Groupe de recherche en

Intelligence Artificielle

U.E.R. de Luminy

Université d'Aix-Marseille

Rapport de recherche
sur le contrat
CRI n° 72-18 de
février 72 à juin 73

UN SYSTEME DE COMMUNICATION HOMME-MACHINE EN FRANCAIS

A. COLMERAUER

H. KANOUI

P. ROUSSEL

R. PASERO

AVANT - PROPOS

En février 1972, le Groupe d'Intelligence Artificielle de Luminy recevait une subvention de 180 000,00 francs dans le cadre du contrat CRI 72-18 intitulé "Communication homme-machine en langue naturelle avec déduction automatique". Ce contrat se termina en juin 1973.

L'objet du contrat était de mettre au point un système expérimental, mais très général de traitement de phrases en français, entrées à partir d'une console d'ordinateur, en vue de dialoguer avec un utilisateur.

L'approche de ce problème difficile fut menée sur trois fronts:

1) Du point de vue linguistique par une étude sous un angle particulier de la syntaxe et la sémantique du français. Signalons à ce sujet la thèse de 3è cycle de R. PASERO: "Représentation du français en logique du 1er ordre en vue de dialoguer avec l'ordinateur".

2) Du point de vue démonstration automatique, plusieurs méthodes ont été étudiées et essayées sur ordinateur. A ce sujet voir la thèse de 3è cycle de P. ROUSSEL: "Définition et traitement de l'égalité formelle en démonstration automatique".

3) Du point de vue purement informatique un langage de programmation très particulier, PROLOG, a été développé. Ce langage à base de démonstration automatique a servi à programmer tout le système décrit dans cette brochure.

Les retombées de ce travail, autres que le système décrit ici, sont multiples. Signalons entre autre que PROLOG a permis de démarer beaucoup d'autres recherches.

Il peut être intéressant de savoir comment l'argent du contrat a été dépensé:

- à payer beaucoup d'heures machine

- à inviter R. KOWALSKI d'Edimbourg et J. TRUDEL de Montréal.

- à prendre contact avec les principaux laboratoires d'intelligence artificielle des Etats-Unis et d'Angleterre.

- à payer une secrétaire et des frais de secrétariat.

Enfin indiquons la part que chacun a prise dans ce travail d'équipe:

P. ROUSSEL s'est occupé de l'écriture de l'interpréteur du langage de programmation PROLOG.

A. COLMERAUER a travaillé sur l'analyseur et le sémantiseur du Français.

R. PASERO a travaillé sur la partie sémantique, c'est à dire, le synthétiseur d'énoncés logiques et le "déducteur".

H. KANOUI a participé à l'écriture de PROLOG et a écrit les règles de morphologie utilisées dans l'analyseur du français. L'ensemble de son travail est décrit dans un rapport de DEA présenté en octobre 1972 au Département de Mathématique-Informatique de Luminy.

TABLE DES MATIERES

PRESENTATION GENERALE

Texte soumis

TOUT PSYCHIATRE EST UNE PERSONNE.
CHAQUE PERSONNE QU'IL ANALYSE, EST MALADE.
*JACQUES EST UN PSYCHIATRE A *MARSEILLE.
EST-CE QUE *JACQUES EST UNE PERSONNE?
OU EST *JACQUES?
EST-CE QUE *JACQUES EST MALADE?

Réponse

OUI

A MARSEILLE

JE NE SAIS PAS

Ceci est un début de conversation que nous avons eue récemment avec l'ordinateur. Nous aurions pu en avoir beaucoup d'autres puisque le système, dont cette brochure est l'objet, permet d'une façon générale:

-de décrire un certain "monde" à l'ordinateur, en français, ou si l'on veut, de lui raconter une histoire;

-de poser une suite de questions, toujours en français concernant ce "monde" ou cette histoire. L'ordinateur est censé y répondre.

Bien entendu l'ordinateur n'accepte qu'un sous-ensemble du français, mais ce sous-ensemble est cependant très vaste et du fait qu'il comprend notamment tous les noms et tous les verbes il permet de décrire n'importe quoi .

La base de notre système est la démonstration automatique. Nous avons développé un langage de programmation appelé PROLOG où chaque instruction n'est rien d'autre qu'un énoncé de logique de 1er ordre et où exécuter un programme revient à démontrer un théorème. Ce langage nous a permis de programmer ou plus exactement d'axiomatiser:

-un analyseur qui convertit un texte, écrit en français, et formé de phrases affirmatives et interrogatives en une arborescence faisant apparaître les relations syntaxiques entre les éléments des phrases. Cette arborescence est appelée structure syntaxique.

-un "sémantiseur" qui, à partir de la structure syntaxique, produit une structure sémantique où les liens sémantiques entre les éléments du texte sont davantage mis en évidence.

-un synthétiseur d'énoncés logiques qui traduit la structure sémantique en un ensemble de formules logiques du 1er ordre, facilitant l'accès aux informations contenues dans le texte.

-un "déducteur" qui "raisonne" sur les énoncés précédemment synthétisés et produit les réponses aux différentes questions.

Dans cette brochure on trouvera successivement les descriptions, pas toujours complètes, de PROLOG, de l'analyseur, du "sémantiseur" du synthétiseur d'énoncés logiques et du "déducteur".

LE LANGAGE DE PROGRAMMATION PROLOG

INTRODUCTION

Prolog a été conçu comme un langage de programmation se situant entre la logique du premier ordre et des langages comme LISP ou PLANNER par exemple.C'est,à la base, un démonstrateur automatique qui permet de traiter aisément les données formelles telles que formules algébriques, arbres, listes, langages naturels, etc...

La plupart des démonstrateurs automatiques existant sont souvent, parce que trop éloignés d'un langage de programmation, peu efficaces en dehors des problèmes mathématiques. L'utilisateur n'a, en effet, que peu de moyens pour contrôler l'éxécution de son "programme" et de plus la syntaxe des langages acceptés par ces démonstrateurs (en général des formules écrites sous forme fonctionnelle) est trop rudimentaire pour permettre une vision synthétique des objets que l'on traite.

Ce sont les écueils que nous avons voulu éviter en concevant PROLOG.

PROLOG fondamentalement travaille sur des fichiers qui sont des ensembles de "clauses". Programmes, données et résultats ont tous cette structure. Un ensemble de commandes simples permet de lire, écrire et créer ces fichiers, la "sortie" résultant de l'éxécution d'un premier fichier pouvant être éxécutée à son tour ou servir de donnée à un autre fichier. Les "programmes" écrits peuvent être non-déterministes et récursifs. Cette récursivité s'apparente d'ailleurs à la récursivité utilisée en logique plus que la notion de récursivité employée dans les langages comme Algol (bien qu'en définitive on puisse prouver que toutes ces notions sont équivalentes). Outre ces quelques points PROLOG permet la génération dynamiques d'informations structurées de type arbre.

Afin de rendre la lecture et l'écriture des clauses plus claires, l'utilisateur peut créer ses propres conventions de langage en redéfinissant par exemple les symboles de bases ou en fixant lui-même les opérateurs qu'il utilisera.Les quelques principes et définitions de bases nécessaires à une bonne compréhention de PROLOG seront d'abord explicités. Suivra ensuite l'étude des commandes constituant un programme PROLOG.

Nous avons toujours essayé de montrer dans ces études l'analogie entre l'éxécution d'un fichier de clauses et l'éxécution de procédures des langages habituels, en espérant montrer comment la logique peut, parfois, s'apparenter à un langage de programmation.

TERMINOLOGIE ET PRINCIPES DE BASE

Les concepts de base propres à PROLOG sont les suivants: clause, unification et résolution. Nous donnerons ici des définitions et des détails volontairement sommaires, les lecteurs désireux de trouver un support théorique plus consistant pouvant se référer à la bibliographie finale portant sur la démonstration automatique.

Clauses

Une clause représente en fait une formule de logique du premier ordre d'un type un peu particulier. Elle consiste en une disjonction de formules atomiques, disjonction quantifiée universellement. Une formule atomique est simplement une fonction logique précédée ou non d'une négation. Chaque argument de la fonction logique (ou prédicat) est un terme. Ces termes sont soient des variables, soient des arbres construits à partir de symboles fonctionnels et d'autres termes.

Au départ les formules atomiques sont écrites sous forme fonctionnelle mais nous verrons que l'utilisateur peut aisément les écrire autrement. Donnons un exemple pour préciser la terminologie employée.
La formule suivante:

$\forall x \ \forall y \ \forall z$ Frère(y,z)$\vee\neg$ Père(x,y)$\vee\neg$ Père(x,z) sera représentée

par la clause suivante:

+Frère(*x,*z)-Père(*x,*y)-Père(*x,*z)

Les variables d'une clause sont précédées de * pour les distinguer des autres symboles. On peut remarquer que le signe de disjonction (ou) est omis, que la négation est exprimée par — et l'affirmation par +.
Une formule atomique précédée d'un signe s'appelle un littéral.
Dans l'exemple précédent

-Père(*x,*y) est un littéral

Père(*x,*y) est une formule atomique

*x est un terme (variable)

*y est un terme (variable)

Les littéraux d'une clause PROLOG sont ordonnées (à l'inverse d'une formule logique où l'ordre importe peu la loi ou étant commutative) de telle sorte qu'une clause puisse être interprétée comme une implication logique ne pouvant s'utiliser que d'une certaine façon.

Rappelons qu'en logique la formule

(A et B et C....et D) → E est équivalente à

E ∨¬A ∨¬B∨ ¬C∨ .....∨ ¬D

Ainsi la clause précédente peut s'interpréter par:

quelque soient x,y,et z si y et z ont pour père x alors y est frère de z.
De manière générale toute clause de la forme $L_0\overline{L}_1\overline{L}_2\ldots\overline{L}_n$, où $L_0,L_1,\ldots,L_n$ sont des littéraux ($\overline{L}_i$ étant le littéral L_i changé de signe), pourra s'interpréter par la règle suivante:

-si l'on peut démontrer L_1, puis L_2,..., puis L_n alors L_0 est démontré.

Par contre on ne pourra pas utiliser cette clause pour démontrer L_1 à partir de $L_0,L_2,\ldots$ et L_n.
Une clause réduite à un seul littéral pourra être interprétée comme l'affirmation ou la négation d'un fait.

Ensembles de clauses

Alors qu'une clause est une disjonction de littéraux, un ensemble de clauses est la conjonction de ces clauses.
Chaque clause ayant toutes ses variables quantifiées universellement, la "portée" d'une variable est donc limitée à la clause où elle figure. On peut donc considérer que chaque clause comporte en quelque sorte les déclarations implicites de ses variables.
Deux variables (même si elles ont le même nom) figurant dans deux clauses différentes n'ont aucun lien entre elles.
De même que pour les littéraux d'une clause, les clauses d'un fichier sont ordonnées et nous verrons que cet ordre est quelquefois prépondérant dans l'éxécution d'un fichier, chaque clause pouvant être assimilée en quelque sorte à une suite d'appels à des procédures. En ordonnant ses clauses l'utilisateur pourra donner la priorité à telle clause plutôt qu'à telle autre lorsqu'il voudra démontrer un fait.

Voici un exemple de fichier de clause où les termes utilisés sont des variables ou des fonctions sans arguments (constantes).

+Frere(*y,*z)-Pere(*x,*y)-Pere(*x,*z).

+Pere(Paul,Pierre)

+Mere(Marie,Jacques)

+Mari(Paul,Marie)

+Pere(*x,*y)-Mari(*x,*z)-Mere(*z,*y)

Substitutions - Instances

Nous avons dit que les arguments d'un prédicat (ou formule atomique) étaient des termes. Ces termes peuvent être des variables ou des arbres formés à partir de symboles fonctionnels et d'autres termes.
Ainsi:

*x,F(*x,*y),+(A,*x),...F(+(A,B),*y) etc.... sont des termes

ici F et + sont des symboles fonctionnels à deux arguments
A et B des symboles fonctionnels sans arguments (constantes).

Une substitution est une suite de couples

$(*x_1 \rightarrow t_1, *x_2 \rightarrow t_2, \ldots, *x_n \rightarrow t_n)$ où $t_1, \ldots$ et t_n sont des termes, $*x_1, \ldots$ et $*x_n$ des variables dinstinctes.

Appliquer une substitution s sur une expression E (terme, littéral, formule atomique ou clause) consiste à remplacer dans E les variables figurant dans s par le terme associé.

L'expression alors obtenue sera notée par E.s et on dira que c'est une instance de E.

Ainsi:

la substitution s={*x → A} appliquée sur *X donne A

s={*x → A,*y → B}.....F(*x,*y)..F(A,B)

s={*x → G(*y)} ..... F(*x,*x)..F(G(*y),G(*y))

s={*x → A}.....H(A,B,*y)..H(A,B,*y)

Par contre F(A,B) ne peut être une instance de F(*x,*x).

Etant quantifiée universellement on peut remarquer qu'une clause représente l'ensemble infini de toutes ses instances.

Unification

L'unification est l'algorithme de base de toute démonstration automatique. Nous verrons que dans certains cas il s'apparente à un appel de procédure par nom, mais il est en fait beaucoup plus puissant puisque les données que nous manions ne sont pas toujours constantes, mais peuvent comporter des variables.

Soient E_1 et E_2 deux expressions (termes ou formules atomiques), ayant éventuellement des variables communes. On dit que E_1 et E_2 sont unifiables lorsqu'il existe une substitution s qui les rends égales, c'est à dire telle que $E_1.s=E_2.s$
Ainsi:

F(*x,+(A,B)) et F(C,*y) sont unifiables par s={*x→ C,*y→ +(A,B)}

par contre:

F(*x) et F(G(*x)) ne sont pas unifiables puisqu'aucune substitution ne peut rendre *x égal à G(*x).

De même pour F(A) et F(H(*x,*y)).

On montre en fait que si E_1 et E_2 sont unifiables, il existe une substitution (appelée plus grand unifieur ou p.g.u) telle que $E_1.s=E_2.s$ et telle que pour toute autre substitution s' qui unifie E_1 et E_2 (i-e que $E_1.s'=E_2.s'$), l'expression $E_1.s'$ est une instance de $E_1.s$ (i-e que il existe s" telle que $E_1.s'=E_2.s'=E_1.s.s"=E_2.s.s"$).

Autrement dit lorsqu'on applique le p.g.u de deux expressions unifiables on obtient la "plus générale" des unifications, toute autre unification en étant un cas particulier.

Lorsque nous parlerons de l'unification de deux expressions c'est en général l'expression commune obtenue en applicant le p.g.u que nous considérerons.

Résolution

La résolution est une règle qui permet de générer une clause à partir de deux autres. Cette règle d'inférence est celle qui permet "d'éxécuter" un fichier de clauses.

Considérons un fichier axiomes et une clause C d'un fichier données. Chaque clause A de axiomes peut, sous certaines conditions, se résoudre sur C et générer ainsi une résolvante R.

Résolution dans le cas de clauses sans variables:

Considérons tout d'abord le cas où les clauses en question sont sans variables.
Soit par exemple axiomes et données les deux fichiers suivants:

Axiomes:

1: +Frere(Paul,Pierre)-Pere(Jacques,Paul)-Pere(Jacques,Pierre);;

2: +Pere(Jacques,Paul);;

3: +Pere(Jacques,Pierre);;

Données:

4: -Frere(Paul,Pierre)+Réponse(oui)..

(La ponctuation terminant les clauses sera étudiée dans l'étude de la commande démontrer).

La transitivité de l'implication logique permet alors de déduire les clauses suivantes:

1 se résoud sur 4:

5: -Père(Jacques,Paul)-Père(Jacques,Pierre)+Réponse(oui);.

2 se résoud sur 5:

6: -Père(Jacques,Pierre)+Réponse(oui);.

3 se résoud sur 6:

7: +Réponse(oui);.

La règle est donc simple: pour pouvoir déduire une résolvante de deux clauses sans variables, il suffit que leurs premiers littéraux soient opposés (c'est à dire constitués de la même formule atomique précédée de signes opposés).

En fait une clause PROLOG est constituée d'une partie principale (celle dont les littéraux vont être unifiés) et d'une partie réponse qui sert à stocker certaines informations et à constituer les clauses de sortie résultant de l'éxécution d'un fichier.
La partie réponse d'une clause si elle existe est séparée de la partie principale par /. En fin une ponctuation termine la clause pour permettre au programmeur de contrôler l'éxécution du fichier. Nous étudirons cette ponctuation dans le chapitre DEMONSTRATION.
Donc dans le cas de clauses sans variables, on dit qu'une clause

$D=L_1L_2.....L_n/R_1....R_n$ peut se résoudre sur la clause

$C=M_1....M_p/S_1....S_q$ si $L_1=\overline{M}_1$. La résolvante générée alors est

$R=L_2....L_nM_2....M_p/R_1.....R_nS_1....S_q$ c'est à dire que les

deux parties principales (sauf L_1 et M_1) sont concaténées, de même que les parties réponses.

Dès qu'une clause ayant sa partie principale vide, est générée elle est ajoutée au fichier de sortie.

Résolution dans le cas général

Le principe reste le même que pour les clauses sans variables, la seule différence résidant dans le fait que L_1 et $\overline{M}_1$ sont unifiables (au lieu d'être égaux), la clause C étant tout d'abord recopiée.

Donnons un exemple de résolution sur des clauses avec variables.

Axiomes:

1 +Frere(*y,*z)-Pere(*x,*y)-Pere(*x,*z)

2 +Pere(Paul, Jacques)

3 +Pere(Paul,Pierre)

Données:

4 -Frere(Jacques,*x)/+Reponse(*x)

1 se résoud sur 4:

5 -Pere(*x,Jacques)-Pere(*x,*z)/+Reponse(*z)

2 se résoud sur 5:

6 -Pere(Paul,*z)/+Reponse(*z)

3 se résoud sur 6:

7 /+Reponse(Pierre)

Clause qui constitue une sortie.

Soient deux clauses C et D sans variables communes (PROLOG effectue le changement de nom des variables si cela est nécessaire), telles que $C=L_1 \ldots L_n/R_1 \ldots R_m$ et

$D=M_1 \ldots M_p/S_1 \ldots S_q$. On dit que C peut se résoudre sur D si

les littéraux L_1 et $\overline{M}_1$ sont unifiables (c'est à dire que L_1 et M_1 sont des signes contraires et que leurs formules atomiques peuvent s'unifier). Si s est alors le p.g.u de ces deux littéraux, la résolvante de C sur D est

$R=L_2.s\ L_3.s \ldots L_n.s\ M_1.s/R_1.s \ldots R_m.s S_1.s \ldots S_q.s$

La résolution est donc analogue au cas de clauses sans variables sauf que l'on considère deux instances C.s et D.s de C et D qui sont telles que $L_1.s=\overline{M}_1.s$.
Ainsi la résolvante de

+P(.(*x,+(*y,*z)),*u)-P(+(.(*x,*y),.(*x,*z)),*u)

sur

-P(.(+(A,B),+(C,D)),*x)/+Réponse(*x)

sera

-P(+(.(+(A,B),C),.(+(A,B),D)),*x)/+Réponse(*x).
(remarquons le changement des variables dans la première clause).
Par contre sur

-P(.(A,.(A,B)),*u)/+Réponse(*x) nous n'aurions pas obtenu de résolvante puisque les deux formules atomiques ne sont pas unifiables.
Nous verrons avec plus de précisions dans le chapitre consacré à la commande "démontrer" comment faire agir un fichier de clauses sur un autre par la résolution.

Les quelques principes et définitions que nous venons de donner permettent de comprendre les mécanismes de base de PROLOG.
Nous pouvons maintenant étudier les commandes utilisables dans ce langage.

COMMANDES ET ORGANISATION DES FICHIERS

Un programme écrit en PROLOG consiste en un ensemble de commandes qui gérent des fichiers de clauses. Chaque fichier a nécessairement un nom, et une bibliothèque des fichiers activés en mémoire tient compte de leurs emplacements respectifs.

Tout programme PROLOG est constitué d'une suite d'unités composées à partir des caractères autres que le blanc.

Dans la suite de cet exposé nous indiquerons la syntaxe d'un nouveau concept PROLOG par une grammaire context-free écrite sous forme de Backhus.

Unités

<unité>::=<unité A-N>|<unité non A-N>
<unité A-N>::=<suite de car A-N>
<suite de car A-N>::=<car A-N><suite de car A-N>|<car A-N>
<car A-N>::=<A|...|Z|0|....|9
<unité non A-N>::=+|-|/|.|,|.....etc

les blancs ne sont jamais pris en compte sauf pour séparer deux unités alpha-numériques.

Certaines unités ont quelquefois un emploi réservé .Ce sont les suivantes:

* ( ) , et AMEN.

AMEN ne doit jamais être utilisé sauf en fin de commande de lecture ou en fin de programme.

Programme PROLOG

<programme>::=<suite de commandes>AMEN.
<suite de commandes>::=<commande><suite de commandes>
<commande>::=<commande de lecture>|<commande fonctionnelle>
<commande de lecture>::=<lecture de fichier>|<lecture d'interface>|
<conventions d'opérateurs>
<commande fonctionnelle>::=<écriture de fichier>|<concatenateur>|
<supprimer>|<démontrer>

Un programme PROLOG est une suite de commandes qui permettent de traiter des fichiers de clauses. Certaines de ces commandes (autre que celles de lectures) s'écrivent sous forme fonctionnelle les arguments étant en général des noms de fichiers.
Les commandes sont éxécutées séquentiellement jusquà AMEN

Organisation des fichiers

Chaque fichier est représenté par un nom, une bibliothèque étant tenue à jour pour éventuellement permettre la récupération de place lors de la suppression de certains fichiers.

Un système de parties communes permet des copies de fichiers très rapides, les recopies étant indépendantes de la taille des clauses, mais liées uniquement aux nombres de clauses du fichier.

D'autre part un système de clefs, dont l'utilisateur n'a pas à se soucier, permet un accès à l'information relativement rapide.

Remarque: le nom d'un fichier peut être n'importe quelle unité hormis AMEN

COMMANDES DE LECTURES

Nous avons toujours jusqu'ici écrit les arguments des littéraux sous forme fonctionnelle. Une telle écriture est certe facile à analyser mais elle présente l'inconvénient majeur de multiplier les parenthèses et les virgules, et de ne pas être toujours claire.

PROLOG dispose de deux commandes, INTERFACE et OPERATEURS qui permettent de remédier à ce désagrément.

Interface

<lecture d'interface>::= INTERFACE<nominterface><suite de définition>

AMEN

<nominterface>::=<unité>

<suite de définitions>::=<définition><suite de définitions>|

<définition>::=<unité>=<unité>

Cette commande permet à l'utilisateur de redéfinir lui-même les unités dans la lecture ou l'écriture future des fichiers de clauses.

Une fois l'interface (un seul interface existe en mémoire) défini, l'utilisateur peut lire ou écrire un fichier. L'interface reste valable tant qu'une autre commande INTERFACE n'est pas éxécutée. Le nom donné à l'interface n'a pas d'importance. Il est utilisé uniquement dans les messages d'erreur.
La définition <unité$_1$>=<unité$_2$> signifie que:

-dans la lecture future d'un fichier de clauses, chaque fois que <unité$_1$> sera lue, elle sera remplacée par< unité$_2$> en mémoire.

-dans l'écriture d'un fichier de clauses résidant en mémoire, chaque fois que <unité$_1$> devra être écrite elle sera remplacée à l'impression par <unité$_2$> .

