CHAPITRE 12

LES SYSTEMES-EXPERTS L'UNIFICATION

L'objectif de l'intelligence artificielle est à long terme de faire effectuer par un système informatique, tout ce que peut faire l'homme en matière de raisonnement. Elle est née des recherches en démonstration automatique et en traduction automatique amorçées dès les années soixante.
Le problème central rencontré dans tous les domaines de l'intelligence artificielle est celui de la modélisation de la connaissance.
Au premier stade de la connaissance, il y a des données brutes telles que celles que l'on représente habituellement dans un fichier, puis les concepts qui sont des regroupements d'objets partageant une propriété, puis des relations : liens non nécessairement déductifs, et enfin des algorithmes, stratégies ou heuristiques.
Aujourd'hui, la réalisation la plus connue de l'intelligence artificielle est le système-expert. C'est un système de déduction automatique utilisant deux types de connaissances, des faits et des règles de production de connaissance servant à enrichir, par de nouveaux faits, la base des faits initiaux.

Les domaines d'utilisation des systèmes-expert sont extrêmement divers, soit il s'agit de reconnaître un objet ou un évenement à partir de connaissances parcellaires voire imprécises ou incertaines, c'est le diagnostic, soit, mais cela revient au même, il s'agit de déterminer une action à partir de la description d'un contexte, et c'est l'aide à la décision.
Les premiers systèmes les plus célèbres sont MYCIN (diagnostic médical 1970), PROSPECTOR (recherche en géologie), et DENDRAL (composition chimique 1965).
Citons encore TAIGER et SNARK (J.L. Laurière) qui est lui un langage de représentation de connaissances, déductif, permettant l'écriture de systèmes-expert sur des thèmes variés comme l'intégration des fonctions, l'archéologie, le bridge, la conduite du R.E.R. etc...

Fonctionnement d'un système-expert

Si l'on s'en tient à l'essentiel, le fonctionnement dans le sens déductif (chaînage-avant), est extrêmement simple à expliquer. Il y a une partie algorithmique : le programme, dont le noyau est le "moteur d'inférence", et une partie déclarative constituée d'un dictionnaire des faits, d'une liste de variables utilisables, éventuellement d'une liste de faits terminaux, et d'une base de règles généralement toutes de la forme "clauses de Horn", c'est-à-dire : P_1 et P_2 et...et $P_n \rightarrow Q$ (les P_i sont les prémisses ou hypothèses, et Q est la conclusion).
Une session consiste alors à choisir des faits dans le dictionnaire, faits dont on sait qu'ils sont vrais ou faux, et à lancer le moteur, lequel balaye la liste des règles dans l'ordre. Dès qu'une règle peut se déclencher parce que ses prémisses sont parmi les faits assurés (qu'ils soient sûrement vrais ou sûrement faux), alors la conclusion est incorporée à cette "base de faits" et le moteur repart au début de la base de règles. Le moteur s'arrête dès qu'un fait terminal est obtenu, ou bien si un balayage complet ne produit plus rien, auquel cas un dialogue peut être provoqué avec l'utilisateur.
L'ensemble des règles est donné par un expert, il peut atteindre plusieurs centaines, celles-ci peuvent être groupées en paquets, ou hiérarchisées, on parle en ce cas de méta-règles voire de méta-méta-règles..., mais en tout cas ce n'est pas un ensemble définitivement immuable ou

complet, il peut être sujet à des réajustements, c'est ce qui fait la souplesse des systèmes-experts.

Un exemple de moteur sans variables, les quadrilatères du plan :

Le dictionnaire utilisé	La base de règles
L losange	L → P
P parallélogramme	P et X → L
R rectangle	P → T
T trapèze	A et R → C
S les 4 côtés sont égaux	C → R
X les diagonales sont perpendiculaires	S → L
D il y a un angle droit	A et X → V
A deux côtés adjacents sont égaux	D et S et T → R
C carré	A et P et X → L
V cerf-volant	S → A

Qu'obtient-on avec la base de faits initiale (S D) ?

Un exemple de système-expert avec variables, caractérisation des fromages [Gacôgne 88]

Le but est de déterminer un fromage grâce à des caractéristiques d'aspect (blanc, bleu, croûte lavée, croûte fleurie, etc ...) mais aussi sur les dimensions ou le goût (maigre, fort, gras ...), afin d'arriver à un fait terminal (brie, st- nectaire ...).
Ce système utilise donc une base d'une trentaine de règles que l'on pourrait facilement étendre:

(mou)(fleuri)(vache)(-brie)(DM =10)--->(camembert)	(mou)(vache)(lavé)(MT = 40)--->(livarot)
(mou)(jaune)(-cuit)(5 > DT)--->(st-nectaire)	(mou)(lavé)(vache)(DM > 15)--->(maroilles)
(gras)--->(MT > 60)	(mou)(fleuri)(vache)(DM > 20)--->(brie)
(chèvre)--->(MT = 45)	(maigre)--->(20 > MT)
(-trous)--->(DT = 0)	(dur)(jaune)--->(cuit)
(-vache)(-chèvre)--->(brebis)	(bleu)(-fourme)--->(lavé)
(dur)(brebis)(5 > DT)(jaune)--->(pyrénées)	(cuit)(-trous)--->(hollande)
(DT > 4)(10 > DT)(cuit)--->(gruyère)	(hollande)(MT = 60)--->(gouda)
(bleu)(brebis)--->(roquefort)	(hollande)(-gouda)--->(édam)
(vache)(5 > DT)(cuit)--->(comté)	(cuit)--->(dur)
(-brebis)(-chèvre)--->(vache)	(10 > DM)(dur)(-cuit)--->(chèvre)
(lavé)(DM > 10)--->(-vache)	(chèvre)--->(-trous)
(jaune)--->(-bleu)	(bleu)(mou)(lavé)--->(bleu-d-auvergne)
(DM > 20)(bleu)(lavé)--->(gorgonzola)	(5 > DM)(chèvre)--->(picodon)

Le système utilise une méta-règle (la contraposition) consistant en une règle de production de connaissances, non pas sur les propriétés d'un fromage particulier, mais sur les règles elles-mêmes. Plus précisément, chaque fois qu'une règle n'ayant qu'une seule prémisse P → Q se présente, alors naturellement si P est un fait acquis, Q en sera déduit; sinon on regarde si la négation de Q est dans la base de faits, en ce cas la négation de P devient un fait prouvé. La contraposition est une propriété de logique formelle exprimant que (P → Q) ⇒ (-Q → -P) en désignant par -P la négation de P.

Ainsi la déduction "Max est marié → Max a plus de 18 ans" est équivalente à "Max a moins de 18 ans → Max est célibataire".

On a d'autre part une partie dialogue demandant à l'utilisateur des précisions supplémentaires au cas où aucun fait terminal n'est obtenu après épuisement des règles. Ce dialogue est fondé sur une liste des règles comportant deux prémisses vraies sur trois, règles qui sont mises en réserve au cours du balayage de la base.

Le dictionnaire est constitué par la liste (fleuri lavé dur mou bleu blanc jaune gras maigre fort frais cuit trous chèvre brebis vache), les faits terminaux sont (camenbert st-nectaire pyrénées gruyère rocquefort comté gorgonzola livarot maroilles bris gouda édam bleu-d-auvergne picodon).

Les variables utilisables sont DM (dimension en cm), DT (diamètre des trous en mm), MT (matière grasse en %), les relations possibles sont < et = .

Exemples de listings d'utilisation :

Donnez la liste des faits initiaux: (gras lavé bleu) Donnez des faits valués: (DM = 12) (gras)--->(MT > 60) contraposée de: (chèvre)--->(MT = 45) (lavé) (DM > 10)--->(-vache) (-vache)(-chèvre)--->(brebis) (bleu) (brebis)--->(roquefort) voilà!	Donnez la liste des faits initiaux: (-trous jaune mou) Donnez des faits valués: () (-trous)--->(DT = 0) (jaune)--->(-bleu) Savez-vous si (oui non ?) cuit: ? Savez-vous si (oui non ?) dur: non élimination de: (dur) (jaune)--->(cuit) contraposée de: (cuit)--->(dur) (mou) (jaune) (-cuit)(5 > DT)--->(st-nectaire) voilà!

