CHAPITRE 16

LE TRAITEMENT DE L'INCERTAIN

Ce dernier chapitre présente un autre langage issu de Prolog, et se poursuit sur le thème de l'incertain et de la commande floue. Fril est né dans les années 80 à l'université de Bristol sous l'impulsion de J.F. Baldwin, il s'agit d'un Prolog calculant des intervalles de confiance que l'on peut accorder aux diverses solutions d'un problème à partir de ceux donnés aux faits initiaux et aux règles du type $P \rightarrow Q$ et $\neg P \rightarrow Q$.
On présente en outre l'algorithme de la commande floue qui permet de réaliser quelques simulations graphiques.

Langage FRIL (Fuzzy relationnal inference language)

Programmer en Prolog, c'est énoncer une suite de règles et de faits (des règles sans prémisses) 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.
Etant donné une base de règles 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, on pose un but Q.
L'interpréteur va alors chercher à unifier les différentes propositions (faits) de Q avec la conclusion de chaque règle, ou 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".
Tout un arbre de recherche est ainsi développé suivant la stratégie dite "racine-gauche-droite", les branches ayant conduit à un effacement total donnant les solutions.

Ce qui fait l'originalité de Fril, c'est que dès qu'une solution est trouvée l'interpréteur va lui calculer un couple de confiance en remontant dans le sens déductif, jusqu'à la racine suivant certaines modalités de calcul :
Dans Fril, chaque fait est donné avec un "support" (x, y) 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. On notera cr (crédibilité) et pl (plausibilité) ces bornes.

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 pour une règle c'est celui de $\neg P \Rightarrow Q$, donc un intervalle pour la probabilité conditionnelle pr (Q/¬P)
En termes de probabilités cela se justifie par $a \leq pr\ (Q/P) \leq b$ et $c \leq pr\ r\ (Q/\neg P) \leq d$

Négation : la relation entre les supports de P et ¬P est nécessairement :

$$\neg(x, y) = (1-y, 1-x)$$

Consistance d'une base de connaissances, prenons l'exemple de probabilités conditionnelles pr (A/Q) = 0,5 pr (A/¬Q) = 0,4 pr (A/S) = 0,8 pr (A/¬S) = 0,4 pr (Q) = 0,7 pr (S) = 0,175 alors pr (A) = pr (A/Q)pr (Q) + pr (A/¬Q)pr (¬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 : si dans un corps de prémisses, on a deux faits P et Q de supports respectifs (x, y) et (x', y'). On suppose par défaut que ces faits sont indépendants et donc "P et Q" aura une définition probabiliste (x, y) et (x', y') = (xx', yy').

Disjonction, elle est définie par (x, y) ou (x', y') = (x + x' -xx', y + y' -yy') en suivant les lois de Morgan.
Dans le cas où la dépendance est précisée cr (P et Q) = max(0, cr (P)+cr (Q)-1) pl (P et Q) = min (pl (P), pl (Q)) et pour la disjonction cr (P ou Q) = max (cr (P), cr (Q)) pl (P ou Q) = min (1, pl (P)+pl (Q))

Mécanisme d'inférence de Jeffrey : Fril prévoit trois types de règles, nous ne parlerons que de la première pour la quelle une conclusion Q peut être observée avec P ou sans P. Le second type de règle en est une généralisation où Q pourrait être observé avec tout un système exhaustif et mutuellement exclusif d'hypothèses P_1, P_2, ... P_n.

Le mécanisme de Jeffrey est une opération d'inférence inspirée, pour $P \Rightarrow Q$, de la formule des probabilités totales de Bayes pour laquelle pr (Q) = pr (Q/P)pr (P) + pr (Q/¬P)pr (¬P)

Si $P \Rightarrow Q$ est connu avec les supports (a, b) et (c, d), et lorsque P est connu avec (x, y), le calcul de (X, Y) pour Q se fait donc en notant z = pr (P) :

$$X = \min\{pr(Q/P)pr(P) + pr(Q/\neg P)pr(\neg P)\} = \min\{az + c(1-z)\}$$

en regardant le min de la fonction $z \in [x, y] \rightarrow az - cz + c$, dans le cas $a \leq c$, ce 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), en résumé :

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 (a, b) seul présent, alors (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 connu 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 (ou plausibilité) égale à 1. Il serait d'ailleurs difficile de prendre une autre position.

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). L'inconvénient est d'avoir par exemple une conclusion Q (0, 1) traduisant l'indétermination maximale lorsque P est faux (0, 0) et que la règle est exacte $P \Rightarrow Q$ (1, 1).

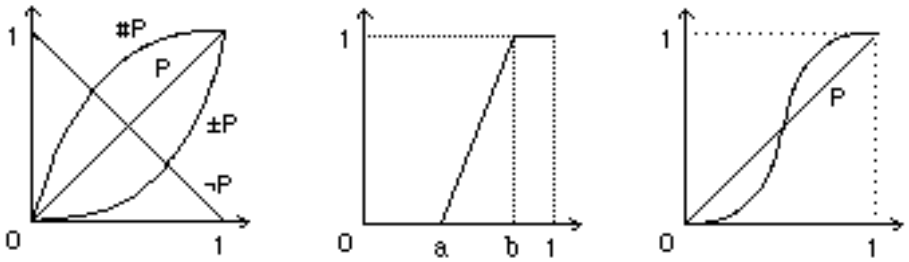
Agrégation des différentes preuves
1 Premier cas, l'intersection : si P se trouve avoir les supports (n1, p1) et (n2, p2) par deux preuves différentes, alors on lui donne le support (max (n1, n2), min (p1, p2)) mais sous réserve que $n \leq p$. C'est la règle d'intersection notée "inter" des supports utilisée par défaut, elle suppose la base de règles consistante.

2 On peut utiliser la règle de Dempster, 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, P_2 de supports (x, y), (x', y') avec c = x(1-y') + x'(1-y) le coefficient de conflit. La règle est une opération commutative et associative pour laquelle (0, 1) est neutre, (0, 0) et (1, 1) sont absorbants :

$$(x, y) \oplus (x', y') = \left(\frac{x + x' - xx' - c}{1 - c}, \frac{yy'}{1 - c}\right) = \left(\frac{xy' + x'y - xx'}{1 - x - x' + xy' + x'y}, \frac{yy'}{1 - x - x' + xy' + x'y}\right)$$

3 Troisième cas : par souci de mieux traduire les problèmes concrets où une conclusion peut très souvent être acquise par différentes voies, Fril un troisième type de règle :
(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))

dans laquelle les différentes hypothèses éventuellement structurées comme (and (or P Q) (neg R)) exactes ou floues, sont affectées de poids et séparées par des point-virgules ; notant "ou bien" comme en Prolog, la somme des poids $\sum w_i$ devant être 1, et la fonction S réalise une sorte d'accentuation" vers une valeur de vérité plus proche de 0 ou de 1.
Si (a_i, b_i) est le support de $(x_i \text{ is } A_i)$, plus généralement de la prémisse P_i, alors le support de la conclusion est calculé par l'application de Jeffrey sur le couple $(S(\sum w_i a_i), S(\sum w_i b_i))$
Si P représente une valeur de vérité, on a (figure de 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" (figure 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".



S réalise une sorte d'"accentuation" simplifiant la règle de Dempster.
En effet, si on cherche l'opérateur rationnel f : P → #P le plus simple vérifiant les trois conditions : (1/2, 1/2) centre de symétrie pour obtenir #¬P = ¬#P, f(1) = 1 et f(0) = 0 (points fixes) et la dérivée f ' (x) = 0 pour x = 0 ou 1. (T et F sont atteint plus vite par #P que par P), nous obtenons $\#x = x^2 / (2x^2 - 2x + 1)$ (figure de droite) qui n'est autre que la restriction de la règle de Dempster restreinte aux couples identiques (x, x).

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 JEAN, JEan, J, J1...
Les clauses s'écrivent dans une fenêtre d'édition ou dans la fenêtre d'éxecution, ainsi que les questions qui peuvent avoir plusieurs formes dont les deux plus commodes sont :
qh (fait1 fait2 ...) donnant les solutions à cette conjonction de faits,
qs (fait1 fait2 ...) donnant en plus les supports
On passe dans l'interpréteur grâce à "reload selection". Le "qs = query-support" est d'ailleurs défini à partir d'un prédicat de fril : (supp_query F (N P)) établissant le lien entre un fait F et son support, ce qui permet d'accéder aux valeurs N et P.

Exemple de base de connaissances

```
((aime Alexandre Johan)) : (0.5 0.8)        ((aime Johan Alexandre)) : (0.6 0.8)
((aime Valerian Aymeric)) : (0.8 0.9)       ((aime Aymeric Valerian)) : (0.6 1)
((freres Valerian Alexandre)) : (1 1)
((amis X X)) : (0 0)        /* Une façon de noter une négation, antiréflexivité */
((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 transitive. On pose une question */

qs ((amis Valerian B))     /* Voilà les réponses */
((amis Valerian Valerian)) : (0 0)          ((amis Valerian Aymeric)) : (0.336 1)
((amis Valerian Alexandre)) : (0 1)         ((amis Valerian Johan)) : (0.063 0.937)
/* On souhaite une "transitivité" de l'amitié, mais sans avoir des appels récursifs infinis avec "amis" seul */
((relation X Z) (amis X Y) (amis Y Z)) : (0.5 0.7)

qs ((relation A B))         /* question qui donne : */
((relation Alexandre Alexandre)) : (0.076832 0.953901)
((relation Johan Johan)) : (0.02205 0.98677)
```

Exemple d'utilisation de la règle de Dempster

Le programme est constitué par 6 clauses dont seules les 2 premières sont incertaines :

```
((vole X) (oiseau X)) : (0.9 1)
((vole X) (pingouin X)) : (0 0)
((oiseau X) (pingouin X))
((oiseau X) (aigle X))
((pingouin hector))
((aigle edgar))

/* Exemple on veut connaître le support du fait qu'Hector vole */
qs ((vole Hector))
<support error 400> : no overlap between supports (0.9 1) and (0 0)
/* On rajoute alors au début, avant les clauses, la "déclaration" : */
dempster vole
qs ((vole X))
((vole hector)) : (0 0)
((vole edgar)) : (0.9 1) no (more) solutions
```

Si on rajoute

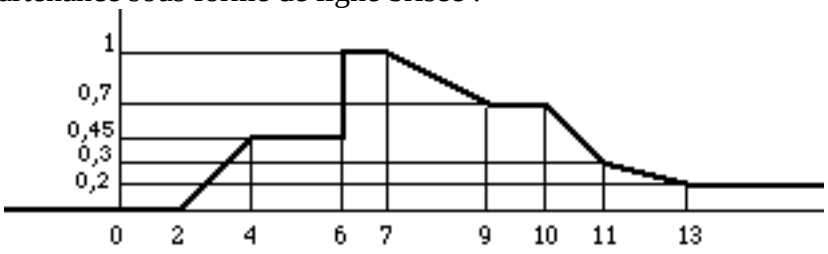
```
((vole X) (pingouin X)) : (0 01)
((vole X) (poule X)) : (0.1 0.2)
((poule daisy))
```

Alors à la question qs ((vole X)), on obtient :

```
((vole hector)) : (0.47 0.52)
((vole daisy)) : (0.67 0.71)
```

Traitement des ensembles flous en Fril

Un sous-ensemble flou de E est la donnée d'une fonction "d'appartenance" μ d'un ensemble E dans [0, 1]. Par exemple [2:0 4:0.45 6:1 7:1 9:0.7 10:0.7 11:0.3 13:0.2] permet de définir une fonction d'appartenance sous forme de ligne brisée :



Cela sert à apprécier les nuances pour des prédicats tels que "grand", "petit" etc.. 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.
Sur un univers U, lors de la confrontation du fait "X est A" avec la valeur floue A' connue pour la variable X, la plausibilité de ce fait est pl = $\sup_U$ min (μ_A, $\mu_{A'}$), et la crédibilité est donnée par cr (X est A) = 1 - pl (X est ¬A) = 1 - $\sup_U$ (min ($\mu_{A'}$, 1 - μ_A)) = $\inf_U$ (max (1 - $\mu_{A'}$, μ_A)).

```
(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 encore [80:0 95:1 lim:1] qui signifie jusqu'à l'infini. */
((taille Max grand))            ((taille Luc moyen))            ((taille Fred 150))

qs ((taille X grand))           /* On pose un but */

((taille Max [165:0 175:1 185:0])) : (0.5 1)
((taille Luc [165:0 175:1 185:0])) : (0.25 0.666666)    /* Expliquez pourquoi ces valeurs */
((taille Fred [165:0 175:1 185:0])) : (0 0)
```

