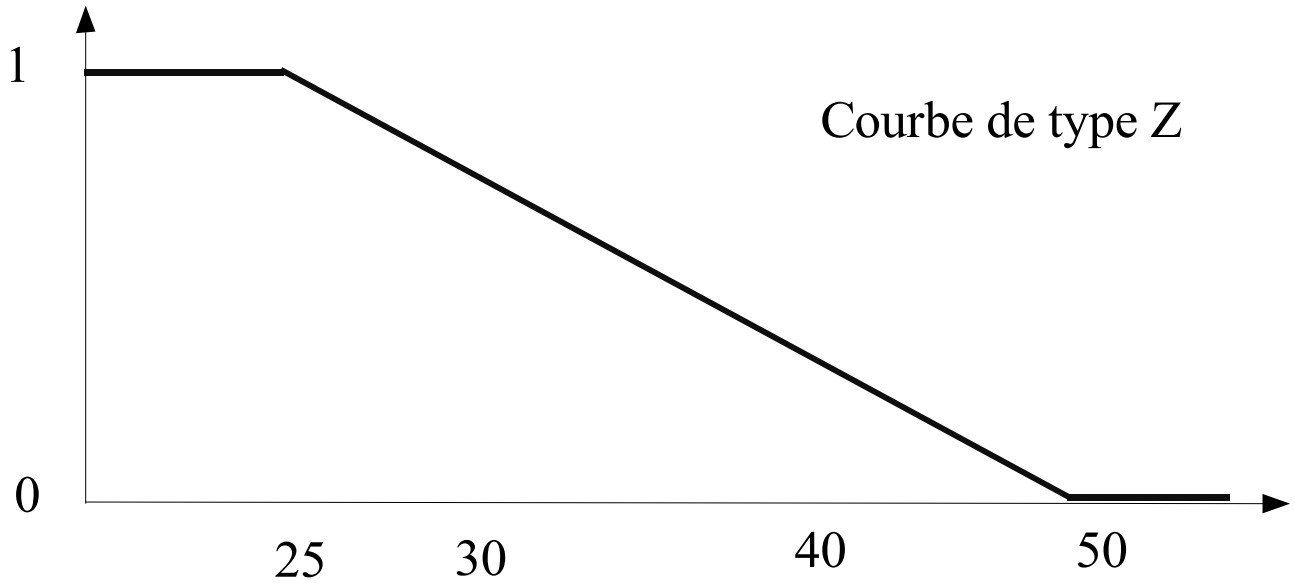

FIG. 3.1 – L'ensemble flou (age jeune)

Les ensembles flous sont munis de quatre opérations fondamentales :

L'intersection Soient F_A et F_B deux ensembles flous définis sur le même univers de discours U. L'intersection des deux ensembles flous est un nouvel ensemble flou qui sera défini par $inter(F_A, F_B)(u) = min(\mu_{F_B}(u), \mu_{F_A}(u))$ (voir figure 3.3)

L'union Soient F_A et F_B deux ensembles flous définis sur le même univers de discours U. L'union des deux ensembles flous est un nouvel ensemble flou qui sera défini par $union(F_A, F_B)(u) = max(\mu_{F_B}(u), \mu_{F_A}(u))$ (voir figure 3.4)

Le complément Soit F_A un ensemble flou défini sur l'univers de discours U. Le complément de cet ensemble sera défini par $complement(F_A)(u) = 1 - \mu_{F_A}(u)$ (voir figure 3.5)

Le produit Soient F_A et F_B deux ensembles flous définis sur les univers de discours U et V. Le produit des deux ensembles flous est un nouvel ensemble flou qui sera défini sur $U * V$ par $produit(F_A, F_B)(u) = f(\mu_{F_B}(u), \mu_{F_A}(u))$, f peut être la fonction min, max etc.

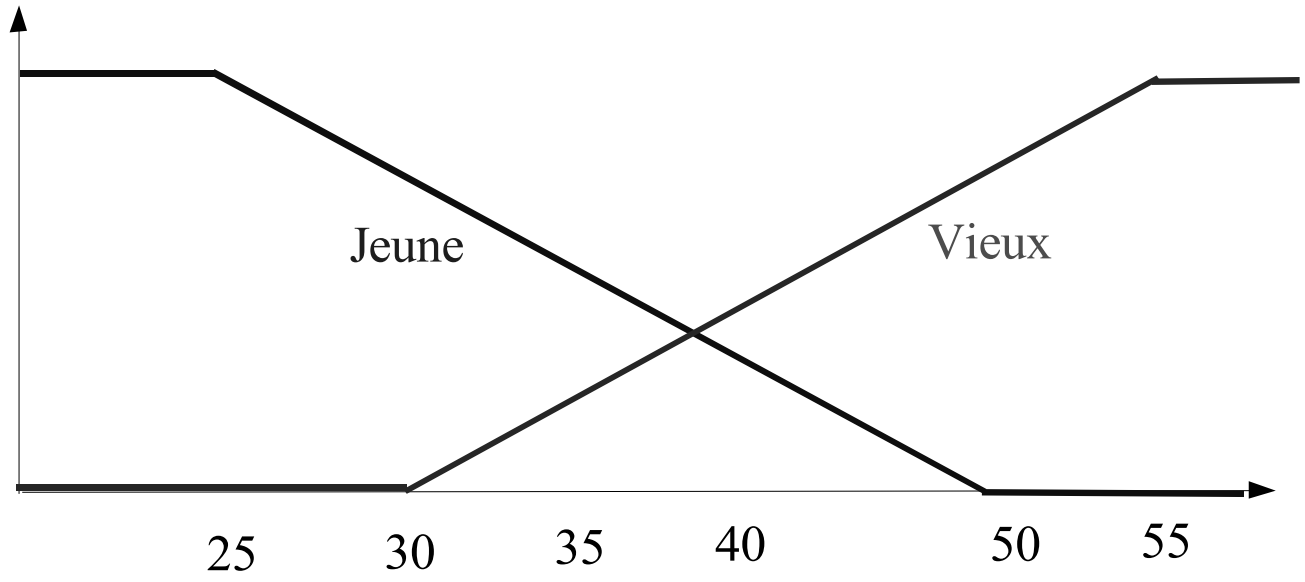

FIG. 3.2 – Les deux ensembles flous (age jeune) & (age vieux)

Règles de type CRISP-CRISP Les règles de type CRISP-CRISP sont des règles qui ne contiennent pas de termes flous dans leurs parties prémisses ni dans leurs parties conclusions. Par contre, elle peuvent avoir des facteurs de certitude. Les règles de propagation de certitudes vues dans la section précédente sont applicables à ce type de règles.