On remarquera la contraposition, et le fait (DM = 12) qui a permis d'évaluer à vrai la prémisse (DM > 10) dans le premier exemple. Il est intéressant de noter par ailleurs qu'un fait qualitatif "gras" entraîne un autre fait qualitatif "non chèvre", par l'intermédiaire d'un fait quantitatif "MT > 60".
On remarque dans le second exemple, que le dialogue permet d'enregistrer de nouveaux faits et de supprimer des règles qui ne serviront plus.

Description sommaire de l'algorithme

Moteur pour BR, base de règles, B état courant de la liste des règles restant à passer en revue, BF liste des faits, et enfin FT liste des faits sur lesquels on doit s'arrêter.

On décrit maintenant l'algorithme du moteur d'inférence à la manière lispienne, c'est-à-dire par une suite de condition qui doivent être regardées de haut en bas. Il est assez facile de le formaliser par deux fonctions mutuellement récursives "moteur" et "dialogue", il est par contre plus délicat de réfléchir à tous les arguments devant intervenir dans ces fonctions.

_ Si BF $\cap$ FT $\neq \emptyset$ alors fini
_ Si B = $\emptyset$ alors échec mais éventuellement dialogue
_ Si les prémisses de la première règle de B sont dans BF alors on relance le moteur avec
BR auquel on retire cette règle et dans son entier (on repart au début)
et avec BF augmenté de la conclusion de la règle.
_ Si la première règle de B n'a qu'une prémisse et si le contraire de la
conclusion est dans BF alors le contraire de la
prémisse est mis dans BF, la règle est retirée,
et on relance le moteur au début des règles qui restent.
_ Sinon moteur avec les règles suivantes.

Nous abordons avec l'exempe suivant un aspect plus général aux systèmes de déduction, les systèmes de réécriture.

Exemple d'unification : l'analyse syntaxique des phrases

Lors de l'analyse d'un texte en français, un certain nombre d'étapes sont à faire :
1° Analyser la morphologie des mots, par exemple "ferme" reçoit quatre valeurs possibles (nom fém sing) (adv manière) et (verbe présent sing. 1° ou 3° pers.). Nous ne parlerons pas de ce problème ici.
2° La "désambiguïsation "syntaxique permet ensuite de lever la plupart des choix possibles, ainsi "le soir il lit dans son lit", correspond à la séquence grammaticale : (DETerminant NOM PROnom Verbe PREPosition ADJectif-possessif NOM), alors que "La belle porte le voile" peut s'interpréter par (DET NOM V DET NOM) ou par (DET ADJ NOM PRO V).
3° L'étape suivante consiste à reconnaître le ou les verbes principaux, et à constituer les groupes nominaux, verbaux, circonstanciels etc... en vue d'une représentation interne sous forme de "graphes conceptuels" reliant les groupes verbaux à leurs sujets, compléments, etc...

Le programme proposé maintenant, est un simple mécanisme de reconnaissance, non pas de la correction grammaticale, mais de la compréhensibilité, ce qui est fort différent, mécanisme intervenant une fois la chaîne des catégories DET NOM ADJ etc... obtenue pour une phrase. Il n'est pas exhaustif et ne prétend pas résoudre la question dans tous les cas, mais constituer un exercice.
On écrit pour cela quelques règles récursives de formation des groupes nominaux comme NOM $\rightarrow$ GN puis GN ADJ $\rightarrow$ GN .
Pour des raisons de commodité de programmation, on écrit la conclusion en premier, ce qui peut se lire comme "on peut avoir un GN (groupe nominal) si on a déjà un GN suivi d'un ADJ". De même, un GV (groupe verbal) suivi d'un autre GV, constitue un GV.
Les catégories INF, GP, DA, et d'autres ne correspondent pas nécessairement à celles de la grammaire scolaire traditionnelle.

Règle de formation des groupes nominaux et exemples

```
(ADJ DA ADJ)        ; très beau (adjectif modifié par un adverbe)
(GN ADJ GN)         ; bon travail
(GN GN ADJ)         ; renard roux
(GN GN PPS)         ; chat pelé
(GN DET GN)         ; un beau chat tigré
(GN NOM)            ; chien
(GN PRO)            ; celui-là
(GN GN GP)          ; chien dans la niche, GP est groupe propositionnel
```

Règles de formation des groupes verbaux

```
(ADV DA ADV)        ; extrêmement peu , "très très beaucoup" sera compris comme
                    ; un adverbe ( DA signifie adverbe adjectivant )
(GV ADV GV)         ; bien faire
(GV GV ADV)         ; faire bien
(GV CONJ GV)        ; et venir
(GV GV GV)          ; aller et venir
(GV V)              ; dort
```

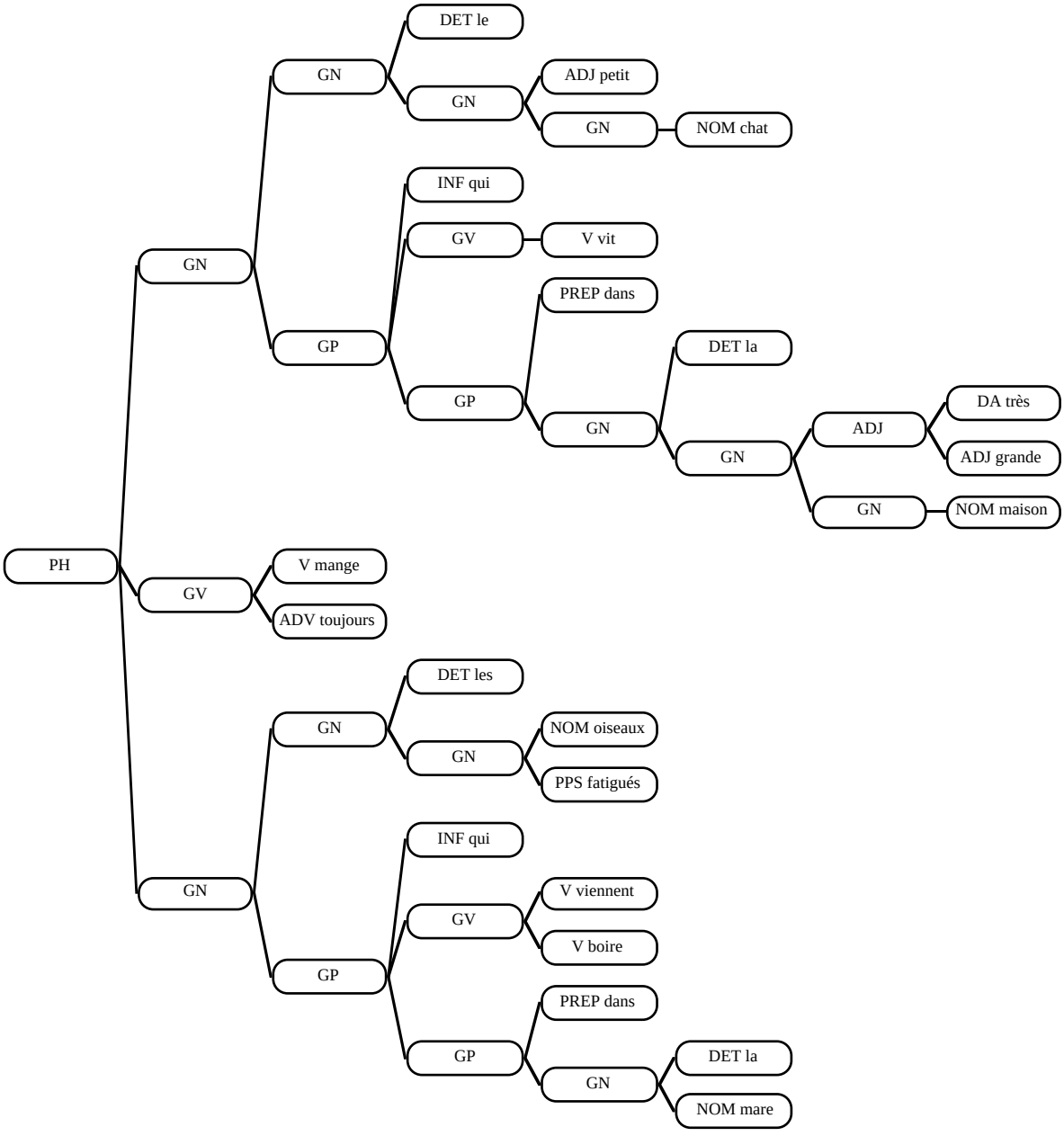
Règles de formation des groupes infléchis

```
(GP PREP GN)        ; sur le mur
(GP INF GV GN)      ; que mange le chat , INF est inflexion (qui que quoi dont où)
(GP INF GV GP)      ; qui est sur le mur
(GP INF GV ADJ)     ; qui est roux
(GP INF GV)         ; qui marche
(GP INF PH)         ; dont on sait l'histoire
```

Règles de formation de la phrase

```
(PH GN GV GP)       ; le petit chat dort sur le mur
(PH GN GV GN)       ; le chat mange le rat
(PH GN GV ADJ)      ; ce chat est noir
(PH GN GV)          ; le chat dort
```

Exemple de phrase compréhensible : Le petit chat qui vit dans la très grande maison, mange toujours les oiseaux fatigués qui viennent boire dans la mare.



