Annexe 3

Langage Fril

A.3.1. Rappel sur le langage Prolog

Prolog est un langage déclaratif [Colmerauer 85] d'une conception totalement différente des langages classiques. Programmer en Prolog, c'est énoncer une suite de faits et de règles puis poser des questions.
Tout programme Prolog constitue un petit système-expert qui va fonctionner en «chaînage-arrière» ou «induction», c'est-à-dire qui va tester les hypothèses pour prouver une conclusion.
Soit une base de règles BR constituée d'une part de règles sans prémisses : des faits comme : (frères Caïn Abel) ou non nécessairement clos comme : (égal X X), et d'autre part de règles sous forme de clauses de Horn comme :
(père X Y) et (père Y Z) $\Rightarrow$ (grand-père X Z) que l'on écrira dans le sens de la réécriture, la conclusion en premier, celle-ci devant s'effacer afin d'être remplacée par les hypothèses : C si H_1 et H_2 et ... et H_n.
Une liste de clauses Q étant écrites, l'interpréteur va alors chercher à unifier les différentes propositions élémentaires (faits) constituant le but Q avec la conclusion de chaque règle. Pour cela, il n'y a pas de distinction dans la base de clauses entre les faits et les règles, la conclusion est toujours en tête, les prémisses suivent et si elles ne sont pas présentes, c'est que la «règle» est un fait initial, en quelque sorte un axiome.
Si par un jeu de substitutions de variables, une telle unification est possible, alors avec ces mêmes substitutions partout dans Q, cette conclusion est remplacée par les hypothèses qui pourraient l'entraîner. C'est «l'effacement» et la constitution d'une «résolvante».
En Prolog une clause (Q si P_1 et P_2 et ... et P_n) s'interprète comme : pour prouver Q, il faut prouver P_1, prouver P_2 etc... Dans les principales implémentations du langage Prolog, la syntaxe est : Q :- P_1 , P_2 , ... , P_n.

Le point virgule, note le «ou» Q :- P ; R. étant équivalent aux deux clauses successives Q :- P. et Q :- R.
Les commentaires, comme en Fril, sont encadrés par /* ... */.
Prouver signifie «effacer» (unification avec un fait). Prolog ne se contente pas de fournir une telle «preuve», c'est-à-dire une instanciation ad hoc des variables, mais va les donner toutes, c'est en cela que l'on parle de non-déterminisme.
Il se pose alors un problème pour la négation, le «faux» est remplacé par le concept de «non prouvable».
Prolog fonctionne grâce au principe de résolution : un programme est une «forme normale» c'est à dire une conjonction de «clauses de Horn». Chaque clause de Horn est une disjonction $Q \vee P_1 \vee P_2$, ... , $\vee P_n$ de littéraux (faits ou négations de faits) dont au plus un peut être positif, ce qui revient à dire que chaque clause est une proposition de la forme $P_1 \wedge P_2$, ... , $\wedge P_n \rightarrow Q$. Le principe est que le programme est équivalent à sa "résolvante» obtenue en effaçant le fait p de la clause C et le littéral ¬p de C' et en remplaçant C et C' par la disjonction obtenue ainsi, et ceci chaque fois qu'un tel couple (p, ¬p) est décelable.