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)

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

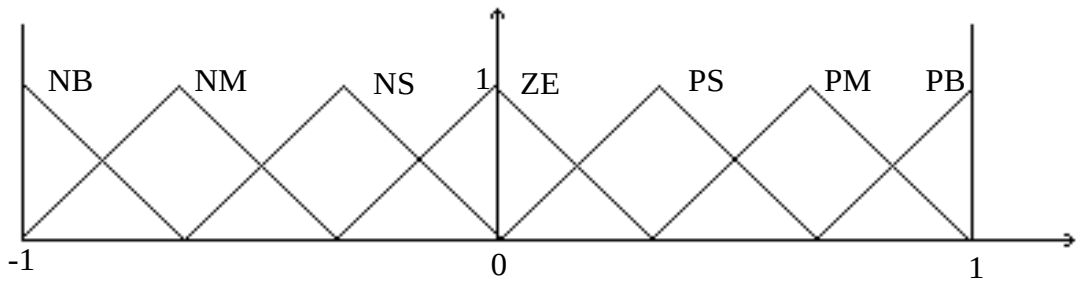
Opérations sur les ensembles flous

Soient :
(A [0:0 6:1 9:0])
(B [2:0 3:1 4:0])

qs (sum A B X)	/* réponse */	X = [2:0 9:1 13:0]
qs (- A B X)	/* réponse */	X = [-4:0 3:1 7:0]

La commande floue

Un algorithme très simple dû à [Mamdani 79] permet de modèliser une fonction par interpolations à partir d'un petit nombre de points "flous".
Plus précisément, à partir de règles de la forme : "si (X est A1) et (Y est A2) alors (U est B)" où A1, A2, B sont des prédicats mentionnés comme "grand positif", "moyen", "presque nul" etc... on peut décrire entièrement une fonction (x, y) → u
Les applications concrètes existant à l'heure actuelle utilisent principalement des fonctions triangulaires ou trapézoïdales, ou du type courbe de Gauss, Arctan, ou homographiques recouvrant l'intervalle [-1, 1], ci-dessous avec 7 (le plus souvent 5) prédicats flous :

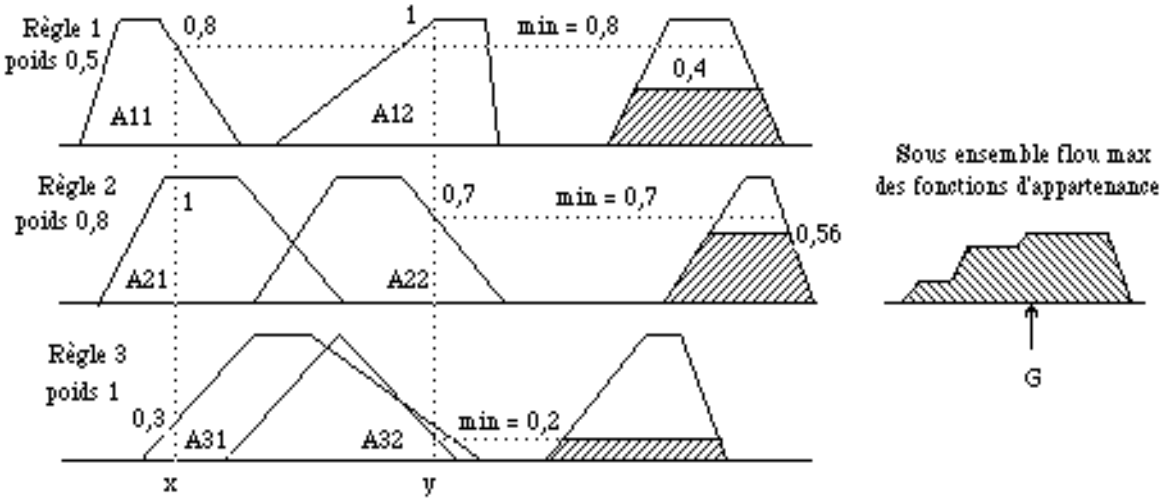


Cette disjonction d'un petit nombre de règles (entre 5 et 20), (on ne veut pas voir de contradictions entre ces règles, mais au contraire, les prédicats se chevauchant), concourent à la détermination du paramètre U dit "de contrôle".

La plupart du temps ces règles ont une, deux voire trois prémisses portant souvent sur un paramètre d'entrée e (erreur ou écart entre une valeur mesurée et une valeur cible ou "consigne" c) et sur la variation Δe de ce paramètre entre deux mesures consécutives (e et sa dérivée e' si on préfère). Ces systèmes sont utilisés pour des "régulateurs" mettant en cause des règles du type "si e est trop grand positif il faut le réduire, d'autant plus que de est grand algébriquement". Les transversales représentant à peu près des situations équivalentes.

de \ e	NB	NS	ZE	PS	PB
NB	PB				ZE
NS				ZE	NZ
ZE			ZE	NZ	NS
PS		ZE	NZ	NS	NM
PB	ZE	NZ	NS	NM	NB

La conjonction (des prémisses) est usuellement interprétée par l'opération min (Zadeh), et la disjonction (des règles) comme le max, ainsi pour deux valeurs précises x et y et trois règles affectées de poids, l'algorithme prévoit que la conclusion B soit tronquée à la hauteur p.min($\mu A_1(x)$, $\mu A_2(y)$) où μ_A représente la fonction d'appartenance (schéma ci-dessous). On obtient un sous-ensemble flou non obligatoirement normalisé pour la sortie U, et enfin une "défuzzification" ou "déflousification" par centre de gravité permet de donner une valeur exacte pour la sortie u. Il y a là un passage [quantitatif → qualitatif → quantitatif]. Par ailleurs il n'y a pas d'enchaînement logique de déductions, sauf à faire fonctionner deux contrôleurs successivement dans le but de déterminer deux paramètres.



Remarque : si les règles s'appliquent peu ou si leurs poids sont faibles (zone peu élevée) il y aura quand même une valeur G déduite. D'autre part il arrive que le produit soit choisi pour la conjonction (logique probabiliste) ou pour l'implication (opérateur de Larsen) ou encore, que des contrôleurs mixent ces lois.

Les principaux opérateurs logiques multivalués sont :

	Négation	**Conjonction**	**Disjonction**	**Implication**
Lukasiewiecz	1-a	max (a+b-1, 0)	min (1, a+b)	min(1- a + b, 1)
Probabiliste	1-a	ab	a+b-ab	si a<b alors a/b sinon 1
Zadeh	1-a	min (a,b)	max (a,b)	si a<b alors a sinon 1 (implication de Gödel non contraposable)

L'agrégation

Si un paramètre U est acquis de plusieurs façons floues par plusieurs règles, il se pose le problème de lui donner une représentation floue approximative, ou une valeur précise (defuzzification). On prend alors la médiane qui fait le partage de l'aire en deux, un mode (valeur donnant le max de u), ou le plus souvent la moyenne (centre de gravité)[Mamdani 75].
Une autre méthode [Sugeno 85] consiste à écrire des règles avec une conclusion précise, ce qui revient à définir une fonction par quelques points dont on étend les antécédents. La défuzzyfication se fait alors par le calcul de la moyenne pondérée des conclusions, ce qui est plus simple à réaliser.

Les systèmes experts flous, et le modus ponens généralisé

Le modus ponens généralisé consiste en la donnée d'une règle (X est A) $\rightarrow$ (Y est B) et d'un fait X est A', on en déduit Y est B', naturellement suivant certaines modalités où A' étant "proche" de A, alors B' est construit comme "voisin" de B.
La définition donnée par [Zadeh 73] en est : $\mu_{B'}(y) = \sup_{x \in U} (T (\mu_{A'}(x), \pi(x,y))$

(analogie avec la formule probabiliste $p(B) = \Sigma p(Ai).p(B/Ai)$)
T est une t-norme, c'est à dire une conjonction, et $\pi (x, y) = [\mu_A(x) \rightarrow \mu_B(y)]$ où $\rightarrow$ est une implication.

16-1° Longueur de liste en Fril, construire le prédicat "longueur" établissant un lien entre une liste et son nombre d'éléments. Les listes se délimitent avec des parenthèses plutôt que des crochets.

```
((longueur() 0))
((longueur (X|Y) C) (longueurY C1) (sum C1 1 C))
/* exemple qs ((longueur (s d f g) X )) provoquera une réponse unique X=4 */
```

16-2° Appartenance

```
((app X (X|Y) ))
((app X (Y|Z)) (app X Z))
/* exemple qs ((app a (f a c)))    qs ((app a (s n c f))) donne successivement */
((app a (f a c))) : (1 1) no (more) solutions
((app a (s n c f))) : (0 1)
/*Faire les calculs pour expliquer ce support, où rajouter une coupure pour éviter celà ?*/
```

16-3° Concaténation (concat X Y Z) concatène la liste X à la liste Y pour donner Z

```
((concat () X X))
((concat (X|Y) L (X|R)) (concat Y L R))
/* exemple qh ((concat X Y ( a b))) donne les réponses */
((concat () (a b) (a b)))
((concat (a) (b) (a b)))
((concat (a b) () (a b))) no (more) solutions
```

16-4° Miroir d'une liste

```
((reverse () () ))
((reverse (X|Y) Z)(reverse Y L) (append L (X) Z))
/* seconde définition maintenant classique */
((reverse X Y) (rev X () Y))
((rev (X|L1) A Y) (rev L1 (X|A) Y))
((rev () Y Y))
/* exemple dans les deux cas        qs ((reverse (a b c) X)) va donner */
((reverse (a b c) (c b a))) : (1 1) no (more) solutions
```

16-5° Tête Y est la liste préfixe de X formée par les N premiers éléments.

```
((tete N X Y) (concat Y _  X) (len Y N) )              /* la différence entre qh et qs */
qh ((tete 3 (a b c d e) X))
((tete 3 (a b c d e) (a b c))) no (more) solutions

/* une autre question, afin d'observer les confiances calculées  */ qs ((tete 3 (a b c d e) X))
((tete  3 (a b c d e) ())) : (0 1)
((tete  3 (a b c d e) (a))) : (0 1)
((tete 3 (a b c d e) (a b))) : (0 1)
((tete  3 (a b c d e) (a b c))) : (1 1) /* la seule intéressante */
((tete  3 (a b c d e) (a b c d))) : (0 1)
((tete  3 (a b c d e) (a b c d e))) : (0 1) no (more) solutions

((tete _ () ()))       /* autre définition */
((tete 0 _  ()))
((tete N (X | L) (X | M)) (sum N 1 N1) (tete N1 L M))
/* exemple  qs ((tete 3 (a b c d e) X))
((tete 3 (a b c d e) (a b c d e))) : (1 1) no (more) solutions */
```

16-6° Dernier élément d'une liste, éléments consécutifs

```
((dernier X L) (concat _ (X) L))
 /* éléments consécutifs */
((consecutifs X Y L) (concat _ (X Y |_) L))
/* paire d'éléments consécutifs */              ((adjacent X Y L) (consecutifs X Y L))
                                                ((adjacent X Y L) (consecutifs Y X L))
((sousens () Y))   /* sous-ensemble non ordonné */
((sousens (A | X) Y) (member A Y Y1) (sousens X Y1))
/* le "member" du lisp */
((member H (H | T) T))
((member H (H1 | T) (H1 | T1) ) (member H T T1))
```

16-7° Substitution, (substitute A L1 B L2) vrai ssi tout A de L1 est substitué par B dans L2

```
((substitute A () B () ))
((substitute A (A|L) B (B|L1)) (substitute A L B L1) (!))
/* la coupure étant là pour empêcher l'unification de A avec Y dans la 2° clause, en cas de succès à la 1°. */
((substitute A (Y|L) B (Y|L1)) (substitute A L B L1))

((subst A () B () ))
((subst A (A|L) B (B|L1)) (subst A L B L1))
((subst A (Y|L) B (Y|L1)) (neg (eq A Y)) (subst A L B L1))
/* Cette autre version contraignant la seconde clause si A ≠ Y grâce à la définition "non (X) si ( X et coupure et
impasse) ou (non (X))" de la négation par l'échec. */
```

Exemple qh ((substitute a (a b a c a) f X) (subst a (a b a c a) f Y)) donne successivement :

```
((substitute a (a b a c a) f (f b f c f)) (subst a (a b a c a) f (f b f c f)))
```

16-8° Compte des éléments d'une liste à toutes les profondeurs.

```
((liste () ))
((liste (_ | _) ))
((compte () 0))
((compte X 1) (neg (liste X)))       /* neg est la négation */
((compte (X | Y) N) (compte X P) (compte Y Q) (sum P Q N))
```

Exemple qh ((compte (a (b c) ((a)) (a (b (c) d) e) a) X)) donne :

```
((compte (a (b c) ((a)) (a (b (c) d) e) a) 10)) no (more) solutions
```

16-9° Enlevement de toutes les occurrences d'un X dans une liste.