Programmation de l'unification pour l'analyse des phrases

Pour comparer une liste L à une règle R de conclusion C, on parcourt simultanément L et les prémisses (cdr R) , LR et PR désignent ce qu'il faut encore comparer. La liste L consiste en ce qui est vu LV et ce qui reste à voir qui est noté LR. On voit la notion, chère aux pré-informaticiens, de rubans qui défilent. "uf" (fonction d'unification d'une liste L de catégories grammaticales, par une règle) va appeler dès que cela est possible dans la lecture de L, la fonction "ufbis".

```
(de ufbis (LV LR PR C R) (cond
                ((null PR) (cons C LR))
                        ; il n'y a plus de prémisses, la conclusion s'écrit en ce cas.
                ((null LR) (append LV LR)) ; pas d'unification possible.
                ((eq (car PR) (car LR))
                        (ufbis (append LV (list (car LR))) (cdr LR) (cdr PR) C R))
                (t  (append LV (UF LR R)))))  ; pas d'unification, mais peut être plus loin.
```

"uf" donne la liste L transformée par la règle R plusieurs fois si nécessaire, elle appelle "ufbis" pour le début de la liste L s'il y a une chance "d'unification", et sinon elle appelle UF pour ce qui suit. Ainsi pour la liste (DET NOM V DET NOM) et la règle (GN DET NOM), "uf" donnera (GN V GN).

```
(de uf (L R) (cond
                ((null L) L)
                ((eq (car L) (cadr  R)) (ufbis (list (car L)) (cdr L) (cddr R) (car R) R))
                (t (cons (car L) (UF (cdr L) R)))))
```

Dans ce qui suit L1 est le résultat d'une unification de L0 avec une règle donc avec la première de B dans DEM , B désigne ce qui reste de règles, et BR est la base initiale de règles.

```
(de dembis (L0 L1 B BR) (cond
                ((null B) L1)
                ((equal L0 L1) (dembis L0 (UF L0 (car B)) (cdr B) BR))
                ; On passe à la règle suivante au cas où la règle ne s'est pas appliquée.
                (t (dembis L1 (UF L1 (car BR)) (cdr BR) BR))))
                ;dès qu'une unification réussit on revient au début de la base de règles.

(de dem (L B) (dembis L (uf L (car B)) (cdr B) B))          ; est la fonction principale
```

Par exemple "Le très petit beau chat gris noir toujours mange partout les petits fatigués oiseaux", qui n'est sans doute pas correcte, mais est indéniablement compréhensible, sera en correspondance avec la liste de ses catégories (DET DA ADJ ADJ NOM ADJ ADJ ADV V ADV DET ADJ PPS NOM) qui se réduira à PH.
D'autre part, une phrase correcte mais difficilement compréhensible comme "L'homme qui a vu l'homme dont la belle-soeur a des ouistitis qui lui ont été donnés par un explorateur du katanga septentrional où j'avais un oncle qui était atrabilaire, est chauve", se réduira de la même façon.

L'unification en général

Dans un système comme celui qui précède, on a vu qu'il s'agissait de confronter une expression E avec une "règle" $H \rightarrow C$ (H comme hypothèse, C comme conclusion). Ce problème peut s'avérer beaucoup plus délicat lorsqu'il s'exerce avec des formules où certains identificateurs désignent des variables, alors que GN, ADJ, etc... représentaient des constantes.
Soit par exemple $(x+y)^2 \rightarrow x^2 + 2xy + y^2$, cette formule constitue une règle de réécriture qui, appliquée à $(a+2b)^2$ par exemple, doit fournir $a^2 + 2a(2b) + (2b)^2$. Afin d'éviter toute ambiguïté, le premier travail est de renommer les variables de la règle de sorte qu'aucun nom ne soit commun entre la règle et l'expression. D'autre part, et c'est une seconde difficulté, l'unification doit être tentée avec toutes les sous-expressions.
Par exemple $a^2 - (a + (b + c)^2)^2$ peut s'unifier trois fois ou bien une seule fois avec H suivant la stratégie choisie. Naturellement il convient d'ordonner intelligemment les règles, car par exemple $x - x \rightarrow 0$ et $0 \rightarrow x - x$ doivent s'inhiber, quant à $x + y \rightarrow y + x$, elle ne doit être déclenchée que sous certaines conditions.

L'exemple de la dérivation

On souhaite construire un programme calculant les dérivées, en appliquant les théorèmes classiques de dérivation à une fonction qui sera donnée comme une suite de symboles. Considérons les opérateurs suivants + - * / X D SIN COS LOG EXP
La lettre X est tout à fait nécessaire pour distinguer la variable des éventuelles constantes figurant dans la fonction à dériver. L'opérateur D désigne la dérivation. Avant toutes choses, l'expression décrivant la fonction doit être transcrite suivant une notation rationnelle, la notation préfixée. Les règles sont écrites comme des listes dont le "car" est l'hypothèse et le "cdr" la conclusion, U et V pouvant désigner des expressions quelconques.

Soit BD la "base de règles" de dérivation

((D + U V) + D U D V)	dérivée d'une somme en notation préfixée
((D - U V) - D U D V)	différence
((D * U V) + * D U V * U D V)	produit
((D / U V) / - * D U V * U D V * V V)	quotient
((D ^ U N) * * N D U ^ U - N 1)	puissance
((D X) 1)	
((D N) 0)	l'ordre de ces deux dernières règles est impératif.

Soit BS une base de règles de simplification
Celle-ci n'est évidemment pas suffisante, mais on donne là un minimum de règle permettant, une fois la dérivée obtenue, de racourcir singulièrement la longueur de son expression.

((+ N 0) N)		la liste (+ N 0) sera réécrite en N ...			
((+ 0 N) N)	((- N 0) N)	((* N 1) N)	((* N 0) 0)	((* 1 N) N)	((* 0 N) 0)
((/ N 1) N)	((- N N) 0)	((^ N 1) N)	((* / 1 N N) 1)		

L'algorithme d'unification doit, pour une expression E, une hypothèse H, et une conclusion C, retourner l'expression modifiée. Cet algorithme s'écrit très bien récursivement, à l'aide toutefois de fonctions pouvant reconnaître, dans une liste, une suite de symbole pouvant constituer un terme mathématique, de ce qui peut le suivre.
Essayer par exemple pour H = (D * U V) et E = (D * + 5 X * X X)

Les tests suivants sont examinés dans l'ordre tant que l'un d'entre eux n'est pas vérifié, à eux sept ils constituent l'algorithme d'unification d'une expression E avec l'expression formelle H d'une hypothèse.

_ Si H et E sont vides alors on retourne la conclusion C.
_ Si H ou E vides séparément, pas d'unification possible.
_ Si H et E débutent par le même symbole, unification de ce qui suit.
_ Si H et E débutent par 2 opérateurs distincts, échec.
_ Si opérateur dans H seulement, échec.
_ Si opérateur au début de E seulement, alors c'est le cas où il faut substituer une variable de H par un terme de E, cette substitution doit avoir lieu dans la conclusion, mais aussi dans les hypothèses restant à parcourir.
_ Sinon, ce dernier cas est celui de substitution d'une variable par une autre.