EXEMPLE D'UNE AGENCE MATRIMONIALE

On établit un programme comme une liste de faits «bruts» qui vont être traduits au moyen de prédicats que l'on décide d'appeler «caractéristiques», «gouts», «recherche» ... et quelques règles définissant «être bien assortis», soit 23 clauses.

```
caracteristiques (alfred, homme, grand, brun, mur).
        /* tout ce qui commence par une minuscule est constant */
caracteristiques (victor homme moyen blond, jeune).
caracteristiques (hector, homme, petit, brun, mur).
caracteristiques (irma, femme, moyenne, blonde, moyen).
caracteristiques (rosa, femme, petite, blonde, jeune).
caracteristiques (olga, femme, petite, brune, mur).
gouts (alfred, class, aventure, velo).
gouts (victor, pop, sf, ski).
gouts (hector, jazz, polar, ski).
gouts (irma , class, aventure, velo).
gouts (rosa , pop, sf, S) :- dif (S, boxe).
        /* le sport est tout sauf de la boxe pour Rosa */
gouts (olga, M, aventure, velo).      /* M peut être instancié par n'importe quoi */
recherche (alfred, grande, rousse, jeune).      /* les majuscules sont des variables */
recherche (victor, T, blonde, jeune) :- dif (T, grande).
        /* Victor ne veut pas de grande */
recherche (hector, petite, blonde, moyen).
recherche (irma , grand, brun, moyen).
recherche (rosa, moyen, blond, jeune).
recherche (olga, moyen, brun, mur).             /* C'est tout pour les faits bruts */
dif (X, X) :- fail.
dif (X, Y).
/*ces deux clauses ne peuvent pas être permutées, elles définissent «être différents»*/
convient (X, Y) :- caracteristiques (X, homme, T1, C1, A1),
        caracteristiques (Y, femme, T2, C2, A2),
        recherche (X, T2, C2, A2), recherche (Y, T1, C1, A1).
memesgouts (X, Y) :- gouts (X, M, L, S), gouts (Y, M, L, S).
assortis (X, Y) :- convient (X, Y), memesgouts (X, Y).
```

/* Ne pas oublier que la conclusion est en premier */

Le programme étant achevé, on lance une recherche par la question :
assortis (X, Y).
La réponse que donnera Prolog est X = victor et Y = rosa
On pourra vérifier à la main qu'il n'y a pas d'autres solutions.

EXEMPLE DES TOURS DE HANOÏ

Il s'agit de déplacer une tour formée de N disques de diamètres échelonnés (le plus petit en haut), d'un emplacement dit gauche à un autre dit droite, en se servant d'un troisième (milieu). La règle est de ne jamais placer un disque sur un autre dont le diamètre serait plus petit.
Trois clauses suffisent avec le prédicat "mouv" qui traduit récursivement ce qu'il faut faire comme mouvements et le prédicat "hanoi" associé au nombre de disques, qui permet de démarrer.

```
mouv (1, A, _ , C) :-   write ("transport de ", A, " sur ", C) , !.
mouv (N, A, B, C) :-  K is N-1, mouv (K, A, C, B),
                              write ("transport de ", A, " sur ", C), mouv (K, B, A, C).
hanoi (N) :-  mouv (N, gauche, milieu, droite).
```

L'explication est transparente, pour déplacer les N disques de A sur C, il faut déplacer les N-1 premiers sur B, puis le dernier sur C, et enfin les N-1 de B sur C. (Il y a 2^{N-1} transports) Remarque "write" n'accepte qu'un seul argument en C-prolog, il faut donc en écrire plusieurs, et se servir de "nl" (retour à la ligne).

EFFACER UN ÉLÉMENT DANS UNE LISTE

Les listes d'objets abstraits s'écrivent en Prolog par énumération [a, b, c, ...] ou en faisant appel à une liste L, alors [a, b | L] désignera par exemple la liste débutant par a et b suivis de tous les éléments de L. La concaténation de deux listes est l'enchainement des deux.
On construit «eff (X, L, R)» où R est la liste L dans laquelle la première occurence de X est effacée.

```
eff (_, [], []).
eff (A, [A | L], L) .
eff (A, [B | L], [B | M]) :- eff (A, L, M).
```

Mais attention le ";" donne les autres solutions ainsi "eff (a , [a, b, a], X)." donne X = [b, a] puis X = [a, b]. Il faut rajouter la prémisse "coupure" dans les deux premières clauses.
Avec eff2 pour effacer toutes les occurences de X dans L, on remplace la seconde clause par "eff2 (A, [A | L], M) :- eff2 (A, L, M)." Mais là aussi "eff2 (a, [b, a, b, a], X)." donnerait X = [b, b] puis X = [b, a, b] puis X = [b, b, a] d'où la nécessité de la coupure.