La coupure signifiant que les solutions à gauche du "!" nous suffisent .

```
((del _  () () ))
((del X (X|L) M) (!) (del X L M))
((del X (Y|L1) (Y|L2)) (del X L1 L2))      /* attention aux pièges */

((del1 X () () ))
((del1 X (X|L) L))
((del1 X (Y|L1) (Y|L2)) (del1 X L1 L2))

((del2 X () () ))
((del2 X (X|L) M) (del2 X L M))
((del2 X (Y|L1) (Y|L2)) (del2 X L1 L2))
qs ((del1 a ( a b a c) A) )          ((del1 a (a b a c) (b a c))) : (1 1)
                                     ((del1 a (a b a c) (a b c))) : (1 1)
                                     ((del1 a (a b a c) (a b a c))) : (1 1) no (more) solutions
qs ((del2  a ( a b a c) A) )         ((del2 a (a b a c) (b c))) : (1 1)
                                     ((del2 a (a b a c) (b a c))) : (1 1)
                                     ((del2 a (a b a c) (a b c))) : (1 1)
                                     ((del2 a (a b a c) (a b a c))) : (1 1) no (more) solutions
```

16-10° Tri par segmentation, P est le pivot, YL, YR parties gauche et droite.

```
((tri () () ))
((tri (P | X) Y)  (partition X P Y1 Y2) (tri Y1 Z1) (tri Y2 Z2) (concat Z1 (P | Z2) Y))
((partition () X () () ))
((partition (Y1 | Y) X (Y1 | YL) YR) (less Y1 X) (partition Y X YL YR) )
((partition (Y1 | Y) X YL (Y1 | YR)) (neg (less Y1 X)) (partition Y X YL YR) )
Exemple (le "qs" donnant des "solutions" indéterminées)  qh ((tri (5 1 7 4 6 2) X)) donne :
((tri (5 1 7 4 6 2) (1 2 4 5 6 7))) no (more) solutions */
```

16-11° Calculs sur des supports à propos d'une base de recrutement d'employés.

```
((bon_employe X) (qualifie X) (motive X) (travailleur X) ) : (0.9 1)
((bon_employe X) (qualifie X) (motive X) ) : (0.6 1)
((bon_employe X) (qualifie X) ) : (0.3 1)
((qualifie Luc)) : (0.9 1)                ((qualifie Jean)) : (0.9 1)
((qualifie Max)) : (0.9 1)                ((motive Jean)) : (0.7 1)
((motive Max)) : (0.9 1)                  ((travailleur Max)) : (0.9 1)
qs ((bon_employe X))     /* On veut savoir qui peut être un bon employé,  réponses : */
((bon_employe Luc)) : (0.27 1)
((bon_employe Jean)) : (0.378 1)
((bon_employe Max)) : (0.6561 1) no (more) solutions
```

16-12° Former une petite base de données sur l'aspect, sécurité, moteur et coût des voitures.

```
(haut [0.1 : 0, 0.6 : 1])          (moyen [0.2 : 0, 0.5 : 1, 0.8 : 0])          (bas [0.4 : 1, 0.7 : 0])
(cher [30 : 0, 70 : 1])            (correct [20 : 0, 50 : 1, 60 : 1, 90 : 0])   (pascher [30 : 1, 80 : 0])
dempster choix
((choix X ok) (performance X bon) (aspect X moderne)) : (0.9 1)
((choix X ok) (prix X cher)) : (0 0.05)              ((choix X ok) (securite X bas)) : (0 0.2)
((performance X bon) (moteur X ok) (securite X haut)) : (0.9 1)
((performance X bon) (securite X haut)) : (0.6 1)
((aspect panhard moderne)) : (0.8 1)                 ((aspect levassor moderne)) : (0.9 1)
((securite panhard haut)) : (1 1)                    ((securite levassor moyen)) : (1 1)
((moteur panhard ok)) : (0.7 1)                      ((moteur levassor ok)) : (0.9 1)
((prix panhard correct)) : (1 1)                     ((prix levassor pascher)) : (1 1)
qs ((choix X Y)) /* question */
((choix panhard ok)) : (0.0852438 0.751709)
((choix levassor ok)) : (0.0825986 0.75536) no (more) solutions
```

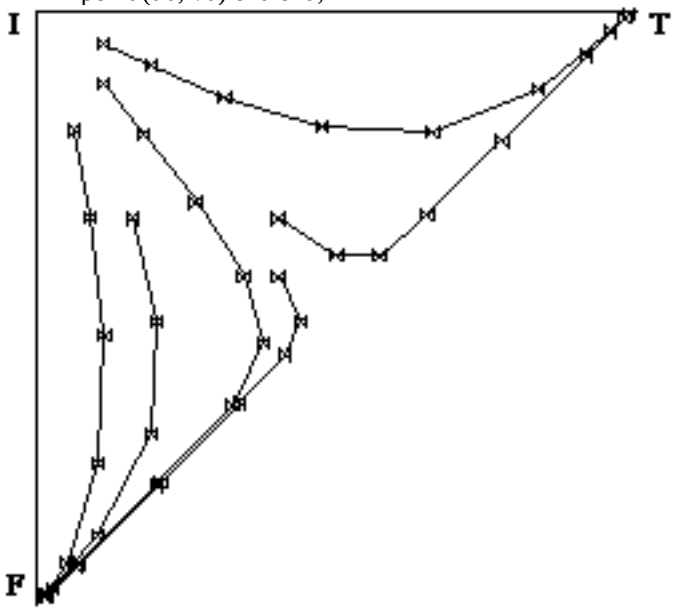
16-13° **La bourse,** On donne la définition des intervalles flous sur [-8, 8] ce sont des pourcentages (negfort [-8:1 -5:0])
(negfaible [-8:0 -5:1 -3:1 0:0])
(zero [-4:0 -1:1 1:1 4:0])
(posfaible [0:0 3:1 5:1 8:0])
(posfort [5:0 8:1])
Chaque titre en bourse (on en prendra 3) peut recevoir instantanément 5 attributs qui sont la variation du cours VC, l'écart avec la moyenne mobile à court terme (50 jours) EMCT, la variation du cours par rapport à la moyenne à court terme VCT, l'écart avec la moyenne à long terme (200 jours) EMLT, et enfin la variation de cet écart VLT.

```
((vc pechiney -3))                              ((emct pechiney 1))
((vct pechiney 2))                              ((vlt pechiney 5)) /* il faut plutôt en acheter */
((emct elf 15)) /* pas assez d'info */         ((emlt unionminiereduhautkatanga  -5))
((vct unionminiereduhautkatanga  1))            ((vlt unionminiereduhautkatanga 3)) /* il faut en acheter */
((vct galerieslafayette -1))                    ((vlt galerieslafayette -9))
((vc galerieslafayette 9))                      ((emlt galerieslafayette 3)) /* il faut en vendre */
/* Les règles pour déterminer un achat ou une vente sont, après avis des experts : */
((opinion A achat) (vc A posfaible) (emct A zero) (vct A negfaible)) : (0.6  0.8)
((opinion A achat) (vc A posfort) (emct A zero) (vct A negfort)) : (0.8  1)
((opinion A achat) (vc A posfaible) (emct A zero) (vct A negfort)) : (0.8  0.9)
((opinion A achat) (vc A posfort) (emct A zero) (vct A negfaible)) : (0.8  1)
((opinion A achat) (emct A posfaible) (vct A posfaible)) : (0.5  0.8)
((opinion A achat) (emct A posfaible) (vct A posfort)) : (0.7  0.9)
((opinion A achat) (vct A negfaible) (vlt A posfaible)) : (0.6  0.9)
((opinion A achat) (vct A negfort)) : (0.5  0.8)
((opinion A achat) (emlt A zero) (vlt A posfaible)) : (0.7  0.9)
((opinion A achat) (emlt A zero) (vlt A posfort)) : (0.8  1)
((opinion A achat) (vlt A negfort)) : (0.5  0.9)
((opinion A vente) (vc A negfaible) (emct A zero) (vct A posfaible)) : (0.6  0.8)
((opinion A vente) (vc A negfort) (emct A zero) (vct A posfort)) : (0.9  1)
((opinion A vente) (vc A negfaible) (emct A zero) (vct A posfort)) : (0.8  0.9)
((opinion A vente) (vc A negfort) (emct A zero) (vct A posfaible)) : (0.8  1)
((opinion A vente) (emct A negfaible) (vct A negfaible)) : (0.5  0.8)
((opinion A vente) (emct A negfaible) (vct A negfort)) : (0.7  0.9)
((opinion A vente) (vct A posfaible) (vlt A negfaible)) : (0.6  0.9)
((opinion A vente) (vct A posfort)) : (0.7  0.9)
((opinion A vente) (vc A negfaible) (emlt A zero) (vlt A negfaible)) : (0.7  0.9)
((opinion A vente) (vc A negfaible) (emlt A zero) (vlt A negfort)) : (0.8  1)
((opinion A vente) (vc A negfort) (emct A zero) (vct A zero)) : (0.4  0.9)
((opinion A vente) (emlt A zero) (vct A negfaible)) : (0.7  0.9)
((opinion A vente) (emlt A zero) (vct A negfort)) : (0.8  0.9)
((opinion A vente) (vlt A posfort)) : (0.5  0.9)
/* On posera la question : */          qs ((opinion A B))
((opinion pechiney achat)) : (0.111111 0.955556)
((opinion galerieslafayette achat)) : (0.5 0.9)
((opinion elf achat)) : (0 1)
((opinion unionminiereduhautkatanga achat)) : (0 1)
((opinion pechiney vente)) : (0.4 0.866667)
((opinion galerieslafayette vente)) : (0.0777778 0.988889)
((opinion elf vente)) : (0 1)
((opinion unionminiereduhautkatanga vente)) : (0 1) no (more) solutions
/* Conclusion, il faut plutôt vendre "Péchiney" et acheter "Galeries Lafayette" */
```

16-14° Etudier les itérés de la règle de Dempster pour un même couple (x, y) de crédibilité - plausibilité. Ecrire en Pascal, les procédures "dempster" pour deux couples (x, y) de [0, 1] tels que $x \leq y$. Et, pour des couples pris au hasard dans ce domaine on constatera le mouvement de leurs itérés vers le "vrai" ou le "faux" suivant la position de départ. On pourra aussi étudier la règle de Dempster, en voyant la déformation d'un maillage de l'ensemble des couples de confiance S.