Ainsi, la commande suivante exécutée:

```
INTERFACE     DEF

DEBUT=(     FIN=  )  ONA= +     SI=-  PROG1=P

PROG2=Q     PROG3=R     ET=,     STOP=.

AMEN
```

La lecture de la clause suivante:

```
ONA   PROG1

DEBUT
      F(*x)  ET  F(*y)  ET  F(*z)
FIN

SI     PROG2

DEBUT
       F(*x)
FIN

SI  PROG3

DEBUT
       F(*y)  ET  F(*z)
FIN

STOP;
```

donnera en mémoire la clause:

```
+P(F(*x),F(*y),F(*z))-Q(F(*x))-R(F(*y),F(*z)).;
```

Remarques

- Dans une commande INTERFACE il est interdit de faire figurer plus d'une fois la même unité à gauche du signe =. Ceci voudrait dire en effet qu'une même unité peut se lire de plusieurs façons. Par contre elle peut très bien figurer plusieurs fois à droite de =.
- Au moment de l'éxécution de INTERFACE, l'interface est initialisé pour chaque unité à <unité>=<unité> (ce qui correspond à l'éxécution d'un interface vide de définitions).
- Au moment de l'éxécution d'un programme PROLOG, une commande d'interface vide est initialisée (chaque unité étant donc lue ou écrite comme elle-même dans les lectures ou écritures de clauses). L'utilisateur désireux de modifier ces conventions doit donc commencer par éxécuter un interface avant de lire ou d'écire des clauses.

Opérateurs

Cette commande permet à l'utilisateur de redéfinir la syntaxe des clauses, en définissant des unités comme opérateurs unaires ou binaires. En outre elle permet de définir un "concaténateur" fort appréciable dans l'écriture de listes ou de formules algébriques.

<conventions d'opérateurs>::= OPERATEURS <nomopérateurs>
<suite de définitions d'opérateurs> AMEN

<nomopérateurs>::=<unité>

<suite de définitions d'opérateurs>::=<définitions d'opérateurs>
<suite de définitions d'opérateurs>|

<définitions d'opérateurs>::=<parite><sens de prenthèsage><listed'op
-érateur>.|CONCATENATEUR<sens de parenthésage><opérateur>.

<parite>::= BINAIRE |UNAIRE

<sens de parenthèsage>::= GD| DG

<liste d'opérateurs>::=<opérateur >,<liste d'opérateurs>|
<opérateur >

<opérateur>::=<unité>

De manière analogue à INTERFACE, OPERATEUR définit la liste des opérateurs unaires ou binaires dont PROLOG devra tenir compte lors des lectures ou écritures de clauses. Cette liste de conventions reste bien sur valable tant qu'une autre commande OPERATEURS n'est pas éxécutée.

Chaque définitions d'opérateurs définit une liste d'opérateurs de même priorité. La première définition donne la liste des opérateurs de priorité 1, la deuxième celle des opérateurs de priorité2 etc..... .
Ces opérateurs sont soit unaires(un seul argument) soit binaires (deux arguments). Ces conventions d'opérateurs permettent de totalement parenthèser une expression où figurent les unités définies comme opérateurs et donc, d'écrire les termes d'une clause de la même façon qu'une expression arithmétique par exemple. Le sens de parenthèsage indique l'ordre dans lequel s'effectue le parenthèsage d'expressions ayant des opérateurs de même priorité.

Ceci implique de plus que les opérateurs unaires dont le sens est GD doivent être placés à droite de l'opérande, ceux dont le sens est DG étant placés à gauche.
Ainsi si ' a été défini comme unaire GD et ABS comme unaire DG, les expressions suivantes sont correctes:

F(*x)' ABS F(*x) ABS F(*x)'

Outre ces conditions l'utilisateur peut définir un concaténateur. Cette unité est alors considérée comme étant un opérateur binaire normal, mais elle peut très bien être omise des clauses.
A la lecture PROLOG reconnait son absence, à l'écriture il l'omet.

Remarques
-Une même unité ne peut être définie plus d'une fois dans les conventions d'opérateurs.
-Une unité réservée : * ou , ou (ou), ne peut être définie comme opérateur.
-Un seul concaténateur peut être définit dans ces conventions.
-Une commande OPERATEURS nomopérateurs AMEN (sans définitions) a pour effet de ne déclarer aucune unité de type opérateur. Cette commande est toujours lancée à l'éxécution d'un programme PROLOG.

Expressions

Chaque argument d'une formule atomique peut s'écrire comme une expression au lieu d'un arbre fonctionnel grâce aux conventions d'opérateurs. Nous allons étudier comment une telle expression est transformée à la lecture en arbre fonctionnel et à l'écriture comment un arbre fonctionnel est transformé en une expression.
On suppose dans la syntaxe suivante que la priorité maximum des opérateurs définis par OPERATEURS est N (éventuellement nul).

<expression>::=<exp 1>

<exp n>::=<exp n+1><opérateur binaire DG n><exp n>|<exp n+1>
<exp n>::=<exp n><opérateur binaire GD n><exp n+1>|<exp n+1>
<exp n>::=<exp n ><opérateur unaire GD n>|<exp n+1>
<exp n>::=<opérateur unaire DG n><exp n >|<exp n+1>
} 1≤n≤N

<opérateur binaire GD n>::=<unité>

<opérateur binaire DG n>::=<unité>

<opérateur unaire GD n>::=<unité>

<opérateur unaire DG n>::=<unité>

<exp N+1>::= <primaire>

<primaire>::=<constante >|<symbole fonctionnel >(<liste d'expressions>)|
<variable > |(<expression >)

<variable>::= *<unité>

<constante>::=<unité>

<symbole fonctionnel>::=<unité>

<liste d'expressions>::=<expression>,<liste d'expressions>|
<expression>

Lorsqu'une expression est analysée par PROLOG elle est transformée en expression totalement parenthésée qui peut également se représenter par un arbre. Le parenthèsage s'effectue en accord avec les priorités et le sens GD ou DG pour les opérateurs de même priorité, les expressions ayant un opérateur de plus forte priorité étant parenthésées en premier.
Ainsi si la commande suivante a été précédemment éxécutée:

```
OPERATEURS   OPER

BINAIRE   GD  +,- .

BINAIRE   DG  .,/.

UNAIRE    DG  ABS.  UNAIRE GD ' .

AMEN
```

l'expression suivante:

F(*x+*y.*z/2.A-ABS F(*x)').G(*x) sera parenthésée par:

F((*x+(*y.(*z/(2.A)))) -ABS((F(*x))')) . G(*x))

qui s'écrit sous forme d'arbre par:

.(F(-(+(*,.(*y,/(*z,.(2.A)))),ABS('(F(*x))))),G(*x))

On peut voir sur cet exemple simple l'intérêt de convention d'opérateurs.

A l'écriture, un arbre résidant en mémoire (et c'est en fait toujours sous cette forme que sont codés les termes d'une clause) est écrit comme une expression, le nombre de parenthèses étant minimisée, sans créer d'ambiguités. L'expression est telle que relue avec les même conventions elle donnerait le même arbre.
Ainsi, en utilisant les mêmes conventions que précédemment l'arbre suivant:

F(+(+(+(A,B),C),+(+(E,F),G))) sera imprimé

F((A+B+C)+(E+F+G))

Voici maintenant un exemple d'utilisation du concaténateur:

OPERATEURS TESTCONC

BINAIRE GD +,-. CONCATENATEUR GD. .

AMEN

Avec ces conventions le littéral suivant:

+P((A+B)(C+D)E F) sera analysé par:

+P(.(.(.(+(A,B),+(C,D)),E),F))

le littéral codé en mémoire par:

+P(.(.(.(A,B),C),.(.(E,F),G))) sera imprimé

+P(A B C)(E F G)

Remarques

-L'omission de l'unité définie comme concaténateur n'apporte jamais d'ambiguitée sauf dans les expressions du type suivant:

<symbole fonctionnel><concaténateur>(<expression>) où <concaténateur> ne peut être omis.
Par exemple A(B+C) signifie toujours que A est pris comme fonction a un argument et cette expression est déjà un arbre pour l'analyseur.

-Toute unité employée comme opérateur dans une clause doit avoir été définie en tant que tel dans les conventions.

-Aucune unité réservée ne peut être employée comme symbole fonctionnels ou comme opérateur dans une clause.

-Une même unité ne doit pas figurer dans une clause à la fois comme opérateur et comme symbole fonctionnel .
Toutes ces remarques s'appliquent une fois la clause transformée par l'interface.

-Les conventions d'opérateurs s'appliquent toujours après le passage de la clause par l'interface en lecture et avant le passage par l'interface en écriture. Donc si on a déclaré . comme opérateur binaire et si l'interface comporte la définition POINT=, l'expression A POINT B sera codée par . (A,B) en lecture.

Lecture de clauses

C'est la commande lire qui permet d'effectuer cette opération.

<lecture de fichier >:= LIRE<nomde fichier><suite de clauses> AMEN

<nom de fichier>::=<unité >

<suite de clauses>::=<clause><suite de clauses> |

<clause >:=<partie principale><partie réponse><condition de retour>|
<clause commentaire >

<clause commentaire>::= *<suite d'unités>..
<partie principale>::=<suite de littéraux >

<partie réponse >:=|/<suite de littéraux>

<condition de retour>::=<impossible>|<littéral non supprimé >|
<réponse non obtenue>|<obligatoire >

<impossible>::= ..

<littéral non supprimé>::= . ;

<réponse non obtenue>::= ; .

<obligatoire>::= ; ;

<suite de littéraux>::=<littéral><suite de littéraux>|<littéral>

<littéral>::=<signe><formule atomique>|<littéral évaluable>

<signe>::=<affirmation>|<négation>

<affirmation>::= +

<négation>::= -

<formule atomique>::=<symbole de prédicat>|<symbole de prédicat>
[<liste d'expressions>]

<symbole de prédicat>::=<unité>

<littéral évaluable>::=<littéral évaluable translatable>|
<littéral évaluable non translatable >

<littéral évaluable non translatable>::=<signe> COPIE(<expression>,
<expression>) |<signe>DAF(<expression>,<expression>)|
<signe>ECRIT (<expression>)
<littéral évaluable translatable>::= -BOUM(<expression>,
<expression>)| -DIF(<expression>,<expression>)

L'éxécution de la commande LIRE provoque la création d'un fichier dont le nom est <nom de fichier>. Ce fichier nouveau écrase tout ancien fichier de même nom.
La lecture des clauses s'effectue en tenant compte des conventions d'interface et d'opérateurs présentes en mémoire au moment de l'éxécution de cette lecture. Chaque clause est lue successivement, les expressions arguments des prédicats et des fonctions étant transformées en arbres fonctionnels.

Remarque:
-Aucune unité réservée ne doit être utilisée comme symbole de prédicat ou comme symbole fonctionnel. (Une fois la clause passée par l'interface bien sur).
-Les explications concernant les littéraux évaluables et les conditions de retour seront données dans le chapitre sur la démonstration.

ECRITURE ET COPIE DES FICHIERS

Tout fichier présent en mémoire peut être édité, copié ou supprimé.

Ecriture des fichiers

<écriture de fichiers>::=<écrire>(<liste de fichiers>)
<liste de fichiers>::=<nom de fichier><liste de fichiers>|
<nom de fichier>

Tout fichier nommé dans cette commande doit être présent en mémoire au moment de l'éxécution de la commande. L'impression des clauses se fait en accord avec l'interface et les conventions d'opérateurs présents en mémoire au moment de la commande.
PROLOG imprime alors le nombre minimum de parenthèses en respectant ces conventions, c'est à dire que tout arbre dominé par un symbole déclaré comme opérateur est écrit sous forme d'expression, le concaténateur étant omis.

Nota-bene

La "résolution" de deux clauses, utilisée par PROLOG, ne faisant intervenir que les premiers littéraux de ces clauses (en testant s'ils sont unifiables) l'ordre des clauses d'un fichier n'a d'importance que pour les clauses ayant un premier littéral de même signe et de même symbole de prédicat.
Aussi, les clauses d'un fichier sont-elles regroupées en fonction du signe et du symbole de prédicat de leur premier littéral. L'impression d'un fichier respecte lui aussi ce regroupement, l'ordre des clauses d'un même groupe étant bien sur important. Ce regroupement permet à PROLOG d'augmenter son efficacité dans la recherche des clauses d'un fichier susceptibles de se résoudre sur une clause donnée.

Copie des fichiers

<concaténer>::= CONCATENER(<fichier crée>,<liste de fichiers>)

Cette commande permet copier des fichiers et de concaténer ces copies pour créer un nouveau fichier. Les fichiers nommés dans la liste doivent déjà éxister et le fichier crée peut éventuellement écraser tout fichier de même nom.
Grâce à un système de parties communes, la copie d'un fichier de clauses ne dépend, en taille mémoire et en temps d'éxécution que du nombre de clauses et non de la taille de ces clauses.

Suppression de fichiers

<supprimer>::= SUPPRIMER (<liste de fichiers>)

Cette commande a pour effet de supprimer les fichiers nommés dans la liste et donc de récupérer éventuellement des zones libérées en mémoire.

DEMONSTRATIONS

La commande que nous allons étudier maintenant est de loin la plus importante.
C'est elle qui permet d'"éxécuter" un fichier de clauses à partir de la méthode de résolution citée précédemment.

<démontrer>::= DEMONTRER(<axiomes>,<données>,<sortie>
<liste d'options>)
<axiomes>::=<nom de fichier>
<données>::=<nom de fichier>
<sortie>::=<nom de fichier>
<liste d'option>::=,<option><liste d'options>|
<option>::=<écriture pas à pas>*écriture des impasses>|
<utilisation d'occur>
<écriture pas à pas>::=ECRIRE
<écriture des impasses>::=IMPASSES
<utilisation d'occur>::= OCCUR

<axiomes> est un fichier déjà existant qui sera le fichier éxécuté,donc en quelque sorte le "programme" ou plutôt, l'ensemble des "procédures".
Le fichier <données> constitue l'ensemble des données de ce "programme". Il doit lui aussi être présent en mémoire.
<sortie> est le fichier dans lequel seront rangés les clauses résultant de l'éxécution. Ce fichier une fois l'éxécution terminée écrase tout fichier de même nom déjà éxistant (qui pourrait être par exemple <données> ou <axiomes>).

Principe général de la démonstration

L'éxécution de démontrer s'effectue de la manière suivante
Les clauses de <données> sont traités successivement par <axiomes>. Le traitement d'une clause C par <axiomes> s'effectuant ainsi : Le fichier <axiomes> est parcouru du début à la fin jusqu'à trouver une clause qui puisse se résoudre sur C. Certaines conditions dans la résolvante R sont alors examinés. Si ces conditions sont satisfaites on teste si la partie principale de cette clause est non vide (auquel cas elle constitue une réponse) et alors on effectue son traitement par le fichier <axiomes>.

Une fois ce traitement terminé (des réponses ayant pu être générées au cours de celui-ci) et sous certaines conditions de retour, le fichier <axiomes> est parcouru à partir de l'axiome utilisé pour créer R, jusqu'à trouver une nouvelle résolution possible sur C. La résolvante est traitée comme précédemment et ainsi de suite jusqu'à la fin du fichier <axiomes>.

Déductions - Backtracking

Lors d'une "démonstration" (traitement d'une clause de <données>) des résolvantes sont crées en cascade, chacune étant obtenue à partir de la précédente. Une telle suite de résolution s'appelle une déduction. Une telle déduction s'arrête sur une impasse. Cette impasse est une clause de plusieurs types possibles.

1) ou bien la clause comporte des conditions qui ne sont plus satisfaites
2) ou bien la clause a une partie principale vide ou réduite à des conditions satisfaites. Dans ce cas elle constitue une clause de sortie.
3) ou bien la clause ne peut se résoudre avec aucune clause de <axiomes>.

Une clause de l'un de ces types ne peut plus générer de résolvantes :

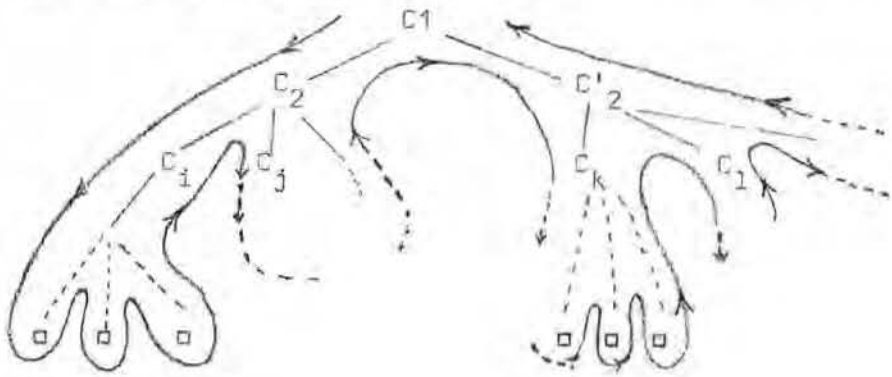
Une déduction à partir d'une clause C_1 est donc constituée d'une suite de clauses

C_1 C_2.....C_n où C_1 est une clause de <données>, C_2 une résolvante d'un axiome sur C_3,.....,C_{i+1} une résolvante d'un axiome sur C_i,.....etc.....
On dit alors que C_i est le parent immédiat de C_{i+1}, C_{i+1} un descendant immédiat de C_i.

Lorsqu'on aboutit à une impasse C_n, PROLOG effectue alors un "backtracking" et revient au traitment de C_{n-1}.

Sous certaines conditions de retour C_{n-1} est résolu avec d'autres axiomes et une nouvelle chaine de résolutions est ainsi crée puis le contrôle revient à C_{n-1} qui est de nouveau résolu, ceci jusqu'à épuisement des axiomes on a un test de retour insatisfait. Le contrôle est alors redonné à C_{n-2}, ainsi de suite.

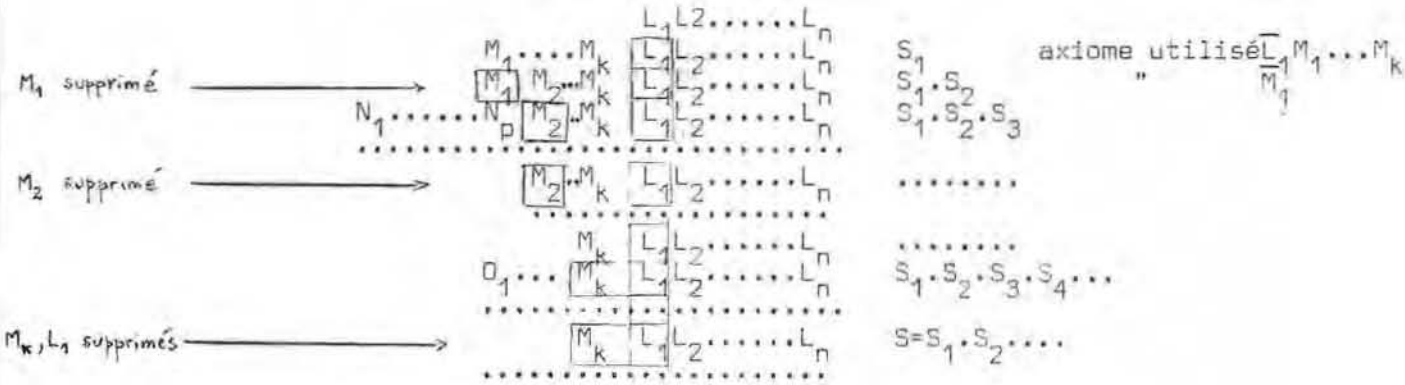
On peut représenter les descendants d'une clause C_1 de <données> sous forme d'un arbre:



La génération des clauses de cet arbre s'effectue suivant le parcours fléché, les □ représentant des impasses, une branche de cet arbre représentant une déduction.
La génération de tous les descendants de chaque clause de <données> s'appelle une <u>démonstration</u>.

<u>Suppression d'un littéral</u>

On peut remarquer que dans une déduction, les littéraux des clauses obéissent à une structure de pile. Une déduction à partir d'une clause consiste en effet à "supprimer" successivement chacun des littéraux de la clause puisque le but à atteindre est d'obtenir une partie principale vide. Nous omettons ici les parties dont les littéraux ne sont jamais résolus mais servent uniquement à garder trace des substitutions effectuées.
Exemple d'une partie de déduction:



Nous avons représenté les clauses par des littéraux auxquels il convient d'appliquer les substitutions successives qui ont servi à unifier et qui sont à droite du schéma.
Les littéraux encadrés sont ceux qui ont été unifiés avec les littéraux des axiomes. Ils ne figurent plus en fait dans les clauses mais montrent la structure de pile.
L'exemple montre comment on a pu "supprimer" L_1 par résolutions successives. $M_1,.....M_k$ ont été introduits par résolution (et la substitution S_1 a été appliquée aux deux clauses qui ont servi à former la résolvante). $M_1,.....,M_k$ sont ensuite "supprimés" etc....

Une fois L_1 "supprimé" il reste à supprimer L_2, ceci jusquà L_n.

Lorsqu'un littéral ne peut plus être supprimé celà signifie qu'on est sur une impasse du type 1 ou 3.
Notons que au cours de cette déduction, la clause $(L_2L_3...L_n).S$ obtenue est une instance de $L_2...L_n$ c'est à dire que la déduction rend les clauses de moins en moins "générales".

Nous n'avons écrit ici qu'une partie de déduction permettant de canceller L_1. Il peut évidemment y en avoir plusieurs, en utilisant d'autres axiomes. On peut ainsi obtenir une autre clause $(L_2L_3...L_n).S'$ distinctes de $(L_2...L_n).S$, les substitutions effectuées au cours des deux déduction ayant pu être différentes.

Conditions de retour

Nous avons dit que du backtracking, PROLOG continuait le parcours du fichier axiomes pour trouver d'autres résolvantes, sous certaines conditions. Ces conditions sont fixées par la ponctuation des clauses utilisées en axiomes.

Supposons que C_n ait été résolu avec un axiome A pour générer une résolvante C_{n+1}, qui à son tour a pu servir à la génération d'autres clauses.
Lors du backtracking sur C_n, le parcours des axiomes à partir de A est effectué pour chercher un axiome susceptible de se résoudre sur C_n, à la condition suivante:

-Si la ponctuation de A est .. C_n n'est résolu avec aucun autre axiome.

-Si " " .; C_n n'est résolu avec d'autres axiomes que si dans les déductions à partir de C_{n+1} tous les littéraux de A ont pu être supprimés

-Si " " ;. C_n n'est résolu avec d'autres axiomes que si des réponses ont pu être générées dans les déductions à partir de C_{n+1}

-Si " " ;; PROLOG va tuojours essayer de résoudre C_n sur les axiomes qui suivent A.

Ces quatre options permettent au programmeur de contrôler l'éxécution du backtracking et de programmer en quelque sorte des stratégies. Ces définitions conduisent aux remarques suivantes:

Utiliser un fichier d'axiomes ponctués par .. revient à écrire un programme déterministe puisque une clause traitée par ce fichier ne pourra se résoudre qu'avec au plus au seul axiome. Cette ponctuation évite un traitement récursif de la clause et donc est plus performante qu'une autre ponctuation. Elle est à conseiller lorsqu'on est certain d'écrire un "programme" déterministe.

Ecrire un ensemble de clauses (ayant par exemple le même signe et le même symbole de prédicat) ponctuées par .; permet d'affirmer que pour "supprimer " un littéral d'une clause, un seul parmi ces axiomes (et toujours le même) pourra servir dans le premier pas des déductions qui ont permi de supprimer le littéral. La différence avec .. est que si la clause en .; est utilisée pour tenter de supprimer un littéral, et si la suppression ne peut se faire, une autre clause peut être utilisée.