Grâce à eff3, on efface X à tous les niveaux de L.

```
eff3 (_, X, X).
eff3 (X, [X | L], M) :- eff3 (X, L, M).
eff3 (X, [Y | L], [Z | M]) :- eff3 (X, Y, Z), eff3 (X, L, M).
```

EXEMPLE DE LA CONCATÉNATION DE DEUX LISTES

Deux clauses vont suffire: conc ([], L, L).
conc ([X | L], M, [X | N]) :- conc (L, M, N).

Avec ce programme très réduit, il est possible de l'interroger sous forme fonctionnelle, par exemple en demandant «conc ([a], [b, c], X).». En ce cas Prolog répondra X = [a, b, c]. Si on demande à présent une résolution d'équation «conc ([a], X, [a, b, c, d]).», on obtiendra X = [b, c, d], mais pour «conc ([a], X, [b, c]).» il n'y aura pas de solution. Pour «conc ([a], X, Y).», il y a une infinité de solutions, enfin, pour «conc ([a], [b], [a, b]).», la réponse est booléenne.

Maintenant, à la question «conc (U, V, [a, b]).» dans laquelle U et V sont les variables demandées, Prolog va fournir une première solution U = [] et V = [a, b], puis les autres si l'utilisateur le désire.

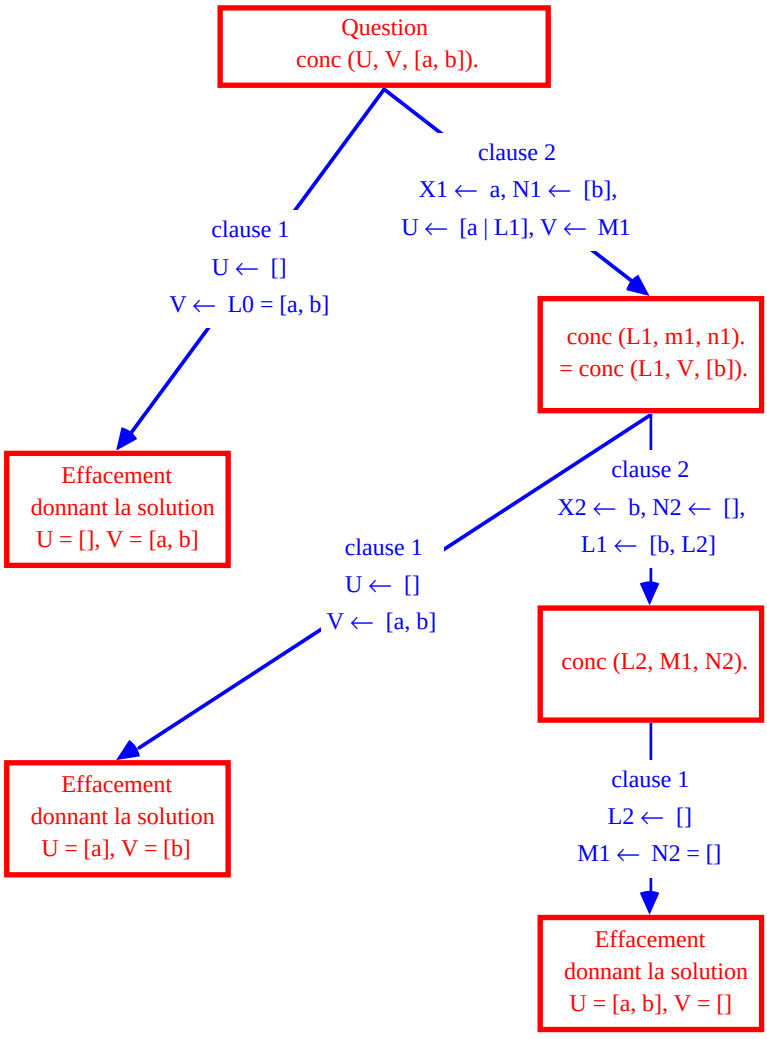

Figure A.3.1 Arbre de résolution du programme «conc» à partir de la question initiale.