Solution : les procedures "place" et "trace" étant toujours les mêmes :

```
function conflit (x1, y1, x2, y2 : real) : real;          begin conflit := x1*(1 - y2) + x2*(1 - y1) end;
procedure dempster (x1, y1, x2, y2 : real; var x, y : real);
        var c : real; begin c:= conflit(x1, y1, x2, y2); x := (x1 + x2 - x1*x2 - c)/(1 - c); y := y1*y2/(1 - c) end;
procedure point (x, y : real);
        begin place(x, y); trace (x + 0.01, y + 0.01); trace (x + 0.01, y - 0.01);
        trace (x - 0.01, y + 0.01); trace (x - 0.01, y - 0.01); trace (x, y) end;
procedure dessin; begin clearscreen; place (0, 0); trace (0, 1); trace (1, 1); trace (0, 0) end;
function hasard : real; {fournit un réel entre 0 et 1} begin  hasard := random/65536 + 0.5 end;
function xa (x, y : real) : real; {accentuation # d'un point (x, y) (dempster avec lui-même) de S }
        begin xa := (2*y - x)*x/(1 - 2*x + 2*x*y) end;
function ya (x, y : real) : real; begin ya := y*y/(1 - 2*x + 2*x*y) end;
procedure iter (x, y : real; n : integer); {trace les n itérations de # sur x, y}
        var i : integer; u0, v0, u1, v1 : real;
        begin point (x, y); u0 := x; v0 := y;
        for i := 1 to n do begin u1 := xa (u0, v0); v1 := ya (u0, v0); u0 := u1; v0 := v1; trace (u0, v0);
                        point (u0, v0) end end;
```



Si F (le faux) est le couple (0, 0), T (1, 1) le vrai, et I (0, 1) l'incertain, les itérés de quelques points au hasard, convergent vers F ou T.

```
procedure deformation1 {déformation par # du maillage i = y - x,w = (x + y) / 2};
        var i , w, x, y : real;
        begin w := 0;
        repeat {place (w,w);}place (xa (w, w), ya (w, w)) ; i := 0;
                repeat x := w*(1 - i); y := i + x; {trace(x,y)} trace(xa (x, y), ya (x, y)); i := i + 0.1 until i >1.1;
                w := w +0.1
        until w >1.1;
        i := 1;
        repeat place (0,1); w := 0; repeat x := w*(1-i); y := i + x; trace(xa (x, y), ya (x, y)); w := w + 0.1
                                        until w > 1.1;
        i := i - 0.1
        until i < -0.1 end;
procedure deformation2 {déformation par # du maillage x, y};
        var x, y : real;
        begin x := 0; y := 0; repeat place(x, y) { (xa(x, x), ya(x, x)) }; y := x;
                                        repeat trace(x,y) {(xa(x, y), ya(x, y)) }; y := y + 0.1 until y > 1.1;
                                        x := x + 0.1
                                until x > 1.1;
        y:= 1;
        repeat place (0, 1); x := 0; repeat trace (x, y) {(xa (x, y), ya (x, y)) }; x := x  + 0.1 until x > y + 0.1;
                                y := y - 0.1
        until y < -0.1 end;
```

16-15° Programmer un contrôleur flou en Pascal, et l'appliquer au problème du suivi d'une trajectoire sinusoïdale par un robot qui mesure à chaque période sa distance latérale suivant un degré d'ouverture ouv = 70° par rapport à sa direction, et la variation de cette distance. On utilisera un angle de braquage entre -A et A, une distance mesurée entre -B et B et sa variation entre -C et C.

dD \ D	NB	NS	ZR	PS	PB
NB				NS	
NS		NS	.	ZR	.
ZR	NB		ZR		PB
PS		ZR		PS	
PB		PS			

Règles de [Kamada Yoshida 90]

```
program ligne;
        uses memtypes, quickdraw; {uses crt, graph; en turbo 6}
        const H = 270  ; L = 480;          {hauteur, largeur de l'écran}
        type predicat = (NEG, POS, NB, NM, NS, ZE, PS, PM, PB, ANY);
        type trapeze = record  g, pg, d, pd : integer end;
                {bornes droite et gauches et pentes des trapèzes tous définis dans[-10,10]}
        type regle = record poids: real ; prem1, prem2, conc: predicat end;
                {chaque règle est de la forme si dl est A et dr est B alors u est C}
        type distribution = array [-10 .. 10] of integer;
                {représente après division par 10 les possibilités entre 0 et 100 pour -1, -0,9, -0,8 .... 0,9, 1
                (coder les possibilités de 0 à 100 évent. sur 7 bits serait raisonnable)}
        type baserel = array [NEG .. ANY] of trapeze;
        type basereg = array [1..10] of regle; {Ces 2 types imposés par pascal dans les en-têtes de procedures}
        var BP : baserel;  BR : basereg;
                ch : char; no, nsi, A, B, C, dm, ouv, borne, ds, k, dep1, dep2, rot,  m, d  : integer;

procedure initialisation (var BP : baserel; var BR : basereg; nr, p : integer);
        begin  {NO = nombre de règles, p = pente des trapèzes, nr = numéro de base de règles }
        BP[NB].g := -100 ; BP[NB].pg := 0 ; BP[NB].d :=-100 ; BP[NB].pd := p ;
        BP[NM].g := -60 ; BP[NM].pg := 20 ; BP[NM].d := -40 ; BP[NM].pd := 20 ;
        BP[NS].g := -p ; BP[NS].pg := p ; BP[NS].d := -p ; BP[NS].pd := p ;
        BP[ZE].g := 0 ; BP[ZE].pg := p ; BP[ZE].d := 0 ; BP[ZE].pd := p ;
        BP[PS].g := p ; BP[PS].pg := p ; BP[PS].d := p ; BP[PS].pd := p ;
        BP[PM].g := 40 ; BP[PM].pg := 20 ; BP[PM].d := 60 ; BP[PM].pd := 20 ;
        BP[PB].g := 100 ; BP[PB].pg := p ; BP[PB].d := 100 ; BP[PB].pd := 0 ;
        BP[ANY].g := -100 ; BP[ANY].pg := 0 ; BP[ANY].d := 100 ; BP[ANY].pd := 0;
{9 règles pour suivre une ligne blanche } NO := 9;
        BR[1].poids := 1; BR[1].prem1 := NB; BR[1].prem2 := PS; BR[1].conc := PS;
        BR[2].poids := 1; BR[2].prem1 := NS; BR[2].prem2 := NS; BR[2].conc := PS;
        BR[3].poids := 1; BR[3].prem1 := NS; BR[3].prem2 := PS; BR[3].conc := ZE;
        BR[4].poids := 1; BR[4].prem1 := any; BR[4].prem2 := NB; BR[4].conc := PB;
        BR[5].poids := 1; BR[5].prem1 := ZE; BR[5].prem2 := ZE; BR[5].conc := ZE;
        BR[6].poids := 1; BR[6].prem1 := any; BR[6].prem2 := PB; BR[6].conc := NB;
        BR[7].poids := 1; BR[7].prem1 := PS; BR[7].prem2 := NS; BR[7].conc := ZE;
        BR[8].poids := 1; BR[8].prem1 := PS; BR[8].prem2 := PS; BR[8].conc := NS;
        BR[9].poids := 1; BR[9].prem1 := PB; BR[9].prem2 := NS; BR[9].conc := NS  end;

function courbe (u : real ) : integer; begin courbe := m + round (H*sin(pi*u*nsi/L)/3) end;

procedure dessin (n : integer);{réalise le dessin de différents tunnels, 5: une ligne blanche }
        var u : integer; begin moveto (0, 0); lineto (L, 0); lineto (L, H); lineto (0, H); lineto (0, 0);
        u := 0 ; moveto (u, m) ; repeat u := u + 8 ; lineto (u, courbe (u) ) until u > L end;
```

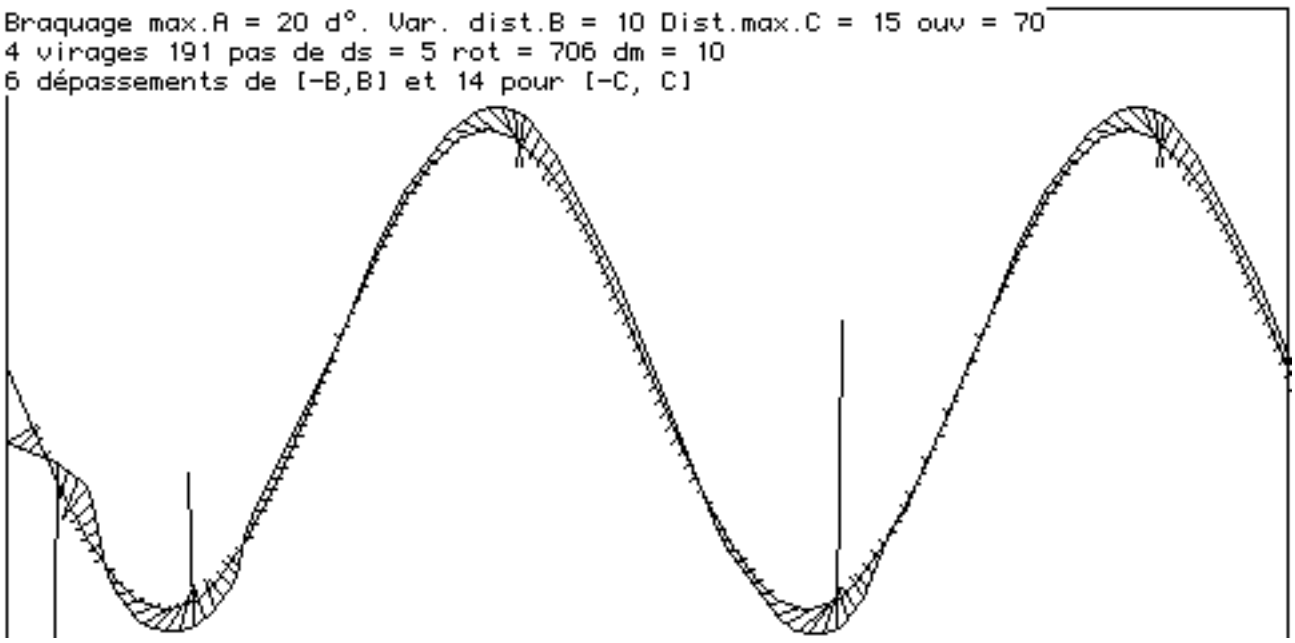
```
function ver (x : integer; t : trapeze) : integer; {donne son résultat en 100°}
        begin
        if x < t.d+1      then if t.g-1 < x then ver := 100
                                        else if t.g -t.pg <x      then ver := 100-(100*(t.g-x)) div (t.pg)
                                                                  else ver := 0
                          else if x < t.d+t.pd      then ver := 100-(100*(x-t.d)) div (t.pd)
                                                    else ver := 0 end;
function conj (a, b : integer) : integer; {une t-norme} begin if a < b  then conj := a else conj := b end;
procedure confrontation (x, y : integer; R: regle ; var U: distribution ); { construit une distribution U en
confrontant 2 valeurs x, y  à une règle R déterminant U, si aucune règle ne s'applique U=0 et l'angle déduit sera
nul}      var poss, i : integer;
        begin poss := round ( (R.poids)*conj (ver (x, BP[R.prem1]), ver (y, BP[R.prem2])));
        for i:= -10 to 10 do U[i]:= conj (poss, ver(i*10, BP[R.conc])) end;
procedure fuzzy (x, y : integer; var U : distribution ) ; { réalise les max en examinant les NO règles du tableau
BR x sera la variation de distance à gauche, y à droite ramenées à [-100,100]}
        var UR : distribution; i, r : integer ;
        begin for i := -10 to 10 do U[i] := 0 ; { U est construit suivant max}
        for   r := 1   to   NO   do       begin confrontation (x, y, BR[r], UR) ;
                                          for i := -10 to 10 do if U[i] < UR[i] then U[i] := UR[i] end end;

function moyenne (U : distribution) : integer;  {donne la moyenne entre -100 et 100 d'une distribution}
        var i, c, s : integer;
        begin c := 0; s := 0; for i := -10 to 10 do begin s := s + i*U[i] ; c := c + U[i] end;
        if c = 0 then moyenne := 0 else moyenne := (10*s) div c end;
function khi (x : integer) : integer; {ramene les variations dl, dr observées sur l'écran à [-B, B], à ajuster}
        begin if x > B  then begin dep1 := dep1 + 1; khi := 100 end
                        else if x < -B    then begin dep1 := dep1 + 1; khi := -100 end
                                          else khi := (100*x) div B end;
function phi  (x : integer) : integer; {ramène les distances observées sur l'écran à [-C, C], à ajuster}
        begin if x > C            then begin dep2 := dep2 + 1; phi := 100 end
                                  else if x <- C    then begin dep2 := dep2 + 1; phi:=-100 end
                                                    else phi := (100*x) div C end;
function psi (x : integer):  integer; {retourne un angle en degrés entre -A et A pour x dans [-100,100] }
        begin psi := (A*x) div 100 end;
function angle (x, y : integer) : integer;
{fonction générale calculant l'angle en d° à prendre suivant les observations d'écran}
        var   U : distribution; begin  fuzzy (khi(x), phi(y), U); angle := psi (moyenne(U))  end;
function tan (a : integer) : real; {tangente en degrés } begin tan := sin (a*pi/180) / cos (a*pi/180) end;
```

Nous présentons ici 3 solutions de calcul de distance sur l'écran, les deux dernières étant réservées à des courbes dont on connaît l'équation.