Lorsqu'un ensemble d'axiomes est ponctué par ;. celà signifie qu'une seule réponse peut être obtenue.

Des clauses en ;; signifient que qoiqu'il arrive, pour traiter une clause,toutes les résolutions seront tentées.

Dans l'étude des analogies entre les démonstrations et les éxécutions de procédures nous reviendrons sur ces remarques. Nous n'avons pas parlé encore des littéraux <u>évaluables</u> qui sont traités d'une manière spéciale.

Littéraux évaluables

Ces littéraux ont tous la particularité de ne jamais être résolus avec les axiomes, c'est à dire qu'ils ne sont jamais unifiés avec les premiers littéraux des axiomes. Certains de ces littéraux sont non translatables, c'est à dire qu'il sont ordonnés comme les littéraux normaux de la clause.

1.Littéraux non translatables

Les symboles de prédicat de ces littéraux sont DAF,COPIE,ECRIT. La différence dans leur traitement avec un littéral ordinaire est qu'au lieu d'être supprimés par résolution avec des axiomes ils sont évalués (à vrai ou faux) et mènent alors ou bien à une impasse, ou bien à leur suppression. On dit qu'ils sont résolus par évaluation.

Soit $L_1 \ldots L_n / R_1 \ldots R_p$ une clause devant être traitée, L_1 étant un littéral non translatable. La formule atomique L_1 est alors évaluée à vrai ou faux, le littéral L_1 évalué lui aussi à vrai ou faux (le signe - jouant le même rôle qu'une négation).
-Si l'évaluation de L_1 est VRAI la clause $L_1 \ldots L_n / R_1 \ldots R_n$ est une impasse (analogue à une clause qui ne pourrait se résoudre avec aucun axiome).
-Si l'évaluation de L_1 est FAUX le littéral L_1 est purement et simplement supprimé de la clause, la clause obtenue alors étant traitée à son tour. Ce résultat apparemment surprenant s'explique si on considère que l'on cherche à "démontrer" $\overline{L_1}$.
Donc si L_1 est VRAI $\overline{L_1}$ est impossible à démontrer
si L_1 est FAUX $\overline{L_1}$ est donc démontré.

Notons que l'évaluation d'un littéral de cette sorte a très bien pu créer des substitutions et modifier la clause $L_1 \ldots L_n / R_1 \ldots R_n$ en $(L_2 \ldots L_n / R_1 \ldots R_n).S$

Voici les résultats d'évaluation des 3 prédicats DAF,COPIE et ECRIT.

*ECRIT (<expression>)
Le résultat de l'évaluation est toujours vrai, mais il s'accompagne de l'impression de <expression>

*DAF (<expression 1>,<expression 2>)
Le résultat de l'évaluation est FAUX si <expression 1> est formellement égal à <expression 2>, VRAI sinon.

Exemples

DAF(*x,*x) FAUX

DAF(F(*y),A) VRAI

DAF(*x,*y) VRAI

*COPIE (<expression 1>,<expression 2>)

A l'inverse des deux premiers prédicats, l'évaluation de ce prédicat peut créer des substitutions dans la clause où il figure. L'évaluation est faite ainsi:
<expression 1> est recopiée, l'arbre constituant la copie étant le même à la seule différence que les variables de <expression 1> sont recopiées en de nouvelles variables qui ne figurent pas dans la clause où se trouve le littéral ± COPIE.
La recopie est alors unifiée avec <expression 2>. Si cette unification est possible, S étant le p.g.u, S est appliquée à toute la clause et le résultat de l'évaluation est VRAI. Sinon le résultat est FAUX et aucune substitution n'est appliquée sur la clause.

Exemple:

-COPIE(F(*x,G(*x),*y),*x)+P(*x,*y,*u) ..

l'évaluation de ce littéral provoque les opérations suivantes:

F(*x,G(*x),*y) est recopié en F(*z,G(*z),*v) *z et*v nouvelles variables
F(*z,G(*z),*v) est unifié avec*x, le p.g.u étant

s={*x→F(*z,G(*z),*v)}.

le littéral est donc supprimé et on obtient la clause:

+P(F(*z,G(*z),*v),*y,*u] ..

Le littéral a été supprimé car évalué à FAUX (la formule atomique à VRAI).

Ce prédicat permet donc d'insérer de nouvelles variables dans une clause en créant un arbre isomorphe à un autre.
Rappelons donc que les littéraux,évaluables non translatables se comportent comme les autres littéraux, sauf que au lieu de se résoudre avec des axiomes ils sont évalués.

Littéraux translatables

Ces littéraux sont eux aussi évaluables et donc jamais résolus avec les axiomes. Cependant, alors que les littéraux non translatables sont évalués lorsqu'ils sont en première position dans la clause, ceux-ci le sont constamment.
Leur position dans une clause est donc sans importance.
Dès qu'une clause est crée (résolvante d'une clause sur un axiome) les conditions que constituent les littéraux translatables sont évalués, quelque soit l'endroit où ils figurent dans la clause.
Trois cas peuvent se produire:
1) l'évaluation est impossible à faire
2) l'évaluation donne VRAI
3) l'évaluation donne FAUX

Dans le premier cas le littéral est rejeté en fin de clause.
Dans le deuxième cas on aboutit à une impasse.
Dans le troisième cas le littéral est supprimé.
Notons que l'évaluation a pu éventuellement provoquer des substitutions et donc modifier la clause (auquel cas les autres conditions sont de nouveau testées).

L'intéret de tels prédicats est de pouvoir retarder leur évaluation lorsqu'on ne peut décider de leur valeur.
Les deux littéraux de cette forme sont -DIF(<expression 1>,<expression 2>) et -BOUM(<expression 1>,<expression 2>).

*DIF (<expression 1>,<expression 2>)

lorque les deux expressions sont sans variables, ce prédicat s'évalue comme DAF (les 2 expressions doivent être deux arbres différents pour que le résultat soit VRAI).
Lorsque des variables interviennent dans la formule atomique DIF(expression 1 , expression 2) on considère que cette formule peut être évaluée à VRAI que si toutes les instances sans variables de cette formule sont vraies.

D'où les trois résultats:
1)<expression 1> et <expression 2> égales formellement : FAUX
2) " " " unifiables mais non égales : évaluation impossible puisque une substitution peut rendre les deux expressions égales.
3)<expression 1> et <expression 2> non unifiables: VRAI puisque aucune substitution ne peut rendre les deux expressions égales.

En fait le littéral contient toujours - comme signe donc toute clause nouvellement crée qui contient -DIF est traitée de cette façon:
Dans le premier cas la clause est une impasse.
Dans le deuxième cas le littéral est rejeté à la fin de la partie principale.
Dans le troisième cas le littéral est supprimé.

L'emploi de ce littéral est utile pour empêcher qu'une variable prenne certaines valeurs dans une clause.

Exemples:

-P(*x,*y)+Q(*y,*z)+R(*x,A) -DIF(*x,*y) -DIF(*x,B) ..

précise que cette clause pourra être utilisée dans une déduction tant que *x et *y ne deviendrons pas égaux et tant que *x ne deviendra pas égal à B.

Remarque importante

Si on résoud une clause C avec un axiome A de la forme

..... -DIF(<expression 1>,<expression 2>).....

la ponctuation étant .., et si dans la résolvante R obtenue le littéral DIF mène à une impasse la clause C n'est plus résolue avec les axiomes.
De la même façon le test pour savoir si un littéral a pu être "supprimé" à partir d'une clause R se fait avant l'évaluation des littéraux évaluables translatables de R.

BOUM (<expression 1>,<expression 2>)

Ce prédicat, lorsque les expressions sont sans variables, est VRAI si <expression 2> est "l'éclatée" de <expression 1> i-e si les conditions suivantes sont réalisées:

1)<expression 1> est un symbole fonctionnel sans arguments (donc une unité de type fonction constante), de la forme $a_1....a_n$ (a_i caractère).

2)<expression 2> est le peigne formé à partir de ces caractères, c'est à dire l'arbre binaire gauche droite:

$-(-(.....-(-(NIL,a_1),a_2),.....,a_{n-1}),a_n)$

qui peut se représenter par:

Dans le cas où des variables figurent dans les expressions

BOUM(<expression 1>,<expression 2>) est évalué de la manière suivante:

1) si <expression 1> est un symbole fonctionnel constant, le peigne associé est crée et unifié avec <expression 2> .
Si l'unification est possible, S le p.g.u, l'évaluation est VRAI et S est appliqué à la clause, sinon l'évaluation est FAUX.

2) si <expression 2> est sans variables alors:
si c'est un peigne l'unité associée est crée.
si l'unité est correcte syntexiquement elle est unifiée avec <expression 1> le résultat de l'évaluation étant identique à1.
si l'unité est incorrecte l'évaluation est FAUX.
si <expression 2> n'est pas un peigne l'évaluation est FAUX.

Dans les autres cas l'évaluation est impossible. Le traitement du littéral -BOUM est analogue à -DIF en fonction de cette évaluation.
L'emploi de BOUM est fort intéressant car il permet de créer dynamiquement de nouvelles unités, d'accéder aux caractères des unités et donc d'être très utile par exemple dans les phases de morphologie lors de traitement de langages naturels.
Donnons un court exemple de traitement des pluriels:

```
Axiomes:
1  +PLURIEL(*x,*y) -T(*xx,*yy) -BOUM(*x,*xx) -BOUM(*y,*yy) ..

2  +T(NIL-C-H-A-C-A-L,NIL-C-H-A-C-A-L -S).

3  +T(NIL-  E-T-A-L  ,NIL- E-T-A-L -S)...
.....
10 +T(*x-A-L,*x-A-U-x) ..
```

Données:

```
11  -PLURIEL(METAL,*x)/+RESULTAT(*x) ..
```

Traitement de Données:

```
11 et 1 → 12
-T(*xx,*yy) -BOUM(METAL,*xx) -BOUM(*y,*yy)/+RESULTAT(*y)
```

évaluation de 12

```
13 -T( NIL-M-E-T-A-L,*yy) -BOUM(*y,*yy)/+RESULTAT(*y)
```

```
13 et 10 → 14
-BOUM(*y, NIL-M-E-T-A-U-X)/+RESULTAT(*y)
```

évaluation de 14

```
15  /+RESULTAT(METAUX);.
```

En supposant que ces clauses ont été écrites avec la convention d'opérateurs:

```
        BINAIRE   GO
```

Traitement des réponses

Lorsqu'une clause générée C n'a plus, dans sa partie principale, que des littéraux évaluables translatables (et après évaluation de ceux-ci), la clause constitue une clause réponse.

Une clause S est crée de la façon suivante et ajoutée au fichier <sorties>.

La partie réponse de C est recopiée et constitue la partie principale de S, à laquelle sont ajoutés les littéraux évaluables encore présents dans C.
Tout littéral figurant plusieurs fois dans S est alors effacé sauf l'occurence la plus à gauche.
La partie réponse de S est vide et la ponctuation est ;.
Après quoi S est ajoutée au fichier <sorties> .
Voici un exemple :

R: -DIF(*x,A)-DIF(*y,B)/+REPONSE1(*x)-REPONSE2(*y)+REPONSE1(*x)

S: +REPONSE1(*x)-REPONSE2(*y)-DIF(*x,A)-DIF(*y,B) ;.

Traitement des options

Trois options sont disponibles dans la commande DEMONTRER: IMPASSES,ECRIRE,OCCUR. Ces options sont mises à FAUX par défaut. Elles permettent l'édition de résultats intermédiaires et l'optimisation du temps d'éxécution.
-ECRIRE provoque l'impression de toutes les résolutions effectuées (donc de tous les pas de chaque déduction).
-IMPASSES provoque uniquement l'impression des impasses.
Ces impression tiennent compte de l'interface et des conventions d'opérateurs alors en mémoire.
-OCCUR permet d'éliminer, dans la procédure d'unification, le test qui décide si une variable apparait dans une listes d'arbres. Cette option réduit les temps d'éxécution mais peut introduire des erreurs si de tels tests sont nécessaires.

Analogies entre démonstrations et appels de procédures

Lorsqu'une clause C de la forme $\overline{L}_1\overline{L}_2.....\overline{L}_n/R_1....R_n$ est traitée par un ensemble d'axiomes PROLOG tente de déduire de cette clause des réponses, donc tente de "supprimer" successivement $\overline{L}_1,\overline{L}_2,...$puis $\overline{L}_n$ au cours de déductions. Il essaie en fait de "démontrer" $L_1,.....$, puis L_n à partir des axiomes.
Pour pouvoir démontrer L_i il faut commencer par utiliser un axiome de la forme $M_0\overline{M}_1....\overline{M}_k/S_1....S_1$ dont le premier littéral M_0 est de même signe que L_i et unifiable avec L_i. Après quoi il faut "démontrer" $M_1,....M_k$.

Appels de procédures

Si l'on considère qu'un signe (+ou-) et un symbole de prédicat représentent un nom de procédure, "démontrer" un littéral de la forme <signe><symbole de prédicat>(<liste d'expressions>) c'est commencer par appeler la procédure <signe><symbole de prédicat> avec comme arguments la liste d'expressions.

Textes d'une procédure

Le texte d'une procédure est constitué de tous les axiomes (ordonnés) dont le premier littéral est identique en signe et en symbole de prédicat.
Exemple:

```
+P(*x,+(*y,*z)).....

+P(*x,.(*y,*z)).....

+P(*x,*x).....
```

Chaque axiome représente alors une suite d'appels à d'autres (ou à la même) procédures. Ainsi

$M_0\overline{M}_1.....\overline{M}_k/S_1.....S_1$ représente des appels successifs à k procédures.

Choix des axiomes

Etant donné une liste d'arguments d'un appel de procédure, cette liste va déterminer le choix des axiomes qui seront éxécutés lors de l'appel. Ce choix est fait en fonction de l'unification possible ou non de la liste d'arguments d'appels et de la liste des "paramètres effectifs" de chaque axiome .

La différence avec un langage comme Algol 60 ou LISP par exemple réside dans le fait qu'un appel donné peut générer l'éxécution de plusieurs axiomes, éxécution simultanée (simultanéité simulée bien sur puique les axiomes sont éxécutés dans un certains ordre, l'éxécution d'un axiome n'effectuant pas la liste des arguments d'appels puisque le backtracking rétablit les variables dans leur état initial) donc non déterministe.

Chaque axiome représente donc une "instruction", plusieurs pouvant être éxécutées parallèlement de façon simulée.

Instruction conditionnelle

L'unification d'une liste d'appel avec une liste de paramètres effectifs d'un axiome constitue en quelque sorte des conditions d'éxécution d'une instruction. Si on note ces conditions par <condition> on peut représenter un axiome par une instruction conditionnelle fonction des conditions de retour.

* $M_0\overline{M}_1.....\overline{M}_k$ .. se représente par

if <condition> then <appels> else

* $M_0\overline{M}_1.....\overline{M}_k$;; se représente par

if <condition> then <appels>

Transmission des arguments

Par le jeu de l'unification les appels de procédures se font par nom puisque des substitutions formelles ont lieu. Le nouveau concept introduit par PROLOG est que les paramètres d'appel peuvent être des arbres avec ou sans variables. C'est ce dernier point qui permet à PROLOG la génération de "programmes" par exemple, ou la possibilité d'écrire des arguments de type "résultat".

DEUX PROGRAMMES

Le premier de ces programmes est déterministe et consiste en un traitement d'arbres. Le second est un système de questions-réponses élémentaire.

Développement d'un produit de facteurs

Le programme écrit ici a pour but, étant donnée une expression arithmétique à développer tous les produits de facteurs.

La syntaxe de l'expression d'entrée est la suivante:

<expression> ::= <facteur>|< facteur>+<expression>

<facteur> ::= <primaire>|<primaire>.<facteur>

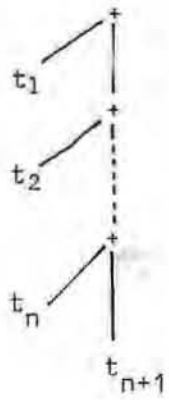
<primaire> ::= <constante>|(<expression>)

L'expression sera analysée en un arbre binaire A ayant + et . comme symboles fonctionnels, les feuilles étant des constantes. Les unités + et . seront déclarés comme opérateurs binaires DG.

L'expression de sortie aura la syntaxe suivante:

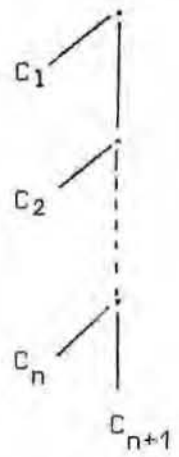
<exp développée> ::= <produit>+<exp développée>|<produit>
<produit> ::= <constante>|<constante>.<produit>

Elle sera produite par écriture d'un arbre B ayant la forme suivante:



ou t_1

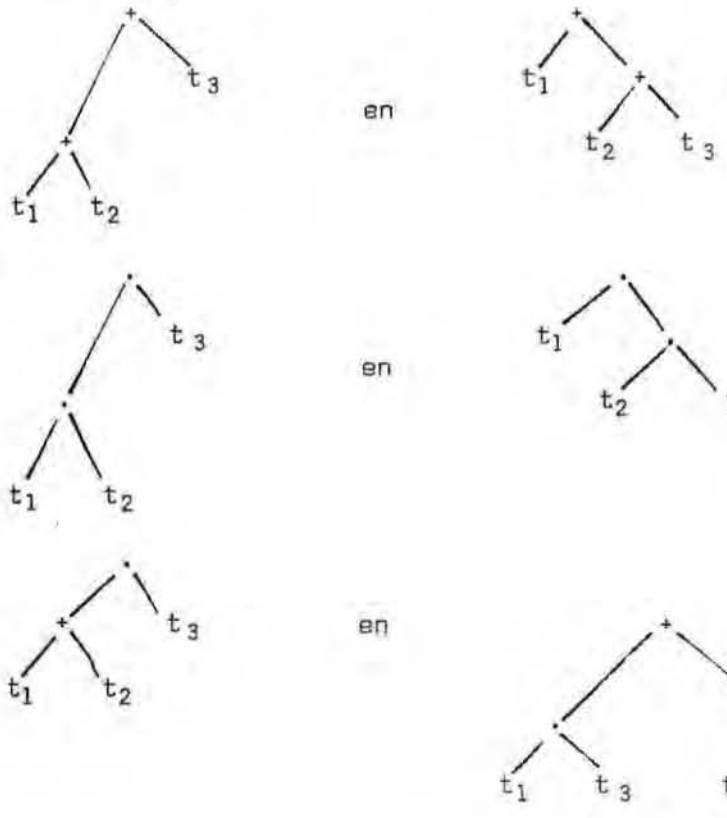
$t_1, \ldots, t_{n+1}$ étant des arbres de la forme



ou C_i

Le programme consiste donc à transformer l'arbre A en B grâce aux règles d'associativité et de distributivité.

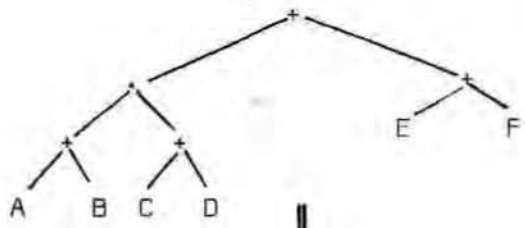
Ainsi on pourra transformer:



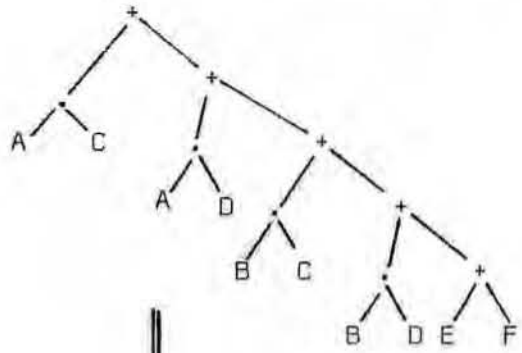
Exemple:

(A+B).(C+D)+E+F

transformation par lecture



transformation par application des règles



transformation par impression

A.C + A.D + B.C + B.D + E + F

```
OPERATEURS ARITH
  BINAIRE DG + .
  BINAIRE DG . .
AMEN

LIRE AXIOMES
 +T(*A + *B,*C)-T(*A,*A1)-T(*B,*B1)-TPLUS(*A1,*B1,*C)..
 +T(*A . *B,*C)-T(*A,*A1)-T(*B,*B1)-TMULT(*A1,*B1,*C)..
 +T(*A,*A)..

 +TPLUS(*A + *B,*C,*A++D)-TPLUS(*B,*C,*D)..
 +TPLUS(*A,*B,*A++B)..

 +TMULT(*A,*B++C,*D)-TMULT(*A,*B,*D1)-TMULT(*A,*C,*D2)
                    -TPLUS(*D1,*D2,*D)..
 +TMULT(*A++B,*C,*D)-TMULT(*A,*C,*D1)-TMULT(*B,*C,*D2)
                    -TPLUS(*D1,*D2,*D)..
 +TMULT(*A.*B,*C,*A.*D)-TMULT(*B,*C,*D)..
 +TMULT(*A,*B,*A.*B)..
AMEN

LIRE DONNEES
-T( (A+B).(C+E.F+A).(E+H)+I+(A+B.C) , *X) /+REP(*X)..
AMEN

DEMONTRER(AXIOMES,DONNEES,RESULT)
ECRIRE(RESULT)
AMEN
```

```
 FICHIER  RESULT

 +REP(A.C.E+B.C.E+A.E.F.E+B.E.F.E+A.A.E+B.A.E+A.C.H+B.C.H+A.E.F.H+B.E.
F.H+A.A.H+B.A.H+I+A+B.C) ;.

 FIN DU FICHIER  RESULT

0006.03 SECONDS IN EXECUTION
```

Système élémentaire de questions-réponses

Ce système consiste à poser des questions sur le fichier suivant:

Pierre est le père de Paul et de Jean.
Marie est la mère de Jacques
Pierre est l'époux de Marie
Deux individus distincts ayant le même père sont frères
L'époux de la mère d'un individu est le père de cet individu.

On pose alors la question: Qui est le frère de Paul?

On traduit ce texte en utilisant les prédicats PERE, MERE, EPOUX et FRERE la question posée consistant à trouver *X tel que FRERE(PAUL,*X)

QUELQUES REMARQUES

Sur les existentiels

Les lecteurs désireux de travailler sur des formules de logique nécessitant l'emploi de quantificateurs existentiels pourront éliminer ceux-ci (et donc aboutir à des formules clausales) grâce à de nouveaux symboles fonctionnels dits "fonctions de Skölem".
Pour plus de détails on pourra se référer à la bibliographie.

Sur l'utilisation possible de PROLOG

Nous pensons que PROLOG, dans son état actuel, peut fournir un bon outil aux linguistes intéressés au traitement automatique des langages naturels. Des grammaires ont d'ailleurs déjà été écrites dans ce langage.

Le traitement algorithmique de formules algébriques (intégration, opérations sur les polynômes, etc...) peut se formuler aisément en PROLOG.

L'écriture de programmes PROLOG peut sembler plus difficile que dans des langages déjà existant. Il nous a fallu d'ailleurs un certain temps pour entrevoir toutes les possibilités qu'il offre, les concepts qu'il introduit (maniement des variables par exemple) étant nouveaux en programmation.

L'écriture est cependant assez concise et formelle pour servir de support à des recherches théoriques sur la programmation (génération et validation de programmes etc...)