Règles simples de type FUZZY-CRISP En général, les règles de type FUZZY-CRISP sont des règles dont la partie prémisses contient un test sur un fait composé d'une variable linguistique avec une valeur floue. La conclusion ne contient pas de terme flou. Nous définissons une règle *simple de type FUZZY-CRISP* comme étant une règle qui comporte un seul test flou dans sa partie prémisses. Supposons que F_A représente l'ensemble flou du fait flou A, et que F_B représente un test sur la même variable linguistique contenue dans A dans une règle donnée.
Nous calculons la similarité entre le fait et la règle selon :

$$S = P(F_B|F_A)$$

si $N(F_B|F_A) > 0.5$ et par

$$S = (N(F_B|F_A) + 0.5) * P(F_B|F_A)$$

sinon.

N,et P étant les mesures de nécessité et de possibilité respectivement calculées par (voir figures 3.6 & 3.7) :

$$P(F_B|F_A) = max(min(\mu_{F_B}(u), \mu_{F_A}(u))$$

Et

$$N(F_B|F_A) = 1 - P(COMP(F_B)|F_A)$$

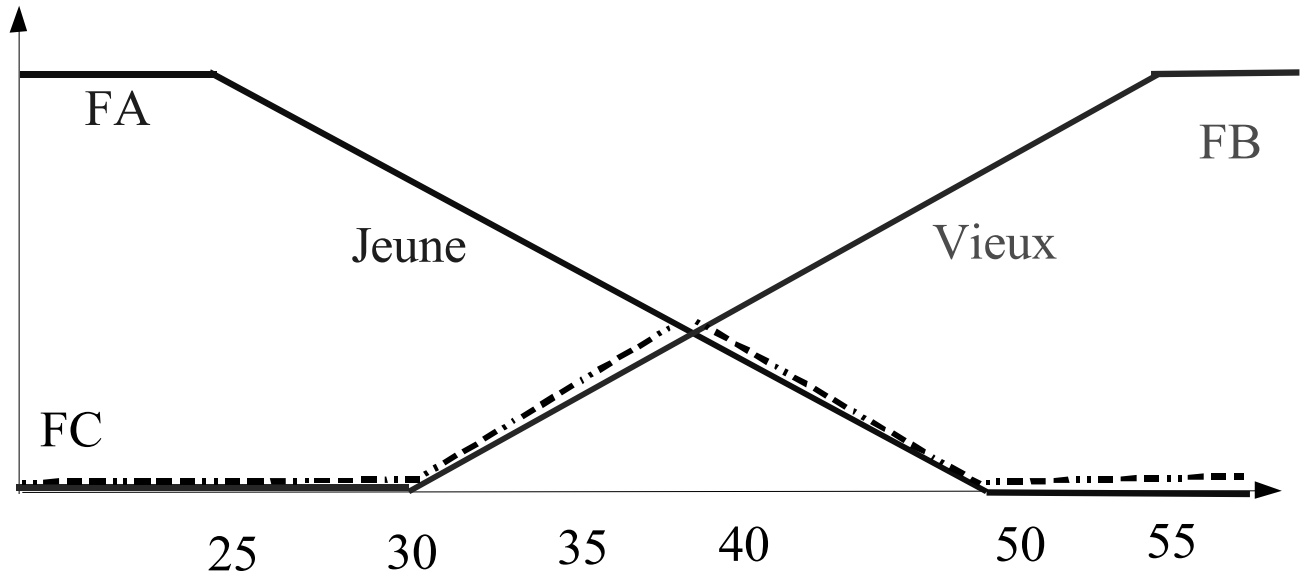

FIG. 3.3 – Intersection des deux ensembles flous (age jeune) & (age vieux)

$COMP(F_B)$ étant le complément de F_B et est décrit par

$$mu_{COMP(F_B)}(u) = 1 - \mu_{F_B}(u)$$

Une fois l'inférence est effectuée le facteur de certitude propagé à la conclusion : $CF(C)$ de la règle sera multiplié par la mesure de similarité précédente.

Règles simples de type FUZZY-FUZZY En général, les règles de type FUZZY-FUZZY sont des règles qui contiennent dans leurs parties premisses des tests sur des faits flous et dans leurs parties conclusions des manipulations sur de faits floues [6]. De même, *une règle simple de type FUZZY-FUZZY* est une règle dont la partie prémisse est composée d'un seul test flou, et la partie conclusion contient une manipulation sur un seul fait flou. En fait, la règle est vue comme une relation entre les deux faits (fait prémisse et fait conclusion).
Soit F_A l'ensemble flou représentant un fait existant dans la base de faits sur le quel est effectué un test dans la partie prémisse de la règle R. Soit F_C le fait dans la conclusion de la règle floue R. La règle représente une relation floue $R = F_A * F_C$. On peut par exemple, représenter cette relation par la fonction d'appartenance suivant :

$$\mu_R(u, v) = min(\mu_{F_A}(u), \mu_{F_C}(v))$$

Soit maintenant F_B le test sur la valeur de la variable linguistique contenu dans le fait F_A alors la fonction d'appartenance $\mu_{F_{C'}}$ de la conclusion résultante C' sera donné par :

$$\mu_{F_{C'}}(v) = max_u(min(\mu_{F_B}(u), \mu_R(u, v))$$

[6] Souvent, dans la partie conclusion, on ajoute un ou plusieurs faits flous dans la base de faits.

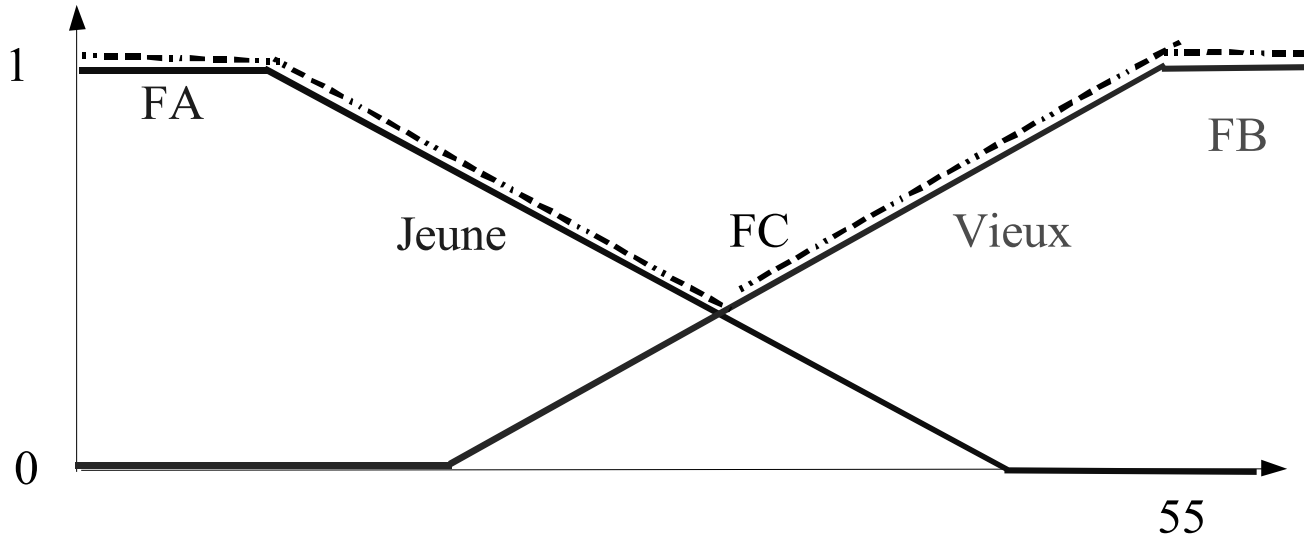

FIG. 3.4 – Union des deux ensembles flous (age jeune) & (age vieux)