A.3.2. Langage Fril (Fuzzy Relational Language de Baldwin)

Fril est un langage déclaratif comme Prolog, cependant, contrairement à Prolog l'arborescence complète est toujours explorée, car toutes les preuves d'une même conclusion vont être confrontées [Baldwin 90].
Dans Fril, chaque fait est donné avec un «support» (n, p). Ce support est considéré dans l'esprit des auteurs comme l'intervalle pouvant contenir la probabilité, il s'agit donc des notions de probabilités basse et haute parfois notées L et U (chapitre 2).
Chaque règle $P \Rightarrow Q$ est donnée avec le support (a, b) de l'événement conditionnel Q/P. Si un deuxième support (c, d) est donné, c'est celui de $\neg P \Rightarrow Q$, donc un intervalle pour la probabilité conditionnelle $p(Q/\neg P)$.
En termes de probabilités cela se justifie par $a \leq p(Q/P) \leq b$ et $c \leq p(Q/\neg P) \leq d$.

NÉGATION

La relation entre les supports de P et ¬P est nécessairement $\neg(n, p) = (1-p, 1-n)$

CONSISTANCE D'UNE BASE DE CONNAISSANCES

Prenons l'exemple de probabilités $p(A/Q) = 0,5$ $p(A/\neg Q)=0,4$ $p(A/S)=0,8$ $p(A/\neg S)=0,4$ $p(Q)=0,7$ $p(S)=0,175$ alors $p(A)=p(A/Q)p(Q) + p(A/\neg Q)p(\neg Q) = 0,47$ et le même calcul avec S donnant 0,47. Si cela n'avait pas été le cas la base aurait été dite inconsistante.

CONJONCTION ET DISJONCTION

Si dans un corps de prémisses, deux faits F_1 et F_2 de supports respectifs (n_1, p_1) et (n_2, p_2) se présentent, on suppose par défaut que ces faits sont indépendants et donc «F_1 et F_2» aura dans ce système le support (n_1n_2, p_1p_2) conformément à la logique probabiliste. «F_1 ou F_2» aura le support $(n_1+n_2-n_1n_2, p_1+p_2-p_1p_2)$
Dans le cas où la dépendance est précisée la conjonction peut être calculée par :
$L(P \text{ et } Q) = \max(0, L(P) + L(Q) - 1)$ et $U(P \text{ et } Q) = \min(U(P), U(Q))$ et la disjonction par $L(P \text{ ou } Q) = \max(L(P), L(Q))$ et $U(P \text{ ou } Q) = \min(1, U(P) + U(Q))$, en suivant un panachage de Zadeh et Lukasiewicz.

MÉCANISME D'INFÉRENCE DE JEFFREY

Etant donnée une règle $P \Rightarrow Q$ connue avec les couples (a, b) et (c, d) et si l'hypothèse P est connue avec le support (x, y), l'opération consistant à calculer le support (X, Y) de Q est inspirée, pour $P \Rightarrow Q$, de la formule de Bayes dite formule des probabilités totales $p(Q) = p(Q/P)p(P) + p(Q/\neg P)p(\neg P)$.
Le calcul de (X, Y) pour la conclusion Q se fait donc :
$X = \min\{p(Q/P)p(P) + p(Q/\neg P)p(\neg P)\} = \min\{az + c(1-z)\}$ en notant $z = p(P)$
en regardant simplement le min de la fonction $z \in [x, y] \rightarrow az - cz + c$, dans le cas de $a \leq c$, le min a lieu pour z = y, c'est $X = ay + c(1-y)$ sinon $X = ax + c(1-x)$
De manière analogue Y = si $b \leq d$ alors $bx + d(1-x)$ sinon $by + d(1-y)$
Remarque : dans le cas où un seul support est donné $(X, Y) = (ax, bx + 1 - x)$ on remarque que y n'intervient pas. Ainsi il suffit que la règle $P \Rightarrow Q$ soit donnée avec un support (a, 1) pour qu'une hypothèse P (x, y) même mauvaise infère une conclusion Q (ax, 1).