Sur l'amélioration de PROLOG

PROLOG est loin d'être définitivement fixé. Le traitement de données numériques ainsi qu'une utilisation conversationnelle de PROLOG sont en cours d'élaboration.
Des interfaces d'entrée-sortie plus évoluées sont également à l'étude. Ces problèmes sont d'ailleurs rattachés au problème plus général des prédicats et fonctions évaluables en démonstration automatique.
La dernière modification prévue, est de formaliser le langage des commandes sous forme clausale et donc d'unifier tout le langage en un "super" démonstrateur.

BIBLIOGRAPHIE

Pour une introduction à la démonstration automatique

*KOWALSKI R.A., HAYES P.J.
Lectures notes on automatic theorem-proving. Memo 40 Mars 7I Departement of computational logic. Edimbourgh University

*NILSSON N.J.
Problem-solving methods in Artificial Intelligence. Mc Graw-Hill Computer Science series.

*ROBINSON J.A.
A machine-oriented logic based on resolution principle. J.A.C.M vol 12 (1965) pp 23-41

Sur les prédicats evaluables

*ROUSSEL Ph.
Définition et traitement de l'égalité formelle. Thèse de troisième cycle. Mai 1972. Université d' AIX-MARSEILLE

Langages de programmation

*COLMERAUER A.
Les systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur. Publication interne n°43 Département d'informatique Faculté des sciences Université de Montréal

*Mc CARTHY and al.
LISP 1.5 Progranner's Manual. MIT Press Cambridge (U.S.A)

*HEWITT C.
"PLANNER": A language for Proving Theorems in robots. Proc. of IJCAI, 1969 pp 295-301

L' ANALYSEUR DU FRANCAIS

Voici tout d'abord le sous-ensemble du Français auquel on s'intéresse.

Sous-ensemble du Français accepté par l'analyseur

<texte>::=<vide>|<phrase><texte>

<phrase>::=<proposition>|EST-CE QUE <proposition>

<proposition>::=<suite non vide de sp><groupe verbal><suite de sp><fin de proposition>

<suite non vide de sp>::=<sp><suite de sp>

<suite de sp>::=<vide>|<sp><suite de sp>

<sp>::=<sn>|<prep><sn>|<advtemps>|<sprel>|< spintero><esteq>|<sppron>

<sn>::=<art><card><nom><relative>|<snrel>|<snintero>|<snpron>

<relative>::=<proposition>|,<proposition>

<groupe verbal>::=<ne><se><le><lui><y><en><verbe><til><pas><fin de verbe>

<ne>::=<vide>|NE |N'

<se>::=<vide>| SE

<le>::=<vide>| LE |LA |L'| LES

<lui>::=<vide>|LUI | LEUR

<y>::=<vide> | Y

<en>::=<vide> |EN

<verbe>::= "n'importe quel verbe à un temps simple à la 3ème personne"

<til>::= - <il> |- T - <il>|<vide>

<il>::= IL | ELLE | ILS | ELLES

<pas>::=<vide>|PAS

<fin de verbe>::="n'importe quelle suite de participes d'adjectifs, d'adverbes et de noms communs"
(exemple: mangé,bien mangé, très soif, etc..)

<prep>::=A|APRES|AVANT|DANS|DE|DURANT|EN|PAR | PENDANT|POUR|SOUS|SUR|VERS

<art>::=<vide>|AUCUN|AUCUNE|CE|CERTAINES|CERTAINS|CES|CET|CHAQUE|CETTE|COMBIEN DE|
LA|L'|LE|LES|PAS|UN|PAS|UNE|QUEL|QUELLE|QUELS|QUELLES|QUELQUES|PLUSIEURS|
TOUT|TOUTE|TOUS|LES|UN|UNE|UN CERTAIN |UNE CERTAINE|1|TOUTES LES

<card>::= DEUX |TROIS|...|DIX|2|3|4|5 ...etc...

<sprel>::= AUQUEL|AUXQUELLES|AUXQUELS | DESQUELLES | DESQUELS | DUQUEL|DONT | OU|
QUE | QUI | QU'

<spintero>::= OU | QUAND | QUE | QUI | QU'

<estceq>::= EST-CE QU' | EST-CE QUE | EST-CE QUI|<vide>

<sppron>::= ELLE |ELLES | IL | ILS

<snrel>::= LAQUELLE | LEQUEL | LESQUELS | OU | QUI

<snintero>::= OU | OU EST-CE QUE | QUI|QUI EST-CE QUE | QUOI |QUOI EST-CE QUE

<snpron>::= ELLES | ELLE | EUX | LUI

<advtemps>::= AUJOURD'HUI | AUSSITOT | BIENTOT | DEMAIN | HIER | TANTOT | TARD | TOT

<fin de proposition>::=<vide>|,|.|?

<nom>::=<nom commun>|*<nom propre>

<nom commun>::="tout nom commun"

<nom propre>::="tout nom propre"

<vide>::=

Voici la forme de la structure syntaxique produite par l'analyseur

Voici la forme de la structure syntaxique produite par l'analyseur

Structure syntaxique

<structure syntaxique> ::= STRUCTURE SYNTAXIQUE (<suite de pr>)

<suite de pr> ::= NIL | (<proposition>).<suite de pr>

<proposition> ::= <type de pr> . (<suite de sp>) . <verbe>

<type de pr> ::= AFF | INTERO | OUINON | COMBIEN | APOS | REST

<suite de sp> ::= NIL | (<sp>) . <suite de sp>

<sp> ::= (<preps>) . <type de sp> . <genre> . <cardinal> . (<nom>) . (<proposition>) . NIL

<preps> ::= NIL |<prep> . <preps>

<prep> ::= SUJ | OBJ | TEMPS | A | APRES | AVANT | DANS | DE | DURANT | EN | PAR |
PENDANT | POUR | SOUS | SUR | VERS

<type de sp> ::= DEF | INDEF | CHAQUE | AUCUN | COMBIEN | INTERO | PRON | REL | REF

<genre> ::= MAS | FEM | *G

<cardinal>::= PLU | 1 | 2 | 3 | 4 .....

<nom> ::= <advtemps> | <genre> . <nom commun s> . <nom commun p> | * <nom propre> |
*<nom propre> . <genre> . <cardinal>

<advtemps> ::= AUJOURDHUI | AUSSITOT | BIENTOT | DEMAIN | HIER | TANTOT | TARD | TOT

<nom commun s> ::= "tout nom commun singulier éclaté par boum"

<nom commun p> ::= "tout nom commun pluriel éclaté par boum"

<nom propre> ::= "tout nom propre"

<verbe> ::= <oui> . (<verbe s> . <verbe p>) . <fin de verbe>

<verbe s> ::= "tout verbe, à la 3ème personne du singulier d'un temps simple,éclaté
par le prédicat boum"

<verbe p> ::= "tout verbe,à la 3ème personne du pluriel d'un temps simple,éclaté
par le prédicat boum"

<fin de verbe> ::= NIL |<mot inconnu> . <fin de verbe>

<mot inconnu> ::= "tout adjectif, participe, adverbe éclaté par le prédicat boum et
auquel on a éventuellement enlevé le e et le s final".

<oui> ::= oui|non

Si l'on reprend l'exemple donné tout au début de ce document, l'analyseur transforme le texte

TOUT PSYCHIATRE EST UNE PERSONNE.
CHAQUE PERSONNE QU'IL ANALYSE, EST MALADE.
*JACQUES EST UN PSYCHIATRE A *MARSEILLE.
EST-CE QUE *JACQUES EST UNE PERSONNE?
OU EST *JACQUES?
EST-CE QUE *JACQUES EST MALADE?

en la structure syntaxique

+STRUCTURESYNTAXIQUE((AFF.(((SUJ.NIL).CHAQUE.MAS.1.(NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S).NIL).((OBJ.NIL).INDEF.FEM.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).NIL).((TEMPS.NIL).INDEF.FEM.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL).(AFF.(((SUJ.NIL).CHAQUE.*X.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).(REST.(((OBJ.NIL).REL.*Y.*Z.NIL.NIL).((SUJ.NIL).PRON.MAS.1.NIL.NIL).NIL).OUI.(NIL-A-N-A-L-Y-S-E.NIL-A-N-A-L-Y-S-E-N-T).NIL).NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL-M-A-L-A-D.NIL).(AFF.(((SUJ.NIL).PROP.*T.1.JACQUES.NIL).((OBJ.NIL).INDEF.MAS.1.(NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S).NIL).((TEMPS.NIL).INDEF.MAS.1.(NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S).NIL).((A.NIL).PROP.*U.1.MARSEILLE.NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL).(OUINON.(((SUJ.NIL).PROP.*V.1.JACQUES.NIL).((OBJ.NIL).INDEF.FEM.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).NIL).((TEMPS.NIL).INDEF.FEM.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL).(INTERO.(((A.APRES.AVANT.DANS.EN.SOUS.SUR.VERS.NIL).INTERO.*W.*X1.NIL.NIL).((SUJ.NIL).PROP.*Y1.1.JACQUES.NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL).(OUINON.(((SUJ.NIL).PROP.*Z1.1.JACQUES.NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL-M-A-L-A-D.NIL).NIL) ;.

Soit ANALYSE (*X,*Y) le prédicat qui signifie que l'arbre syntaxique de la chaîne *X est *Y.

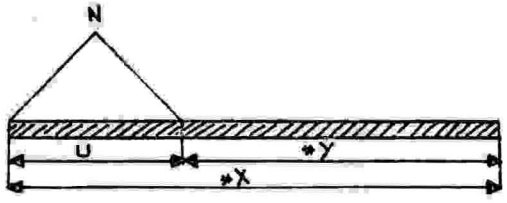
Pour chaque symbole non terminal N de la grammaire on définit un prédicat à trois arguments

N(*X,*V,*Y)

qui affirme que:

-le symbole non terminal N peut se dériver en une chaîne U
-*X = U concaténé avec *Y
-*V est l'arbre syntaxique correspondant à cette dérivation

Tout ceci peut être schématisé par



A chaque règle de la grammaire on peut alors faire correspondre systèmatiquement un axiome de l'analyseur:

```
+ANALYSE(*X,*V) -CHAINE(*X,*V,NIL) ;;

+CHAINE(A.*X1, CHAINE(A,*V,B), *X2) -CHAINE(*X1,*V,B.*X2) ;;

+CHAINE(*X1,CHAINE(*V,*W),*X3) -SUITEDEB(*X1,*V,*X2)
      -SUITEDEA(*X2,*W,*X3) ;;

+SUITEDEA(*X,NIL,*X) ;;

+SUITEDEA(A.*X1,SUITEDEA(A,*V),*X2) -SUITEDEA(*X1,*V,*X2) ;;

+SUITEDEB(*X,NIL,*X) ;;

+SUITEDEB(B.*X1,SUITEDEB(B,*V),*X2) -SUITEDEB(*X1,*V,*X2) ;;
```

Avant de donner l'ensemble des axiomes qui composent l'analyseur, expliquons la façon d'axiomatiser en PROLOG un analyseur pour une grammaire "context-free" donnée.

Soit par exemple la grammaire

<chaine> ::= A <chaine> B | <suite de B> <suite de A>

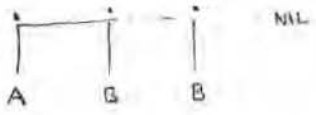
<suite de B> ::= | B <suite de B>

<suite de A> ::= | A <suite de A>

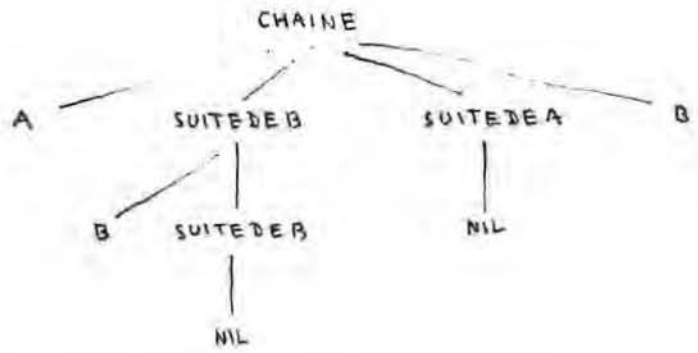
On se propose étant donné une chaîne (terminée par NIL), par exemple,

A B B NIL

qui, si l'on définit "." comme concatenateur droite-gauche, se représente par le "peigne"



d'écrire un analyseur qui donne son arbre syntaxique au sens classique du terme, c'est à dire



donc

CHAINE (A, SUITE DE B (B, SUITE DE B(NIL)), SUITE DE A(NIL),B)

Soit ANALYSE (*X,*Y) le prédicat qui signifie que l'arbre syntaxique de la chaîne *X est *Y.

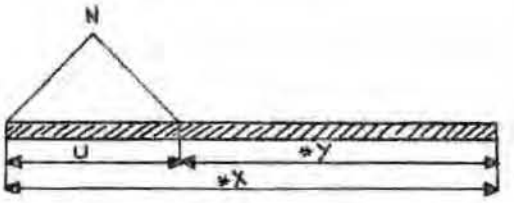
Pour chaque symbole non terminal N de la grammaire on définit un prédicat à trois arguments

N(*X,*V,*Y)

qui affirme que:

-le symbole non terminal N peut se dériver en une chaîne U
-*X = U concaténé avec *Y
-*V est l'arbre syntaxique correspondant à cette dérivation

Tout ceci peut être schématisé par



A chaque règle de la grammaire on peut alors faire correspondre systèmatiquement un axiome de l'analyseur:

```
+ANALYSE(*X,*V) -CHAINE(*X,*V,NIL) ;;

+CHAINE(A.*X1, CHAINE(A,*V,B), *X2) -CHAINE(*X1,*V,B.*X2) ;;

+CHAINE(*X1,CHAINE(*V,*W),*X3) -SUITEDEB(*X1,*V,*X2)
       -SUITEDEA(*X2,*W,*X3) ;;

+SUITEDEA(*X,NIL,*X) ;;

+SUITEDEA(A.*X1,SUITEDEA(A,*V),*X2) -SUITEDEA(*X1,*V,*X2) ;;

+SUITEDEB(*X,NIL,*X) ;;

+SUITEDEB(B.*X1,SUITEDEB(B,*V),*X2) -SUITEDEB(*X1,*V,*X2) ;;
```

Cette méthode d'analyse est donc très générale et de plus on voit tout de suite comment l'utiliser pour obtenir quelque chose de plus évolué qu'un simple arbre syntaxique.