En simplifiant, on retrouve (voir figure 3.8) :

$$\mu_{F_{C'}}(v) = min(Z, \mu_{F_C}(v))$$

avec :

$$Z = max(min((\mu_{F_B}(u), \mu_{F_A}(u))$$

Les règles floues complexes Nous distinguons deux cas :

Il existe plusieurs faits flous dans la conclusion Dans ce cas la règle *SI A alors C1 et C2 et ..Cn* est traitée exactement comme la séquence des règles *SI A alors C1*,*SI A alors C2*,.. *SI A alors Cn*.

Il existe plusieurs tests flous dans la partie prémisse Si nous avons la règle *Si* B_1 *et* B_2 *alors C* B_1 est un test sur le fait flou A_1, B_2 est un test sur le fait flou A_2, alors les ensembles flous F_{C1} et F_{C2} résultant de l'application des deux règles : *Si* B_1 *alors C* et *Si* B_2 *alors C* sont calculés. Le résultat global est

$$F_C = F_{C1} \cap F_{C2}$$

autrement dit :

$$\mu_{F_C}(u) = min(\mu_{F_{C1}}(u), \mu_{F_{C2}}(u))$$

Combinaison de résultats Supposons que nous avons une règle de type FUZZY-FUZZY et dont une partie de la conclusion : C1 concerne l'ajout d'un fait flou dans la base de faits. Supposons qu'avant le déclenchement de cette règle, il existe déjà dans la base de fait un fait F1 qui porte sur la même variable linguistique que C1. Dans ce cas, le déclenchement de la règle précédente implique l'ajout du fait C2 dans la base de fait qui remplacera F1, et ceci selon :

$$F_{C2} = F_{C1} \cup F_{F1}$$

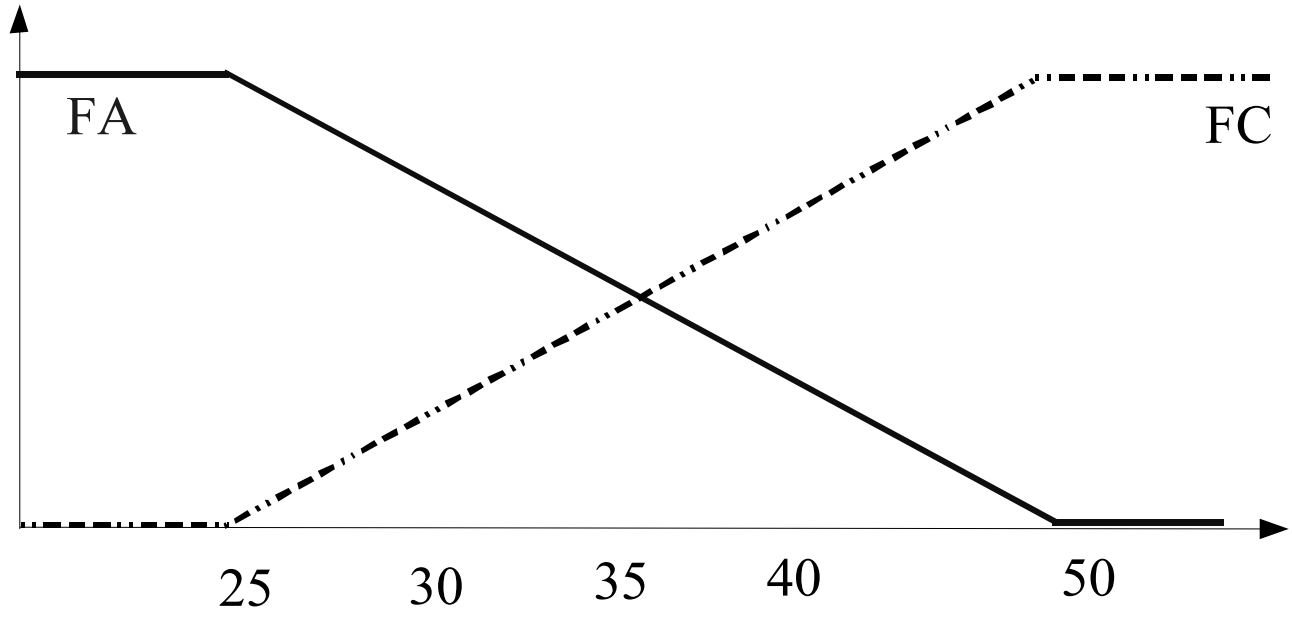

FIG. 3.5 – Le complément d'un ensemble flou

Autrement dit :

$$\mu_{F_{C2}}(u) = max(\mu_{F_{C1}}(u), \mu_{F_{F1}}(u))$$

La notion de defuzzification Dans certaines applications, comme les applications de contrôle il est nécessaire et intéressant de représenter l'ensemble flou par un seul point qui sera considéré comme le point le plus représentatif de l'ensemble. Nous appelons cette opération *defuzzification*. Une méthode est la méthode du centre de gravité, elle est effectuée par le calcul du centre de gravité de tous les points, en pondérant chaque point par son degré flou. Une deuxième méthode qui peut être utilisée est la méthode du maximum, elle suppose que le point le plus représentatif est celui ayant le degré flou maximal. Dans le cas de l'existence de plusieurs points, la moyenne est calculée. Noter bien que les définitions données ici correspondent bien à des ensemble flous définis dans des univers de discours finis. Il est possible d'étendre ces définitions bien sur les univers infinis et ceci en utilisant la notion de l'intégrale mathématique.