La valeur (X, Y) attribuée à la conclusion Q lorsque P vaut (x, y) est :
X = si a ≤ c alors ay + c(1-y) sinon ax + c(1-x)
et : Y = si b ≤ d alors bx + d(1-x) sinon by + d(1-y)
Si seulement (a, b) est fourni, on a (X, Y) = (ax, bx + 1 - x)

REMARQUE

Dans le dernier cas, la valeur y n'a pas d'importance, ainsi par exemple si la prémisse est connue avec le support (0.4 0.8) et la règle avec (0.9 0.9), la conclusion aura (0.36 0.96) et il suffit que b soit 1 (règle pouvant être jusqu'à certaine) pour que la conclusion ait une probabilité supérieure 1. Il serait d'ailleurs difficile de prendre une autre position.

AGRÉGATION

1 Premier cas, l'intersection : si P se trouve avoir les supports (n_1, p_1) et (n_2, p_2) par deux preuves différentes, alors l'interpréteur Fril donne le support $(\max(n_1, n_2), \min(p_1, p_2))$ mais sous réserve que $n \leq p$. C'est la règle d'intersection notée «inter» des supports utilisée par défaut, elle suppose la base de règles consistante.

2 Sinon on utilise la règle de Dempster (chapitre 2), si cela est précisé, tous les faits intervenant dans un prédicat dit «de Dempster» sont supposés indépendants. Deux preuves de P correspondent à des points de vue indépendants P_1, P2 de supports (L_1, U_1), (L_2, U_2)
On a donc $L = (L_1 + L_2 - L_1L_2 - c)/(1-c)$ et $U = U_1U_2/(1-c)$ avec le coefficient de conflit $c = L_1(1-U_2) + L_2(1-U_1)$

3 Troisième cas : par souci de mieux traduire les problèmes concrets où une conclusion peut très souvent être acquise par différentes voies, Fril prévoit la syntaxe suivante :
(conclusion IF (evlog S(x) (x_1 is A_1) w_1 ; (x_2 is A_2) w_2 ; ..... ; (x_n is A_n) w_n)) : ((n_1 p_1) (n_2 p_2))
Règle dans laquelle les différentes hypothèses éventuellement structurées comme (and (or P Q) (not R)) exactes ou floues, sont affectées de poids w_i et séparées par des point-virgules ; notant «ou bien» comme en Prolog, la somme des poids $\sum w_i$ devant être 1.
La fonction S réalise une sorte «d'accentuation» vers une valeur de vérité plus proche de 0 ou de 1.plus simple que la règle de Dempster.
Si (a_i, b_i) est le support de (x_i is A_i), plus généralement de la prémisse P_i, alors le support de la conclusion est calculé par l'application de la règle de Jeffrey sur $(S(\sum w_i a_i), S(\sum w_i b_i))$.

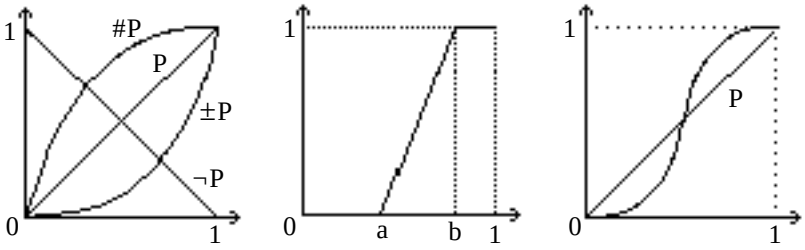

Figure A.3.2 Si P représente une valeur de vérité, on a (à gauche) les représentations de nonP, de «très»P souvent évoquée comme P^2 et de «plus ou moins»P par $\sqrt{P}$. Les fonctions S nommées «filtre» (centrale) proposées par [Baldwin, Martin 94] sont du type ci-dessous avec a et b égaux à 0.5 / 0.8 pour «most», 0.7 / 1 pour «very», 0 / 0.5 pour «fairly» ou encore 0 / 1 pour «trueline». A droite l'opérateur de Dempster.