Objectif du programme, sur l'exemple de sin (2x), on désire obtenir le détail des étapes, et la dérivée 2*cos (2*x).

donnée initiale	(SIN (2 * X))
préfixation	(SIN * 2 X)
dérivée	(D SIN * 2 X)
action de BD	(* D * 2 X COS * 2 X) avec la règle du sinus
	(* + * D 2 X * 2 D X COS * 2 X) avec le produit
	(* + * D 2 X * 2 1 COS * 2 X) avec (D X) → 1
	(* + * 0 X * 2 1 COS * 2 X) avec (D N) → 0
action de BS	(* + 0 * 2 1 COS * 2 X) avec (* 0 N) → 0
	(* * 2 1 COS * 2 X) avec (+ 0 N) → N
	(* 2 COS * 2 X) avec (* N 1) → N
normalisation	(2 * (COS (2 * X))

12-1° Retrouver les enchaînements des deux exemples, en examinant les règles de reconnaissance des fromages, mais en repartant toujours du début lorsqu'une nouvelle connaissance est acquise.

12-2° Traduire l'algorithme du moteur d'inférence, avec contraposition, sur l'exemple des quadrilatères.

On représente les faits atomiques par des lettres C, R, L ... et leur négation par des listes (C), (L), ... ainsi :

```
(de neg (F) (if (atom F) (list F) (car F)))

(de moteur (BF BR B) (cond
                ((null B) BD) ; c'est fini
                ((inclu (cdar B) BR) ; cas où les prémisses sont vérifiées
                        (moteur (cons (caar B) BF) (ret (car B) BR) (ret (car B) BR)))
                ((and (null (cddar B)) (member (neg (caar B)) BF)) ; cas d'une contraposition
                        (moteur (cons (neg (cadar B)) BF) (ret (car B) BR) (ret (car B) BR)))
                (t (moteur BF BR (cdr B))) )) ; règle non déclenchable, on passe à la suite
```

12-3° Ecrire la négation logique de faits qui, propositionnels sont représentés par des atomes préfixés ou non par -, et valués sont du type (> N 4) (> 7 N) etc...

```
(de non (F) (cond
                ((atom F) (if      (eq '- (car (explode F))) (implode (cdr (explode F)))
                                   (implode (cons '- (explode F))) ))
                ((eq (car F) '=) (cons '<> (cdr F))) ; si F est (= N 7) alors nonF sera (<> N 7)
                ((eq (car F) '<>) (cons '= (cdr F)))
                ((numberp (car (last F))) (list '> (1+ (car (last F))) (cadr F))) ; si F=(> N 4) nonF est (> 5 N)
                (t  (list '> (car (last F)) (1- (cadr F)))) )) ; si F=(> 6 N) alors nonF = (> N 5)
```

12-4° Constituer une fonction (ev F) rendant t par exemple si F est une liste du type (> DM 10) et que DM soit affectée par 12, prévoir tous les cas.

On définit d'abord l'inconnue présente dans un fait par :

```
(de inc (F) (if (numberp (cadr F)) (caddr F) (cadr F) ))
```

En notant un fait (< X 5) ou (= 4 Y) ou (> 7 Z) etc ... Par ailleurs une variable sera représentée par sa valeur si elle en a une, ou bien par une liste du type (< 4) si on sait simplement par exemple qu'elle est inférieure à 4. On ne considère que des nombres entiers.

```
(de ev (F BF) (cond
                ((member F BF) t) ; cas d'un fait déjà dans BF
                ((atom F) nil) ; cas d'un fait atomique inconnu
                ((numberp (eval (inc F))) (eval F)) ; cas où la variable est déjà affectée
                ;exemple F = (> N 12) et N=23 donne t
                ((not (boundp (eval (inc F)))) nil) ; on ne sait rien sur la variable
                ((eq (car F) '=) nil) ; cas de F = (= X 3) par exemple
                ((eq (car F) '<>) (cond
                        ((eq (car (eval (inc F))) '>) ; cas de F = (<> N a) et N = (> b)
                                (> (car (last (eval (inc F)))) (1- (car (last F))))) ; on renvoie (> b a-1)
                        ((eq (car (eval (inc F))) '<) (> (car (last F)) (1- (car (last (eval (inc F)))))))
                        ; si N = (< b) on renvoie (> a b-1)
                        ((eq (car (eval (inc F))) '<>) (= (car (last F)) (car (last (eval (inc F)))))) ))
                        ; si N = (<> b) on renvoie (= a b)
                ((and (numberp (car (last F))) (eq (car (eval (cadr F))) '>)) ;F= (> a N) et N= (> b)
                        (> (car (last (eval (cadr F)))) (1- (car (last F))))) ; on renvoie (≥ b a)
                ((and (numberp (cadr F)) (eq (car (eval (car (last F)))) '<)) ; F=(> a N) et N=(< b)
                        (> (cadr F) (1- (car (last (eval (car (last F))))))) ; on renvoie (≥ a b)
                (t nil) ))
```

12-5° Constituer une fonction (verif L) rendant T si tous les éléments de L sont soit dans BF, soit des faits vérifiés par la fonction "ev" de l'exercice précédent.

```
(de verif (L BF) (cond
                ((null L) t)
                ((null (ev (car L) BF)) nil)
                (t (verif (cdr L))) ))
(de fi (L BF) (cond ; renvoie le premier fait inconnu de la liste L
                ((null L) nil)
                ((null (ev (car L) BF)) (car L))
                (t (fi (cdr L) BF)) ))
```

12-6° Faire de même une fonction (quasi L) vérifiant qu'il ne manque qu'un fait sur trois.

```
(de quasi (L BF) (cond
                ((null L) nil)
                ((null (ev (car L) BF)) (verif (cdr L) BF))
                (t (quasi (cdr L))) ))
```

12-7° Construire les fonctions "moteur" et "dialogue" des fromages, se passant mutuellement la main en utilisant les paramètres BR base de règles, B état courant de la base, BF base de faits, RR règles quasi-déclenchables mises en réserve et FT faits terminaux.
Appliquer le tout aux règles de connaissance sur les fromages représentées par des listes de listes, la conclusion étant placée en premier.

Définissons d'abord un "rajout" de fait dans une base de faits BF :

```
(de raj (F BF) (cond ; renvoie la base complétée
                ((atom F) (if (member F BF) BF (cons F BF)))
                ((eq (car F) '=) (set (cadr F) (caddr F)) BF) ; on affecte réellement la variable
                ((eq (car F) '<>) (set (cadr F) (list '<> (caddr F)) BF)
                ((numberd (caddr F)) (set (cadr F) (list '> (caddr F)) BF)
                (t (set (cadr F) (list '< (cadr F)) BF) ))

(de dem (BF BR B RR FT) (cond
                ((null B) (if (null RR) (sortie BF) ; fonction à écrire délivrant les résultats
                        (dial BF BR RR FT)))
                ((verif (cdar B) BF) ; les prémisses sont satisfaites
                        (dem (raj (caar B) BF) (ret B (car B) BR) (ret (car B) BR) RR FT))
                ((and (null (cddar B)) (verif (non (caar B)) BF)) ; contraposition possible
                        (dem (raj (non (cadar B)) BF) (ret (car B) BR) (ret (car B) BR) RR FT))
                ((and (> (length (car B)) 2) (quasi (cdar B))) ; 1prémisse inconnue sur au moins 2
                        (dem BF BR (cdr B) (if (member (car B) RR) RR (cons (car B) RR)) FT))
                (t (dem BF BR (cdr B) RR FT)) ))

(de dial (BF BR RR R FT) (if ; RR= règles en réserve, R=état courant de RR
                (null R) (dem BF BR BR RR FT)
                (print "'Savez-vous si (oui / non / ?) " (fi (cdar R) BF))
                (let ((rep (read))) (cond
                        ((eq rep 'oui) (dem (raj (fi (cdar R) BF) BF) BR BR (ret (car R) RR) FT))
                        ((eq rep 'non)
                                (dem (raj (non (fi (cdar R) BF)) BF) BR BR (ret (car R) RR) FT))
                        (t (dial BF BR RR (cdr R) FT))) )))
```

Il suffira de placer des visualisations de ce qui se passe à chaque étape.

12-8° **Ecrire un "analyseur syntaxique"** vérifiant qu'un programme Pascal est correctement écrit. On se limitera fortement aux types principaux et instructions essentielles, en écrivant des règles récursives de formation des notions de programme, bloc, instructions etc, en consultant les diagrammes de Conway présents dans tous les manuels de pascal.