3.4 La théorie de Dempster -Shafer

La théorie de Dempster-Shafer est fondée sur la modélisation de l'incertitude par un intervalle de probabilités plutôt que par une simple probabilité numérique. Elle intègre deux notions : la notion de *l'incertitude* et le notion de *l'ignorance*. La théorie de Dempster-Shafer associe à une proposition donnée l'intervalle : [Confiance, Plausibilité]. Cet intervalle est compris entre 0 et 1. La plausibilité est définie par $Pl(S) = 1 - conf(\overline{S})$.
Cet intervalle permet de mesurer le niveau de confiance de certaines propositions, mais également la

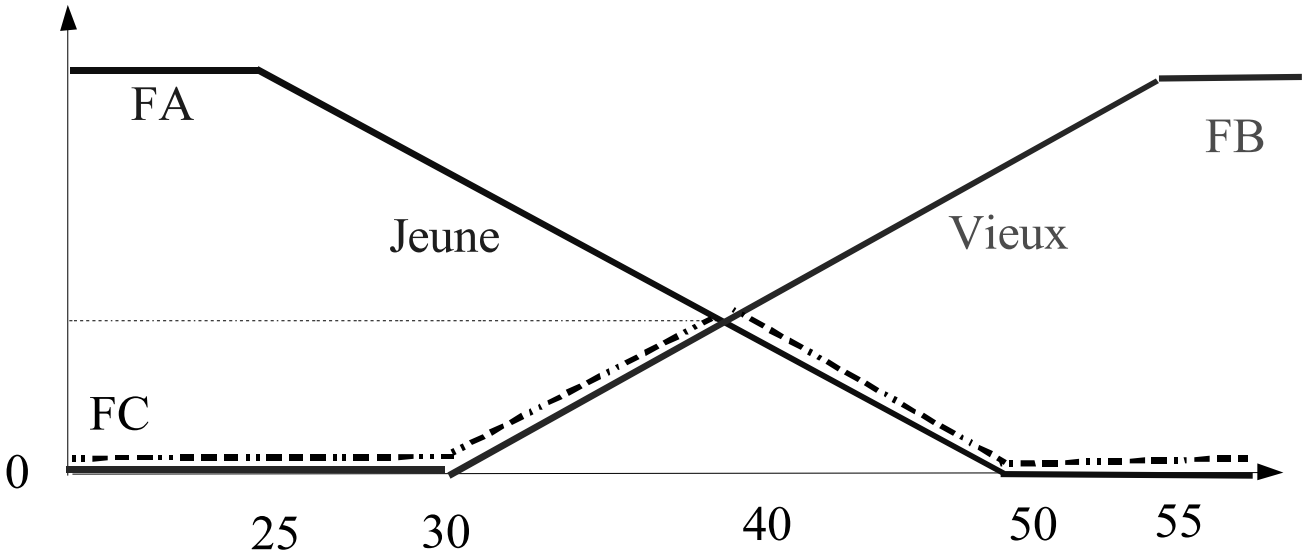

FIG. 3.6 – La possibilité d'avoir FB connaissant FA

quantité d'information disponibles. Supposons que nous ayons trois hypothèses A,B et C . Si aucune information n'est disponible on associe à chacune l'intervalle [0,1]. Noter bien que cela est différente de l'approche bayesienne qui aurait affecté à chaque hypothèse la valeur 0.33.

Contrairement à la théorie des probabilités, qui affecte à l'absence d'une hypothèse H la valeur : $1 - P(H)$, la théorie de Dempster-Shafer n'oblige pas que la croyance en une hypothèse soit assignée à l'ignorance d'une autre hypothèse. Elle demande que l'assignation soit effectuée uniquement au niveau des sous-ensembles de l'environnement.

Exemple Supposons que nous avons l'environnement $V = \{A, B, C\}$ ou A correspond à un avion de ligne, B est un avion de bombardement et C est un avion de chasse. Un ingénieur transmet un signal à un avion. Si c'est un avion ami, ce dernier envoie son code d'identification, sinon, s'il transmet rien il est considéré comme avion ennemi.
Supposons que l'avion ne pouvant envoyer un signal de réponse indique un degré de confiance de 0.7. L'assignation de la confiance se fait alors sur le sous ensemble $\{B, C\}$ de V : $m_1(\{B, C\}) = 0.7$ m_1 étant l'observation d'un seul capteur. Le reste de la confiance se trouve au niveau de l'environnement : $m_1(V) = 0.3$ et non pas au niveau de l'avion ami comme le proposait la théorie des probabilités. Nous appelons cette nouvelle mesure *la masse* pour faire la différence avec la probabilité[7].

Mathématiquement parlant, pour un environnement donné V la masse est une fonction :

$$m : P(V) \rightarrow [0, 1]$$

De plus, $m(\phi) = 0$ et $\sum_{X \in P(V)} m(X) = 1$

[7]car $m(V)$ n'est pas nécessairement égale à 1

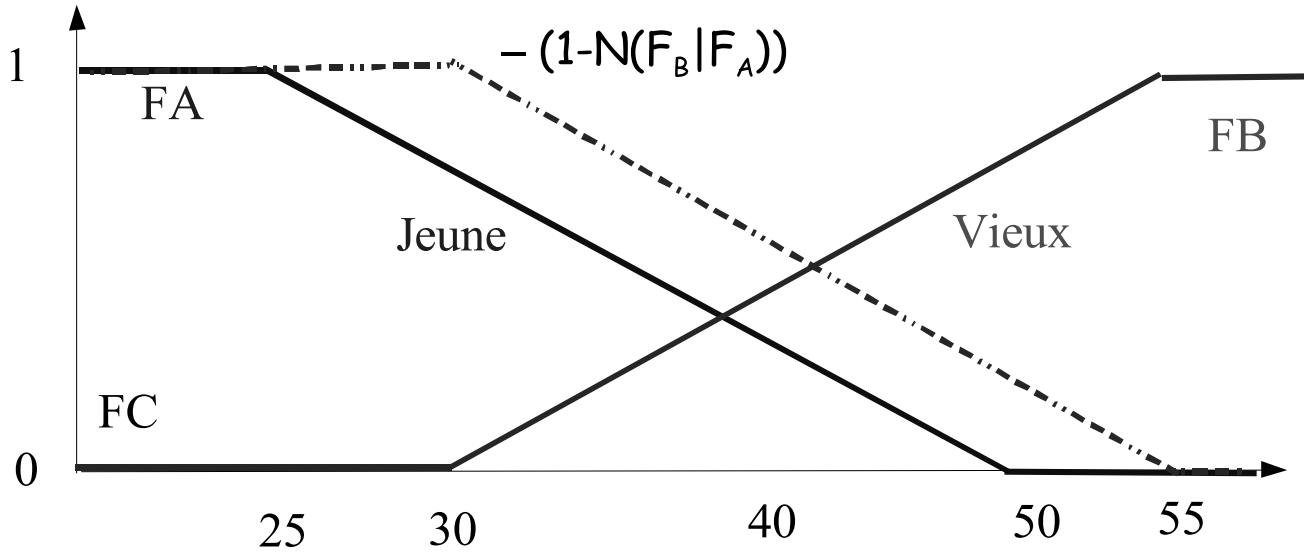

FIG. 3.7 – La possibilité d'avoir le complément de FB connaissant FA

Dempster a proposé une formule de combinaison de différentes masses pour produire une masse résultante représentant un certain consensus. Ce résultat tend à favoriser l'acceptation en incluant les masses se retrouvant dans les intersections :

$$m_1 + m_2(Z) = \sum_{X \cap Y = Z} m_1(X) * m_2(Y)$$

Par exemple, si on suppose qu'on dispose d'un autre capteur pour la détection des avions avec : $m_2(\{B\}) = 0.9$ et $m_2(V) = 0.1$ nous avons :

$$m'_3(\{B\}) = m_1 + m_2(\{B\}) = 0.63 + 0.27 = 0.90$$

mesure trouvée pour l'avion bombardier ;

$$m'_3(\{B, C\}) = m_1 + m_2(\{B, C\}) = 0.07$$

mesure trouvée l'avion bombardier ou chasseur ;

$$m'_3(\{V\}) = m_1 + m_2(\{V\}) = 0.03$$

mesure trouvée pour la totalité de l'environnement.