REMARQUES PRATIQUES

Les constantes de Fril peuvent débuter par une majuscule ou non comme Jean ou jean, et les variables sont les majuscules isolées ou les mots débutant par au moins 2 majuscules J, J1, JEan, JEAN, ...
Un prédicat prédéfini (supp_query X S) établit un lien entre un fait X et son support S, on peut donc accéder aux valeurs par exemple (supp_query X (n p)) puis formuler sur n et p. Ainsi ((supp—querry ((predicat X)) (X Y)) (p (X + Y) / 2))
réalisera une impression «p» de la demi-somme des probabilités basse et haute.
Les clauses s'écrivent dans une fenêtre d'édition (ainsi éventuellement que la question «qs»... «qh»... ne donnant pas les supports). Puis «reload selection» lance l'interpréteur.

EXEMPLE INTRODUCTIF DE BALDWIN

Une boîte contient 6 boules rouges et 4 bleues, 8 sur les 10 sont solides.
On ne peut donc donner exactement p(solide / rouge) mais un encadrement. Les données seront :
((rouge X)) : (0.6 0.6)
((solide X) (rouge X)) : ((0.67 1) (0.5 1))
Si on demande «qs ((solide uneballe)) on aura le calcul (0.402 1), «uneballe» désignant une constante.
Maintenant, avec des prédicats «solide», «rouge», «grand, »voisin«, et des constantes a, b, on a la base de règles :

R1 ((solide a)) : (1 1)
R2 ((solide b)) : (0 1)
R3 ((rouge a)) : (0.8 0.9)
R4 ((rouge b)) : (0 0)
R5 ((grand a)) : (0.85 1)
R6 ((grand X) (solide X)) : (0.6 0.9) (0.3 0.5))
R7 ((voisin X Y) (grand X) (rouge Y)) : ((0.9 1) (0 0.1))
On demande «qs (voisin X Y)»

En posant ce but (voisin A B), les solutions sont comme en Prolog, vues comme les instantiations des variables le long des chemins amenant à un effacement des clauses. Pour chaque solution, son support est calculé en remontant des feuilles vers la racine de l'arbre. Une première solution est trouvée A = a, B = a avec deux preuves en

partant des supports (0.612, 0.91) une autre avec (0.612, 0.829), l'agrégation se fait ici par intersection. Toutes les solutions sont calculées, et on obtient A = B = a (0.612, 0.829) puis A = a B = b (0, 0.1), A = b B = a (0.216, 0.829) enfin A = B = b (0, 0.1). A titre d'exercice, on pourra recalculer les valeurs obtenues 0.216...