12-9° Problème du Professeur Delahaye : supposons les 4 règles suivantes :

Si musicien alors mathématicien — Si grand et brun alors musicien
Si non (lunettes) alors brun — Si lunettes alors musicien

Et l'unique fait : grand, peut-on montrer qu'il s'agit d'un mathématicien ? Que faut-il-faire en général pour compléter le chaînage-avant afin d'avoir ici par exemple la conclusion "mathématicien" ?
Programmer la fonction suivante BR → BR appelée "compilation logique" :
a) Toute règle1 H1 et ... et Hn → C1 ou ... ou Cp, où Hi et Cj sont des littéraux, est réécrite sous la forme disjonctive (¬H1 ou ¬H2 ou ... ou C1 ou C2 ou ... ou Cp).
b) Pou tout couple de règle, si on trouve (X ou L1) et (¬X ou L2), on remplace cette paire par la règle (L1 ou L2).
c) Pour chaque règle (L1 ou ... ou Lk), on construit k règles de la forme (¬L1 et ¬L2 ... ¬Li-1 et ¬ Li+1 ... → Li)
d) On peut utiliser une logique à trois valeurs où les atomes reçoivent une valeur V ou F s'ils sont (ou leur négation) dans BF, sinon ils ont la valeur I (indéterminé) et les connecteurs et, ou sont calculé avec min, max. [P.Matthieu 91]

12-10° Réalisation de la dérivation des fonctions d'une variable.
a) Compléter les règles de dérivation.
b) Développer à la main la dérivation de Log (X) / X puis de $3X^2 + 5$

12-11° Ecrire une fonction "ca" donnant le premier terme mathématique d'une liste
ex. (ca '(+ 4 5 6 - 7)) ⇨ (+ 4 5)
On utilisera pour ce faire une définition du poids d'un opérateur comme 1 moins le nombre de places, exemple p(+) = -1 p(SIN) = 0 p(3) = p(X) = 1, puis le critère de Rosenbloom :
E est une expression bien formée si et seulement si p(F) ≤ 0 pour tout préfixe gauche de f de E, et p(E) = 1
On calcule le poids en avançant de gauche à droite, ce qui donne par exemple pour (* + 2 X SIN / X 3) représentant (2+X)sin(X/3), -1, -2, -1, 0, 0, -1, 0, 1.

```
(de op (X) (cond ; prédicat "être un opérateur"
                 ((numberp X) nil)
                 ((member X '(+ - * / sin cos tan atan exp ln)) t)
                 (t nil) ))
(de ca (L) (cond ; renvoie le premier terme mathématique de L
                 ((null L) nil)
                 ((or (not (op (car L))) (numberp (car L)) (eq (car L) 'X)) (list (car L))
                 ; c'est le cas des constantes, des constantes numériques, et de la lettre X
                 ((member (car L) '(sin cos ln exp tan atan)) (cons (car L) (ca (cdr L))))
                 ;cas des opérateurs à une place
                 (t (append (cons (car L) (ca (cdr L))) (ca (cd (cdr L))))) ))
(de cd (L)  ; renvoie ce qui suit le premier terme de L
                 (nthcdr (length (ca L)) L))
```

12-12° Il faut noter que dans d'autres exemples de dérivation, la base de règle de simplification BS donnée plus haut, ne serait pas suffisante. Ainsi pour $3*x^2 + 5$, le système tel qu'il est décrit ici donnerait (3 * (2 * X)) car l'associativité du produit est ignorée. Ce problème n'est pas facile à résoudre, par contre il est possible de rajouter une étape avant la normalisation : un schéma de règle de simplification bouclant avec l'action de BS. Il s'agit d'une fonction récursive "srs" qui tente d'évaluer ce qui est évaluable, par exemple rempaçant les symboles consécutifs + 3 5 par 8 si ceux-ci sont rencontrés dans une liste. Ecrire cette fonction. On vérifiera que l'on doit obtenir : (srs '(- / + ^ 2 4 * 3 6 2 17)) ⇨ (0)

```
(de srs (L) (cond       ((> 3 (length L)) L)
                        ((and (op (car L)) (numberp (cadr L)) (numberp (caddr L)))
                              (cons (eval (tete L 3)) (srs (cdddr L))) )
                        (t (cons (car L) (srs (cdr L)))) ))
```

12-13° Ecrire la fonction "uf" pour la dérivation, testant l'unification, puis une fonction "unif" réalisant la transformation d'une expression à l'aide d'une règle, en cherchant les unifications avec toutes les sous-expressions.

Supposons (subst Ancien Nouveau liste) renvoie la liste où les subtitutions sont faites, ce qui est à vérifier dans chaque dialecte lisp.

```
(de uf (E H C) (cond ; nil si unification impossible, ou expression E modifiée sinon
                ((and (null H) (null E)) C) ; fin avec succès entre l'expression E et l'hypothèse H
                ((or (null H) (null E)) nil) ; échec
                ((eq (car E) (car H)) (uf (cdr E) (cdr H) C)) ; 2 symboles identiques, on continue
                ((and (op (car H)) (op (car E))) nil) ; 2 opérateurs distincts, donc échec
                ((op (car H)) nil) ; opérateur manquant dans E, échec
                ((op (car E)) (uf (cd E) (subst (car H) (ca E) (cdr H)) (subst (car H) (ca E) C)))
                ; cas de substitution d'une variable de H par un terme de E dans toute la règle
                (t (uf (cdr E) (subst (car H) (car E) (cdr H)) (subst (car H) (car E) C))) ))
                ;cas de la substitution d'une variable par une autre

(de unif (E R) (cond ; donne l'expression E transformée ou non par la règle R
                ((> (length (car R)) (length E)) E)
                ((eq (uf E (car R) (cdr R))) ; pas d'unification globale, on cherche au 1° sous-terme
                        (cons (car E) (append (unif (ca (cdr E)) R) (unif (cd (cdr E)) R))))
                (t (unif (uf E (car R) (cdr R)) R)) )); demanderait à être optimisée
```

12-14° Ecrire la fonction "dem" de paramètres E expression et B base de règles, retournant une expression transformée par toutes les réécritures de B.
Ecrire "der" réalisant éventuellement toutes les étapes pour une expression : préfixation, adjonction du symbole D, action de dem avec BD, action de dem avec BS, évaluation des termes numériques grâce à la fonction "srs" du 12°, transformation en écriture normalisée.

```
(de pref (E) (cond ; donne l'écriture préfixée de E
                ((null E) nil)
                ((atom E) (pref (explode E)))
                ; attention suivant les Lisp utilisés explode joue des tours avec les chiffres en particulier avec 0
                ((member '+ E) (spref '+ E)) ; on regarde les priorités
                ((member '- E) (spref '- E))
                ((member '* E) (spref '* E))
                ((member '/ E) (spref '/ E))
                ((member '^ E) (spref '^ E))
                ((op (car E)) (cons (car E) (pref (cdr E))))
                ((> (prof E) 1) (pref (car E))) ;prof sera le nombre de niveaux de parenthèses
                (t E) ))

(de spref (X E) (append (cons X (pref (tete (1- (rang X E)) E))) (pref (cdr (member X E)))) )

(de par (L) (cond ; retire un niveau de parenthèses seulement dans la liste L
                ((null L) nil)
                ((atom (car L)) (cons (car L) (par (cdr L))))
                (t (concat (car L) (par (cdr L)))) )); sera utile pour écrire3+4*5 plutôt que 3+(4*5)

(de norm (E) (cond ; donne l'écriture normalisée de la liste préfixée E
                ((null (cdr E)) (car E))
                ((atom E) (pref (explode E)))
                ((eq (car E) '+) (par (snorm '+ E))) ; + et - sont moins prioritaires que * et /
                ((eq (car E) '-) (par (snorm '- E)))
                ((eq (car E) '*) (snorm '* E))
                ((eq (car E) '/) (snorm '/ E))
                ((op (car E)) (cons (car E) (list (norm (cdr E)))) ))

(de snorm (X L)  (cons (norm (ca (cdr L) )) (cons X (list (norm (cd (cdr L)))))) )
```

```
(de dem (E BR B) (if ; renvoie E modifiée par la base de règles B de réécritures
                     (null B) E ; on a fini
                     (dembis E (unif E (car B)) BR (cdr B) (car B)) ))

(de dembis (E1 E2 BR B R) (if; R est la règle qui vient de donner E2
                     (equal E1 E2) (dem E1 BR B) ; pas de changement, donc règle suivante
                     (print R "' transforme " E1 "' en " E2) (dem E2 BR BR) ))

(de der (F) ; pour une écriure F d'une fonction de la variable X, doit donner sa dérivée
                     (norm (srs (dem (dem (cons 'D (pref F)) BD) BS)))
```

En fait il faut composer "srs" avec "dem BS" tant que l'expression n'est pas stabilisée.