A partir de ces trois résultats, nous pouvons donner l'intervalle de confiance suivant : [0.9, 1] à l'élément B car la somme des confiances des trois sous ensembles est égale à 1 et les trois sous ensembles incluent B.

D'une manière générale la borne inférieure de l'intervalle de confiance liée à un sous-ensemble donné est calculée par $conf(X) = \sum_{Y \in X} m(Y)$. Par exemple la confiance affectée au sous ensemble

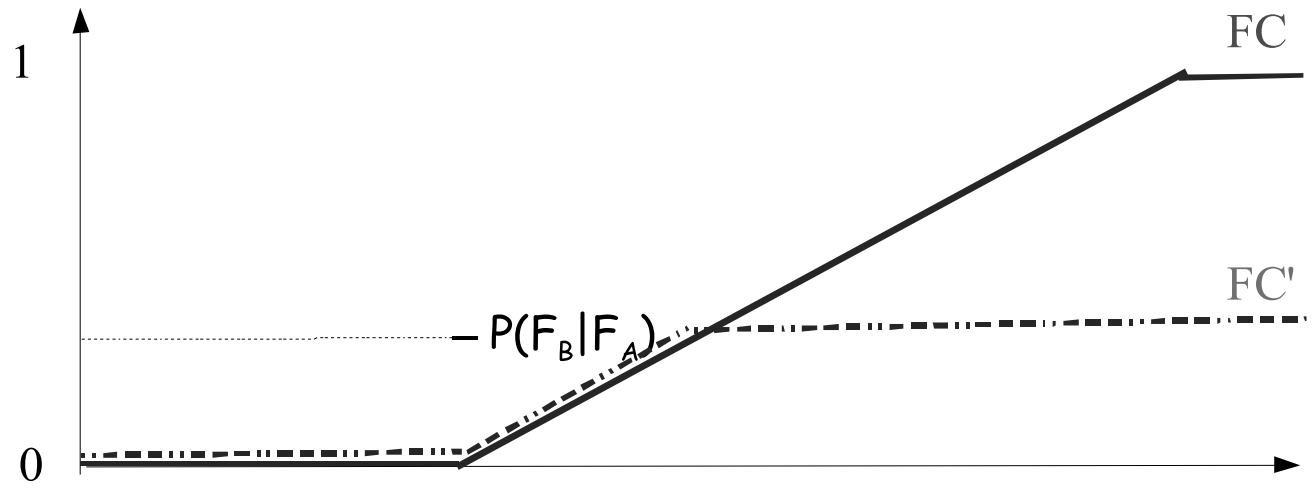

FIG. 3.8 – Règle de type FUZZY-FUZZY : SI B Alors C, A étant un fait flou défini sur le même domaine que B. C' serait le résultat de l'activation de cette règle par A

B,C est $conf'_3(\{B,C\}) = m'_3(\{B,C\}) + m'_3(\{B\}) + m'_3(\{C\}) = 0.07 + 0.9 + 0 = 0.97$

De même la plausibilité associée à un sous ensemble donnée est $pl(X) = 1 - conf(X')$, X' étant le complément de X. Par exemple, $pl(\{B,C\}) = 1 - conf(\{A\} = 1 - m'_3(\{A\}) = 1$. Par conséquent, l'intervalle de confiance associée à $\{B,C\}$ est $I(\{B,C\}) = [0.97, 1]$.

Nous pouvons voir la plausibilité en X comme étant le degré avec lequel l'évidence ne parvient pas à réfuter X. Ainsi, nous définissons *l'ignorance* de X par $igr(X) = pl(X) - conf(X)$, et *le doute* de X par : $dbt(X) = conf(X') = 1 - pl(X)$.

Normalisation de la confiance Supposons dans l'exemple suivant, que nous ayons un troisième capteur avec les masses suivantes : $m_3(\{A\}) = 0.95$ et $m_3(V) = 0.05$. La combinaisons des trois évidences nous donnent les résultats suivants :

$$m_1 + m_2 + m_3(\{A\}) = 0.0285$$

$$m_1 + m_2 + m_3(\{B\}) = 0.045$$

$$m_1 + m_2 + m_3(\{B,C\}) = 0.0015$$

$$m_1 + m_2 + m_3(\{V\}) = 0.0015$$

$$m_1 + m_2 + m_3(\{\phi\}) = 0$$

Nous remarquons que la somme de toutes les masses est inférieure à 1. Pour résoudre ce problème nous divisons chaque combinaisons élémentaire par $1 - k$ avec $k = \sum_{X \cap Y = \phi} m_1(X) * m_2(X)$.

Dans l'exemple $k = 0.855 + 0.0665 = 0.9215$; et les nouveaux résultas sont :

$$m_1 + m_2 + m_3(\{A\}) = 0.363$$

$$m_1 + m_2 + m_3(\{B\}) = 0.573$$
$$m_1 + m_2 + m_3(\{B, C\}) = 0.045$$
$$m_1 + m_2 + m_3(\{V\}) = 0.019$$

Par conséquence, la confiance associée à {B} est $conf(\{B\}) = 0.573$ la plausibilité est $pl(\{B\}) = 1 - conf(\{A, C\})$ avec $conf(\{A, C\}) = m_1 + m_2 + m_3(\{A, C\}) + m_1 + m_2 + m_3(\{A\}) + m_1 + m_2 + m_3(\{C\}) = 0 + 0.363 + 0 = 0.363$.

Et donc, l'intervalle de confiance associé à B est $I(\{B\}) = [0.573, 0.637]$.

Retenons bien que la formule générale de la règle de Dempster de combinaison est :

$$m_1 + m_2(Z) = \frac{\sum_{X \cap Y = Z} m_1(X) * m_2(Y)}{1 - k}$$

.

$$k = \sum_{X \cap Y = \phi} m_1(X) * m_2(X)$$

.

Chapitre 4

Représentation de connaissances

4.1 Les langages de représentation de connaissances

Quand on conçoit un système à base de connaissances, la première mission est de déterminer les objets du domaine réel - porteurs de connaissances - et les relations entre ces objets, et de les représenter en utilisant un langage formel.