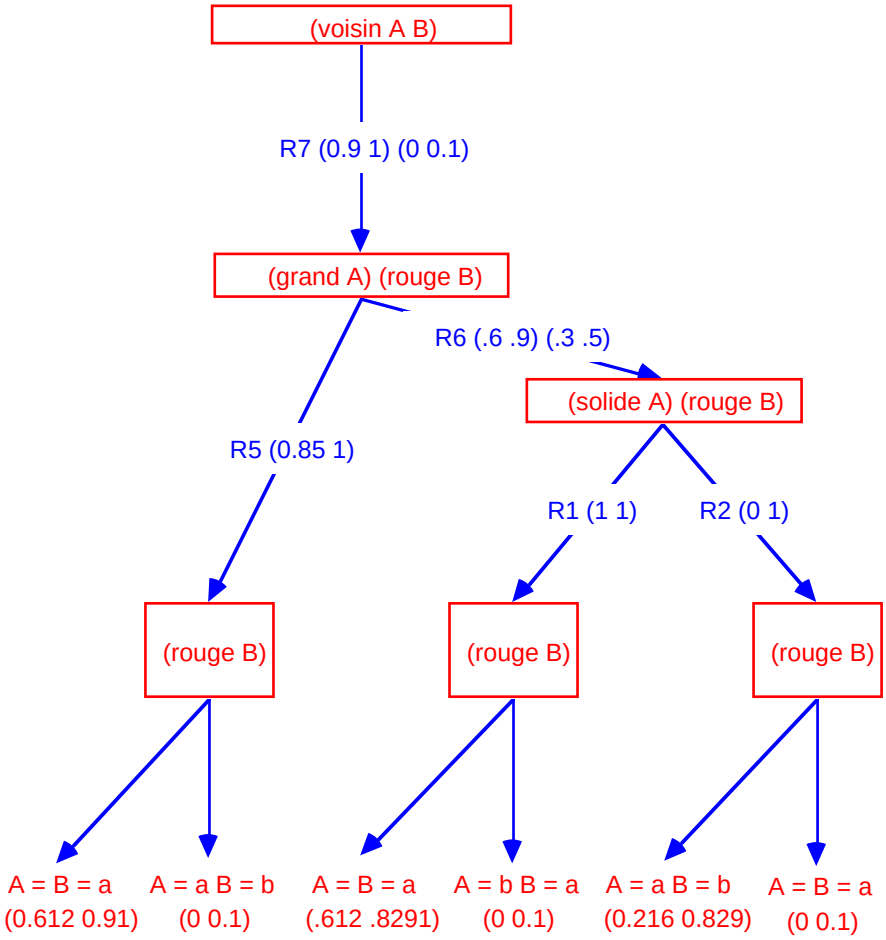

Figure A.3.3 Arbre de résolution de l'exemple de Baldwin.

EXEMPLE DE BASE DE CONNAISSANCES

((aime Alexandre Johan)) : (0.5 0.8)
((aime Alexandre Julien)) : (0.8 1)
((aime Julien Alexandre)) : (0.7 0.9)
((aime Johan Alexandre)) : (0.6 0.8)
((aime Laureline Valerian)): (0.6 1)
((aime Valerian Laureline)) : (0.8 0.9)
((freres Valerian Alexandre)) : (1 1)
((amis X X)) : (0 0) /* Une façon de rendre compte de la négation */
((amis X Y) (aime X Y) (aime Y X)) : (0.7 1)
((amis X Z) (freres X Y) (amis Y Z)) : (0.3 0.7)

```
/* Difficulté comme en Prolog de définir une relation symétrique et transitive */
/* On pose une question */
qs ((amis Valerian B))
/* Voilà les réponses */
((amis Valerian Valerian)) : (0 0)                /* antireflexivité */
((amis Valerian Laureline)) : (0.336 1)
((amis Valerian Alexandre)) : (0 1)
((amis Valerian Johan)) : (0.063 0.937)
((amis Valerian Julien)) : (0.1176 0.8824)
no (more) solutions

/* On souhaite une «transitivité» de l'amitié, mais attention aux appels récursifs infinis
avec «amis» seul*/
((relation X Z) (amis X Y) (amis Y Z)) : (0.5 0.7)
qs ((relation A B))
((relation Alexandre Alexandre)) : (0.076832 0.953901)
((relation Johan Johan)) : (0.02205 0.98677)
((relation Johan Julien)) : (0.04116 0.975304)
((relation Julien Johan)) : (0.04116 0.975304)
((relation Julien Julien)) : (0.076832 0.953901)
no (more) solutions
/* On voit rapidement une incertitude presque totale pour ces 5 derniers résultats */
```