Nous pouvons donc terminer ce chapitre en donnant l'ensemble des axiomes de l'analyseur du Français.

```
INTERFACE LU
   -=TRAIT '=APOST ,=VIRGULE .=POINT ?=PTINTERO *=ASTER AMEN

OPERATEURS LU
   CONCATENATEUR DG .. AMEN

LIRE CEKONINSERICI    +LU(    ← On insère le texte français ici
));;

AMEN

INTERFACE ANALYSEUR
   AMEN

OPERATEURS ANALYSEUR
   BINAIRE DG .. BINAIRE GD -. AMEN

LIRE ANALYSEUR

** ANALYSE SYNTAXIQUE ..

-LU(*U) -ELI(*U,*V) -CON(*V,*W) -ECRIT(LU(*W)) -PS(*W,*X) /
   +STRUCTURESYNTAXIQUE(*X) ..

**SUITE DE PHRASES ..

+PS(NIL,NIL) ..
+PS(*A,(*X.*Y).*Z) -P(*A,*X.*Y,*B) -PS(*B,*Z) -DIF(*X,REL) ;.
+PS(*A,*Z) +P(*A,*X,*B) -ECRIT(INCOMPRIS(*X)) -PS(*B,*Z) ..

** PHRASE INCORRECTE ..

-P(*X.NIL,*X,NIL) ..
-P(POINT.*A,POINT,*A) ..  -P(PTINTERO.*A,PTINTERO,*A) ..
-P(*X.*A,*X.*Y,*B) +P(*A,*Y,*B) ..

** PROPOSITION ..

+P(EST.TRAIT.CE.QUE.*A,OUINON.*X,*B) -P(*A,OUINON.*X,*B) ..
+P(*A,*T.*S.*O.*X,*E) -SPSPS(*A,*T.*U,*B)
   -GV(*B,*V.*T.*O.*G.*N.*X,*C)
   -ECRIT(GV(*V.*T.*O.*G.*N.*X))
   -SPS(*C,*W,*D) -FIN(*D,*T,*E) -3SPS(*G.*N,*U,*V,*W,*S) ..

** REGROUPEMENT DES 3 LISTES DE SYNTAGMES PREPOSITIONNELS ..

+3SPS(*R,*U1,*V,*W1,*X.*X3) -RSUJ(*R,*U1,*U2,*W1,*W2)
   -EPUR(OBJ.NIL,*Z,*U2,*U3)
   -ROBJ(*W2,*W3) -CONC(*U3.*V.*W3.NIL,*X1) -PREPSNA(*X1,*P)
   -SUPEREPUR(*P,*X1,*X.*X2) -LINEAIRE(*X2,*X3)
   -ECRIT(LINEAIRE(*X.*X3)) ..

** RECHERCHE DU SUJET ..

+RSUJ(*R,((SUJ.NIL).*X).*Y,*S.*Y,*U,*U) -SUJ(*R,*X,*S) ..
+RSUJ(*R,(*P.*X).NIL,*S.NIL,*U,*U) -DANS(SUJ,*P) -SUJ(*R,*X,*S) .;
+RSUJ(*R,*X.*Y1,*X.*Y2,*U1,*U2) -RSUJ(*R,*Y1,*Y2,*U1,*U2) ..
+RSUJ(*R,NIL,NIL,(*P.*X).*Y,*S.*Y) -DANS(SUJ,*P) -SUJ(*R,*X,*S) ..

+SUJ(*G.*N1,*T.*G.*N2.*X,(SUJ.NIL).*T.*G.*N2.*X) -ACNB(*N1,*N2) ..

+DANS(*X,*X.*Y) ..  +DANS(*U,*X.*Y) -DANS(*U,*Y) ..

+ACNB(1,1) ..  +ACNB(PLU,PLU) ..  +ACNB(PLU,*N) -DIF(*N,1) ..
```

```
** RECHERCHE EVENTUELLE DE L'OBJET ..

+ROBJ(((OBJ.SUJ.TEMPS.NIL).*X).*Y1,(*O.*X).*Y2) -RTEMPS(*O,*Y1,*Y2) ..
+ROBJ(*X.*Y1,*X.*Y2) -ROBJ(*Y1,*Y2) ..    +ROBJ(NIL,NIL) ..

+RTEMPS(OBJ.NIL,((OBJ.SUJ.TEMPS.NIL).*X).*Y,((TEMPS.NIL).*X).*Y) ..
+RTEMPS(*O,*X.*Y1,*X.*Y2) -RTEMPS(*O,*Y1,*Y2) ..
+RTEMPS(OBJ.SUJ.TEMPS.NIL,NIL,NIL) ..

** CONCATENATION DE LISTES ..

+CONC(*X.NIL,*X) ..
+CONC((*X.*Y).*Z,*X.*U) -CONC(*Y.*Z,*U) ..
+CONC(NIL.*X,*U) -CONC(*X,*U) ..

** LISTE DES PREPOSITIONS NON AMBIGUES ..

+PREPSNA(((*P.NIL).*X).*Y,*P.*U) -PREPSNA(*Y,*U) ..
+PREPSNA(*X.*Y,*U) -PREPSNA(*Y,*U) ..    +PREPSNA(NIL,NIL) ..

** EPURATION REPETEE DES PREPOSITION AMBIGUES ..

+SUPEREPUR(NIL,*X,*X) ..
+SUPEREPUR(*V1,*X1,*X3) -EPUR(*V1,*V2,*X1,*X2)
   -SUPEREPUR(*V2,*X2,*X3) ..

** EPURATION DES PREPOSITIONS AMBIGUES ..

+EPUR(*U,*V,((*P.NIL).*X).*Y1,((*P.NIL).*X).*Y2)
   -EPUR(*U,*V,*Y1,*Y2) ..
+EPUR(*U,*V1,(*P1.*X).*Y1,(*P2.*X).*Y2) -MOINS(*P1,*U,*P2)
   -TV(*P2,*V1,*V2) -EPUR(*U,*V2,*Y1,*Y2) ..

+TV(*P.NIL,*P.*V,*V) ..    +TV(*P,*V,*V) ..

+EPUR(*U,NIL,NIL,NIL) ..

** SOUSTRACTION DE DEUX ENSEMBLES ..

+MOINS(*X1,*U.*V,*X3) -MOIN(*X1,*U,*X2) -MOINS(*X2,*V,*X3) ..
+MOINS(*X,NIL,*X) ..

** SOUSTRACTION D'UN ELEMENT A UN ENSEMBLE ..

+MOIN(*X.*Y,*X,*Y) ..    +MOIN(*X.*Y1,*U,*X.*Y2) -MOIN(*Y1,*U,*Y2) ..
+MOIN(NIL,*U,NIL) ..

** CREATION DE PLUSIEURS SP POUR UNE PREPOSITION AMBIGUE ..

+LINEAIRE(((*P.NIL).*X).*Y1,((*P.NIL).*X).*Y2) -LINEAIRE(*Y1,*Y2) ..
+LINEAIRE(((*P.*R).*X).*Y1,((*P.NIL).*X).*Y2)
   -LINEAIRE((*R.*X).*Y1,*Y2) ..
+LINEAIRE(NIL,NIL) ..

** FIN DE PROPOSITION ..

+FIN(VIRGULE.*A,REL,*A) ..
+FIN(PTINTERO.*A,OUINON,*A) ..
+FIN(PTINTERO.*A,COMBIEN,*A) ..
+FIN(PTINTERO.*A,INTERO,*A) ..
+FIN(PTINTERO.*A,REL,PTINTERO.*A) ..
+FIN(POINT.*A,AFF,*A) ..
+FIN(POINT.*A,REL,POINT.*A) ..
```

```
** SUITE DE SYNTAGMES PREPOSITIONNELS AUTRE QUE SE LE LUI Y EN ETC ..

+SPSPS(*A,*T.(*X.*Y.*Z).*U,*C) -SP(*A,*X.*Y.*Z,*B) -ACCORDTYPE(*Y,*T)
   -ECRIT(SP(*X.*Y.*Z))
   -SPS(*B,*U,*C) -DIF(*Y,REL) -DIF(*T,REL) .;
+SPSPS(*A,REL.(*X.REL.*Z).*U,*C) -SP(*A,*X.REL.*Z,*B)
   -ECRIT(SP(*X.REL.*Z))
   -SPS(*B,*U,*C) ..

+SPS(*A,NIL,*A) ;.
+SPS(*A,*X.*Y,*C) -SP(*A,*X,*B) -ECRIT(SP(*X)) -SPS(*B,*Y,*C) ..

+ACCORDTYPE(INTERO,INTERO) ..  +ACCORDTYPE(COMBIEN,COMBIEN) ..
+ACCORDTYPE(*X,*Y) -DIF(*X,INTERO) -DIF(*X,COMBIEN)
   -DIF(*Y,INTERO) -DIF(*Y,COMBIEN) ..

** GROUPE VERBAL ..

+GV(*A,*S.*T.*U.*V.*W.*X.*Y,*K) -NE(*A,*U,*B)  -SE(*B,*M,*C)
   -LE(*C,*N,*D) -LUI(*D,*O,*E) -Y(*E,*P,*F) -EN(*F,*Q,*G.*H)
   -VERB(*G,*W.*X) -DAF(*X,NIL) -TIL(*H,*T.*V.*W,*I) -PAS(*I,*U,*J)
   -FINVERB(*J,*Y,*K) -DOSPS(*M.*N.*O.*P.*Q.NIL,*S) ..

+DOSPS(NIL,NIL) ..  +DOSPS(NIL.*X,*Y) -DOSPS(*X,*Y) ..
+DOSPS((*P.*T.*G.*N).*X,(*P.*T.*G.*N.NIL.NIL).*Y) -DOSPS(*X,*Y) ..

** NE PAS ..

+NE(NE.*A,*X,*A) ..          +NE(*A,*X,*A) ..
+PAS(PAS.*A,NON,*A) ..        +PAS(*A,OUI,*A) ..

** PRONOMS JUSTE AVANT LE VERBE ..

+SE(SE.*A,(OBJ.NIL).REF.*G.*N,*A) ..  +SE(*A,NIL,*A) ..

+LE(*X.*A,(OBJ.NIL).PRON.*Y,*A) -ART(*X,DEF.*Y) .; +LE(*A,NIL,*A) ..

+LUI(LUI.*A,(A.NIL).PRON.*G.1,*A) ..
+LUI(LEUR.*A,(A.NIL).PRON.*G.PLU,*A) ..
+LUI(*A,NIL,*A) ..

+Y(Y.*A,(A.DANS.EN.NIL).PRON.*G.*N,*A) .. +Y(*A,NIL,*A) ..

+EN(EN.*A,(DE.NIL).PRON.*G.*N,*A) ..  +EN(*A,NIL,*A) ..

** VERBE ..

+VERB(A,*Y) -VERBE(A,*Y) ..
+VERB(*X,*N.NIL) -CONNU(*X) .;
+VERB(*X,*Y) -VERBE(*X,*Y) ..

** -IL -ELLE -ILS -ELLES JUSTE APRES LE VERBE ..

+TIL(TRAIT.*X.*A,*T.*Y,*A) -SPPRON(*X,(SUJ.NIL).*Y) -DIF(*T,REL)
   -DIF(*T,AFF) ..
+TIL(*A,*X,*A) ..

** FIN DU VERBE ..

+FINVERB(*X.*A,NIL,*X.*A) -CONNU(*X) .;
+FINVERB(*X.*A,*U.*V,*B) -BOUM(*X,*Y) -MOINSES(*Y,*U)
   -FINVERB(*A,*V,*B) ..
```

```
+MOINSES(*X-E-S,*X) ..   +MOINSES(*X-S,*X) ..
+MOINSES(*X-E,*X) ..     +MOINSES(*X,*X) ..

** SYNTAGME PREPOSITIONNEL ..

+SP(*X.*A,(TEMPS.NIL).PROP.MAS.1.*X.NIL,*A) -ADVTEMPS(*X) .;

+SP(*X.*A,*P.PRON.*G.*N.NIL.NIL,*A) -SPPRON(*X,*P.*G.*N) .;
+SP(*X.*A,*P.INTERO.*G.*N.NIL.NIL,*A) -SPINTERO(*X,*P.*G.*N) .;
+SP(*X.*A,*P.REL.*G.*N.NIL.NIL,*A) -SPREL(*X,*P.*G.*N) .;

+SP(*X.*A,(*X.NIL).*Y,*B) -PREP(*X) -SN(*A,*Y,*B) .;
+SP(*A,(OBJ.SUJ.TEMPS.NIL).*X,*B) -SN(*A,*X,*B) ..

** SYNTAGME NOMINAL ..

+SN(*X.*A,PRON.*G.*N.NIL.NIL,*A) -SNPRON(*X,*G.*N) .;
+SN(*X.*A,REL.*G.*N.NIL.NIL,*A) -SNREL(*X,*G.*N) .;
+SN(*X.*A,INTERO.*G.*N.NIL.NIL,*A) -SNINTERO(*X,*G.*N) .;

+SN(*A,*T2.*G.*N2.*M.*R,*E) -ARTV(*A,*T1.*G.*N1,*B)
   -CARDV(*B,*T1.*N1.*N2,*C) -NOM(*C,*T1.*T2.*N2.*M,*D)
   -REL(*D,*R,*E) -DIF(*T2,REL) ..

** ABSENCE EVENTUELLE DE L'ARTICLE ..

+ARTV(TOUS.LES.*A,DEF.MAS.PLU,*A) ..
+ARTV(TOUTES.LES.*A,DEF.FEM.PLU,*A) ..
+ARTV(*X.*A,*Y,*A) -ART(*X,*Y) .;  +ARTV(*A,*T.*G.NIL,*A) ..

** ABSENCE EVENTUELLE DU CARDINAL ..

+CARDV(*X.*A,DEF.PLU.*N,*A) -CARD(*X,*N) .;
+CARDV(*X.*A,DEM.PLU.*N,*A) -CARD(*X,*N) .;
+CARDV(*X.*A,INDEF.NIL.*N,*A) -CARD(*X,*N) .;
+CARDV(*A,PROP.NIL.1,*A) ..
+CARDV(*A,*T.*N.*N,*A) ..

** NOM PROPRE ET NOM COMMUN ..

+NOM(ASTER.*X.*A,*T.PROP.*N.*X,*A) ..
+NOM(*X.*A,*T.*T.*N.*M,*A) -ACCORDNOMBRE(*I,*N)
   -NOMCOM(*X,*I,*M) -DIF(*T,PROP) ..

+ACCORDNOMBRE(1,1) ..
+ACCORDNOMBRE(PLU,*X) ..

** RELATIVE EVENTUELLEMENT ABSENTE ..

+REL(VIRGULE.*A,(APOS.*X).NIL,*B) -P(*A,REL.*X,*B) ..
+REL(*A,(REST.*X).NIL,*B) -P(*A,REL.*X,*B) .;
+REL(*A,NIL,*A) ..

** ADVERBE DE TEMPS ..

+ADVTEMPS(AUJOURDHUI) ..
+ADVTEMPS(AUSSITOT) ..
+ADVTEMPS(BIENTOT) ..
+ADVTEMPS(DEMAIN) ..
+ADVTEMPS(HIER) ..
+ADVTEMPS(TANTOT) ..
+ADVTEMPS(TARD) ..
+ADVTEMPS(TOT) ..
```

```
** SP RELATIF ..

+SPREL(AUQUEL,(A.NIL).MAS.1) ..
+SPREL(AUXQUELLES,(A.NIL).FEM.PLU) ..
+SPREL(AUXQUELS,(A.NIL).MAS.PLU) ..
+SPREL(DESQUELLES,(DE.NIL).FEM.PLU) ..
+SPREL(DESQUELS,(DE.NIL).MAS.PLU) ..
+SPREL(DUQUEL,(DE.NIL).MAS.1) ..
+SPREL(DONT,(DE.NIL).*G.*N) ..
+SPREL(OU,(A.DANS.EN.NIL).*G.*N) ..
+SPREL(QUE,(OBJ.NIL).*G.*N) ..
+SPREL(QUI,(SUJ.NIL).*G.*N) ..

** SP INTEROGATIF ..

+SPINTERO(OU,(A.APRES.AVANT.DANS.EN.SOUS.SUR.VERS.NIL).*G.*N) ..
+SPINTERO(OUESTCEQUE,*X) -SPINTERO(OU,*X) ..
+SPINTERO(QUAND,(A.APRES.AVANT.DANS.DE.DURANT.EN.PENDANT.TEMPS.VERS
.NIL).*G.*N) ..
+SPINTERO(QUANDESTCEQUE,*X) -SPINTERO(QUAND,*X) ..
+SPINTERO(QUE,(OBJ.NIL).*G.*N) ..
+SPINTERO(QUESTCEQUE,(OBJ.NIL).*G.*N) ..
+SPINTERO(QUESTCEQUI,(SUJ.NIL).*G.*N) ..
+SPINTERO(QUI,(OBJ.SUJ.NIL).*G.*N) ..
+SPINTERO(QUIESTCEQUE,(OBJ.NIL).*G.*N) ..
+SPINTERO(QUIESTCEQUI,(SUJ.NIL).*G.*N) ..

** SP PRONOM BANAL ..

+SPPRON(ELLE,(SUJ.NIL).FEM.1) ..
+SPPRON(ELLES,(SUJ.NIL).FEM.PLU) ..
+SPPRON(IL,(SUJ.NIL).MAS.1) ..
+SPPRON(ILS,(SUJ.NIL).MAS.PLU) ..

** SN RELATIF ..

+SNREL(LAQUELLE,FEM.1) ..
+SNREL(LEQUEL,MAS.1) ..
+SNREL(LESQUELS,*G.PLU) ..
+SNREL(OU,*G,*N) ..
+SNREL(QUI,*G.*N) ..


** SN INTERROGATIF ..

+SNINTERO(OU,*G.*N) ..
+SNINTERO(OUESTCEQUE,*G.*N) ..
+SNINTERO(QUI,*G.*N) ..
+SNINTERO(QUIESTCEQUE,*G.*N) ..
+SNINTERO(QUOI,*G.*N) ..
+SNINTERO(QUOIESTCEQUE,*G.*N) ..

** SN PRONOM BANAL ..

+SNPRON(ELLE,FEM.1) ..
+SNPRON(ELLES,FEM.PLU) ..
+SNPRON(EUX,MAS.PLU) ..
+SNPRON(LUI,MAS.1) ..
```

```
+ART(AUCUN,AUCUN.MAS.1) ..
+ART(AUCUNE,AUCUN.FEM.1) ..
+ART(CE,DEM.MAS.1) ..
+ART(CERTAINES,INDEF.FEM.PLU) ..
+ART(CERTAINS,INDEF.MAS.PLU) ..
+ART(CES,DEM.*G.PLU) ..
+ART(CET,DEM.MAS.1) ..
+ART(CHAQUE,CHAQUE.*G.1) ..
+ART(CETTE,DEM.FEM.1) ..
+ART(COMBIENDE,COMBIEN.*G.PLU) ..
+ART(LA,DEF.FEM.1) ..
+ART(LAPOSTROPHE,DEF.*G.1) ..
+ART(LE,DEF.MAS.1) ..
+ART(LES,DEF.*G.PLU) ..
+ART(QUEL,INTERO.MAS.1) ..
+ART(QUELLE,INTERO.FEM.1) ..
+ART(QUELLES,INTERO.*G.PLU) ..
+ART(QUELQUES,INDEF.*G.PLU) ..
+ART(PLUSIEURS,INDEF.*G.PLU) ..
+ART(TOUT,CHAQUE.MAS.1) ..
+ART(TOUTE,CHAQUE.FEM.1) ..
+ART(UN,INDEF.MAS.1) ..
+ART(UNE,INDEF.FEM.1) ..
+ART(1,INDEF.*G.*N) ..

** CARDINAL ..

+CARD(DEUX,2) ..
+CARD(TROIS,3) ..
+CARD(QUATRE,4) ..
+CARD(CINQ,5) ..
+CARD(SIX,6) ..
+CARD(SEPT,7) ..
+CARD(HUIT,8) ..
+CARD(NEUF,9) ..
+CARD(DIX,10) ..
+CARD(*X,*X) -NOMBRE(*X) -DIF(*X,0) -DIF(*X,1) ..

** PREPOSITION ..

+PREP(A) ..
+PREP(APRES) ..
+PREP(AVANT) ..
+PREP(DANS) ..
+PREP(DE) ..
+PREP(DURANT) ..
+PREP(EN) ..
+PREP(PAR) ..
+PREP(PENDANT) ..
+PREP(POUR) ..
+PREP(SOUS) ..
+PREP(SUR) ..
+PREP(VERS) ..


** NOMBRE ..

+NOMBRE(*X) -BOUM(*X,*Y) -CHIFFRES(*Y) ..

+CHIFFRES(NIL-0) +COUIC ..
+CHIFFRES(*X-*Y) -CHIFFRE(*Y) -CHIFFRES(*X) ..
+CHIFFRES(NIL) ..
```

```
** CHIFFRE ..

+CHIFFRE(0) ..
+CHIFFRE(1) ..
+CHIFFRE(2) ..
+CHIFFRE(3) ..
+CHIFFRE(4) ..
+CHIFFRE(5) ..
+CHIFFRE(6) ..
+CHIFFRE(7) ..
+CHIFFRE(8) ..
+CHIFFRE(9) ..

** MOT CONNU ..

+CONNU(ASTER) ..
+CONNU(LEUR) ..
+CONNU(NE) ..
+CONNU(NIL) ..
+CONNU(PAS) ..
+CONNU(POINT) ..
+CONNU(PTINTERO) ..
+CONNU(SE) ..
+CONNU(TOUS) ..
+CONNU(TOUTES) ..
+CONNU(TRAIT) ..
+CONNU(VIRGULE) ..
+CONNU(Y) ..

+CONNU(*X) -ADVTEMPS(*X) .;
+CONNU(*X) -SPREL(*X,*Y) .;
+CONNU(*X) -SPINTERO(*X,*Y) .;
+CONNU(*X) -SPPRON(*X,*Y) .;
+CONNU(*X) -SNREL(*X,*Y) .;
+CONNU(*X) -SNINTERO(*X,*Y) .;
+CONNU(*X) -SNPRON(*X,*Y) .;
+CONNU(*X) -ART(*X,*Y) .;
+CONNU(*X) -CARD(*X,*Y) .;
+CONNU(*X) -PREP(*X) .;

** ELISIONS ..

+ELI(D.APOST.*A,DE.*B) -ELI(*A,*B) ..
+ELI(L.APOST.*A,LAPOSTROPHE.*B) -ELI(*A,*B) ..
+ELI(N.APOST.*A,NE.*B) -ELI(*A,*B) ..
+ELI(QU.APOST.*A,QUE.*B) -ELI(*A,*B) ..
+ELI(S.APOST.*A,SE.*B) -ELI(*A,*B) ..
+ELI(*X.*A,*X.*B) -ELI(*A,*B) ..
+ELI(*X,*X.NIL) ..

** CONTRACTIONS ET DECONTRACTIONS ..

+CON(AU.*A,A.LE.*B) -CON(*A,*B) ..
+CON(AUX.*A,A.LES.*B) -CON(*A,*B) ..
+CON(DES.*A,DE.LES.*B) -CON(*A,*B) ..
+CON(DU.*A,DE.LE.*B) -CON(*A,*B) ..
```

```
+CON(AUJOURD.APOS.HUI.*A,AUJOURDHUI.*B) -CON(*A,*B) ..
+CON(COMBIEN.DE.*A,COMBIENDE.*B) -CON(*A,*B) ..
+CON(OU.EST.TRAIT.CE.QUE.*A,OUESTCEQUE.*B) -CON(*A,*B) ..
+CON(PAS.UN.*A,AUCUN.*B) -CON(*A,*B) ..
+CON(PAS.UNE.*A,AUCUNE.*B) -CON(*A,*B) ..
+CON(QUAND.EST.TRAIT.CE.QUE.*A,QUANDESTCEQUE.*B) -CON(*A,*B) ..
+CON(QUE.EST.TRAIT.CE.QUE.*A,QUESTCEQUE.*B) -CON(*A,*B) ..
+CON(QUE.EST.TRAIT.CE.QUI.*A,QUESTCEQUI.*B) -CON(*A,*B) ..
+CON(QUI.EST.TRAIT.CE.QUE.*A,QUIESTCEQUE.*B) -CON(*A,*B) ..
+CON(QUI.EST.TRAIT.CE.QUI.*A,QUIESTCEQUI.*B) -CON(*A,*B) ..
+CON(QUOI.EST.TRAIT.CE.QUE.*A,QUOIESTCEQUE.*B) -CON(*A,*B) ..
+CON(TRAIT.T.TRAIT.*A,TRAIT.*B) -CON(*A,*B) ..
+CON(UN.CERTAIN.*A,UN.*B) -CON(*A,*B) ..
+CON(UNE.CERTAINE.*A,UNE.*B) -CON(*A,*B) ..
+CON(*X.*A,*X.*B) -CON(*A,*B) ..
+CON(NIL,NIL) ..

** MORPHOLOGIE DU NOM ..

+NOMCOM(*X,*S) -BOUM(*X,*Y) -NOMECLATE(*Y,*S) .;

+NOMECLATE(*X,1.*X.*Y) -PLUR(*X,*Y) .;
+NOMECLATE(*X,PLU.*Y.*X) -PLUR(*Y,*X) ..

+PLUR(NIL-O-E-I-L,NIL-Y-E-U-X) ..
+PLUR(NIL-C-I-E-L,NIL-C-I-E-U-X) ..
+PLUR(*X-A-I-E-U-L,*X-A-I-E-U-X) ..
+PLUR(*X-A-L,*X-A-L-S) -DS(*X,NIL-B.NIL-C-A-R-N-A-V.NIL-C-E-R-E-M-O-N-
   I.NIL-C-H-A-C.NIL-C-O-R.NIL-F-E-S-T-I-V.NIL-P.NIL-R-E-C-I-T.NIL-R-E
   -G.NIL-S-A-N-T) .;
+PLUR(*X-A-I-L,*X-A-U-X) -DS(*X,NIL-B.NIL-C-O-R.NIL-E-M.NIL-S-O-U-P-I-
   R.NIL-T-R-A-V.NIL-V-A-N-T.NIL-V-I-T-R) .;
+PLUR(*X-E-A-U,*X-E-A-U-X) ..
+PLUR(*X-A-L,*X-A-U-X) ..
+PLUR(*X-A-U,*X-A-U-S) -DS(*X,NIL-L-A-N-D.NIL-S-A-R-R) .;
+PLUR(*X-O-U,*X-O-U-X) -DS(*X,NIL-B-I-J.NIL-C-H.NIL-G-E-N.NIL-H-I-B.
   NIL-J-O-U-J.NIL-P.NIL-C-A-I-L-L) .;
+PLUR(*X-O-U,*X-O-U-S) ..
+PLUR(*X-E-U,*X-E-U-S) -DS(*X,NIL-P-N.NIL-B-L) .;
+PLUR(*X-E-U,*X-E-U-X) ..
+PLUR(*X-A-S,*X-A-S) ..
+PLUR(*X-O-I-S,*X-O-I-S)-DIF(*X,NIL-L) -DIF(*X,NIL-F) -DIF(*X,NIL-A-L)
   -DIF(*X,NIL-P-A-R-R)-DIF(*X,NIL-C-H-A-R-R) -DIF(*X,NIL-D-E-S-A-R-R)
   -DIF(*X,NIL-E-F-F-R) -DIF(*X,NIL-O-C-T-R) -DIF(*X,NIL-R) ;.
+PLUR(*X-A-I-S,*X-A-I-S) -DIF(*X,NIL-B-A-L) -DIF(*X,NIL-D-E-R-L)
   -DIF(*X,NIL-D-E-L) -DIF(*X,NIL-E-S-S) -DIF(*X,NIL-E-T) ;.
+PLUR(*X-*Y-I-S,*X-*Y-I-S) -DIF(*X-*Y,NIL-A-L-I-B)
   -DIF(*X-*Y,NIL-E-T-A-B-L) -DIF(*X-*Y,NIL-F-O-U-R-M)
   -DIF(*X-*Y,NIL-O-U-B-L) -DIF(*X-*Y,NIL-P-L) -DIF(*X-*Y,NIL-R-E-P-L)
   -DIF(*X-*Y,NIL-A-M) -DIF(*X-*Y,NIL-E-N-N-E-M)
   -DIF(*X-*Y,NIL-E-P) -DIF(*X-*Y,NIL-M-A-R) -DIF(*X-*Y,NIL-P-A-R)
   -DIF(*X-*Y,NIL-A-B-R) -DIF(*X-*Y,NIL-C-R) -DIF(*X-*Y,NIL-T-R)
   -DIF(*X-*Y,NIL-S-O-U-C) -DIF(*Y,A) -DIF(*Y,O) ;.
+PLUR(*X-U-S,*X-U-S) -DIF(*X,NIL-T-R-I-B) -DIF(*X,NIL-A-C-C)
   -DIF(*X,NIL-E-C) -DIF(*X,NIL-E- L) -DIF(*X,NIL-G-L)
   -DIF(*X,NIL-D-E-T-E-N ) -DIF(*X,NIL-F-E-T) -DIF(*X,NIL-V-E-R-T)
   -DIF(*X,NIL-I-M-P-R-E-V) ;.
+PLUR(*X,*X-S) ..
+PLUR(*X-X,*X-X) ..
+PLUR(*X-Z,*X-Z) ..
```

```
** MORPHOLOGIE DU VERBE ..

+VERBE(*X,*S) -BOUM(*X,*Y) -VERBECLATE(*Y,*S) .;

+VERBECLATE(*X,PLU.*Y.*X) -COUPLE(*Y,*X) .;
+VERBECLATE(*X,1.*X.*Y) -COUPLE(*X,*Y) ..

+COUPLE(NIL-E-S-T,NIL-S-O-N-T) ..
+COUPLE(NIL-C-H-A-T-I-E,NIL-C-H-A-T-I-E-N-T) ..
+COUPLE(NIL-E-N-V-I-E,NIL-E-N-V-I-E-N-T) ..
+COUPLE(NIL-A-M-N-I-S-T-I-E,NIL-A-M-N-I-S-T-I-E-N-T) ..
+COUPLE(*X-R-O-M-P-T,*X-R-O-M-P-E-N-T) ..
+COUPLE(*X-V-I-E-N-T,*X-V-I-E-N-N-E-N-T) ..
+COUPLE(*X-T-I-E-N-T,*X-T-I-E-N-N-E-N-T) ..
+COUPLE(*X-S-O-R-T,*X-S-O-R-T-E-N-T) ..
+COUPLE(*X-D-O-R-T,*X-D-O-R-M-E-N-T) ..
+COUPLE(*X-S-E-R-T,*X-S-E-R-V-E-N-T) ..
+COUPLE(*X-C-U-I-T,*X-C-U-I-S-E-N-T) ..
+COUPLE(*X-S-U-I-T,*X-S-U-I-V-E-N-T) ..
+COUPLE(*X-D-U-I-T,*X-D-U-I-S-E-N-T) ..
+COUPLE(*X-T-R-U-I-T,*X-T-R-U-I-S-E-N-T) ..
+COUPLE(*X-N-U-I-T,*X-N-U-I-E-N-T) ..
+COUPLE(*X-V-O-I-T,*X-V-O-I-E-N-T) ..
+COUPLE(*X-F-I-T,*X-F-I-S-E-N-T) ..
+COUPLE(NIL-S-A-I-T,NIL-S-A-V-E-N-T) ..
+COUPLE(*X-A-U-T,*X-A-L-E-N-T) ..
+COUPLE(NIL-V-E-U-T,NIL-V-E-U-L-E-N-T) ..
+COUPLE(*X-P-R-E-N-D,*X-P-R-E-N-N-E-N-T) ..
+COUPLE(*X-E-T,*X-E-T-T-E-N-T) -DIF(*X,NIL-V) -DIF(*X,NIL-D-E-V)
   -DIF(*X,NIL-R-E-V) ;.
+COUPLE(*X-F-A-I-T,*X-F-O-N-T) ..
+COUPLE(*X-P-L-A-I-T,*X-P-L-A-I-S-E-N-T) ..
+COUPLE(NIL-T-A-I-T,NIL-T-A-I-S-E-N-T) ..
+COUPLE(NIL-C-R-O-I-T,NIL-C-R-O-I-E-N-T) ..
+COUPLE(*X-C-R-O-I-T,*X-C-R-O-I-S-S-E-N-T) ..
+COUPLE(*X-C-L-O-T,*X-C-L-O-S-E-N-T) ..
+COUPLE(*X-C-L-U-T,*X-C-L-U-E-N-T) ..
+COUPLE(*X-S-O-U-T,*X-S-O-L-V-E-N-T) ..
+COUPLE(*X-V-I-T,*X-V-I-V-E-N-T) -DIF(*X,A-S-S-E-R) ;.
+COUPLE(*X-D-I-T,*X-D-I-S-E-N-T) ..
+COUPLE(*X-L-I-T,*X-L-I-S-E-N-T) -DS(*X,NIL-E.NIL-R-E-E) .;
+COUPLE(*X-C-R-I-T,*X-C-R-I-V-E-N-T) ..
+COUPLE(*X-R-I-T,*X-R-I-E-N-T) -DS(*X,NIL.NIL-S-O-U) .;
+COUPLE(*X-C-O-N-N-A-I-T,*X-C-O-N-N-A-I-S-S-E-N-T) ..
+COUPLE(*X-P-A-R-A-I-T,*X-P-A-R-A-I-S-S-E-N-T) ..
+COUPLE(*X-A-T,*X-A-T-T-E-N-T) ..
+COUPLE(*X-A-I-T,*X-A-I-E-N-T) ..
+COUPLE(*X-O-I-T,*X-O-I-V-E-N-T) -DIF(*X,NIL-A-S-S) -DIF(*X,NIL-C-H)
   -DIF(*X,NIL-D-E-C-H)-DIF(*X,NIL-E-C-H) ;.
+COUPLE(*X-O-I-T,*X-O-I-E-N-T) ..
+COUPLE(*X-I-T,*X-I-S-S-E-N-T)
   -DIF(*X,NIL-L) -DIF(*X,NIL-G-L) -DIF(*X,NIL-T-A-P)
   -DIF(*X,NIL-H-E-R) -DIF(*X,NIL-T) -DIF(*X,NIL-V)
   -DIF(*X,NIL-D-E-V) ;.
+COUPLE(*X-I-E-D,*X-E-Y-E-N-T) ..
+COUPLE(*X-R-A-I-N-T,*X-R-A-I-G-N-E-N-T) ..
+COUPLE(*X-E-I-N-T,*X-E-I-G-N-E-N-T) ..
+COUPLE(*X-O-I-N-T,*X-O-I-G-N-E-N-T) ..
+COUPLE(*X-V-A-I-N-C,*X-V-A-I-N-Q-U-E-N-T) ..
+COUPLE(*X-U-T,*X-U-V-E-N-T) ..
+COUPLE(*X-C-O-U-D,*X-C-O-U-S-E-N-T) ..
+COUPLE(*X-M-O-U-D,*X-M-O-U-L-E-N-T) ..
```

```
** MORPHOLOGIE DU VERBE ..

+VERBE(*X,*S) -BOUM(*X,*Y) -VERBECLATE(*Y,*S) .;

+VERBECLATE(*X,PLU.*Y.*X) -COUPLE(*Y,*X) .;
+VERBECLATE(*X,1.*X.*Y) -COUPLE(*X,*Y) ..

+COUPLE(NIL-E-S-T,NIL-S-O-N-T) ..
+COUPLE(NIL-C-H-A-T-I-E,NIL-C-H-A-T-I-E-N-T) ..
+COUPLE(NIL-E-N-V-I-E,NIL-E-N-V-I-E-N-T) ..
+COUPLE(NIL-A-M-N-I-S-T-I-E,NIL-A-M-N-I-S-T-I-E-N-T) ..
+COUPLE(*X-R-O-M-P-T,*X-R-O-M-P-E-N-T) ..
+COUPLE(*X-V-I-E-N-T,*X-V-I-E-N-N-E-N-T) ..
+COUPLE(*X-T-I-E-N-T,*X-T-I-E-N-N-E-N-T) ..
+COUPLE(*X-S-O-R-T,*X-S-O-R-T-E-N-T) ..
+COUPLE(*X-D-O-R-T,*X-D-O-R-M-E-N-T) ..
+COUPLE(*X-S-E-R-T,*X-S-E-R-V-E-N-T) ..
+COUPLE(*X-C-U-I-T,*X-C-U-I-S-E-N-T) ..
+COUPLE(*X-S-U-I-T,*X-S-U-I-V-E-N-T) ..
+COUPLE(*X-D-U-I-T,*X-D-U-I-S-E-N-T) ..
+COUPLE(*X-T-R-U-I-T,*X-T-R-U-I-S-E-N-T) ..
+COUPLE(*X-N-U-I-T,*X-N-U-I-E-N-T) ..
+COUPLE(*X-V-O-I-T,*X-V-O-I-E-N-T) ..
+COUPLE(*X-F-I-T,*X-F-I-S-E-N-T) ..
+COUPLE(NIL-S-A-I-T,NIL-S-A-V-E-N-T) ..
+COUPLE(*X-A-U-T,*X-A-L-E-N-T) ..
+COUPLE(NIL-V-E-U-T,NIL-V-E-U-L-E-N-T) ..
+COUPLE(*X-P-R-E-N-D,*X-P-R-E-N-N-E-N-T) ..
+COUPLE(*X-E-T,*X-E-T-T-E-N-T) -DIF(*X,NIL-V) -DIF(*X,NIL-D-E-V)
   -DIF(*X,NIL-R-E-V) ;.
+COUPLE(*X-F-A-I-T,*X-F-O-N-T) ..
+COUPLE(*X-P-L-A-I-T,*X-P-L-A-I-S-E-N-T) ..
+COUPLE(NIL-T-A-I-T,NIL-T-A-I-S-E-N-T) ..
+COUPLE(NIL-C-R-O-I-T,NIL-C-R-O-I-E-N-T) ..
+COUPLE(*X-C-R-O-I-T,*X-C-R-O-I-S-S-E-N-T) ..
+COUPLE(*X-C-L-O-T,*X-C-L-O-S-E-N-T) ..
+COUPLE(*X-C-L-U-T,*X-C-L-U-E-N-T) ..
+COUPLE(*X-S-O-U-T,*X-S-O-L-V-E-N-T) ..
+COUPLE(*X-V-I-T,*X-V-I-V-E-N-T) -DIF(*X,A-S-S-E-R) ;.
+COUPLE(*X-D-I-T,*X-D-I-S-E-N-T) ..
+COUPLE(*X-L-I-T,*X-L-I-S-E-N-T) -DS(*X,NIL-E.NIL-R-E-E) .;
+COUPLE(*X-C-R-I-T,*X-C-R-I-V-E-N-T) ..
+COUPLE(*X-R-I-T,*X-R-I-E-N-T) -DS(*X,NIL.NIL-S-O-U) .;
+COUPLE(*X-C-O-N-N-A-I-T,*X-C-O-N-N-A-I-S-S-E-N-T) ..
+COUPLE(*X-P-A-R-A-I-T,*X-P-A-R-A-I-S-S-E-N-T) ..
+COUPLE(*X-A-T,*X-A-T-T-E-N-T) ..
+COUPLE(*X-A-I-T,*X-A-I-E-N-T) ..
+COUPLE(*X-O-I-T,*X-O-I-V-E-N-T) -DIF(*X,NIL-A-S-S) -DIF(*X,NIL-C-H)
   -DIF(*X,NIL-D-E-C-H)-DIF(*X,NIL-E-C-H) ;.
+COUPLE(*X-O-I-T,*X-O-I-E-N-T) ..
+COUPLE(*X-I-T,*X-I-S-S-E-N-T)
   -DIF(*X,NIL-L) -DIF(*X,NIL-G-L) -DIF(*X,NIL-T-A-P)
   -DIF(*X,NIL-H-E-R) -DIF(*X,NIL-T) -DIF(*X,NIL-V)
   -DIF(*X,NIL-D-E-V) ;.
+COUPLE(*X-I-E-D,*X-E-Y-E-N-T) ..
+COUPLE(*X-R-A-I-N-T,*X-R-A-I-G-N-E-N-T) ..
+COUPLE(*X-E-I-N-T,*X-E-I-G-N-E-N-T) ..
+COUPLE(*X-O-I-N-T,*X-O-I-G-N-E-N-T) ..
+COUPLE(*X-V-A-I-N-C,*X-V-A-I-N-Q-U-E-N-T) ..
+COUPLE(*X-U-T,*X-U-V-E-N-T) ..
+COUPLE(*X-C-O-U-D,*X-C-O-U-S-E-N-T) ..
+COUPLE(*X-M-O-U-D,*X-M-O-U-L-E-N-T) ..
```

```
+COUPLE(*X-D,*X-D-E-N-T) ..
+COUPLE(*X-Q-U-I-E-R-T,*X-Q-U-I-E-R-E-N-T) ..
+COUPLE(*X-C-O-U-R-T,*X-C-O-U-R-E-N-T) ..
+COUPLE(NIL-M-E-U-R-T,NIL-M-E-U-R-E-N-T) ..
+COUPLE(*X-*Y-*Z-E,*X-*Y-*Z-E-N-T) -DIF(*X-*Y-*Z,NIL-C-O-N-S)
   -DIF(*X-*Y-*Z,NIL-P-R-E-S-S) -DIF(*X-*Y-*Z,NIL-R-E-S-S)
   -DIF(*X-*Y-*Z,NIL-M) -DIF(*X-*Y-*Z,NIL-D-E-M)
   -DIF(*Y-*Z,V-I) -DIF(*Y-*Z,T-I) ;.
+COUPLE(*X-T,*X-T-E-N-T) ..
+COUPLE(*X-A,*X-O-N-T) ..

** PREDICAT DS ..

+DS(*X,*X) ..
+DS(*X,*X.*Y) ..
+DS(*X,*Z.*Y) -DS(*X,*Y) ..

AMEN

DEMONTRER(ANALYSEUR,CEKONINSERICI,STRUCTURESYNTAXIQUE)

ECRIRE(STRUCTURESYNTAXIQUE)

AMEN
```