Ce langage doit aussi supporter les inférences effectuées sur ces objets. Le résultat de l'inférence doit correspondre à une action ou à une observation sur le domaine.

Plusieurs questions liées au domaine, doivent se poser lors du choix d'un langage de représentation, par exemple : est-ce que le langage doit représenter des classes, de hiérarchie entre classes, d'exception, des relation de type cause - conséquence, de l'incertitude, etc.

Il est nécessaire de distinguer entre les *schémas de représentation* et le *moyen de l'implémentation*. Les langages de programmation servent de moyens d'implémentation. En général, les schémas de représentation de connaissances sont plus spécifiques que le calcul des prédicats ou des langages comme Prolog ou Clips. Prolog et Clips restent les langages de programmation les plus adaptés pour l'implémentation de ces schémas, mais il est toujours possible d'utiliser des langages plus conventionnels comme C, Java, etc.

Dans la littérature, plusieurs schémas de représentation de connaissances ont été proposés et implémentés. Nous pouvons les classer en quatre catégories :

les schémas de représentations logiques La logique est utilisée pour représenter la base de connaissances. Les règles d'inférence appliquent cette connaissance aux éléments du problèmes. La logique du premier ordre (le calcul des prédicats) est un exemple de cette catégorie. Prolog est un langage idéal pour l'implémentation de ces représentations.

les schémas de représentations procédurales Les connaissances sont représentées par un ensemble d'instructions pour la résolution d'un problème. Par exemple dans les systèmes à base de connaissances chaque règle de la forme *si .. alors* est une procédure qui permet de résoudre un sous but du problème.

les schémas de représentations en réseaux sémantiques Les réseaux sémantiques représentent les connaissances comme étant un graphe. Les nøeuds correspondent aux objets (concepts dans le

domaine). Les arcs représentent les relations ou les associations. On parle des réseaux sémantiques et de graphes conceptuels.

les schémas de représentations en objets structurés Ces schémas de représentation peuvent être considérés comme une extension des réseaux sémantiques, en admettant que le nøeud peut correspondre à une structure complexe qui représente un objet encapsulant des attributs, des valeurs et encore des procédures pour effectuer des tâches données. On parlera des *Scripts*, des *Frames* et des *objets*.

4.2 Aspects de la représentation de connaissances

Plusieurs questions doivent se poser par rapport à la représentation des connaissances pour un domaine spécifique.

Que représenter exactement ? Une fois les objets du monde sont déterminés, la vraie question est qu'est ce qu'il faut représenter. Quelle relation doit-on mettre en avant ? Par exemple, en calcul de prédicats on peut dire que la voiture est rouge : couleur(voiture, rouge). Mais comment pourrait on dire que la voiture de X est plus rouge que celle de Y.

A quel niveau de granularité Par exemple, en calcul de prédicats et en réseaux sémantiques, les objets sont représentés par des simples symboles. Par contre, La représentation par objets structurés nous permet de définir des objets complexes en encapsulant des attributs pour désigner un seul objet.

Comment peut - on faire la différence entre extension et intention ? l'extension d'un concept est défini par l'ensemble d'objets. L'intention signifie la définition des structures décrivant un concept donné. Par exemple, le concept étudiant à l'Eisti est défini par extension par l'ensemble sandrine, stéphanie, corinne, etc.. Ce même concept peut être défini par intention par : *Etudiant qui a préparé des classes préparatoires et qui a présenté sa candidature à l'Eisti etc.*. En fait, un système à base de connaissances contient une représentation par intention tandis qu'une base de données contient une représentation par extension. Ces deux représentations doivent être liées en modélisant la correspondance entre données dans la base de données et schémas de représentation par intention dans la base de connaissances.

comment représenter la méta-connaissance ? La méta-connaissance correspond à la connaissance ayant comme objectif de traiter les schémas de connaissances sur le domaine. Puisque les données traitées sont des connaissances, on appelle ce type de connaissances méta-connaissance.

Comment organiser les connaissances en hiérarchie ? Les représentations par objets structurés constituent un exemple de la hiérarchisation des connaissances (voir plus loin).

4.3 Les réseaux sémantiques

4.3.1 Limitations de la représentation logique par rapport à l'expression des humains

L'utilisation de la logique pour formaliser le raisonnement humain qui s'exprime en utilisant un langage donné posent certains problèmes. Prenons par exemple la phrase suivant *si X est un oiseau alors X est capable de voler*. En calcul de prédicat cette phrase est traduite par :

$$(oiseau(X) \rightarrow peutvoler(X))$$

qui est logiquement équivalent à

$$(\neg peutvoler(X)) \rightarrow \neg oiseau(X))$$

A partir de cela, supposons que X est le chat Félix, Félix ne peut pas voler et Félix n'est pas un oiseau ; donc la valeur de vérité de la deuxième formule est vraie. Par conséquent, on en déduit que le fait que félix est un chat qui ne peut pas voler constitue une modèle pour l'hypothèse : *tous les oiseaux volent* ! !

De plus, le calcul des prédicats nous permet de formaliser des expressions qui n'ont aucun sens mais qui soient vrais quand même ! ! comme l'expression :

$$(2 + 3 = 8) \rightarrow couleur(ciel, vert)$$

Cette expression est vraie car $0 \rightarrow 0$ est vraie.

4.3.2 Théorie d'associations

La théorie d'associations définit un sens d'un objet en terme de réseau d'association avec d'autres objets de la base de connaissances.

La figure 4.1 montre comment créer des associations entre les objets. En 1969, Collins et Quillian ont modélisé le stockage et le traitement de l'information chez l'humain avec un réseau sémantique. Ils ont mis en place plusieurs types d'associations :

- les associations décrivant les caractéristiques ou les propriétés de l'objet ou du concept. Ces associations sont libellées avec : *est,a*
- les associations décrivant les actions que peut entamer l'objet. Ces associations sont libellées avec : *peut,..*
- Les associations décrivant l'héritage libellées par *est un*.
- Les associations décrivant les exceptions libellées par *n'est pas, n'a pas , ne peut pas*.

L'organisation de ce réseau a été inspirée des expériences psychologiques menées sur les humains. Par exemple, Collins et Quillian ont constaté que le temps de réaction à la question *est ce que le canary peut voler ?* est plus long que celui de la question *est ce que le canary peut chanter ?* L'explication qu'ils ont fournie est que : l'information est toujours stockée dans le niveau le plus abstrait. Citons un autre exemple concernant les expériences, le temps de réaction qui correspond à la question *est ce que l'autruche respire ?* est plus long que celui de la question *est ce que l'autruche vole ?* Une explication possible serait que les exceptions sont toujours stockées dans le niveau le plus spécifique.