TRAITEMENT DES ENSEMBLES FLOUS EN FRIL

On les définit par une suite de couples x : μ(x), l'interpréteur fera alors des interpolations linéaires et prendra les deux valeurs des extrêmes pour tout ce qui est inférieur ou supérieur à ces extrêmes. Ainsi «grand» est-il définit ci-dessous par un triangle, et «trèsgros» par une marche.
La confrontation d'une assignation floue A' de la variable X face à une référence floue A**, s**i A représente un prédicat tel que «grand», de la proposition «X est A» est calculée par le support (n p) :
p = sup min (μ_A, $\mu_{A'}$) comme cela est fait habituellement et :
n = 1 - p(X est non A) = 1 - sup min (μ_A, 1 - $\mu_{A'}$) = inf max(1 - $\mu_{A'}$, μ_A) contrairement à [Dubois, Prade 85].

```
(grand [165:0 175:1 185:0])
(moyen [160:0 170:1 175:0])
(gros [60:0 75:1 90:0])
(tresgros [80:0 95:1])      /* ou bien [80:0 95:1 lim:1] qui signifie jusqu'à l'infini */
((taille Max grand))
((taille Luc moyen))
((taille Fred 72))

qs ((taille X grand))                    /* On pose un but */
/* Le schéma des triangles explique les supports, les trois réponses sont */
((taille Max [165:0 175:1 185:0])) : (0.5 1)
((taille Luc [165:0 175:1 185:0])) : (0.25 0.666666)
((taille Fred [165:0 175:1 185:0])) : (0 0)
```

A titre d'exercice, en définissant (petit [0:0 4:1 7:1 9:0]) et donnant le fait (salaire Max [2:0 3:1 9:0]), la réponse pour (salaire Max petit) sera (0.6 0.9), faire le dessin.

DÉFINITIONS RÉCURSIVES DES ENSEMBLES FLOUS

Une façon intéressante de définir récursivement des ensembles flous est de dire que si H1 est une grande taille, alors H0 = H1-1 l'est encore, mais un peu moins, d'où un support (0.9 1) par exemple à cette règle.

```
((grand H)(less 175 H)(!))
/* définition par une contrainte, essayer sans la coupure ! ferait boucler indéfiniement
avec la seconde clause */

((grand H0) (sum H0 1 H1)(grand H1)) : (0.9 1) /* définition récursive */
((petit H) (less H 160) (!))
((petit H0) (sum H1 1 H0) (petit H1)) : (0.9 1)
```

/* voici les questions */	/* et voilà les réponses */
qs ((grand 175))	((grand 175)) : (0.9 1) yes
qs ((grand 172))	((grand 172)) : (0.6561 1) yes
qs ((grand 170))	((grand 170)) : (0.531441 1) yes
qs ((grand 160))	((grand 160)) : (0.185302 1) yes
qs ((petit 165))	((petit 165)) : (0.531441 1) yes
qs ((petit 160))	((petit 160)) : (0.9 1) yes
qs ((petit 155))	((petit 155)) : (1 1) yes

```
/* on rajoute des prédicats de Dempster en assumant la non-contradiction des règles */
dempster grand_ht
((grand_ht H) (grand H)) : (1 1)
((grand_ht H) (petit H)) : (0 0)
dempster petit_ht
((petit_ht H) (petit H)) : (1 1)
((petit_ht H) (grand H)) : (0 0)
```

Questions	Réponses
qs ((grand_ht 170))	((grand_ht 170)) : (0.437658 0.823531) yes
qs ((petit_ht 165))	((petit_ht 165)) : (0.437658 0.823531) yes
qs ((grand_ht 160))	((grand_ht 160)) : (0.022239 0.120015) yes
qs ((petit_ht 155))	((petit_ht 155)) : (1 1) yes

A titre d'exercice, rajouter ((poids X gros) (taille X grand)) (0.7 0.9) et demander (gros Max) etc.

OPÉRATIONS

Soient deux ensembles flous A et B définis comme suit :

```
(A [0:0  6:1  9:0])
(B [2:0  3:1  4:0])
qs (+ A B)       /* réponse */    [2:0  9:1  13:0]
qs (- A B)       /* réponse */    [-4:0  3:1  7:0]
qh ((sum [0:0  1:1  2:0]  [3:0  4:1  5:1  7:0]  X))
/* réponse approximative) */
X = [3:0  5:1  6:1  9:0]
```

Plusieurs applications sont en projet, on verra notamment [Baldwin 96].