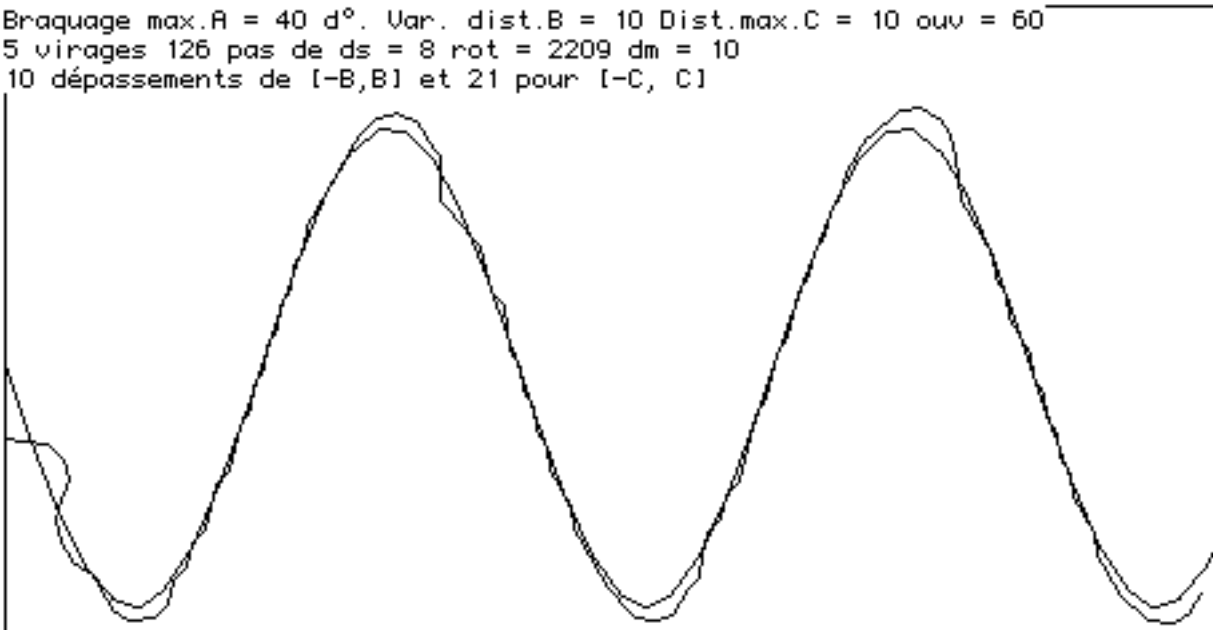
```
function dis1 (u, v, d: integer) : integer ; {renvoie la distance du point u,v à la paroi la plus proche, dans la
direction d  en degrés, par convention l'origine des angles est le point cardinal est et l'orientation est positive }
        var i, u0, v0 : integer; c, s: real;
        begin c := cos (d*pi/180); s:= sin (d*pi/180); i := 1;
        repeat i := i + 1; u0:= u + round (i*c); v0 := v - round(i*s)
        until getpixel (u0, v0) or (i>dm) or getpixel (u0-1, v0) or getpixel (u0, v0-1)  or getpixel (u0+1, v0)  ;
        dis1 := round (sqrt (sqr (u-u0)+sqr (v-v0))) end;
function dis2 (u, v, a : integer) : integer;{renvoie la distance algébrique à la courbe dans la direction a ° }
        var r, s : real; d : integer;
        function f (x : real) : real ; begin  f := v - (x - u)* tan(a) - courbe(x) end;
        function dicho (a, b : real) : integer; {donne à l'unité près l'abscisse d'écran de f(x)=0 }
                var p, q, r : real ;
                begin p:= a; q := b;
                repeat r := (p+q) /2; if f(p)*f(r) < 0 then q := r else p := r  until q - p < 1  dicho := round(r) end;
        begin  if abs(a) = 90 then r := u else r := dicho (u - L/nsi, u + L/nsi);
        s := courbe(r ); d := round (sqrt (sqr(u-r) + sqr (v-s)))
                {round (abs (u-r) + abs (v-s) ) serait la distance de Hamming};
        if (abs(a) - 90)*(r - u) < 0      then if abs(a) = 90       then if s > v     then dis2 := d
                                                                                      else dis2 := -d
                                                                    else dis2 :=  d
                                          else dis2 := -d end;
```

```
function dis3 (u, v, dir : integer) : integer ; {cherche suivant que le point est au dessus de la courbe ou non, le
moment où on traverse celle-ci. C'est cette distance qui est bien sûr la plus rapide à calculer}
        var p, q, t : real; a, c, s, i : integer; {c=1 est la droite, -1 la gauche}
        begin
        if dir = 0 then dis3 := v - courbe(u)
        else    begin  if v > courbe(u) then c := 1  else c := -1 ; a := dir  - c*ouv ;
                if abs (a) < 90 then s := 1 else s := -1; {s=1 si |a| < 90°}
                t := tan(a); i := 0;  {on augmente jusqu'à traverser la courbe}
                repeat p := v + s*i*t - courbe (u + s*i) ; i := i + 1 until (p*c <= 0) or (i > dm);
                        {lineto (u+s*i, round (v + s*i*t)) ; lineto (u, v); patte de mouche du dessin ci-dessous}
                        dis3 := c*round (i*sqrt (1 + t*t))
                end end;

procedure ligneblanche (u0, v0, dir0 : integer );
        var u, v, d, d0, dir, al :  integer;  {en degrés}
        begin moveto (u0, v0); dir := dir0; u: = u0; v := v0; d0 := ds; k := 0 ;
        repeat d := dis3 (u, v, dir ); {arrivé en u0, v0, on calcule dir }
                al := angle (d - d0, d ); rot := rot + abs (al); dir := dir + al;
                u := round (u + ds*cos (pi*dir/180)); v := round (v + ds*sin (pi*dir/180));
                d0 := d ; lineto (u, v) ; k := k+1
        until (L-u < d) or (k > borne) or (v > H) end;

begin clearscreen ; k := 0 ; dep1 := 0; dep2 := 0; rot := 0; d := H div 10; m:= H div 2;
initialisation (BP, BR, 5, 50);  ouv := 70; {demi-angle de vision} nsi := 4; {degré de sinuosité de la courbe}
        ds := 5;{avance à chaque pas} borne := 30; dm:= 10;  A := 20; {braquage maximum en degrés}
        B := 10; {variation de distance} C := 15 ; {distance considérée comme la plus grande }
        dessin (5);   ligneblanche (0, 6*d,  40) ;
        writeln ('Braquage max.A = ', A, ' d°. Var. dist.B = ', B,' Dist.max.C = ', C , ' ouv = ', ouv );
        writeln (nsi, ' virages ', k,' pas de ds = ', ds ,' rot = ', rot, ' dm = ', dm);
        write (dep1,' dépassements de [-B,B] et ', dep2 , ' pour [-C, C] ') end.
```

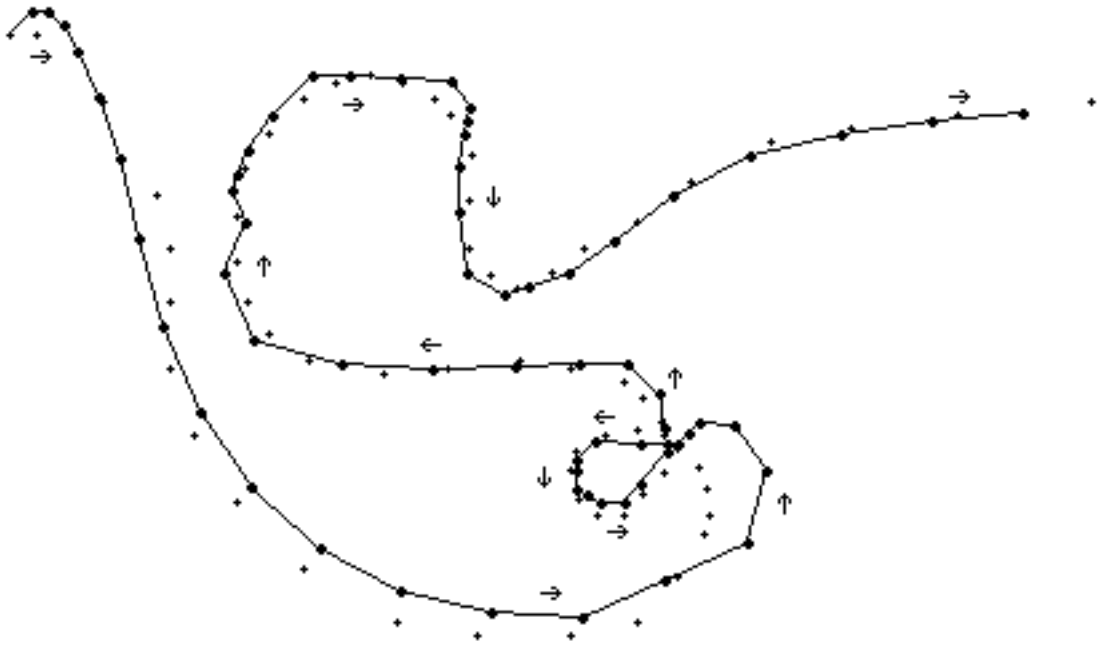
On a considéré pour l'application, une distance maximale dm = 10, au delà de laquelle on ne cherche plus à mesurer. Résultat obtenu en visualisant les prises de mesure des distances :



Résultat obtenu en modifiant les règles :
En remplaçant les deux règles "d est NB et dd est ZE → da est NB" ainsi que "d est PB et dd est ZE → da est PB" par les règles "d est NB → da est NB" et "d est PB → da est PB" ce qui revient à décrire les deux colonnes extrêmes de la table par seulement deux règles, on obtient un meilleur suivi :



16-16° Suivi d'une ligne de marqueurs, à chaque instant un robot mesure sa distance x et l'angle a de sa direction avec la prochaine borne.
En utilisant les règles suivantes (a ZE) → (∂dir ZE et∂p PB), (x ZE et a ZE) → (∂dir ZE et ∂p NB), (x PB et a PB) → (∂dir NS et ∂p is PB), (x PB et a NB) → (∂dir PS et ∂p PB), (x PB et a PS) → (∂dir NB et ∂p ZE), (x PB et a NS) → (∂dir PB et ∂p PB), (x ZE et a PB) → (∂dir NB et ∂p NB), (x ZE et a NB) → (∂dir PB et ∂p NB). On fixe les paramètres dm = 50, acc = 7, vm = 90°, braquage am = 50° pm = 35.