Le fait d'utiliser l'héritage présente plusieurs avantages :

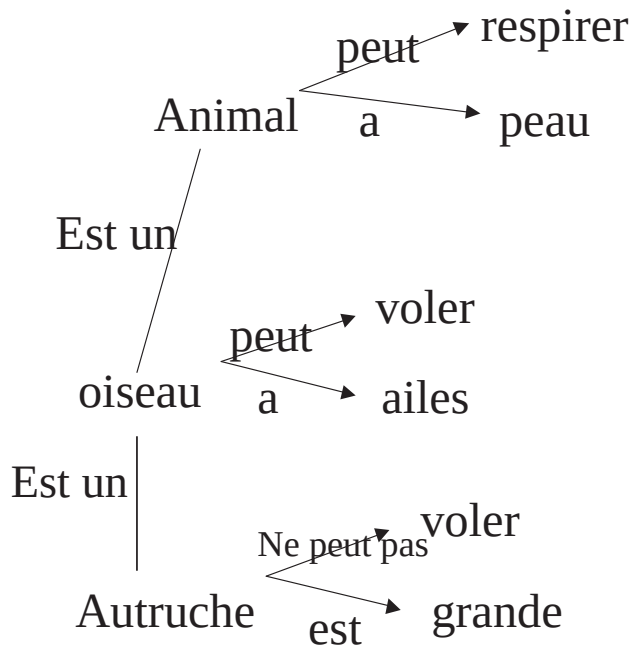

FIG. 4.1 – Théorie d'association : exemple

– réduction de la taille de la base de connaissances en " factorisant " les connaissances communes.
– Maintenance et consistance de la base de connaissances surtout quand on ajoute des nouvelles concepts ou objets.

4.3.3 Besoin de standardisation

Malgré le fait que les réseaux sémantiques sont plus riches en notions (l'héritage, les associations, les exceptions, etc.) que le calcul logique, il n'est pas évident qu'ils puissent supporter la complexité de n'importe quel domaine. La plus part des travaux qui ont suivi ceux de Quillian ont focalisé sur la définition d'un ensemble plus riche. L'utilisation d'un formalisme inspiré par le langage naturel dans la mise en øeuvre d'un réseau sémantique a été beaucoup utilisé et amené à la réalisation de bases de connaissances plus générales et plus consistantes qu'avant. En 1973, Simmons a souligné l'intérêt de la mise en øeuvre des relations standardisées en définissant une *structure de cas* des verbes en anglais. La structure d'un cas inclut *des agents*, *des objets*, *des instruments*, *des lieux* et *du temps*.

Il y a eu des travaux encore plus ambitieux et complexes comme le travail de Schank sur la théorie de dépendances conceptuelles. Cette théorie est fondée sur quatre primitives de conceptualisation : les actions symbolisées, les objets, les modificateurs d'action et les modificateurs d'objets.

Les actions symbolisées par *ACTs* peuvent être de plusieurs types selon leurs significations sémantiques, nous citons quelques exemples :

ATRANS transformer une relation ;

PTRANS transformer un lieu physique d'un objet ;

PROPEL appliquer une force physique à un objet.

Les modifications apportées aux relations sont libellées par : passé(*p*), futur(*f*) et conditionnelles (*c*) et ceci dans le but de leur associer un temps d'action.

4.4 Les graphes conceptuels

4.4.1 Introduction

Nous détaillons dans cette section, un cas particulier des réseaux sémantiques : les graphes conceptuels. Les nøeuds d'un graphe représentent des concepts ou des relations entre concepts. Les graphes conceptuels n'utilisent pas des arcs libellés mais des nøeuds représentant les relations entre concepts. Par exemple dans la figure 4.2, les concepts *chien* et *noir* sont représentés par des *nøeuds concepts* et la couleur est représentée par *un nøeud relation*. Les concepts sont représentés par des *rectangles* et les relations par des *ellipses*. Les nøeuds concepts représentent des objets concrets comme *voiture, téléphone* ou des objets abstraits comme *beauté*. Les nøeuds de relation indique une relation concernant un ou plusieurs concepts. Une relation d'arité n est représentée par un nøeud de relations ayant n arcs.

FIG. 4.2 – Les graphes conceptuels

4.5 Types, individus et noms

Dans un graphe conceptuel, chaque rectangle est libellé par un type qui indique le type (ou la classe) d'individus représentés par ce nøeud. Les types sont organisés en hiérarchie. Par exemple, le type *chien* est un sous type du type *carnivore*. Chaque rectangle est libellé par le type de l'individu et son nom (ou son identificateur) séparés par un *:* (voir figure 4.2). Chaque individu possède un *identificateur unique* précédé par le symbole #. La différence entre les identificateurs et les noms est que deux individus différents du même type peuvent avoir le même nom mais jamais le même identificateur. D'un autre côté, dans un graphe conceptuel nous pouvons utiliser le symbole générique *, pour indiquer un concept générique. Par convention, un nøeud contenant le type *chien* est équivalent au nøeud contenant *chien :*X* qui indique le concept générique du chien sans indiquer un individu donné. Pour résumer, un *nøeud concept* peut contenir un *concept individuel* ou un *concept générique*.

4.5.1 Hiérarchie de types

La hiérarchie de types est un ordre partiel défini sur les types. Supposons que *S* et *T* sont deux types. La relation $T \leq S$ indique que *T* est un sous-type de *S* et que *S* est un super-type de *T*. Supposons que *S*, *T* et *U* sont trois types. Si $T \leq S$ et $T \leq U$, alors *T* est un sous-type commun de *S* et *U*. De même, si $S \leq V$ et $U \leq V$, alors *V* est un super-type commun de *S* et *U*. La hiérarchie de type forme un treillis. Les types peuvent avoir plusieurs parents et plusieurs enfants. Chaque pair de types ont un super-type

commun minimal et un sous-type commun maximal. Dans le treillis, $\top$ indique le type universel qui est supérieur à tous les types. Et $\bot$ indique le sous type de tous les types.

4.5.2 Généralisation et spécialisation

La théorie des graphes conceptuels nous permet de former des nouveaux graphes à partir des graphes existants, et ceci en généralisant ou spécialisant certains graphes. Quatre opérations sont utilisées pour cette fin.

Copy nous permet de créer un nouveau graphe g qui est la copie exacte de g_1

Join nous permet de combiner deux graphes dans un seul graphe. Si il existe un nøeud concept c_1 dans S_1 qui est identique à un nøeud concept c_2 dans S_2. Un nouveau graphe est formé en supprimant c_2 de S_2 et en liant toutes les relations incidentes de c_2 avec c_1.