LE SEMANTISEUR

Le semantiseur s'occupe surtout de résoudre les problèmes de référence posés par les pronoms. Ces références peuvent aussi bien se faire à l'intérieur d'une même phrase que d'une phrase à une autre. Le sémantiseur a aussi pour rôle de fractionner le texte en une suite de paragraphes indépendants (chaque paragraphe est une suite de phrases) c'est à dire qu'aucun paragraphe ne contient un pronom référent à quelquechose de contenu dans un autre paragraphe.

Voici la forme de la structure sémantique produite par le sémantiqeur.

Structure sémantique

<Structure Semantique> ::= STRUCTURESEMANTIQUE (<Texte>)

<Texte> ::= NIL | <Paragraphe> . <Texte >

<paragraphe> ::= NIL | <phrase> . <paragraphe>

<phrase> ::= P (<type phrase> , <liste synt.prep> , <Groupe verbal>)

<type phrase> ::= AFF | INTERO | COMBIEN | OUINON | REST | APOS

<liste synt.prep> ::= NIL | <synt.prep> . <liste synt.prep>

<synt.prep> ::= SP (<liste prep> , <synt.nominal>)

<liste prep> ::= NIL | <prep.>.<liste prep.>

<prep> ::= SUJ | OBJ | TEMPS | A | POUR | SUR ....

<synt.nominal> ::= SN (<nom Prop.ou indice> , <quantif> , <paragraphe>)

<nom Prop. ou indice> ::= *I |* <nom propre>

<quantif> ::= DEF | INDEF | CHAQUE | AUCUN | PROP | INTERO | COMBIEN | PRON

<Groupe verbal> ::= <oui> ((<verbe S> . <verbe P>) . <fin de verbe>))
NOM (<nom commun S> . <nom commun P>) CARD (<cardinal>)

<oui> ::= OUI | NON

<cardinal> ::= PLU | 1 | 2 | 3 .....

<nom propre> ::= "tout nom propre"

<nom commun S> ::= "tout nom commun Singulier éclaté par boum"

<nom commun P> ::= "tout nom commun pluriel éclaté par boum"

<verbe S> ::= "tout verbe, à la 3ème personne du singulier d'un temps simple,
éclaté par boum"

<verbe P> ::= "tout verbe, à la 3ème personne du pluriel d'un temps simple,
éclaté par boum"

<fin de verbe> ::= NIL | <mot inconnu> . <fin de verbe>

<mot inconnu> ::= "tout adjectif, participe, adverbe éclaté par le prédicat boum
et auquel on a éventuellement enlevé le e et le s final"

Si l'on reprend l'exemple le sémantiseur transforme

+STRUCTURESYNTAXIQUE((AFF.(((SUJ.NIL).CHAQUE.MAS.1.(NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S).NIL).((OBJ.NIL).INDEF.FEM.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).NIL).((TEMPS.NIL).INDEF.FEM.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL).(AFF.(((SUJ.NIL).CHAQUE.*X.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).(REST.(((OBJ.NIL).REL.*Y.*Z.NIL.NIL).((SUJ.NIL).PRON.MAS.1.NIL.NIL).NIL).OUI.(NIL-A-N-A-L-Y-S-E.NIL-A-N-A-L-Y-S-E-N-T).NIL).NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL-M-A-L-A-D.NIL).(AFF.(((SUJ.NIL).PROP.*T.1.JACQUES.NIL).((OBJ.NIL).INDEF.MAS.1.(NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S).NIL).((TEMPS.NIL).INDEF.MAS.1.(NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S).NIL).((A.NIL).PROP.*U.1.MARSEILLE.NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL).(OUINON.(((SUJ.NIL).PROP.*V.1.JACQUES.NIL).((OBJ.NIL).INDEF.FEM.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).NIL).((TEMPS.NIL).INDEF.FEM.1.(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S).NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL).(INTERO.(((A.APRES.AVANT.DANS.EN.SOUS.SUR.VERS.NIL).INTERO.*W.*X1.NIL.NIL).((SUJ.NIL).PROP.*Y1.1.JACQUES.NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL).(OUINON.(((SUJ.NIL).PROP.*Z1.1.JACQUES.NIL).NIL).OUI.(NIL-E-S-T.NIL-S-O-N-T).NIL-M-A-L-A-D.NIL).NIL) ;.

en

+STRUCTURESEMANTIQUE((P(AFF,SP(SUJ.NIL,SN(*X,CHAQUE,P(APOS,SP(SUJ.NIL,SN(*X,PRON,NIL)).NIL,CARD(1)).P(REST,SP(SUJ.NIL,SN(*X,PRON,NIL)).NIL,NOM(NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S)).NIL)).SP(OBJ.NIL,SN(*Y,INDEF,P(APOS,SP(SUJ.NIL,SN(*Y,PRON,NIL)).NIL,CARD(1)).P(REST,SP(SUJ.NIL,SN(*Y,PRON,NIL)).NIL,NOM(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S)).NIL)).SP(TEMPS.NIL,SN(*Z,INDEF,P(APOS,SP(SUJ.NIL,SN(*Z,PRON,NIL)).NIL,CARD(1)).P(REST,SP(SUJ.NIL,SN(*Z,PRON,NIL)).NIL,NOM(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S)).NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL)).P(AFF,SP(SUJ.NIL,SN(*T,CHAQUE,P(APOS,SP(SUJ.NIL,SN(*T,PRON,NIL)).NIL,CARD(1)).P(REST,SP(SUJ.NIL,SN(*T,PRON,NIL)).NIL,NOM(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S)).P(REST,SP(OBJ.NIL,SN(*T,PRON,NIL)).SP(SUJ.NIL,SN(*X,PRON,NIL)).NIL,OUI((NIL-A-N-A-L-Y-S-E.NIL-A-N-A-L-Y-S-E-N-T).NIL)).NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL-M-A-L-A-D.NIL)).NIL).(P(AFF,SP(SUJ.NIL,SN(JACQUES,PROP,P(APOS,SP(SUJ.NIL,SN(JACQUES,PRON,NIL)).NIL,CARD(1)).NIL)).SP(OBJ.NIL,SN(*U,INDEF,P(APOS,SP(SUJ.NIL,SN(*U,PRON,NIL)).NIL,CARD(1)).P(REST,SP(SUJ.NIL,SN(*U,PRON,NIL)).NIL,NOM(NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S)).NIL)).SP(TEMPS.NIL,SN(*V,INDEF,P(APOS,SP(SUJ.NIL,SN(*V,PRON,NIL)).NIL,CARD(1)).P(REST,SP(SUJ.NIL,SN(*V,PRON,NIL)).NIL,NOM(NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S)).NIL)).SP(A.NIL,SN(MARSEILLE,PROP,P(APOS,SP(SUJ.NIL,SN(MARSEILLE,PRON,NIL)).NIL,CARD(1)).NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL)).NIL).(P(OUINON,SP(SUJ.NIL,SN(JACQUES,PROP,P(APOS,SP(SUJ.NIL,SN(JACQUES,PRON,NIL)).NIL,CARD(1)).NIL)).SP(OBJ.NIL,SN(*W,INDEF,P(APOS,SP(SUJ.NIL,SN(*W,PRON,NIL)).NIL,CARD(1)).P(REST,SP(SUJ.NIL,SN(*W,PRON,NIL)).NIL,NOM(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S)).NIL)).SP(TEMPS.NIL,SN(*X1,INDEF,P(APOS,SP(SUJ.NIL,SN(*X1,PRON,NIL)).NIL,CARD(1)).P(REST,SP(SUJ.NIL,SN(*X1,PRON,NIL)).NIL,NOM(NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S)).NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL)).NIL).(P(INTERO,SP(A.APRES.AVANT.DANS.EN.SOUS.SUR.VERS.NIL,SN(*Y1,INTERO,NIL)).SP(SUJ.NIL,SN(JACQUES,PROP,P(APOS,SP(SUJ.NIL,SN(JACQUES,PRON,NIL)).NIL,CARD(1)).NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL)).NIL).(P(OUINON,SP(SUJ.NIL,SN(JACQUES,PROP,P(APOS,SP(SUJ.NIL,SN(JACQUES,PRON,NIL)).NIL,CARD(1)).NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL-M-A-L-A-D.NIL)).NIL).NIL) ;.

Voici le sémantiseur proprement dit:

```
INTERFACE SEMANTISEUR
AMEN

OPERATEURS SEMANTISEUR
   BINAIRE DG .. BINAIRE GD -. AMEN

LIRE SEMANTISEUR

** CALCUL DE LA STRUCTURE SEMANTIQUE ..

-STRUCTURESYNTAXIQUE(*U) -TPS(NIL,*X,*U,*V) -TEXTE(*V,*W)
   / +STRUCTURESEMANTIQUE(*W) ..

** TEXTE ..

+TEXTE((*O.*X).*A,(*X.*Y).*Z) -PS(*A,*Y,*B) -TEXTE(*B,*Z) ..
+TEXTE(NIL,NIL) ..

** SUITE DE PHRASES ..

+PS((MILIEU.*X).*A,*X.*Y,*B) -PS(*A,*Y,*B) ..
+PS(*X.*A,NIL,*X.*A) ..  +PS(NIL,NIL,NIL) ..

** TRANSFORMATION D'UNE SUITE DE PHRASES ..

+TPS(*A1,*A4,*X1.*Y1,(*O.*X2).*Y2)
   -TP(DEBUT.(DIESE.*A1),*O.*A2,*X1,*X2)
   -ECRIT(PRI(*O.*X2))
   -INI(*A2,*A3,*O) -TPS(*A3,*A4,*Y1,*Y2) ..
+TPS(*A,*A,NIL,NIL) ..

+INI(*A,*A,MILIEU) ..
+INI(*A,*B,*X) -DEBUT(*A,*B) ..

+DEBUT(DIESE.*X,NIL) ..
+DEBUT(*X.*Y1,*X.*Y2) -DEBUT(*Y1,*Y2) ..

** TRANSFORMATION D'UNE PHRASE ..

+TP(*A1,*A2,*T.*S1.*V1,P(*T,*S2,*V2))
   -TSPS(NIL.*A1,*B.*A2,*S1,*S2)
   -TV(*V1,*V2) ..

** TRANSFORMATION D'UNE SUITE DE SP ..

+TSPS(*A1,*A3,*X1.*Y1,*X2.*Y2) -TSP(*A1,*A2,*X1,*X2)
   -ECRIT(SP(*X2))
   -TSPS(*A2,*A3,*Y1,*Y2) ..
+TSPS(*A,*A,NIL,NIL) ..
```

```
** TRANSFORMATION D'UN SP ..

+TSP(*B1.*A1,*B2.*A3,*P.*T1.*G.*C.*N.*R,SP(*P,SN(*I,*T2,*X)))
   -INDICE(*B1.*A1,*B2.*A2,*P.*T1.*G.*C.*N,*I) -TTYPE(*T1,*T2)
   -ECRIT(INDICE(*I,*B2.*A2))
   -TC(*T2.*N,*I,*C,*X1) -TN(*T2,*I,*N,*X2) -TRS(*A2,*A3,*R,*X3)
   -DOPS(*X1.*X2.*X3,*X) ..

+DOPS(NIL.*X1,*X2) -DOPS(*X1,*X2) ..
+DOPS(*X.*Y1,*X.*Y2) -DOPS(*Y1,*Y2) ..
+DOPS(*X,*X) ..

** TRANSFORMATION D'UNE SUITE DE RELATIVES ..

+TRS(*A1,*A3,*X1.*Y2,*X2.*Y2) -TP(*A1,*A2,*X1,*X2)
   -ECRIT(REL(*X2))
   -TRS(*A2,*A3,*Y1,*Y2) ..
+TRS(*A,*A,NIL,NIL) ..

** CALCUL DE L'INDICE ..

+INDICE(*X.*Y,*X.*Y,*P.REF.*U,*J) -INDICESUJ(*X,*J) ..

+INDICE(*X.*O.*Y,((*P.*N).*X).*O.(*P.*N.*G.*C.*N).*Y,
   *P.PROP.*G.*C.*N,*N) ..

+INDICE(*X.*O.*Y,((*P.*I).*X).*O.(*P.*I.*U).*Y,*P.*T.*U,*I)
   -DAF(*T,REL) -DAF(*T,PRON) -DAF(*T,DEM) .;

+INDICE(*X.*O1.*Y,((*P.*I).*X).*O2.*Y,*P.*T.*U,*I)
   -DANS(*O1,*O2,*I.*U,*Y) -HORS(*I,*X) ..

** RECHERCHE DE L'INDICE DU SUJET ..

+INDICESUJ(((SUJ.NIL).*I).*Y,*I) ..
+INDICESUJ(*X.*Y,*I) -INDICESUJ(*Y,*I) ..

** RECHERCHE DE L'ANTECEDANT ..

+DANS(*O,MILIEU,*X,DIESE.*Y) -DANS(MILIEU,MILIEU,*X,*Y) ..
+DANS(*O,*O,*I.*G.*C1.*N1,(*P.*I.*G.*C2.*N2).*X) -ACNB(*C1,*C2)
   -ACNOM(*N1,*N2) -DAF(*P,TEMPS.NIL) ;.
+DANS(*O1,*O2,*X,*Y.*Z) -DANS(*O1,*O2,*X,*Z) ..
+DANS(*O,*O,*I.*X,NIL) ..

+HORS(*I,(*P.*J).*X) -DAF(*I,*J) -HORS(*I,*X) .. +HORS(*I,NIL) ..

+ACNB(*X,*X) .. +ACNB(*X,*Y) -DIF(*X,1) -DIF(*Y,1) ..
+ACNOM(*X,*X) .. +ACNOM(NIL,*X) ..
```

```
** TRANSFORMATION DU TYPE ..

+TTYPE(REL,PRON) ..  +TTYPE(DEM,PRON) ..
+TTYPE(REF,PRON) ..  +TTYPE(*T,*T) ..

** TRANSFORMATION DU CARDINAL ..

+TC(COMBIEN.*N,*I,*C,NIL) ..
+TC(PRON.*N,*I,*C,NIL) .. +TC(INTERO.NIL,*I,*C,NIL) ..
+TC(PROP.*N,*I,PLU,*X) -TC(PROP.*N,*I,1,*X) ..
+TC(*T,*I,*C,P(APOS,SP(SUJ.NIL,SN(*I,PRON,NIL)).NIL,CARD(*C))) ..

** TRANSFORMATION DU NOM ..

+TN(PROP,*I,*N,NIL) ..  +TN(PRON,*I,*N,NIL) ..
+TN(*T,*I,NIL,NIL) ..
+TN(*T,*I,*N,P(REST,SP(SUJ.NIL,SN(*I,PRON,NIL)).NIL,NOM(*N))) ..

** TRANSFORMATION DU VERBE ..

+TV(OUI.*X,OUI(*X)) .. +TV(NON.*X,NON(*X)) ..

AMEN


LIRE CEKONINSERICI
```

← on insère ici les données du sémantiseur

```
AMEN

DEMONTRER(SEMANTISEUR,CEKONINSERICI,BOB)

ECRIRE(BOB)

AMEN
```

SYNTHETISEUR D'ENONCES LOGIQUES

L'objet de cette phase est de transformer la structure sémantique en un ensemble de formules logiques.