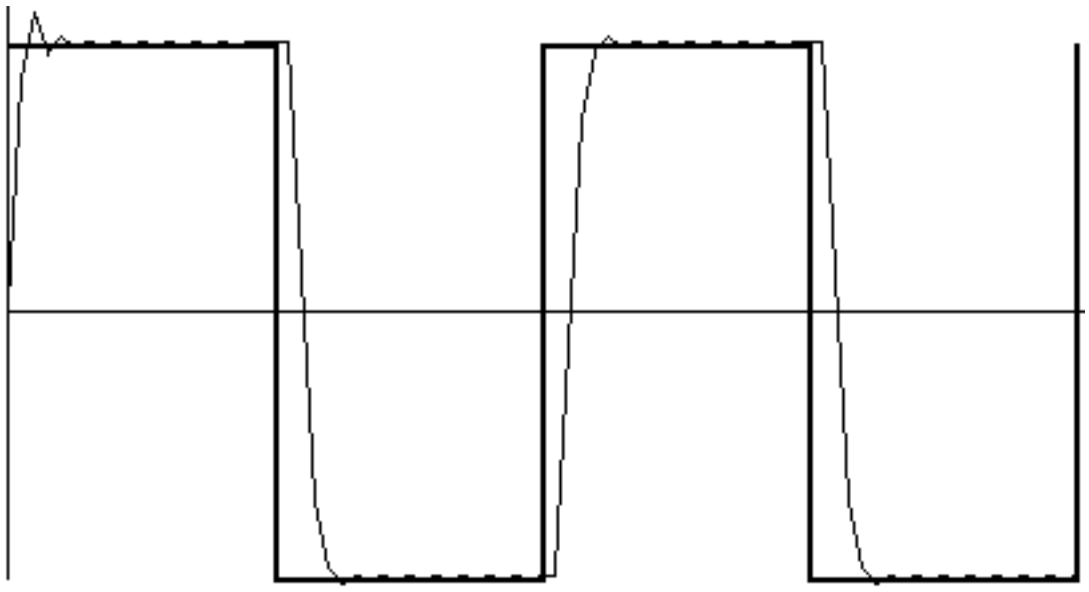
16-17° Ecrire en Lisp un contrôleur flou, version Mamdani, pour une table de règles qui sera lue suivant l'ordre de Cantor. On construira des histogrammes par 21 valeurs régulièrement répartis entre -1 et 1, et on parcourera la table des 25 règles (pour 5 prédicats) dans l'ordre de x + y croissant.

```
(de histog (TR) (cond; TR est le nom d'un trapèze que l'on modifie en histogramme de 21 valeurs entre -1 et  1
        ((eq TR 'ANY) (compte 21 1))  ; ANY est la fonction 1 (prédicat toujours vrai)
        ((eq TR 'NUL) (compte 21 0))  ; NUL est la fonction 0 (prédicat toujours faux)
        (t (let ((k 1) (d 0.1) (H nil) (a (car (eval TR))) (b (cadr (eval TR))) (al (caddr (eval TR)))
                (bt (cadddr (eval TR)))) ; compteur, décrément, liste des valeurs, a, b, al, bt est le trapèze TR
                (while (< (+ b bt) k) (set 'H (cons 0 H)) (set 'k (- k d)))
                (while (< b k) (set 'H (cons (1+ (divide (- b k) bt)) H)) (set 'k (- k d)))
                (while (< a k) (set 'H (cons 1 H)) (set 'k (- k d)))
                (while (< (- a al) k) (set 'H (cons (1+ (divide (- k a) al)) H)) (set 'k (- k d)))
                (while (< -1 k) (set 'H (cons 0 H)) (set 'k (- k d)))
                (cons (if (eq TR 'NB) 1 0) H)) )))

(de compte (n v) (if (eq n 0) nil (cons v (compte (1- n) v))))
(de poss (x P) (cond ; donne la fonction d'appartenance de x au prédicat P
        ((eq P 'NUL) 0)
        ((eq P 'ANY) 1)
        ((and (<=  x -1) (eq P 'NB)) 1)
        ((and (<= 1 x) (eq P 'PB)) 1)
        (t (apply 'posbis (cons x (eval P)))) ))
(de posbis (x a b al bt) (cond ; le prédicat trapézoïdal (a b alpha beta), de noyau [a, b], et support [a-α, b+β]
        ((<= x (- a al)) 0)
        ((< x a) (divide (- (+ x al) a) al))
        ((<= x b) 1)
        ((< x (+ b bt)) (divide (- (+ b bt) x) bt))
        (t 0)))
(de mini (x y) (if (< x y) x y))

(de maxi (L M)  (cond ; réalise la liste composée des max dans les listes L et M
        ((null L) M)
        ((null M) L)
        ((< (car L) (car M)) (cons (car M) (maxi (cdr L) (cdr M))))
        (t (cons (car L) (maxi (cdr L) (cdr M))))))
(de troncature (L v) ; L est un histogramme, on renvoie le min avec v
        (if (null L) nil (cons (mini (car L) v) (troncature (cdr L) v))))
(de moyenne (L) (mbis L -1 0 0)) ; donne l'abscisse du barycentre
(de mbis (LR k S pr ) (if (null LR) (if (eq pr 0) 0 (divide S pr))
                (mbis (cdr LR) (+ k 0.1)(+ (* k (car LR)) S) (+ pr (car LR)))))

(de cantor (i j) (if (<= (+ i j) (1+ npr))
        (+ (1- j) (div (* (- (+ i j) 2) (1- (+ i j))) 2))
        (- (* npr (1+ npr)) i (div (* (- (1+ (* 2 npr)) i j) (- (+ 2 (* 2 npr)) i j)) 2))))

(de fuzzy (C x y) ; x et y sont les entrées, C l'ensemble des règles, PRED une variable globale ci-dessous
        (fuzzybis C x y PRED PRED PRED 1 1 nil nil))
(de fuzzybis (C x y PRED Px Py i j R v)
(cond ; donne le sous ensemble flou max, Px et Py sont les listes de prédicats correspondant à une flèche
        ((null Px) (* am (moyenne R))); am désigne plus loin l'angle maximal
        ((or (null Py) (eq v 0)) (fuzzybis C x y PRED (cdr Px) PRED (1+ i) 1 R nil))
        ((null v)  (fuzzybis C x y PRED Px Py i j R (poss x (car Px))))
        (t (fuzzybis C x y PRED Px (cdr Py) i (1+ j)
                (maxi R (troncature (cassoc (nth (cantor i j) C) HISTO) (mini v (poss y (car Py))))) v) )))

(setq PRED '(NB NS ZE PS PB) NB '(-1 -1 0 0.5) NS '(-0.5 -0.5 0.5 0.5) ZE '(0 0 0.5 0.5)
        PS '(0.5 0.5 0.5 0.5) PB '(1 1 0.5 0) npr 5 ; nb de prédicats
        HISTO (mapcar (lambda (x) (cons x (histog x))) PRED) am 1) ; définit les constantes
```

16-18° Application à un suivi de créneaux : soit une consigne c alternativement 1 et -1, que doit suivre un contrôleur flou dont les entrées seront l'erreur e = c - x et la variation ∂e de cette erreur entre deux prises de mesures. On prendra 10 mesures par palier et 4 paliers, mesurer alors l'erreur relative moyenne pour ces 40 étapes sachant qu'initialement x = ∂e = 0 et que les règles sont : (e est NB) et (∂e est NB) --> (u est PB)
(e est NB) et (∂e est ZE) → (u est PB) (e est NB) et (∂e est PB) → (u est PB)
(e est NS) et (∂e est NS) → (u est PS) (e est NS) et (∂e est PS) → (u est PS)
(e est ZE) et (∂e est NS) → (u est ZE) (e est ZE) et (∂e est ZE) → (u est ZE)
ainsi que les configurations symétriques.

Solution :
La liste des 25 règles parcourue dans l'ordre de Cantor va s'écrire (pb nul nul nul ps pb nul ze nul nul nb ns ze ps pb nul nul ze nul nb ns nul nul nul nb) dont seules les 13 premières valeurs seront considérées dans la liste C plus bas.

```
(de moitie (C) (moitiebis C nil))
(de moitiebis (C R) (cond ; donne la moitié symétrique R de C
        ((null C) R)
        ((atom C) (moitiebis (list C) R))
        ((eq (car C) 'nb) (moitiebis (cdr C) (cons 'pb R)))
        ((eq (car C) 'ns) (moitiebis (cdr C) (cons 'ps R)))
        ((eq (car C) 'pb) (moitiebis (cdr C) (cons 'nb R)))
        ((eq (car C) 'ps) (moitiebis (cdr C) (cons 'ns R)))
        (t (moitiebis (cdr C) (cons (car C) R)))))

(de complete (C) (append C (cdr (moitie C)))) ; réalise la table complète à partir des 13 premiers termes

(de valeur (C) (repeat 20 (print)) (move-to 10 30) (pen-size 1 1) (line-to 10 250)
        (line-to 10 150) (line-to 500 150) (move-to 10 50) (pen-size 2 2)
        (let ((i 0) (co -1) (u 0) (x 10) (e -1) (de 0) (val 0) (CH (complete C)))
        (repeat 2 (line-to (+ 110 (* 200 i)) 50) (line-to (+ 110 (* 200 i)) 250)
                (line-to (+ 210 (* 200 i)) 250) (line-to (+ 210 (* 200 i)) 50) (set 'i (1+ i)))
        (pen-size 1 1) (move-to 10 150)
        (repeat 4 (set 'co (- co))
                (repeat 20 (set 'u (+ u (fuzzy CH e de)))
                        (set 'de (- u co e)) (set 'e (- u co)) (set 'x (+ x 5))
                        (line-to x (truncate (- 150 (* 100 u)))) (set 'val (+ val (abs e)))))))

(valeur '(pb nul nul nul ps pb nul ze nul nul nb ns ze))
```



16-19° Construire un contrôleur flou symbolique de type Sugeno très général de R^n dans R^p, où les règles sont du type ((c1 c2 ... cp) p1 p2 ..... pn) pour n entrées et p sorties (que des symboles pour les prémisses et des symboles dont on prendra les sommets, ou bien des nombres en conclusions)

```
(de repete (x L) ; produit une liste d'autant de fois x qu'il y a d'éléments dans L
        (ifn (null L) (cons x (repete x (cdr L)))))
(de maxi (a b) (if (< a b) b a))
(de ramene (x a) (cond ; réalise une bijection de [-a, a] dans [-1, 1]
        ((< x (- a)) - 1)
        ((< a x) 1)
        (t  (divide x a))))
(de sommet (S) ; donne l'abscisse du sommet du prédicat (r = 1/2)
        (selectq  S (NB -1) (NS -0.5) (ZE 0) (ANY 0) (PS 0.5) (PB 1)))
(de opp (pr) ; donne l'opposé du prédicat pr
        (selectq  pr (NB 'PB) (NS 'PS) (ZE 'ZE) (ANY 'ANY) (PS 'NS) (PB 'NB)))

(de coef (E P) (coefbis E  P nil))) ; coef d'application de la règle de liste de prémisses P pour l'entrée E = (e1 e2

(de coefbis (E P LC) ; LC liste de coefficients
        (if (or (null E) (null P)) (tnorm LC) (coefbis (cdr E) (cdr P)
        (cons (if (eq (car P) 'ANY) 1 (maxi 0 (- 1 (divide (abs (- (sommet (car P)) (car E))) r)))) LC))))
(de tnorm (L) (if (null L) 1 (minibis (car L) (cdr L)))) ; on a choisit la t-norme de Zadeh
(de minibis (X L) (cond
        ((null L) X)
        ((> X (car L)) (minibis (car L) (cdr L)))
        (t (minibis X (cdr L)))))
(de mini (a b) (if (< a b) a b))
(de fuzzy (E LR) ; fonction principale, donne une liste en sortie dans [-1, 1]^p avec LR liste de règles
        (fuzzybis (coef E (cdar LR)) E (caar LR) (repete 0 (caar LR)) 0 (cdr LR)))
(de fuzzybis (co E LC S SC LR) ; c coeff E entrées LC liste des conclusions S sortie, SC somme des coef
        (fuzzyter E (ajou co (if (numberp (car LC)) LC (mapcar 'sommet LC)) S) (+ co SC) LR))
(de fuzzyter (E S SC LR) ; E vecteur entrée, S vecteur sortie
        (if  (null LR) (if (eq SC 0) (repete 0 S) (mapcar (lambda (x) (divide x SC)) S))
        (fuzzybis (coef E (cdar LR)) E (caar LR)  S SC (cdr LR))))

(de ajou (co L M) ; produit la liste co*L + M terme à terme
        (if (or (null L) (null M)) nil (cons (+ (* co (car L)) (car M)) (ajou co (cdr L) (cdr M)))))
```

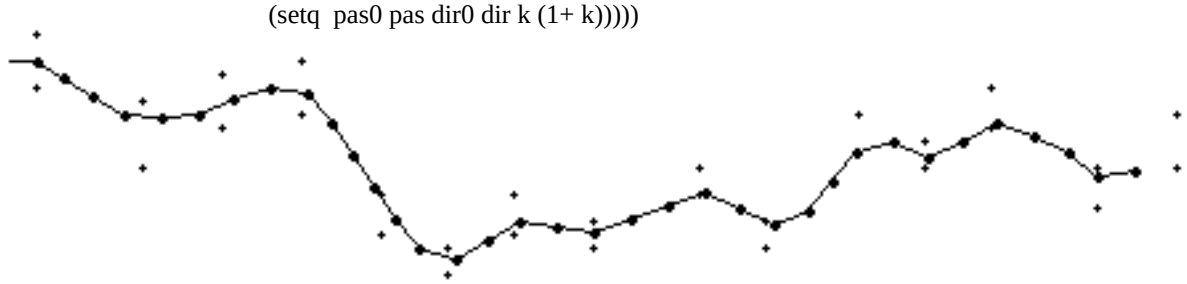
16-20° **Application à un parcours de slalom.**
On définit des règles portant sur deux entrées, les angles que font la direction du robot avec les deux portes de la prochaine porte à passer, et deux sorties, la variation d'angle dans ±1 radian et l'accélération dans [-4, 4]. Les règles sont : a1 est quelconque et a2 est NB alors ∂a est PB et ∂pas est NB) ainsi que (any nb pb pb) (any ns ze ps), (any ps nb pb), (ns ps pb nb), (ze ps ps pb) et leurs symétriques. On donne une piste où chaque triplet correspond à une abscisse de porte et aux deux ordonnées des côtés de cette porte : (setq PISTE '((20 70 90) (60 95 120) (90 85 105) (120 80 100) (150 130 145) (175 150 160) (200 130 145) (230 140 150) (270 120 130) (295 140 150) (330 100 115) (355 110 120) (380 90 105) (420 120 135) (450 100 120)))
Définir la symétrique d'une règle et le parcours du robot dont la performance sera jugée par le nombre de portes ratées (somme des distances à ces portes) plus 5 fois le nombre de portes non atteintes en 35 pas plus le nombre de règles utilisées.