12-15° Automate fini. Construire une fonction prédicat répondant vrai ou faux suivant qu'une liste de charactères proposée est reconnaissable ou non par un automate fini. Un automate fini est la donnée d'un alphabet A, d'un ensemble Q d'états comprenant un état initial q0 et un sous-ensemble Q' d'états finaux, et d'une fonction de transition f de Q*A vers Q. Par exemple A = {a, b}, Q = {q0, q1, q2} et Q' = {q0} avec f définie par f(q0, a) = q1, f(q1, a) = q1, f(q1, b) = q2, f(q2, b) = q2 et f(nil, q2) = q0.
A partir de q0, un mot sur l'alphabet A correspond donc à un chemin dans l'ensemble des états. Si ce chemin aboutit dans Q', le mot est dit reconnu par l'automate. Dans l'exemple donné (faire un schéma) ce sont les mots du type $a^n b^m$.
Autre test : A = {a, b} et q0 initial et final, avec q0, a → q1, q0, b → q2, q1, b → q0, q2, a → q0. $L = (ab + ba)^n$ est dit langage rationnel.

12-16° Automate à pile. Construire une fonction prédicat répondant vrai ou faux suivant qu'une liste de charactères proposée est reconnaissable ou non par un automate à pile. Un automate à pile est la donnée d'un alphabet A, d'un ensemble Q d'états comprenant un état initial q0, et d'une fonction de transition f de Q*A vers Q*∪{0, 1}. f s'étend également aux mots sur A, grâce à f(u, 0) = 0, f(nil, q) = q. Le langage (dit algébrique) reconnu par l'automate est l'ensemble des mots u tels que f(u, q0) = 1.

12-17° Construire un système expert où chaque fait est associé à un couple (probabilité basse, probabilité haute) (x, y) appelé support de même que chaque règle.

Nous adopterons dans ci-dessous la démarche suivante :
Si une conclusion Q est inférée avec les valeurs (xi, yi) directement ou par l'effet d'une contraposition.
a) Si ni Q ni ¬Q ne sont connus c'est à dire présents dans la base de faits, alors évidemment (Q xi yi) lui est rajouté.
b) Si Q est connu avec les valeurs (xa, ya), alors il est remplacé par (Q, max(xi, ya), max(yi, ya))
c) Si ¬Q est connu avec les valeurs (xn, yn), alors il est naturel de penser que la connaissance de ¬Q présente plus d'intérêt que celle de son contraire, et en ce cas c'est ¬Q qui est remplacé par (¬Q, max(xn, 1-yi) max(yn, 1-xi))
Dans tous les cas un fait déjà présent ne sera donc modifié lors d'une inférence que si sa confiance augmente, ce qui signifie éventuellement que la confiance en son contraire diminue.

```
(de neg (F) (cond       ((atom F) (list F))
                        ((null (cdr F)) (car F)) ((list (neg (car F)) (- 1 (caddr F)) (- 1 (cadr F))))))
; renvoie la négation d'un fait sous la forme ¬ P = (P) et ¬ (P) = P, ou alors ¬ (P n p) = (P 1-p 1-n)

(de contrap (R) (list (- 1 (cadr R)) (- 1 (car R)) (neg (cadddr R)) (neg (caddr R))))
; donne la contraposée (1-p 1-n ¬H ¬C) d'une règle R = (n p C H)
```

```
(de app (F L) (cond         ; établit si le fait F est présent dans la liste L de faits
                            ; et en ce cas renvoie son couple (n p), mais le calcule si ¬F est présent
                  ((null L) nil)
                  ((equal (caar L) F)(cdar L))
                  ((equal (caar L) (neg F)) (list (- 1 (caddar L)) (- 1 (cadar L))))
                  (t (app F (cdr L)))))

(de conf1 (LF LH n p) (cond         ; LH est une liste d'hypothèse, LF une liste de faits
                  ((null LH) (list n p))
                  ((conf2 (app (car LH) LF) LH n p))
                  (t nil)))
(de conf2 (V LH n p) (if (null V) nil (conf1 LF (cdr LH) (min n (car V)) (min p (cadr V)))))
                  ; renvoie nil si non déclenchable, sinon renvoie le couple de vérification des hypothèses
; démarre avec (conf LF (cdddr R) 1 1)

(de norm1 (LF n p C nc pc) ; n p pour la règle, nc pc pour la conclusion C
                  ; renvoie LF modifié avec le fait C acquis et déjà connu ou non
                  (norm2 nil LF C (min n nc) (min p pc) ))
;les systèmes en vigeur prennent ce "et" par un produit
(de norm2 (LV LR C ni pi) (cond ; LV liste vue LR liste restante ni pi valeurs inférées
                  ((null LR) (print "'conclusion inconnue, donc acquise") (cons (list C ni pi) LV))
                  ((equal (caar LR) C) (print "' conclusion modifiée ")
                        (norm3 LV (cdr LR) (caar LR) ni pi (cadar LR) (caddar LR)))
                  ((equal (caar LR) (neg C)) (print "' contraire de la conclusion modifiée ")
                        (norm3 LV (cdr LR)(caar LR)(- 1 pi)(- 1 ni)(cadar LR)(caddar LR)))
                  (t (norm2 (cons (car LR) LV) (cdr LR) C ni pi)) ))
(de norm3 (LV LR F ni pi na pa)
                  (append LV (cons (list F (max ni na) (max pi pa)) LR)))

(de retirer (X L) (cond     ((null L) L) ; renvoie la liste L privée de la 1° occurrence de X
                            ((equal X (car L)) (cdr L))
                            (t (cons (car L)(retirer X(cdr L))))))

(de moteur1 (LF BR B sc) ; sc = nil si contraposée non encore examinée
                  ; B règles restantes, sc signal indiquant que la contraposée a été vue
(if (null B) LF     ; test d'arrêt
                  (moteur2 LF BR B (conf1 LF (cdddar B) 1 1) sc)))        ; on examine la première règle de B
(de moteur2 (LF BR B V sc) (cond
                  ((null V) ;première règle de B non déclenchable
                  (cond    ((and (null sc) (eq (length (car B)) 4))    ; on essaie la contraposée de cette 1° règle
                                          (moteur1 LF BR (cons (contrap (car B)) (cdr B)) t))
                           (t (moteur1 LF BR (cdr B) nil))))
                  (t       (print "'règle : " (car B) "' avec : " V)
                           (moteur1 (norm1 LF (car V) (cadr V) (caddar B) (caar B) (cadar B))
                                    (retirer (car B) BR) (retirer (car B) BR) nil)))))

(de sef (LF BR) (moteur1 LF BR BR nil))          ; fonction principale de démarrage
```

Exemple **:** On considère les propositions P = "Il pleut", M = "Je suis mouillé", S = "Je sort," A = "J'ai un parapluie", R = "Je prends un parapluie".

On donne les cinq règles exprimées avec leur conclusion suivie des prémisses (les négations sont entre parenthèses) :

```
(set 'BR '       ((6 8 (S) P)          ; s'il pleut, je ne sort pas
                  (6 10 R A)           ; si j'ai un parapluie, je le prend
                  (8 10 R S P)         ; si je sort et qu'il pleut, je prend un parapluie
                  (7 10 (M) R)         ; j'ai un parapluie, alors je ne suis pas mouillé
                  (9 10 (S) P M)))     ; il pleut et je suis trempé, je rentre
```

Afin de simplifier la visualisation, les coefficients sont donnés en dixièmes.

(set 'LF '((P 4 8) (S 7 7))) ; sont les deux faits initiaux donnés avec leurs deux valeurs, le déroulement des inférences est le suivant :

(sef LF BR)
règle : (6 8 (s) p) avec la confiance : (4 8) contraire de la conclusion modifiée
règle : (8 10 r s p) avec la confiance : (4 7) conclusion inconnue, donc acquise
règle : (0 4 (a) (r)) avec la confiance : (3 6) conclusion inconnue, donc acquise
règle : (6 10 r a) avec la confiance : (6 10) conclusion modifiée
règle : (7 10 (m) r) avec la confiance : (6 10) conclusion inconnue, donc acquise
règle : (9 10 (s) p m) avec la confiance : (0 4) contraire de la conclusion modifiée
= ((p 4 8) ((a) 0 4) (r 6 10) ((m) 6 10) (s 7 10))

On remarquera que, dans la base de faits finale, le fait S s'est trouvé renforcé par rapport à ses valeurs initiales.