Nous avons donné déjà la syntaxe de la structure sémantique.

Les règles suivantes définissent la syntaxe des énoncés logiques

Structure des énoncés logiques

<énoncé logique> ::= <clause> <énoncé logique> | <clause>

<clause> ::= <littéral> <clause> | <littéral>

<littéral> ::= <signe> <Prédicat>

<signe> ::= + | -

<Prédicat> ::= <P> | <Q> | <EG> | <ETRE> | <DS> | <R> | <CARD> | <NQ>

<P> ::= P (<liste de SP> , <verbe> , <nombre>)

<Q> ::= Q (<terme> , <terme> , <nombre>)

<EG> ::= EG (<terme> , <terme> , <nombre>)

<ETRE> ::= ETRE (<terme> , <nom commun S> , <nom commun P>)

<DS> ::= DS (<prep> , <liste de prep>)

<R> ::= R (<indice> , <réponse>)

<CARD> ::= CARD (<terme> , <cardinalité>)

<NQ> ::= NQ (<indice>)

<liste de SP> ::= *S | <SP> . <liste de SP> | NIL

<SP> ::= *S | SP (<prep> , <terme>)

<prep> ::= *P | OBJ | SUJ | TEMPS | A | POUR | .....

<terme> ::= *T | <nom propre> | F (<indice> , <liste de termes >)

<verbe> ::= <oui> ((<verbe S> . <verbe P>) . <fin de verbe>)

<oui> ::= OUI | NON

<liste de prep> ::= <prep> . <liste de prep> | NIL

<indice> ::= <indice> ' | 0

<réponse> ::= OUI | NON | SP (<prep>,*R) | SP (<prep> , CARD (*I))

<cardinalité> ::= E (<nombre>) | S (<nombre>

<nombre> ::= *N | 1 | 2 | 3 | .....

<liste de termes> ::= <terme> . <liste de termes> | NIL

Si nous prenons la structure sémantique de l'exemple présenté dans le sémantiseur, le synthétiseur d'énoncés logiques la transforme en un ensemble de clauses suivant:

```
+P(SP(SUJ,*X).SP(TEMPS,F(O',*X.NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).N
IL),*Y)-ETRE(*X,NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S)-CAR
D(*X,E(1)) ;.

+P(SP(SUJ,*X).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL-M-A-L-A-D.NIL),*Y)-P
(SP(OBJ,*X).SP(SUJ,*Z).NIL,OUI((NIL-A-N-A-L-Y-S-E.NIL-A-N-A-L-Y-S-E-N-
T).NIL),*T)-ETRE(*X,NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S)-CARD(*X
,E(1))-ETRE(*Z,NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S)-CARD
(*Z,E(1)) ;.

+P(SP(SUJ,JACQUES).SP(TEMPS,F(O''',NIL)).SP(A,MARSEILLE).NIL,OUI((NIL-
E-S-T.NIL-S-O-N-T).NIL),*X) ;.

+CARD(F(O,*X.NIL),E(1))-ETRE(*X,NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-
I-A-T-R-E-S)-CARD(*X,E(1)) ;.

+CARD(F(O',*X.NIL),E(1))-ETRE(*X,NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H
-I-A-T-R-E-S)-CARD(*X,E(1)) ;.

+CARD(JACQUES,E(1)) ;.

+CARD(F(O'',NIL),E(1)) ;.

+CARD(F(O''',NIL),E(1)) ;.

+CARD(MARSEILLE,E(1)) ;.

+R(O',OUI)-P(SP(SUJ,JACQUES).SP(TEMPS,*X).NIL,OUI((NIL-E-S-T.NIL-S-O-N
-T).NIL),*Y)-ETRE(*X,NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S)-CARD(*
X,E(1))-EG(JACQUES,*Z,*T)-ETRE(*Z,NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-
N-E-S)-CARD(*Z,E(1))-CARD(JACQUES,E(1))-DIF(JACQUES,*Z) ;.

+R(O',NON)+CARD(JACQUES,E(1)) ;.

+R(O',NON)+CARD(F(O'''',NIL),E(1)) ;.

+R(O',NON)+ETRE(F(O'''',NIL),NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S
) ;.

+R(O',NON)+EG(JACQUES,F(O'''',NIL),*X) ;.

+R(O',NON)+CARD(F(O'''''',NIL),E(1)) ;.

+R(O',NON)+ETRE(F(O'''''',NIL),NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-
S) ;.
```

```
+R(O',NON)+P(SP(SUJ,JACQUES).SP(TEMPS,F(O'''',NIL)).NIL,OUI((NIL-E-S-
T.NIL-S-O-N-T).NIL),*X) ;.

+R(O'',SP(*X,*Y))-DS(*X,A.APRES.AVANT.DANS.EN.SOUS.SUR.VERS.NIL)-P(SP(
*X,*Y).SP(SUJ,JACQUES).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL),*Z)-CARD(J
ACQUES,E(1)) ;.


+R(O''',OUI)-P(SP(SUJ,JACQUES).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL-M-A
-L-A-D.NIL),*X)-CARD(JACQUES,E(1)) ;.

+R(O''',NON)+CARD(JACQUES,E(1)) ;.

+R(O''',NON)+P(SP(SUJ,JACQUES).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL-M-A
-L-A-D.NIL),*X) ;.

+NQ(O'''') ;.

+ETRE(F(O,*X.NIL),NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S)-ETRE(*X,N
IL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S)-CARD(*X,E(1)) ;.

+ETRE(F(O',*X.NIL),NIL-P-E-R-S-O-N-N-E.NIL-P-E-R-S-O-N-N-E-S)-ETRE(*X,
NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S)-CARD(*X,E(1)) ;.

+ETRE(F(O'',NIL),NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S) ;.

+ETRE(F(O''',NIL),NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I-A-T-R-E-S) ;
.

+EG(*X,F(O,*X.NIL),*Y)-ETRE(*X,NIL-P-S-Y-C-H-I-A-T-R-E.NIL-P-S-Y-C-H-I
-A-T-R-E-S)-CARD(*X,E(1))-DIF(*X,F(O,*X.NIL)) ;.

+EG(JACQUES,F(O'',NIL),*X) ;.
```

Voyons comment nous pouvons représenter une phrase par une formule logique d'une manière systématique:
Prenons par exemple les phrases du texte:

Tout psychiatre est une personne.

Chaque personne qu'il analyse est malade.

qui correspondent en fait à un paragraphe.

Nous allons faire les transformations en balayant le paragraphe de gauche à droite et en appliquant des règles générales de transformation sur les syntagmes nominaux rencontrés, règles qui sont décrites dans le programme présenté plus loin.

Nous supposons que le connecteur entre les phrases est "et". Nous allons donc commencer par faire des transformations sur Tout psychiatre qui est quantifié par CHAQUE, et qui donne:

$\forall x$(psychiatre(x) $\supset$ x est une personne
et chaque personne que x analyse est malade)

Nous passons alors au prochain SN:

$\forall x$(psychiatre(x) $\supset$ $\exists$ y(personne(y) et etre(x,y)
et chaque personne que x analyse est malade))

et ainsi de proche en proche:

$\forall x$(psychiatre(x) $\supset$ $\exists$ y(personne(y) et etre(x,y)
et $\forall$ z((personne(z)
et analyse(x,z)) $\supset$ estmalade(z))))

ce qui peut s'écrire:

$\forall x \exists y \forall z$(psychiatre(x) $\supset$ personne(y) et etre(x,y)
et (personne(z)
et analyse(x,z)) $\supset$ estmalade(z))

Ce qui en représentation clausale donne:

-psychiatre(x) + personne(F(x))

-psychiatre(x) + etre(x,F(x))

-psychiatre(x) -personne(z) -analyse(x,z) * est malade(z)

Examinons maintenant le cas des questions.
Prenons l'exemple:

Ex Est ce que Jacques est une personne?

ceci est interprété par

(Jacques est une personne) ⊃ R(oui)

et

¬(Jacques est une personne) ⊃ R(non)

Ce qui ramène le travail à des phrases affirmatives et donne donc pour la première:

(∃x etre(Jacques,x) et personne(x)) ⊃ R(oui)

ce qui s'écrit aussi

∀ x (etre(Jacques,x) et personne(x)) ⊃ R(oui)

et donne en représentation clausale:
-etre(Jacques,x)-personne(x)+R(oui)

pour la deuxième phrase nous obtiendrons de la même façon:

+ etre(Jacques,A) +R(non)

+ personne(A) +R(non)

Prenons maintenant la question

où est Jacques?

Elle est interprétée par:

(Jacques est à "un endroit") ⊃ R(cet endroit)

ce qui se traduit donc par:

(∃x etre-a(Jacques,x)) ⊃ R(x)

soit

∀x etre-a(Jacques,x) ⊃ R(x)

ce qui donne donc:

etre-a(Jacques,x) +R(x)

N.B. R(x) signifie la réponse est x.

Nous avons, pour simplifier, laissé de côté le traitement de certaines "informations", par exemple les informations sur la cardinalité des "êtres", et certains traitements spéciaux comme celui du verbe être, mais nous avons surtout essayé de montrer comment en fait nous interprétons un texte, dégageant les mécanismes généraux de transformation.

Nous présentons dans la suite le programme écrit en PROLOG qui réalise ces transformations et génère les énoncés logiques correspondant.

```
INTERFACE ZOZO AMEN
OPERATEURS ZOZO
UNAIRE DG NO.
UNAIRE GD '.
BINAIRE DG /.
BINAIRE DG IMP,.  .
BINAIRE GD -.
AMEN

LIRE CEKONINSERICI
AMEN                    ← on insère ici les données

LIRE QUANT

** TRANSFORMATION DE TEXTE ..

-STRUCTURESEMANTIQUE(*X) -TTEXTE(0/*I,0/*J,*X,*Y) +ONA(NO(*I').*Y) ..
+TTEXTE(*I/*II,*K/*K1,*X.*Y,*XX.*YY)
    -TPARAGRAPHE(*I/*I2,*K/*K2,0/NIL,*X/*XX,NIL/*Z)
    -TTEXTE(*I2/*II,*K2/*K1,*Y,*YY) ..
+TTEXTE(*I/*I,*K/*K,NIL,NIL) ..

** TRANSFORMATION DE PARAGRAPHE ..

+TPARAGRAPHE(*I/*I,*J/*J,*X,NIL/NIL,*K/*K) ..
+TPARAGRAPHE(*I/*II,*J/*JJ,0/*X,P(OUINON,*L,*V).*Z/(*Y IMP K(*I',OUI))
    .((NO *Y1) IMP K(*I',NON)).*Z1,*K/*KK)
      -COPIE(*K.P(OUINON,*L,*V),*K.P(OUINON,*LL,*V))
      -TPHRASE1(0/0,*J/*J1,1/*X,*L,P(AFF,*L,*V).NIL/*Y,*K/*K1)
      -TPHRASE1(0/0,*J1/*J2,0/*X,*LL,P(AFF,*LL,*V).NIL/*Y1,*K1/*K2)
      -TPARAGRAPHE(*I'/*II,*J2/*JJ,0/*X,*Z/*Z1,*K2/*KK) ..

+TPARAGRAPHE(*I/*II,*J/*JJ,0/*X,P(INTERO,*L,*V).*Z/*w1.*Z1,*K/*KK)
    -TPHRASE1(*I/*I',*J/*J1,0/*X,*L,P(AFF,*L,*V).NIL/*w1,*K/*K1)
    -TPARAGRAPHE(*I'/*II,*J1/*JJ,0/*X,*Z/*Z1,*K1/*KK) ..

+TPARAGRAPHE(*I/*II,*J/*JJ,0/*X,P(COMBIEN,*L,*V).*Z/*w1.*Z1,*K/*KK)
    -TPHRASE1(*I/*I',*J/*J1,0/*X,*L,P(AFF,*L,*V).NIL/*w1,*K/*K1)
    -TPARAGRAPHE(*I'/*II,*J1/*JJ,0/*X,*Z/*Z1,*K1/*KK) ..

+TPARAGRAPHE(*I,*J,*X,P(*Y,*L,*V).*Z/*w,*K)
    -TPHRASE1(*I,*J,*X,*L,P(*Y,*L,*V).*Z/*w,*K) ..


** TRANSFORMATION DE PHRASE DONT LE VERBE EST ETRE ..

+TPHRASE1(*I,*J,*X,*L,P(*Q,*L,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL))
   .*Z/*w,*K)
   -TSP(NIL,*L,P(*Q,*L,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL)),*N)
   -CONC(*N,*Z,*M)
   -TPARAGRAPHE(*I,*J,*X,*M/*w,*K) .;
```

```
+TPHRASE1(*I,*J,*X,*L,P(*Q,*L,NON((NIL-E-S-T.NIL-S-O-N-T).NIL))
   .*Z/*W,*K)
   -TSP(NIL,*L,P(*Q,*L,NON((NIL-E-S-I.NIL-S-O-N-T).NIL)),*N)
   -CONC(*N,*Z,*M)
   -TPARAGRAPHE(*I,*J,*X,*M/*W,*K) .;

+TPHRASE1(*I,*J,*X,*L,*P,*K) -TPHRASE(*I,*J,*X,*L,*P,*K) ..

+TSP(*X,SP(*Y.NIL,*U).*S,*P,*N)  -DS(*Y,OBJ.SUJ.NIL)
   -TSP(SP(*Y.NIL,*U).*X,*S,*P,*N) .;
+TSP(*X,*Y.*S,*P,*N) -TSP(*X,*S,*P,*N) ..
+TSP(*X.*Y.NIL,NIL,*P,*N) -TSPF(*X.*Y.NIL,*P,*N) ..

+TSPF(*X.*Y.NIL,P(*Q,*Y.*X.NIL,*V),P(*Q,*Y.*X.NIL,*V1))
   -VERB(*V,*V1) ..
+TSPF(*X.*Y.NIL,P(*Q,*L,*V),P(*Q,*Y.*X.NIL,*V1).P(*Q,*LL,*V))
   -VERB(*V,*V1) -OS(*L,*LL) ..

** TRANSFORMATION DE PHRASE ..

** REGLES TERMINALES ..

+TPHRASE(*I,*J,*X,NIL,*U.*V/*U1.*VV,*K)    -REPRE(*U,*U1)
   -TPARAGRAPHE(*I,*J,*X,*V/*VV,*K) ..

** SN PRON ..

+TPHRASE(*I,*J/*JJ,*X,SP(*Z,SN(*A,PRON,*U)).*V,*W/*U1.*W1,*K/*KK)
   -TPARAGRAPHE(0/0,*J/*J1,*X,*U/*U1,*K/*K1)
   -TPHRASE(*I,*J1/*JJ,*X,*V,*W/*W1,*K1/*KK) ..

** SN PROPRE ..

+TPHRASE(*I,*J/*JJ,*X,SP(*Z,SN(*A,PROP,*U)).*V,*W/*U1.*W1,*K/*KK)
   -TPARAGRAPHE(0/0,*J/*J1,*X,*U/*U1,*K/*K1)
   -TPHRASE(*I,*J1/*JJ,*X,*V,*W/*W1,*K1/*KK) ..

** SN INDEF ..

+TPHRASE(*I,*J/*JJ,*X,SP(*Z,SN(*A,INDEF,*U)).*V,*W/*U1.*W1,*K/*KK)
    -DEF(*X,*J,*K,*A,*J1,*K1,*B,*C)
    -TPARAGRAPHE(0/0,*J1/*J2,*B,*U/*U1,*K1/*K2)
    -TPHRASE(*I,*J2/*JJ,*B,*V,*W/*W1,*K2/*KK) ..

** SN DEF ..

+TPHRASE(*I,*J/*JJ,*X,SP(*Z,SN(*A,DEF,*U)).*V,*W/*U1.*S1.*W1,*K/*KK)
     -DIV(*U,*REST1,*APOS1) -COPIE(*K.SN(*A,CHAQUE,*REST1),*K.*SN)
     -DEF(*X,*J,*K,*A,*J1,*K1,*B,*C)
     -TPARAGRAPHE(0/0,*J1/*J2,*B,*U/*U1,*K1/*K2)
     -TPARAGRAPHE(0/0,*J2/*J3,*B,P(AFF,SP(SUJ.NIL,*SN).SP(OBJ.
     NIL,SN(*A,PRON,NIL)).NIL,OUI(DANS)).NIL/*S1,*K2/*K3)
     -TPHRASE(*I,*J3/*JJ,*B,*V,*W/*W1,*K3/*KK) ..
```

```
** SN CHAQUE ..

+TPHRASE(*I,*J/*JJ,*X,SP(*Z,SN(*A,CHAQUE,*U)).*V,*W/*REST IMP *APOS.
         *W1,*K/*KK)
    -CHAQ(*X,*J,*K,*A,*J1,*K1,*B,*C)
    -SCARD(*U,*UU)
    -DIV(*UU,*REST1,*APOS1)
    -TPARAGRAPHE(0/0,*J1/*J2,*C,*REST1/*REST,*K1/*K2)
    -TPARAGRAPHE(0/0,*J2/*J3,*B,*APOS1/*APOS,*K2/*K3)
    -TPHRASE(*I,*J3/*JJ,*B,*V,*W/*W1,*K3/*KK) ..

** SN AUCUN ..

+TPHRASE(*I/*I1,*J/*JJ,*X,SP(*Z,SN(*A,AUCUN,*U)).*V,*W.*WW/*REST
    IMP *APOS.(NO *W1).*WW1,*K/*KK)
   -CHAQ(*X,*J,*K,*A,*J1,*K1,*B,*C)
   -SCARD(*U,*UU)
   -DIV(*UU,*REST1,*APOS1)
   -TPARAGRAPHE(0/0,*J1/*J2,*C,*REST1/*REST,*K1/*K2)
   -TPARAGRAPHE(0/0,*J2/*J3,*B,*APOS1/*APOS,*K2/*K3)
   -TPHRASE(*I/*I1,*J3/*J4,*C,*V,*W.NIL/*W1,*K3/*K4)
   -TPARAGRAPHE(*I1/*II,*J4/*JJ,*B,*WW/*WW1,*K4/*KK) ..

** SN INTERO ..

+TPHRASE(*I/*I',*J/*JJ,0/*Y,SP(*Z,SN(*A,INTERO,*U)).*V,*W/(*U1.*W1.
     *PR) IMP R(*I',SP(*Z1,*A)),*K/*KK)
    -PRED(*Z,*PR,*Z1)
    -TPARAGRAPHE(0/0,*J/*J1,1/*A.*Y,*U/*U1,*A.*K/*K1)
    -SBSP(*W,SP(*Z,SN(*A,INTERO,*U)),SP(*Z1.NIL,SN(*A,PRON,NIL)),*W3)
    -TPHRASE(0/0,*J1/*JJ,1/*A.*Y,*V,*W3/*W1,*K1/*KK) ..

** SN COMBIEN ..

+TPHRASE(*I/*I',*J/*JJ,0/*Y,SP(*Z,SN(*B,COMBIEN,*U)).*V,*W/(
   CARD(*B,*A).*U1.*W1.*PR) IMP R(*I',SP(*Z1,CARD(*A))),*K/*KK)
    -PRED(*Z,*PR,*Z1)
    -TPARAGRAPHE(0/0,*J/*J1,1/*B.*Y,*U/*U1,*B.*K/*K1)
    -SBSP(*W,SP(*Z,SN(*B,COMBIEN,*U)),SP(*Z1.NIL,SN(*B,PRON,NIL)),*W3)
    -TPHRASE(0/0,*J1/*JJ,1/*B.*Y,*V,*W3/*W1,*K1/*KK) ..

** REGLES SUR PRED ..


+PRED(*X.NIL,NIL,*X) ..
+PRED(*X,DS(*Z,*X),*Z) ..

**CONSTRUCTIONS DES VARIABLES ..

+DEF(0/*X,*K,*J,F(*K,*X),*K',*J,0/*K,1/*X) ..
+DEF(1/*X,*K,*N,*Y,*K,*Y.*N,1/*Y.*X,0/*Y.*X) ..
+CHAQ(0/*X,*K,*N,*Y,*K,*Y.*N,0/*Y.*X,1/*Y.*X) ..
+CHAQ(1/*X,*K,*J,F(*K,*X),*K',*J,1/*X,0/*X) ..
```

```
** REGLES SUR VERB , OS ET DS ..

+CONC(*X.*Y,*Z,*X.*U) -CONC(*Y,*Z,*U) ..
+CONC(*X,*Z,*X.*Z) ..

+VERB(OUI(*V),OUI(ETRE)) ..
+VERB(NON(*V),NON(ETRE)) ..

+OS(SP(SUJ.NIL,SN(*I,*A,*U)).*Z,SP(SUJ.NIL,SN(*I,PRON,NIL)).*ZZ)
   -OS(*Z,*ZZ) ..
+OS(SP(OBJ.NIL,SN(*I,*A,*U)).*Z,*ZZ) -OS(*Z,*ZZ) ..
+OS(*X.*Y,*X.*YY) -OS(*Y,*YY) ..
+OS(NIL,NIL) ..

+DS(*X,*X.*Y) ..
+DS(*X,*Y.*Z) -DS(*X,*Z) ..

** REGLES SUR SBSP ..

+SBSP(P(*X,*Y,*Z).NIL,*V,*U,P(*X,*YY,*Z).NIL)
    -SBSP(*Y,*V,*U,*YY) ..
+SBSP(P(*X,*Y,*Z).*P,*V,*U,P(*X,*YY,*Z).*PP)
    -SBSP(*Y,*V,*U,*YY) -SBSP(*P,*V,*U,*PP) ..
+SBSP(*X.*Y,*X,*Z,*Z.*Y) ..
+SBSP(*X.*Y,*U,*V,*X.*YY) -SBSP(*Y,*U,*V,*YY) ..


** REGLES DE DIVISION REST-APOS DIV ..

+DIV(P(REST,*X,*Y).*U,P(REST,*X,*Y).*V,*W) -DIV(*U,*V,*W) ..
+DIV(P(APOS,*X,*Y).*U,*V,P(APOS,*X,*Y).*W) -DIV(*U,*V,*W) ..
+DIV(NIL,NIL,NIL) ..

** REGLES DE SORTIE DES CLAUSES ..

-ONA(*X.*Y) +ONA(*X) ;;
-ONA(*X.*Y) +ONA(*Y) ..
-ONA(*X IMP *Y) +ONA(NO *X) +ONA(*Y) ..
-ONA(NO (*X.*Y)) +ONA(NO *X) +ONA(NO *Y) ..
-ONA(NO (*X IMP *Y)) +ONA(*X) ;;
-ONA(NO (*X IMP *Y)) +ONA(NO *Y) ..
-ONA(NO NO *X) +ONA(*X) ..
-ONA(R(*I,*X)) /+R(*I,*X) ..
-ONA(P(*X,*Y)) -DEC(*X,*L) /+P(*L,*Y,*I) ..
-ONA(NO P(*X,*Y)) -DEC(*X,*L) /-P(*L,*Y,*I) ..
-ONA(EG(*X,*Y)) -DIF(*X,*Y) /+EG(*X,*Y,*I) ..
-ONA(NO EG(*X,*Y)) -DIF(*X,*Y) /-EG(*X,*Y,*I) ..
-ONA(Q(*X,*Y)) -DIF(*X,*Y) /+Q(*X,*Y,*I) ..
-ONA(NO Q(*X,*Y)) -DIF(*X,*Y) /-Q(*X,*Y,*I) ..
-ONA(ETRE(*X,*Y)) /+ETRE(*X,*Y) ..
-ONA(NO ETRE(*X,*Y)) /-ETRE(*X,*Y) ..
-ONA(CARD(*X,*Y)) /+CARD(*X,*Y) ..
-ONA(NO CARD(*X,*Y)) /-CARD(*X,*Y) ..
+ONA(NIL) ..
-ONA(NO NIL) ..
-ONA(DS(*X,*Y)) /+DS(*X,*Y) ..
-ONA(NO DS(*X,*Y)) /-DS(*X,*Y) ..
-ONA(NQ(*X)) /+NQ(*X) ..
```

```
**REGLES DE REPRESENTATION DES PHRASES ..

+REPRE(P(*X,SP(*Y,*Z).SP(*U,*V).NIL,OUI(ETRE)),EG(*Z1,*V1))
   -REPRE(*Z.*V,*Z1.*V1) ..
+REPRE(P(*X,SP(*Y,*Z).SP(*U,*V).NIL,NON(ETRE)),NO EG(*Z1,*V1))
   -REPRE(*Z.*V,*Z1.*V1) ..
+REPRE(P(*X,SP(*Y,*Z).SP(*U,*V).NIL,OUI(DANS)),Q(*Z1,*V1))
     -REPRE(*Z.*V,*Z1.*V1) ..
+REPRE(P(*X,*Y,OUI(*Z)),P(*YY,OUI(*Z))) -REPRE(*Y,*YY) ..
+REPRE(P(*X,*Y,NON(*Z)),P(*YY,NON(*Z))) -REPRE(*Y,*YY) ..
+REPRE(P(*X,SP(*Y,*Z).NIL,NOM(*U)),ETRE(*Z1,*U)) -REPRE(*Z,*Z1) ..
+REPRE(P(*X,SP(*Y,*Z).NIL,CARD(PLU)),CARD(*Z1,S(1))) -REPRE(*Z,*Z1) ..
+REPRE(P(*X,SP(*Y,*Z).NIL,CARD(*I)),CARD(*Z1,E(*I))) -REPRE(*Z,*Z1) ..
+REPRE(*X.*Y,*XX.*YY) -REPRE(*X,*XX) -REPRE(*Y,*YY) ..
+REPRE(SP(*X,*Y),SP(*X,*YY)) -REPRE(*Y,*YY) ..
+REPRE(SN(*I,*X,*Y),*I) ..
+REPRE(NIL,NIL) ..

** REGLES SUR SCARD ..

+SCARD(P(APOS,*X,CARD(*Y)).*U,P(REST,*X,CARD(*Y)).*U) ..
+SCARD(*X.*Y,*X.*YY) -SCARD(*Y,*YY) ..
+SCARD(NIL,NIL) ..

** REGLES SUR DEC  ..

+DEC(NIL,NIL) ..
+DEC(SP(*X.NIL,*Y).*Z,SP(*X,*Y).*ZZ) -DEC(*Z,*ZZ) ..
+DEC(SP(*X.*Y,*Z).*U,SP(*X,*Z).*UU) -DEC(*U,*UU) ;;
+DEC(SP(*X.*Y,*Z).*U,*T) -DEC(SP(*Y,*Z).*U,*T) ..
AMEN

DEMONTRER(QUANT,CEKONINSERICI,SORTIE)
ECRIRE(SORTIE)

AMEN
```