Solution :

```
(setq PRED '(ANY NB NS ZE PS PB) NH 2 NC 2 pi 3.14159 r 0.5 km 35 k0 (length piste) L 450 ds 4 pm 20
acc 4  vm 1.0 am (// (* pi 60) 180))
(de sym (R) (cons (list (opp (caar R)) (cadar R)) (list (opp (caddr R)) (opp (cadr R)))))
```

On définit la tenue de route d'une trajectoire comme la moyenne de | (|da| / am) + (∂pas / 2acc) - 1/2| qui est donc d'autant meilleure qu'elle est proche de 0, mauvaise vers 1/2 et dangereuse vers 1.

```
(de valeur (C)  ; C = (((da dx) a1 a2) ((da' dx') a1' a2') .....)
        (let ((che piste) (u  10) (v 80) (dir0 0) (dir 0) (pas0 ds) (ch (complete C))
                (pas (div (+ pm ds) 2)) (F nil) (k 0) (tr 0) (val 0) (od 0) (u1 0)(v1 0))
        (while (and che (< k km)) (setq uc (caar che) vc (cadar che) wc (caddar che) che (cdr che))
                (croix uc vc) (croix uc wc) (move-to u v)
                (while (and (< u uc) (< k km))
                        (set 'F (fuzzy (list (ramene (rpi (- dir (arc u v uc vc))) vm)
                                        (ramene (rpi (- dir (arc u v uc wc))) vm))  CH))
                        (setq dir (+ dir (* (car F) am)) pas (mini (maxi ds (+ pas (* (cadr F) acc))) pm))
                        (setq u1 (+ u (truncate (* pas (cos dir)))) v1 (+ v (truncate (* pas (sin dir)))))
                        (setq od (if (eq u u1) 0 (truncate ( + v (* (- v1 v) (divide (- uc u) (- u1 u)))))) u u1 v v1)
                        (line-to u v) (point u v) ; od est l'ordonnée
                        (setq  pas0 pas dir0 dir k (1+ k)))))
```

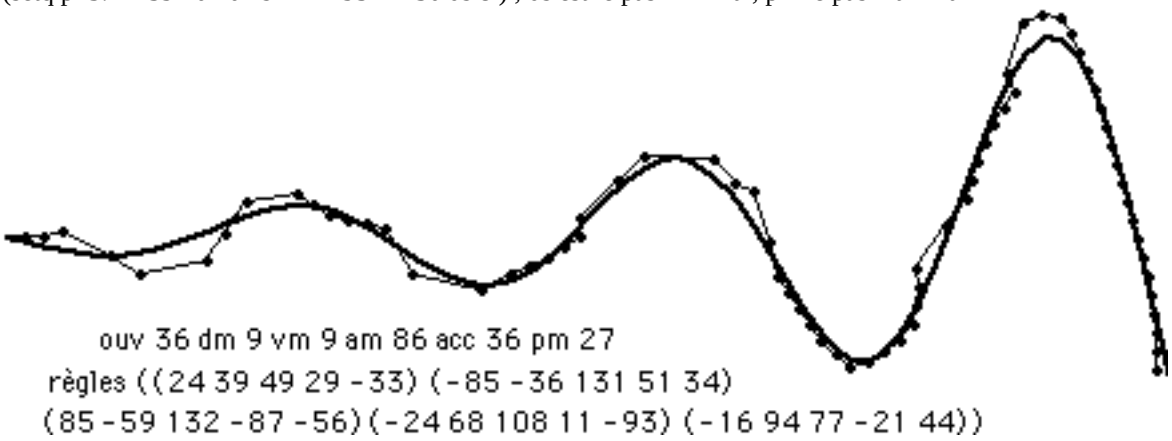


```
(((pb pb) any nb) ((ze ps) any ns) ((nb pb) any ps) ((pb nb) ns ps) ((
ps pb) ze ps) ((ze pb) ze ps) ((pb nb) any nb)) 35 pas. Tenue de route
50 Parcours restant 3 %  vm =68 acc =3 am =49 pm =15 Portes ratées 1
 Valeur 58
```

16-21° Contrôleur flou numérique type Sugeno de R^2 dans R^2 où les règles sont du type (a b r c d) (c, d) étant le couple conclusion (braquage et variation de vitesse) et (a, b) le centre, r le rayon du prédicat flou hypothèse (distance à la route et variation de cette distance).
On l'appliquera au suivi d'une sinusoïde amortie, les données étant structurées par une liste C de la forme (ouv dm vm am pm (a b r c d) (a' b' r' c' d') .....).

```
(de maxi (a b) (if  (< a b) b a))
(de mini (a b) (if (< a b) a b))
(de ramene (x a) ; ramène x à a exprimé en centièmes
        (if (< a (abs x)) (* (sgn x) 100) (* 100 (divide x a))))
(de fuzzy (LR x y) ;renvoie le couple (u v) de [-100, 100] en fonction de x y de [-100, 100]
        (if (null LR) '(0 0) (fuzzybis (car LR) x y 0 0 0 (cdr LR))))
(de fuzzybis (R x y u v Sc RR) ; règle R, u, v numérateurs, Sc somme des coefficients
        (fuzzyter (maxi (- 1 (divide (maxi (abs (- x (car R))) (abs (- y (cadr R)))) (caddr R))) 0)
                (cadddr R) (car (last R)) x y u v Sc RR))
(de fuzzyter (cf c d x y u v Sc RR) ; cf dans [-1, 1] est le coef d'application de la dernière règle étudiée
        (if     (null RR) (if (eq (+ cf Sc) 0) '(0 0)
                (list (divide (+ u (* cf c)) (+ Sc cf)) (divide (+ v (* cf d)) (+ Sc cf))))
                (fuzzybis (car RR) x y (+ u (* cf c)) (+ v (* cf d)) (+ Sc cf) (cdr RR))))
Exemple :
(fuzzy '( (0 0 2 2 4) (2 0 2 6 8) ) 1 0) = (4 6)
(de rpi (a) (cond ; représentation dans ]-pi, pi]
        ((< pi a) (rpi (- a (* 2 pi))))
        ((<= a (- pi)) (rpi (+ a (* 2 pi))))
        (t a)))
(de courbe (u) (truncate (+ 120 (* 150 (exp (divide u -150)) (sin (// (* pi nsi u) L)))))) ; sinusoide amortie
(de courbebis (u) (f (- L u))) ; pour l'avoir dans l'autre sens
(de point (u v) (pen-size 2 2) (line-to (1+ u) v) (line-to u (1+ v))(line-to (1- u) v)
        (line-to u (1- v)) (line-to (1+ u) v) (line-to u v) (pen-size 1 1))
(de dicho (a b phi eps) ; donne l'unique c entre a et b à eps près tel que phi soit nul
        (dichobis a b (divide (+ a b) 2) phi eps))
```

```
(de dichobis (a b c phi eps) (cond
        ((< (abs (- b a)) eps) c)
        ((<= (* (funcall phi a) (funcall phi c)) 0) (dichobis a c (divide (+ a c) 2) phi eps))
        (t (dichobis c b (divide (+ b c) 2) phi eps))))

(de distance (u v dir ouv dm) (dist0 u v dir (courbe u) ouv dm)) ; renvoie (distance et ordonnée) dir en radians
(de dist0 (u v dir w ouv dm) (if (eq v w) (list 0 w) (dist1 u v dir w (if (> v w) 1 -1) ouv dm)))
(de dist1 (u v dir w cote ouv dm)(cond
        ((eq (abs (- dir (* cote ouv))) (// pi 2)) (line-to u w) (line-to u v) (list (- v w) w))
        (t (dist2 u v w cote (rpi (- dir (* cote ouv))) dm))))
(de dist2 (u v w cote a dm) (list (dist3 u v cote (sin a) (cos a) dm) w))
(de dist3 (u v cote si co dm) (dist4 ; calcule par dichotomies une distance bornée dans [-dm, dm]
        (if (< 0 (* cote (+ v (* dm si) (- (courbe (+ u (* dm co))))))) (* cote dm)
        (// (- (dicho u (+ u (* dm co))
                (lambda (x) (- (courbe x) v (// (* (- x u) si) co))) 1) u) (* co cote))) u v si co cote))
(de dist4 (x u v si co cote) ; (line-to (truncate (+ u (* x co cote))) (truncate (+ v (* x si cote)))) (line-to u v)
                x)
(de valeur (C) ;  C=(ouv dm vm am acc pm (a b r c d) (a' b' r' c' d') .....)
(let ((u0 0) (v 120) (u  0) (v0 120) (w0 120) (dir0 1) (dir 1) (pas (nth 5 C)) (pas0 ds)(F nil) (x 0) (x0 0) (k 0)
        (ouv (// (* pi (car C)) 180)) (dm (maxi ds (cadr C))) (am (// (* pi (cadddr C)) 180))
        (vm (caddr C))  (acc (nth 4  C)) (pm (nth 5 C)) (ch (complete C)) )
        (pen-size 2 2) (move-to u v) (while (< u L) (set 'u (+ u 6)) (line-to u (courbe u)))
        (pen-size 1 1) (set 'u 0) (set 'v 120) (move-to u v)
        (while (and(<  k k0)(< u L)(<= 0 u) (< (abs(- v0 w0)) (* 19 ds)))
                (set 'u (+ u0 (truncate (* pas (cos dir))))) (set 'v (+ v0 (truncate (* pas (sin dir)))))
                (line-to u v) (point u v) (set 'x (distance u v dir ouv dm))
                (set 'F (fuzzy (nthcdr 6 ch) (ramene (car x) dm) (ramene(- (car x) x0) vm)))
                (set 'dir (rpi (+ dir (divide (* (car F) am)100))))
                (set 'pas (mini (maxi ds (+ pas (divide (* (cadr F) acc)100))) pm))
                (setq pas0 pas dir0 dir x0 (car x) w0 (cadr x) u0 u v0 v k (1+ k)))
        (print  '"Parcours restant " (set 'u (- 100 (div (* 100. (maxi 1 (abs u))) L)))'" % " k '" pas.) ))

(setq pi 3.14159 k0 40 nsi 7 L 495 H 150 ds 8 ) ; ds est le pas minimal, pm le pas maximal
```



On peut mesurer la norme de la différence des deux fonctions (elle vaut 1697 pour le dessin)

```
(de integ (u) (- (* 120. u) (*  495.5 (exp (divide u -150))
        (+ (sin (divide (* pi nsi u) L)) (// (* pi nsi 150. (cos (divide (* pi nsi u) L))) L)))))
(de integbis (u) (- (integcourbe (- L u)))) ; primitive de la sinusoïde amortie de droite à gauche
(de norm1 (u0 v0 u1 v1)
        (abs (+ (divide (* (- u1 u0) (+ v1 v0)) 2) (integ u0) (- (integ u1)))))
(de norme (u0 v0 w0 u1 v1 w1) (cond ; donne la norme L1 de la diff. "courbe" - segment 0-1
        ((eq u0 u1) 0)
        ((< 0 (* (sgn (- v0 w0)) (- v1 w1))) (norm1 u0 v0 u1 v1))
        (t (norm2 u0 v0 u1 v1
                (dicho u0 u1 (lambda (x) (+ (divide (*(- v1 v0) (- x u0)) (- u1 u0)) v0 (- (courbe x)))) 1)))))
(de norm2 (u0 v0 u1 v1 ui)  (+ (norm1 u0 v0 ui (+ v0 (divide (* (- v1 v0) (- ui u0)) (- u1 u0))))
        (norm1 ui (+ v0 (divide (* (- v1 v0) (- ui u0)) (- u1 u0))) u1 v1)))
```

16-22° Application de R^3 dans R^2 : un robot avance dans un couloir sinusoïdal en prenant la mesure de sa distance frontale et de ses distances à $\pm\pi/3$ de sa direction. On utilisera les règles (0 150 -20 50 40 -25) (100 50 25 50 -50 -100) (100 50 50 50 -100 60) (120 0 60 50 -50 -100) (40 120 60 5 75 0) (75 25 0 60 -75 -50) (0 60 40 5 -100 -100) (80 75 80 15 -5 0) ainsi que leurs symétriques où (a b c r da dp) désigne la règle de conclusion (da dp) pour le point (a b c) autour duquel la règle reste valable avec le rayon r, et les paramètres :
NH = 3, NC = 2 (3 hypothèses, 2 conclusions), pi = 3.14159, r = 0.5, nsi = 6 (6 sommets), L = 495, H = 80, M = 100, ds = 5, pm = 25 (pas minimal et maximal), acc = 5 (accélération maximale), dm = 55, ecart = 40 (entre les côtés), am = 50° (angle maximal de braquage).