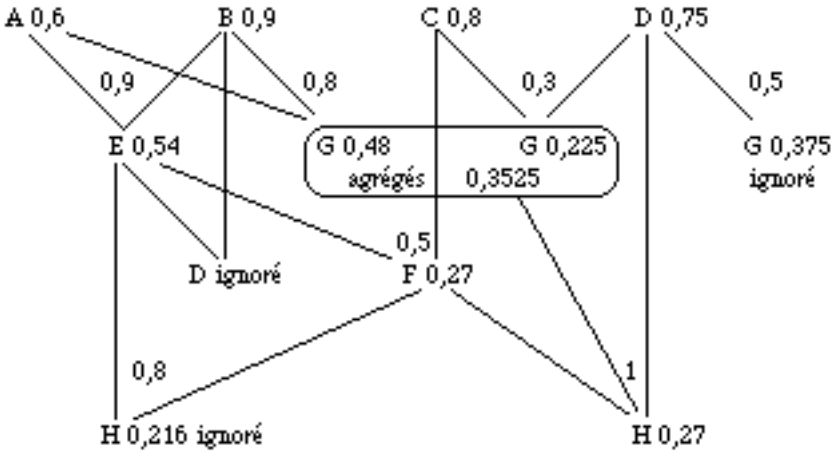
12-18° Système expert fonctionnant avec la logique des défauts
Dans un système déductif où on a des règles du type A → C, et A et B → ¬C, on pourra dire que A entraîne C en général (par défaut d'une information B supplémentaire, auquel cas la première règle ne doit plus se déclencher). Ainsi si T et T' sont des ensembles de règles et d'axiomes avec T inclu dans T', on n'a pas nécessairement la théorie de T incluse dans celle de T' (logique non monotone). On souhaite marier les approches flou et défauts dans un petit système déductif limité au calcul propositionnel. Les propositions y reçoivent un degré de vérité v satisfaisant à v(¬P) = 1 - v(P).
Définitions : Axiome : règle sans prémisse
Preuve de P : arbre de racine P dont les feuilles sont des axiomes et dont chaque noeud est la conclusion d'une règle dont les prémisses sont les fils du noeud.
Longueur d'une preuve : nombre d'arêtes du plus long chemin
Cardinal d'une preuve : nombre de variables propositionnelles ayant une occurrence dans l'arbre.
On considère un système de règles maintenant affectées de coefficients et la théorie engendrée grâce aux métarègles suivantes :
1) Si A_1 et A_2 et A_3 et ... -> C est une règle de coefficient r avec chaque A_i connu avec le coefficient a_i, alors C est admis (provisoirement) avec le coefficient r*min(a_1, a_2, a_3, ...)
2) Si P est obtenu par deux preuves de longueurs distinctes, P aura le coefficient correspondant à la preuve de plus courte longueur. (Ainsi on donne la priorité à des déductions plus immédiates et on évite le fait d'avoir à agréger par exemple un axiome avec son éventuelle démonstration, cas fâcheux où une connaissance initiale pourrait se trouver affaiblie.)
3) Si P est obtenu par deux preuves de mêmes longueur mais de cardinaux distincts, c'est la preuve de plus grand cardinal qui l'emporte (Ainsi des précisions supplémentaires entraînent une priorité sur des règles générales.)
4) Si P est obtenu par deux preuves de mêmes longueur et cardinal, alors on prend simplement la moyenne, mais comme cette opération n'est pas associative, il faudra tenir compte du nombre d'occurrences.
5) On peut également considérer la contraposition

a) Dérouler à la main l'enchaînement des déductions suivant la stratégie ci-dessus.
b) Dessiner et expliquer le graphe des déductions pour la théorie de T1 = {A, E, A → B, B → C, E → B} ainsi que pour T2 = {A, B, D, E, A&B → C, B&C → D, C&E → F, D&F → C}. L'ensemble de toutes les conclusions est-il toujours parfaitement défini?
Donner un algorithme pour le construire.
c) Trouver une représentation qui va permettre de mettre en oeuvre un tel moteur d'inférence et le programmer.
d) Trouver un exemple pour le tester et commenter les résultats.

Solution :
Les faits sont représentés par (F coef long card nb) qui sont les informations relatives à la meilleure preuve de F à chaque instant.
BF est une base de tels faits
BR une liste de règles représentées par (r Q P_1 P_2 P_3 ... P_n)
On explore l'arbre en largeur, lg étant la profondeur à chaque instant

```
(de moteur (BF BR) (moteur1 (init BF) BR BR 1 nil))
(de moteur1 (BF B BR lg s)     ; s est un booléen indiquant la fin
                (if (null B) (if s (moteur1 BF BR BR (+ 1 lg) nil) (presentation (reverse BF )))
                                (moteur2 BF (caar B) (cadar B) (cddar B) 1 lg 0
                                                (assoc (if (atom (cadar B)) (cadar B) (caadar B)) BF) B BR s)))
(de moteur2 (BF r Q2 LP cfp lg cdp Q1 B BR s)
; confrontation d'une conclusion Q2 avec ce qu'on en sait déjà Q1
      (if (null LP) (cond          ;les prémisses ont été vues
                ((null Q1) (moteur1 (cons (if (atom Q2)(list Q2 (* r cfp) lg (+ 1 cdp) 1)
                                (list (car Q2) (- 1 (* r cfp)) lg (+ 1 cdp) 1)) (ret Q1 BF)) (cdr B)
                                (ret (car B) BR) lg t)) ; Q vient d'être acquis pour la première fois
                ((< (caddr Q1) lg) (moteur1 BF (cdr B) (ret (car B) BR) lg s));
                                ; cas où Q a déjà été acquis avec une profondeur inférieure
                ((< (+ 1 cdp)(cadddr Q1)) (moteur1 BF (cdr B) (ret (car B) BR) lg t))
                                ; cas où Q a déjà été prouvé au même passage, mais avec plus de précision
                ((< (cadddr Q1)(+ 1 cdp)) (moteur1
                                (cons (if (atom Q2) (list Q2 (* r cfp) lg (+ 1 cdp) 1)
                                                (list (car Q2) (1-(* r cfp)) lg (+ 1 cdp) 1))
                                                (ret Q1 BF))(cdr B) (ret (car B) BR) lg t))
                                ; Q vient d'être acquis avec une meilleure précision pour la même longueur
                (t (moteur1 (cons (list (car Q1) (/ (+ (if (atom Q2) (* r cfp)(- 1 (* r cfp)))(*(cadr Q1)(occ Q1)))
                        (1+ (occ Q1))) lg (1+ cdp) (1+ (occ Q1))) (ret Q1 BF)) (cdr B) (ret (car B)BR) lg t)) )
                (moteur3 BF r Q2 (assoc (if (atom (car LP)) (car LP) (caar LP)) BF) LP cfp lg cdp Q1 B BR s)))

(de moteur3 (BF r Q2 P LP cfp lg cdp Q1 B BR s) (cond    ; examine la prémisse P
                ((null P) (moteur1 BF (cdr B) BR lg s));P n'est pas connue, on passe à la règle suivante
                ((eq lg (caddr P)) (moteur1 BF (cdr B) BR lg s))
                ; P vient d'être prouvé à la même passe, on réserve l'examen de cette règle pour la passe suivante
                ((atom (car LP)); on passe à la prémisse suivante qui est positive
                                (moteur2 BF r Q2 (cdr LP) (inf cfp (cadr P)) lg (+ cdp (cadddr P)) Q1 B BR s))
                (t (moteur2 BF r Q2 (cdr LP) (inf cfp (- 1 (cadr P))) lg (+ cdp (cadddr P)) Q1 B BR s)) ))
                                ; cas d'une prémisse ¬P dont le coefficient est 1- celui de P connu.

(de occ (L) (caddr (cddr L))); donne simplement le cinquième élément = nb d'occurrence d'un fait
(de inf (X Y) (if (< X Y) X Y))
(de ret (X L) (cond ;retire la première occurrence de X dans la liste L
                ((null L) nil)
                ((equal X (car L)) (cdr L))
                (t (cons (car L) (ret X (cdr L))))))

(de init (BF) (cond ; met les axiomes A sous la forme (A 0 1 1)
                ((null BF) nil)
                ((null (cddar BF)) (cons (append (car BF) '(0 1 1)) (init (cdr BF)))) ))
(de presentation (L) (if (null L) () ; fonction de visualisation d'une liste de faits de la forme (F coef long card nb)
                (prin (caar L) "' coef= "(cadar L));0," (truncate (* 1000 (cadar L))))
                (if (eq (caddar L)0) (print "'  axiome ") (print  "'  preuve de profondeur = "
                                (caddar L) "'  et d'importance = " (cadddr (car L))))
                (presentation (cdr L))))

(de neg (P) (if (atom P) (list P) (car P)))

(de contrap (B) (cond ; renvoie la liste des règles de B augmentée de ses contraposées
                ((null B) nil)
                ((null (cdddar B))
                                (mcons (car B) (list (caar B) (neg (caddar B)) (neg (cadar B))) (contrap (cdr B))))
                (t (cons (car B) (contrap (cdr B))))))
```