Restrict Le but de la restriction est d'effectuer un changement sur un ou plusieurs nøeud(s) concept(s) pour pouvoir effectuer une opération de *join* ultérieurement. Par exemple, un concept générique dans un graphe peut être remplacé par un concept spécifique du même type. De même, la hiérarchie de type peut être utilisée pour nous permettre de remplacer un type par un sous-type. Autrement dit, pour effectuer la restriction, nous utilisons deux types d'héritage :
- l'héritage des propriétés des individus et ceci en spécialisant un concept généralisé pour un type donné.
- l'héritage des propriétés des sous-types et ceci en utilisant la hiérarchie de types.

Simplify nous permet de supprimer une relation dans un graphe conceptuel dans le cas où cette relation est dupliquée. La duplication d'une relation dans un graphe est souvent le résultat de l'application de l'opération *join* sur deux graphes.

Notons que Ces règles ne constituent en aucun cas des règles d'inférence. Ils ne garantissent en aucun cas que l'obtention des graphes dérivés qui seront vrais. Par contre le maintien du sens est garanti.

4.5.3 Nøeuds propositionnels

Nous avons vu que nous pouvons utiliser les graphes pour définir les relations entre les objets du monde. Nous pouvons aussi définir les relations entre les propositions. Exemple, *Thomas croit que Marie aime les pizzas*. Les graphes conceptuels incluent un concept type qui est le concept *proposition* qui définit des relations propositionnelles sur un ensemble de graphes conceptuels (voir figure 4.3).

4.6 Graphes conceptuels et logique

En utilisant les graphes conceptuels, nous pouvons facilement représenter la conjonction entre concepts. Exemple : le chien est grand et a faim. Mais nous n'avons pas jusqu'au là établi une moyenne pour exprimer les relations de disjonction et de négation. Nous pouvons implémenter une relation *neg* qui est une opération unaire agissant sur une proposition. En utilisant donc, des négations et des conjonctions nous pouvons implémenter des disjonctions. Par convention, les concepts génériques correspondent à des quantificateurs existentiels. Par exemple, le graphe 4.2 (en éliminant la spécification

toto) correspond à la formule $\exists X \exists Y (chien(X) \wedge coleur(X, Y) \wedge noir(Y))$. Il existe un algorithme nous permettant d'extraire la formule en logique de prédicats correspondante à un graphe conceptuel :

1. Attribuer à chaque concept générique une variable unique.
2. Attribuer une constante unique à chaque concept individuel. Cette constante peut être le nom ou l'identificateur.
3. Représenter chaque concept par un prédicat unaire ayant comme nom le type du nøeud et comme arguments les variables ou les constantes attribuées à ce nøeud.
4. Représenter chaque relation entre concepts d'arité n par un prédicat d'arité n, qui a également le même nom. Un argument du prédicat doit correspondre à la variable ou la constante attribuée au concept lié à cette relation.
5. Prendre la conjonction des expressions atomiques obtenues. Les variables seront quantifiées par des quantificateurs existentiels.

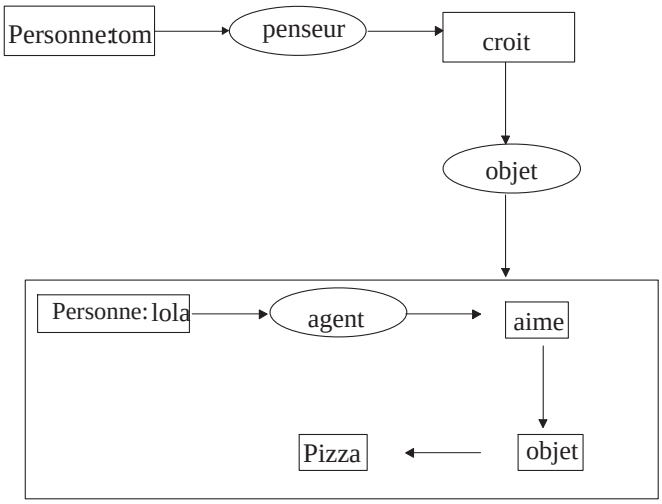

FIG. 4.3 – Les nøeuds propositionnels

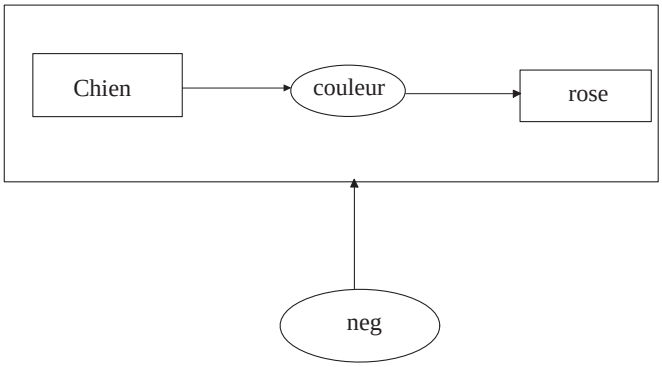

FIG. 4.4 – Les nøeuds propositionnels : la négation

Par exemple le graphe conceptuel donné par la figure 4.2 donne lieu à l'expression logique :

$$\exists X1(dog(toto) \wedge color(toto, X1) \wedge noir(X1))$$

4.7 Les objets structurés

Dans les réseaux sémantiques, ou les graphes conceptuels, le niveau de granularité de la connaissance est limitée au nøeud concept. Les objets structurés nous permettent, en fait de représenter plus finement la connaissance

4.8 Les frames ou les schémas

La connaissance est représentée par des objets structurés contenant des fentes (slot). Chaque fente est représentée par son nom et sa valeur. Chaque schéma individuel peut être vu comme une structure de données (enregistrement en C). Les fentes peuvent contenir plusieurs types d'information :

1. l'identification du schéma ;
2. les descripteurs du schéma ;
3. la relation avec les autres schémas par exemple le téléphone d'hotel est une spécialisation du schéma téléphone ;
4. Les informations procédurales attachées à certaines fentes ;
5. Les informations par défauts représentées par des valeurs attribuées aux fentes par exemple le nombre de pieds d'une chaise est égale à 4 ;

Quand une *instance* d'un schéma est créée, le système attribue des valeurs aux fentes (slot) en utilisant les valeurs par défauts, en exécutant des procédures attachées, ou bien en demandant à l'utilisateur les informations complémentaires. Les schémas peuvent être vus comme des réseaux sémantiques évolués. Ils permettent une organisation hiérarchique de la connaissance plus élaborée qu'en réseaux sémantiques. Les attachements procéduraux permettent de décrire une certaine partie de la connaissance par des étapes d'exécution.

Bibliographie

[1] R. Forsyth. *Expert Systems : Principles and Case Studies*. Chapman and Hall, 1984.

[2] P. Jackson. *Introduction to Expert Systems*. Addition-wesley, 1986.

[3] G.F. Luger and W.A. Stubblefield. *Artificial Intelligence : Structures and Strategies for Complex Problem Solving*. Addition-wesley, 1999.

[4] S. Russel and P. Norvig. *Artificial Intelligence : A Modern Approach*. Prentice-Hall International, Inc., 1995.