Point, rpi, dicho sont identiques à l'exercice précédent.
On définiera la symétrique de la règle (a b c r da dp) comme la règle (c b a -da dp) qui entraîne un comportement symétrique face aux parois de droite et de gauche.

```
(de complete (C) (elim (append C (mapcar 'sym C))))

(de sym (R) (cons (list (opp (caar R)) (cadar R)) (list (cadddr R) (caddr R) (cadr R)))) ; dépend du problème
```

Ici, la distance à une courbe dépend de la fonction f.

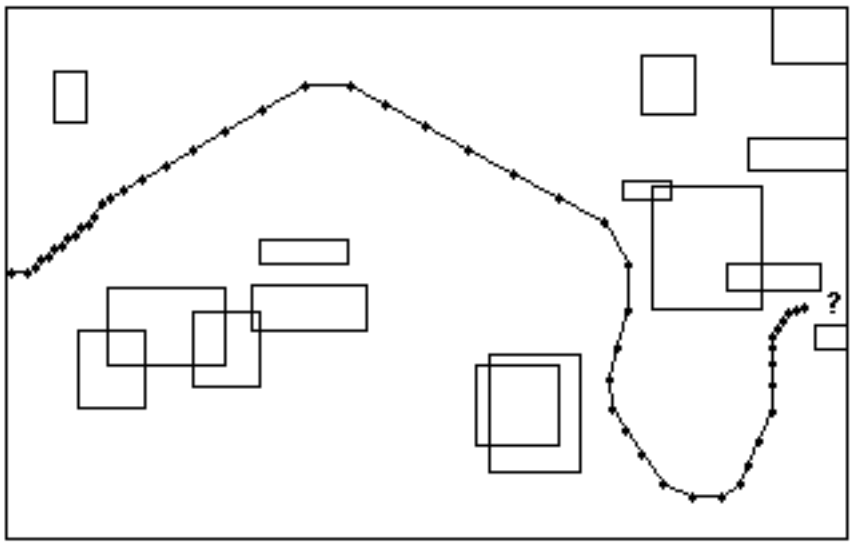
```
(de distance (u v a dm f) (dist1 u v a (funcall f u) dm f)) ; renvoie distance, dir en radians
(de dist1 (u v a w dm f) (cond       ; ici la distance est toujours positive
          ((eq (abs a) (// pi 2)) (line-to u w) (line-to u v) (abs (- v w)) )
          (t (dist3 u v w (sin a) (cos a) dm f))))

(de dist3 (u v w si co dm f) (dist4 ; calcule par dichotomies une distance bornée dans [-dm, dm]
          (if (< 0 (* (- v w) (+ v (* dm si) (- (funcall  f (+ u (* dm co)))))))  dm
          (// (- (dicho u  (+ u (* dm co)) (lambda (x) (-  (funcall f x) v  (// (* (- x u) si) co)))1) u) co))  u v si co))
(de dist4 (x u v si co) x)

(de courbe1 (u) (truncate (+ M (* H ; (exp (divide (- L u) (- H))); facteur pour la sinusoide amortie non utilisé ici
                          (sin (// (* pi nsi (- L u)) L))))))
(de courbe2 (u) (+ ecart (courbe1 u)))

(de valeur (C)  (print '" règles " C) ; C=((a b c  r  al  ac) (a' b' c' r' al' ac') .....)
          (let ((u  0) (v 0) (dir0 0)(dir -1) (pas0 ds) (pas (div (+ pm ds) 2)) (F nil) (k 0) (ch (complete C)) (tr 0))
          (pen-size 2 2) (move-to u (courbe1 u)) (while (< u L) (set 'u (+ u ds)) (line-to u (courbe1 u)))
          (set 'u 0) (move-to u (courbe2 u)) (while (< u L) (set 'u (+ u ds)) (line-to u (courbe2 u)))
          (pen-size 1 1) (set 'u 0) (set 'v (+ M (// ecart 2))) (move-to u v)
          (while (and (<  k k0) (< u L) (<= 0 u) (< (courbe1 u) v) (< v (courbe2 u)))
                    (set 'u (+ u (truncate (* pas (cos dir))))) (set 'v (+ v (truncate (* pas (sin dir)))))
                    (line-to u v) (point u v)
                    (set 'F (fuzzy (list          (ramene (distance u v (- dir (// pi 3)) dm 'courbe1) dm)
                                                  (ramene (mini (distance u v dir dm 'courbe1)
                                                                (distance u v dir dm 'courbe2)) dm)
                                                  (ramene (distance u v (+ dir (// pi 3)) dm 'courbe2) dm)) ch))
                    (set 'dir (+ dir (* (car F) am)))  (set 'pas (mini (maxi ds (+ pas (* (cadr F) acc))) pm))
                    (set 'pas0 pas) (set 'dir0 dir) (set 'k (1+ k))) ))
```

16-23° On matérialise une piste de largeur constante, par une bande pour $0 \le x \le L$ où la pente est donnée par $p_x = -\sin(\pi x / L)$. Un skieur la parcourt de gauche à droite en connaissant à chaque instant sa position x, $h_1 < y < h_2$ et sa direction α par rapport à la plus grande pente. Sa vitesse est alors donnée par un "pas" tel que $ds \le pas < pm$. Il ne peut contrôler que sa direction en modifiant son cap grâce à $-am \le \partial\alpha \le am$. Sa vitesse est modifiée par $\partial p = acc\ (p_x - p_0) / (1- p_0)$. Son parcours est stoppé dès que $pas \ge pm$ où si y ne respecte plus $h_1 < y < h_2$ ou encore si $x \ge L$. Dans tous les cas sa performance est la minimisation du nombre de pas. Un skieur est une liste de règles floues réalisant une fonction (p_x, y, α)--> $\partial\alpha$. Les paramètres L, am, ds, pm, p_0 seront à fixer suivant la simulation désirée. Ce problème s'écarte du suivi de ligne par un robot : il s'agit ici d'aller vite sans suivre la ligne de plus grande pente, ni de trop s'approcher des bords de la piste. Une stratégie d'évolution appropriée doit faire émerger un comportement global : une trajectoire sinueuse dont les oscillations sont d'autant plus ressérée que la pente est forte.

16-24° Parcours d'obstacles, chercher des règles pour éviter des obstacles en accélérant dans les lignes droites et en ralentissant dans les courbes. A chaque instant on captera la distance et l'angle polaire de l'obstacle le plus proche.

Indication, si d_m est par exemple 40, la distance maximale de reconnaissance, et $a_m = 45°$ l'angle maximal de reconnaissance et de braquage, on considère les obstacles comme une liste de couples de coordonnées autour desquels on doit rester à plus de ds = 4 par exemple.

```
(de arc (a b c d) (cond ; donne l'arc en radians du vecteur (a, b) (c, d) avec l'horizontale
        ((eq b d) (if (< a c) 0 pi))
        ((eq a c) (if (< b d) (// pi 2) (- (// pi 2))))
        ((< c a) (if (< b d) (+ pi (arc c d a b)) (- (arc c d a b) pi)))
        (t (atan (divide (- d b) (- c a))))))
(de degre (a) (truncate (* 180 (// a pi))))
(de radar (u v dir L d a) ; donne le couple (d a) de l'obstacle de L le plus proche de (u v) dans la dir
        (if (null L) (list d a)
        (radar1 u v dir (abs (- u (caar L))) (abs (- v (cadar L))) L d a)))
(de radar1 (u v dir x y L d a)
        (if (or (< dm x) (< dm y)) (radar u v dir (cdr L) d a)
        (radar2 u v dir x y (- (arc u v (caar L) (cadar L)) dir) (cdr L) d a)))
(de radar2 (u v dir x y ag L d a)
        (if (or (< am ag) (< ag (- am))) (radar u v dir L d a)
        (radar3 u v dir (- (sqrt (+ (* x x) (* y y))) ds) ag L d a)))
(de radar3 (u v dir di ag L d a) (if (< di d) (radar u v dir L di ag) (radar u v dir L d a)))
```



16-25° Ecrire en lisp la fonction "modus-ponens généralisé" qui à deux fonctions d'appartenance μ_A, $\mu_{A'}$, μ_B, une t-norme et une implication, associe une fonction d'appartenance $\mu_{B'}$ approchant μ_B.

Conjonctions [Bouchon 93]

```
(de weber (a b) (cond ((eq a 1) b) ((eq b 1) a) (t 0)))
(de lukas (a b) (maxi 0 (1- (+ a b))))
(de einstein (a b) (divide (* a b) (- (+ (* a b) 2) a b)))
(de proba (a b) (* a b))
(de hamacher (a b) (if (eq (zadeh a b) 0) 0 (divide (* a b) (- (+ a b) (* a b)))))
(de zadeh (a b) (if (< a b) a b))
```

Implications

```
(de reichenbach (p q) (1+ (- (* p q) p)))
(de willmott (p q) (maxi (- 1 p) (zadeh p q)))
(de mamdani (p q) (zadeh p q))
(de rescher (p q) (if (< p q) 1 0))
(de kleene (p q) (maxi (- 1 p) q))
(de godel (p q) (if (< q p) q 1))
(de goguen (p q) (if (eq p 0) 1 (zadeh 1 (divide q p))))
(de implukas (p q) (- 1 (maxi 0 (- p q))))
(de larsen (p q) (* p q))
```

Modus-ponens

```
(df mpg (norme implic muA muB muAp)
        (list 'lambda '(y) (list 'mpgbis norme implic muA (list 'funcall muB 'y) muAp -1 0)))

(de mpgbis (norme implic muA my muAp x sup)
        (if (> x 1) sup
        (mpgbis norme implic muA my muAp (+ 0.1 x)
        (maxi sup (funcall norme (funcall muAp x) (funcall implic (funcall muA x) my))) ))
(de maxi (a b) (if (< a b) b a))
```

Exemple :

```
(de muA (x) (cond ((< x -1) 0) ((< x -0.5) (* 2 (1+ x))) ((< x -0.2) 1) ((< x 0) (* x -5)) (t 0)))
(de muAp (x) (cond ((< x -0.8) 0) ((< x -0.6) (+ 4 (* 5 x))) ((< x -0.4) (- -2 (* 5 x))) (t 0)))
(de muB (x) (cond ((< x 0) 0) ((< x 4) (* x 0.15)) ((< x 5) (- (* 0.4 x) 1)) ((< x 6) 1)
                ((< x 9) (- 1.6 (* x 0.1))) ((< x 10) (-  7 (* x 0.7))) (t 0)))
(de phi (x) (truncate (+ 10 (* 30  (1+ x)))))
(de phir (u) (1- (divide (- u 10) 30)))
(de psi (y) (truncate (+ 50 (* 100 (- 1 y)))))
(de graphe (f) (move-to 0 150); trace la fonction f pour 0 < y < 1 et -1 <  x < 1
        (let ((u 0)) (repeat 400 (line-to u (psi (funcall f (phir u)))) (set 'u (1+ u))) ))

(de essai (norme implic) (pen-size 2 2) (graphe 'muA) (pen-size 1 1) (graphe 'muAp)
        (pen-size 2 2) (graphe 'muB) (pen-size 1 1) (graphe (mpg norme implic 'muA 'muB 'muAp))
        (print "'t-norme : " norme "' implication : " implic))

(essai 'hamacher 'implukas)
```