Exemple, considérons l'ensemble T = {(A, 0.6), (B, 0.9), (C, 0.8), (D, 0.75), (A&B → E, 0.9), (B&E → D, 0.8), (C&E → F, 0.5), (D&F → E, 1), (D → G, 0.5), (C&D → G, 0.3), (A&B → G, 0.8), (E&F → H, 0.8), (D&F&G → H, 1)}.




```
(set 'BF0 '((A 0.6 ) (B  0.9) (C 0.8) (D 0.75) ))
(set 'BR0 '((0.9 E A B) (0.8 D B E) (0.5 F C E) (0.5 G D) (0.3 G C D) (0.8 G A B) (0.8 H E F) (1 H D F G)))

(moteur  BF0 BR0)
d coef = 0,750  axiome
c coef = 0,800  axiome
b coef = 0,900  axiome
a coef = 0,600  axiome
e coef = 0,540  preuve de profondeur = 1  et d'importance = 3
g coef = 0,352  preuve de profondeur = 1  et d'importance = 3
f coef = 0,270  preuve de profondeur = 2  et d'importance = 5
h coef = 0,270  preuve de profondeur = 3  et d'importance = 10
=( )
```

Soit maintenant la base de connaissances suivante :
En général les oiseaux volent. Les pingouins sont des oiseaux qui ne volent pas, sauf ceux des Kerguelen qui volent légèrement. Hector et Victor sont des pingouins des Kerguelen, mais Victor ne vole pas vraiment. Nestor est un autre pingouin.

```
(set 'BRP '((0.9 vole oiseau) (1 oiseau pingouin) (0.8 (vole) pingouin) (1 pingouin pinkerguelen)
      (0.3 vole pinkerguelen) (1 pinkerguelen victor) (1 pinkerguelen hector) (0.9 (vole) victor)
      (1 pingouin nestor)))
(setq bf1 '((hector 1)) bf2 '((victor 1)) bf3 '((nestor 1)) bf4 '((vole 0.7)))
```

Avec un seul axiome, on a successivement :

```
hector coef = 1  axiome
pinkerguelen coef = 1  preuve de profondeur = 1  et d'importance = 2
pingouin coef = 1  preuve de profondeur= 2  et d'importance = 3
vole coef =  0,3  preuve de profondeur = 2  et d'importance = 3
oiseau coef = 1  preuve de profondeur = 3  et d'importance = 4
```

Puis pour l'unique axiome Victor :

```
victor coef = 1  axiome
pinkerguelen coef = 1  preuve de profondeur = 1  et d'importance = 2
vole coef =  0,1  preuve de profondeur = 1  et d'importance = 2
pingouin coef = 1  preuve de profondeur = 2  et d'importance = 3
oiseau coef = 1  preuve de profondeur = 3  et d'importance = 4
```

Avec Nestor :

```
nestor coef = 1  axiome
pingouin coef = 1  preuve de profondeur = 1  et d'importance = 2
oiseau coef = 1  preuve de profondeur = 2  et d'importance = 3
vole coef =  0,2  preuve de profondeur = 2  et d'importance = 3
```

12-19° Mécanisme d'inférence de Jeffrey, reprendre l'idée du problème 17 où chaque fait est associé à un couple (x, y) vérifiant $0 \leq x \leq y \leq 1$ appelé support et interprété comme un intervalle possible pour la probabilité de P. La règle d'inférence est alors inspirée, pour $P \Rightarrow Q$, de la formule de Bayes $p(Q) = p(Q/P)p(P) + p(Q/\neg P)p(\neg P)$

$P \Rightarrow Q$ connu avec les supports (a, b) et (c, d) pour celui de $\neg P \Rightarrow Q$
Lorsque P est connu avec le support (x, y), le calcul de (X, Y) pour Q se fait donc par $X = \min\{p(Q/P)p(P) + p(Q/\neg P)p(\neg P)\} = \min\{az + c(1-z)\}$ en notant $z = p(P)$.
On obtient facilement :
X = si $a \leq c$ alors $ay + c(1-y)$ sinon $ax + c(1-x)$ et,
Y = si $b \leq d$ alors $bx + d(1-x)$ sinon $by + d(1-y)$
Au cas où seul le premier support (a, b) est donné on a $(X, Y) = (ax, bx + 1 - x)$

En ce cas, 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). On vérifiera qu'il n'y a pas de compatibilité avec la contraposition, c'est à dire que si de P(x, y) on déduit Q(X, Y) alors $\neg Q$ admet le support (1-Y, 1-X) et l'implication $\neg Q \Rightarrow \neg P$ entraine des supports distincts de (1-y, 1-x) sauf si $a = b = 1$ et $c = d = 1$.

12-20° Le dilemme itéré des prisonniers [J.P.Delahaye 92, Axelrod 92].
Deux suspects en détention provisoire sont placés sans pouvoir communiquer dans le choix suivant : si l'un avoue et pas l'autre, il sera libéré (sa charge sera T = 0, T comme trahir) et l'autre aura un coût D = 5 ans de prison (D = duper), si les deux avouent, ils auront chacun C = 2 (C = coopérer) ans de prison, et si aucun n'avoue, ils auront P = 4 ans de prison (P = punir). (Réfléchir aux cas éventuels pour voir où est le dilemme.)
Plus généralement deux partenaires réitèrent un "marché" suivant ces règles et on aimerait trouver une stratégie minimisant en moyenne le coût au cours des itérations.
Le "gain" moyen (en années de prisons pour les prisonniers) est la fonction que l'on va chercher à minimiser. Le fait que $(T+D) / 2 > C$ indique que les deux entités n'aient pas intérêt à s'entendre à tour de rôle pour une série de trahir-duper, la coopération leur étant plus avantageuse.
On gardera toujours $T = 0 < C = 2 < R = 3 < P = 4 < D = 5$.
Lorsque ce dilemme est répété plusieurs fois avec les mêmes adversaires, on dira qu'on a une "partie" résultant de (par exemple 100) "coups". Plusieurs stratégies peuvent être alors imaginées :
1) Gentille : on coopère toujours.
2) Méchante : on trahit toujours.
3) Lunatique : c ou t est tiré au hasard.
4) Donnant-donnant : on coopère la première fois, puis on joue ce que l'adversaire a joué au coup précédent.
5) Rancunier : si l'adversaire a trahit au moins une fois dans les coups précédent, alors on trahira toujours, sinon on coopère.
6) Périodique méchant : quelquesoit le jeu de l'adversaire, on coopère une fois puis on trahit deux fois.
7) Périodique gentil : idem mais une trahison suivie de deux coopérations.
8) Majorité-mou : on joue le jeu de l'adversaire qui a été le plus présent au cours des jeux précédents, et la coopération en cas d'égalité.
9) Méfiant : trahison au début, puis donnant-donnant.
10) Majorité-dur : idem majorité-mou sauf trahison en cas d'égalité.
11) Sondeur : trahison suivie de deux coopérations pour les trois premiers coups, puis si l'adversaire a coopéré aux coups 2 et 3 alors on trahit toujours, sinon donnant)donnant.
12) Donnant-donnant dur : coopération sauf si l'adversaire a trahit lors de l'un des deux coups précédent.
Trouver une repésentation cohérente des stratégies, puis programmer ce qu'est un coup (le coût d'une stratégie face à une autre), puis ce qu'est une partie, puis ce que serait l'évaluation d'une stratégie, c'est à dire la somme de ses coûts face aux 12 statégies ci-dessus.
Que se passe-t-il si on introduit un renoncement R = 3 ce qui signifie qu'une partie n'a plus nécessairement 100 jeux ?