DEDUCTEUR

Le déducteur a pour but essentiel de répondre aux questions qui lui sont posées.

Pour cela il doit "fouiller" les énoncés logiques pour trouver les questions et les réponses à ces questions.

Le déducteur est en fait constitué des énoncés logiques générés dans la phase précédente et d'un ensemble d'axiomes logiques établissant des relations sémantiques générales entre phrases.

Les objets sur lesquels nous travaillons sont des ensembles d'individus et nous avons interprété les phrases par des relations entres ces ensembles.

ex x voit y

signifie en fait que "l'ensemble x" voit "l'ensemble y" ce que nous interprétons par:

$\forall a \in x \; \exists b \in y$ tel que "a voit b"

et

$\forall b \in y \; \exists a \in x$ tel que "a voit b"

Pour permettre de faire les déductions nécessaires nous avons axiomatisé l'inclusion d'ensemble (prédicat Q), et l'"égalité" d'ensemble (prédicat EG).

Par exemple d'après notre interprétation si nous supposons que

x voit y

et

$z \subset x$

nous pouvons alors conclure que

$\exists S \quad S \subset y$ et z voit S

ce qui se traduit par la règle:

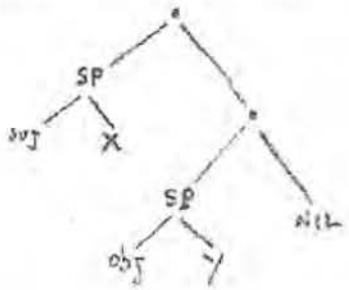
$$-\text{Voir}(x,y)-Q(z,x)+\text{Voir}(z,F(x,z))$$

Nous avons exprimé les relations de type "verbe" à l'aide du **prédicat** P.

x voit y s'écrira en fait

P(L , voit)

où L à la structure de "peigne" suivante:



Les relations de type "nom" seront exprimés à l'aide du prédicat ETRE

ETRE(x,homme.hommes)

signifiant que x a la propriété d'être homme.

La cardinalité d'un ensemble sera exprimée à l'aide du prédicat CARD

CARD(x,i) signifie que le cardinal de x est i

i pourra être de plusieur formes.

E(nombre) pour l'égalité.

S(nombre) pour supérieur
ES(nombre) pour égal ou supérieur.

ex + CARD(x,E(3))

+ CARD(x,ES(1))

Nous trouverons dans ce qui suit l'ensemble des axiomes généraux:

```
INTERFACE DEDUC AMEN
OPERATEURS DEDUC
UNAIRE GD '.
BINAIRE DG . .
BINAIRE GD - .
AMEN

LIRE CEKONINSERICI
** REGLES DE DEDUCTIONS SUR P ..

+P(*T,*V,*I) -DAF(*I,1) -DAF(*I,2) -DAF(*I,3) -DAF(*I,4)
     -P(*L,*V,2)    -SBS(*L,*Y,*X,*I) -DS(SP(*Z,*Y),*L)
     -EG(*X,*Y,*J) -DIF(*X,*Y) ;.

-P(*L,*V,*I) -DAF(*I,1) -DAF(*I,2) -DAF(*I,3) -DAF(*I,4)
    +P(*T,*V,2)       -SBS(*L,*Y,*X,*T) -DS(SP(*Z,*Y),*L)
    -EG(*X,*Y,*J) -DIF(*X,*Y) ;.

+P(*T,*V,*I) -DAF(*I,1) -DAF(*I,2) -DAF(*I,3) -DAF(*I,4)
    -P(*L,*V,3)   -SBT(P(*L,*V),*L,*Y,*X,*T) -DS(SP(*Z,*Y),*L)
    -Q(*X,*Y,7) -DIF(*X,*Y) ;.

-P(*L,*V,*I) -DAF(*I,1) -DAF(*I,2) -DAF(*I,3) -DAF(*I,4)
    +P(*T,*V,3)   -SBT(P(*L,*V),*L,*Y,*X,*T) -DS(SP(*Z,*Y),*L)
    -Q(*X,*Y,*J) -DIF(*X,*Y) ;.

+P(*T,*V,*I) -DAF(*I,1) -P(*L,*V,1)
    -DS1(*T,*L)
    -DIF(*T,*L) ;.
-P(*L,*V,*I) -DAF(*I,1) +P(*T,*V,1) -DS1(*T,*L)
   -DIF(*T,*L) ;.

-P(*L,NON(*V),*I) -DAF(*I,1) -DAF(*I,2) -DAF(*I,3) -DAF(*I,4)
    -CARDL(*L,E(1)) -P(*L,OUI(*V),4) .;
-P(*L,OUI(*V),*I) -DAF(*I,1) -DAF(*I,2) -DAF(*I,3) -DAF(*I,4)
    -CARDL(*L,E(1)) -P(*L,NON(*V),4) .;

** REGLES DE DEDUCTIONS SUR Q,EG, ET CARD ..

+Q(BIDON1,BIDON2,*I) +COUIC ..
-Q(BIDON1,BIDON2,*I) +COUIC ..
+EG(BIDON1,BIDON2,*I) +COUIC ..
-EG(BIDON1,BIDON2,*I) +COUIC ..
+CARD(BIDON,*I) +COUIC ..
-CARD(BIDON,*I) +COUIC ..


+Q(*X,*Y,*I) -DAF(*I,3) -DAF(*I,4) -DAF(*I,7) -EG(*X,*Y,2) ;.
+Q(*X,*Y,*I) -DAF(*I,3) -DAF(*I,4) -DAF(*I,7) -EG(*Y,*X,2) ;.

+EG(*X,*Y,*I) -DAF(*I,5) -DAF(*I,2) -DAF(*I,6)
   -EG(*Y,*X,5) -DIF(*X,*Y) ;.
```

on insère ici les énoncés logiques.

```
+Q(*X,*Y,*I) -DAF(*I,1) -DAF(*I,3) -DAF(*I,4)
   -Q(*X,*Z,1) +COPIE(*Z,BIDON) -Q(*Z,*Y,*J)
   -DIF(*X,*Y) -DIF(*X,*Z) -DIF(*Y,*Z) ;.
-Q(*X,*Z,*I) -DAF(*I,2) -DAF(*I,1) -DAF(*I,3) -DAF(*I,4)
   +COPIE(*Z,BIDON) -Q(*Z,*Y,1) +COPIE(*Y,BIDON) +Q(*X,*Y,*J)
   -DIF(*X,*Y) -DIF(*X,*Z) -DIF(*Y,*Z) ;.
-Q(*Z,*Y,*I) -DAF(*I,2) -DAF(*I,1) -DAF(*I,3) -DAF(*I,4)
   +COPIE(*Y,BIDON) +Q(*X,*Y,1) +COPIE(*X,BIDON) -Q(*X,*Z,*J)
   -DIF(*X,*Y) -DIF(*X,*Z) -DIF(*Y,*Z) ;.

-EG(*X,*Y,*I) -DAF(*I,3) -DAF(*I,4) +Q(*X,*Y,2) ;.

+EG(*X,*X,*I) ;.

-EG(*X,*Y,*I) -DAF(*I,5) +EG(*Y,*X,5) -DIF(*X,*Y) ;.

+EG(*X,*Y,*I) -DAF(*I,2) -DAF(*I,5) -Q(*X,*Y,3) -Q(*Y,*X,3)
   -DIF(*X,*Y) ;.
-Q(*X,*Y,*I) -DAF(*I,2) -DAF(*I,5) +EG(*X,*Y,3) -Q(*Y,*X,3)
   -DIF(*X,*Y) ;.

+EG(*X,*Y,*I) -DAF(*I,2) -DAF(*I,4) -DAF(*I,5) -DAF(*I,6)
   -EG(*X,*Z,6) +COPIE(*Z,BIDON)
       -EG(*Z,*Y,*J) -DIF(*X,*Y) -DIF(*X,*Z) -DIF(*Y,*Z) ;.
-EG(*X,*Z,*I) -DAF(*I,2) -DAF(*I,3) -DAF(*I,4) -DAF(*I,5) -DAF(*I,6)
   +COPIE(*Z,BIDON) -EG(*Z,*Y,6) +COPIE(*Y,BIDON) +EG(*X,*Y,*J)
   -DIF(*X,*Y) -DIF(*X,*Z) -DIF(*Y,*Z) ;.
-EG(*Z,*Y,*I) -DAF(*I,2) -DAF(*I,3) -DAF(*I,4) -DAF(*I,5) -DAF(*I,6)
   +COPIE(*Y,BIDON) +EG(*X,*Y,6) +COPIE(*X,BIDON) -EG(*X,*Z,*J)
   -DIF(*X,*Y) -DIF(*X,*Z) -DIF(*Y,*Z) ;.


+CARD(*Y,*I) -EG(*X,*Y,2) -CARD(*X,*I) -DIF(*X,*Y) ;.
+CARD(*X,ES(*I)) -CARD(*X,E(*I)) ;.
+CARD(*X,ES(*I)) -CARD(*X,S(*I)) ;.

** REGLES SUR LES FONCTIONS DE SKOLEM H(X,Y,Z),G(X,Y) ET CHAQUE(X) ..


+ETRE(CHAQUE(*S.*P),*S.*P) ;.
+CARD(CHAQUE(*S.*P),E(1)) ;.

+CARD(H(*X,*Y,*Z),ES(1)) ;.
+Q(H(*X,*Y,F(*I,*Z)),F(*I,*Z),*J) ..
+ETRE(H(*X,*Y,F(*I,*Z)),*T) -ETRE(F(*I,*Z),*T) ..
+ETRE(H(*X,*Y,*Z),*T) -ETRE(*Z,*T) ..


** REGLES SUR CARDL ..

+CARDL(SP(*Z,*X).NIL,*I) -CARD(*X,*I) ;.
+CARDL(SP(*Z,*X).*Y,*I) -CARD(*X,*I) -CARDL(*Y,*I) -DIF(*Y,NIL) ;.

** REGLES SUR DS1 ET DS ..

+DS1(*X.NIL,*Z) -DS(*X,*Z) ;.
+DS1(*X.*Y,*Z) -DS(*X,*Z) -TRANS(*Z,*X,*T) -DS1(*Y,*T) -DIF(*Y,NIL) ;.
+DS(*X,*X.*Y) ;.
+DS(*X,*Y.*Z) -DS(*X,*Z) -DIF(*X,*Y) ;.
```

```
** REGLES SUR TRANS ..

+TRANS(NIL,*X,NIL) ;.
+TRANS(*X.*Y,*X,*Y) ;.
+TRANS(*X.*Y,*Z,*X.*YY) -TRANS(*Y,*Z,*YY) -DIF(*X,*Z) ;.

** REGLES SUR SBT ET SBS ..

+SBS(NIL,*X,*Z,NIL) ;.
+SBS(SP(*X,*Y).*Z,*Y,*U,SP(*X,*U).*T) -SBS(*Z,*Y,*U,*T) ;.
+SBS(SP(*X,*Y).*Z,*V,*U,SP(*X,*Y).*T) -SBS(*Z,*V,*U,*T) -DIF(*Y,*V) ;.

+SBT(*P,NIL,*X,*Y,NIL) ;.
+SBT(*P,SP(*X,*Y).*Z,*Y,*U,SP(*X,*U).*T) -SBT(*P,*Z,*Y,*U,*T) ;.
+SBT(*P,SP(*X,*Y).*Z,*V,*U,SP(*X,*Y).*T) -CARD(*Y,E(1))
       -SBT(*P,*Z,*V,*U,*T) -DIF(*Y,*V) ;.
+SBT(*P,SP(*X,F(*Y,*R)).*Z,*V,*U,SP(*X,H(*P,*U,F(*Y,*R))).*T)
       -SBT(*P,*Z,*V,*U,*T)
       -DIF(F(*Y,*R),*V) ;.

** REGLES TERMINALES ..

+NQ(*I) -R(*I,*X) +REP(*I,*X);;
+NQ(*I) -NQ(*I') ..
+R(*I,JE.NE.SAIS.PAS) ..


** REGLES DE SORTIE DES REPONSES ..

-REP(*I,JE.NE.SAIS.PAS) -ECRIT(JENESAISPAS) /+SORT(JE.NE.SAIS.PAS) ..
-REP(*I,OUI) -ECRIT(OUI) /+SORT(OUI) ..
-REP(*I,NON) -ECRIT(NON) /+SORT(NON) ..
-REP(*I,SP(*Z,CHAQUE(*X.*Y))) -DAF(*Z,SUJ) -DAF(*Z,OBJ)
       -DAF(*Z,TEMPS) -ECRIT(*Z.CHAQUE.*X) /+SORT(*Z.CHAQUE.*X) .;
-REP(*I,SP(*Z,CHAQUE(*X.*Y))) -ECRIT(CHAQUE.*X) /+SORT(CHAQUE.*X) ..
-REP(*I,SP(*Z,CARD(E(*J)))) -DAF(*Z,SUJ) -DAF(*Z,OBJ)
       -DAF(*Z,TEMPS) -ECRIT(*Z.*J) /+SORT(*Z.*J) .;
-REP(*I,SP(*Z,CARD(E(*J)))) -ECRIT(*J) /+SORT(*J) ..
-REP(*I,SP(*Z,CARD(S(*J)))) -DAF(*Z,SUJ) -DAF(*Z,OBJ)
       -DAF(*Z,TEMPS) -ECRIT(*Z.PLUS.DE.*J) /+SORT(*Z.PLUS.DE.*J) .;
-REP(*I,SP(*Z,CARD(S(*J)))) -ECRIT(PLUS.DE.*J) /+SORT(PLUS.DE.*J) ..
-REP(*I,SP(*Z,CARD(ES(*J)))) -DAF(*Z,SUJ) -DAF(*Z,OBJ)
       -DAF(*Z,TEMPS) -ECRIT(*Z.AU.MOINS.*J) /+SORT(*Z.AU.MOINS.*J) .;
-REP(*I,SP(*Z,CARD(ES(*J)))) -ECRIT(AU.MOINS.*J) /+SORT(AU.MOINS.*J)
..
-REP(*I,SP(*Z,*X))
       -DAF(*Z,SUJ) -DAF(*Z,OBJ) -DAF(*Z,TEMPS)
       -ETRE(*X,*S.*P) -CARD(*X,E(1))
       -ECRIT(*Z.UN.*S) /+SORT(*Z.UN.*S) .;
-REP(*I,SP(*Z,*X)) -ETRE(*X,*S.*P) -CARD(*X,E(1))
       -ECRIT(UN.*S) /+SORT(UN.*S) .;
-REP(*I,SP(*Z,*X))
       -DAF(*Z,SUJ) -DAF(*Z,OBJ) -DAF(*Z,TEMPS)
       -ETRE(*X,*S.*P)
       -ECRIT(*Z.DES.*P) /+SORT(*Z.DES.*P) .;
-REP(*I,SP(*Z,*X)) -ETRE(*X,*S.*P) -ECRIT(DES.*P) /+SORT(DES.*P) .;
-REP(*I,SP(*Z,*X))
       -DAF(*Z,SUJ) -DAF(*Z,OBJ) -DAF(*Z,TEMPS)
       -ECRIT(*Z.*X) /+SORT(*Z.*X) .;
-REP(*I,SP(*Z,*X))
       -ECRIT(*X) /+SORT(*X) ..
AMEN
```

```
LIRE SUPPORT
-NQ(0') ..
AMEN


DEMONTRER(CEKONINSERICI,SUPPORT,RESULT,10)

ECRIRE(RESULT)

AMEN
```

EVALUATION DU SYSTEME

La meilleure façon d'évaluer notre système est bien entendu de regarder tout d'arbord ce qu'il en sort. Voici donc quelques exemples de textes se terminant par une ou plusieurs questions que nous avons soumis au système. Tous ces exemples,à part le dernier qui fait appel à une longue chaine de déductions,ont été improvisés et la pluspart du temps par des visiteurs extérieurs à notre laboratoire. Malheureusement ils n'ont pas toujours marché du premier coup et il a souvent fallu que la personne qui entrait son texte utilise des paraphrases de ses phrase originales.

TEXTE D' ENTREE:

*KAYSER TRAVAILLE A *PARIS.
CHAQUE PERSONNE QUI TRAVAILLE A *PARIS, PREND LE METRO.
*KAYSER EST UNE PERSONNE.
QUE PREND *KAYSER?
OU TRAVAILLE *KAYSER?
REPONSE :

+SORT(UN CERTAIN METRO) ;.

+SORT(A*PARIS) ;.

TEXTE D' ENTREE:

LA GLACE EST UN SOLIDE.
TOUT SOLIDE QUI FOND, EST UN LIQUIDE.
*KAYSER BOIT DE LA GLACE QUI FOND.
QUI BOIT D'UN LIQUIDE?
*KAYSER BOIT-IL D'UN LIQUIDE?

REPONSE :

+SORT(*KAYSER) ;.

+SORT(OUI) ;.

TEXTE D' ENTREE:

LE *MONDE INFORME TOUTE PERSONNE QUI LE LIT.
LE *PRESIDENT EST UNE PERSONNE.
LE *PRESIDENT LIT LE *MONDE.
QUI EST-CE QUI LIT LE *MONDE?
QUI EST-CE QUE LE *MONDE INFORME?

REPONSE :

+SORT(LE*PRESIDENT) ;.

+SORT(LE*PRESIDENT) ;.

TEXTE D' ENTREE:

TOUT HOMME QUI EST CELIBATAIRE, PEUT DEMANDER MARIAGE A TOUTE FEMME QUI EST CELIBATAIRE.TOUT HOMME QUI EST RICHE,PEUT EPOUSER TOUTE FEMME A QUI IL PEUT DEMANDER MARIAGE.TOUT HOMME QUI PEUT EPOUSER LA FEMME QU'IL VEUT EPOUSER,L'EPOUSE.
*MARTIN,QUI EST RICHE,EST UN HOMME QUI EST CELIBATAIRE.
*LEONIE EST UNE FEMME QUI EST CELIBATAIRE.
*MARTIN VEUT EPOUSER *LEONIE.
EST-CE QUE *MARTIN EPOUSE *LEONIE?

REPONSE :

+SORT(OUI) ;.

Nous pouvons faire maintenant une évaluation plus précise sur trois points.

1) Sous-ensemble du français accepté. Ce sous-ensemble est à la fois vaste et restreint. Il est vaste du fait que le vocabulaire est illimité: tous les noms, tous les verbes ou mots jouant un role de verbe sont acceptés. Il est restreint du fait que l'absence de dictionnaire oblige à restreindre les constructions syntaxiques possibles pour pouvoir avoir suffisament de points de repère pour reconnaitre la fonction de chaque mot.

2) Pouvoir de déduction. Du fait que nous n'avons pas de dictionnaire de nom de verbes et d'adjectifs, nous nous intéressons qu'à la sémantique provenant des articles de pronoms et des enchessements de phrases. Mais ceci était une hypothèse de travail. Si quelqu'un veut exprimer les liens sémantiques qui existent entre le verbe "haïr" et "aimer" il suffit qu'il l'exprime dans son texte par une phrase du genre: "chaque personne qui hait une personne ne l'aime pas".

3) Souplesse et efficacité du système. C'est ici que nous pouvons faire le plus de reproche au système: il est très lent et très figé. Ceci provient du fait que tout le système a été écrit en PROLOG à une époque où PROLOG était encore expérimental et n'avait pas encore les capacités de l'interpréteur que nous terminons actuellement. Signalons que la nouvelle version de PROLOG tourne 25 fois plus vite que l'ancienne et donne toutes les possibilités conversationnelles qui manquaient à cette ancienne version. Ceci nous ouvre de vastes horizons pour l'écirutre du prochain système.