à mes parents,
Robert

UNIVERSITÉ D'AIX-MARSEILLE

THÈSE

présentée

A L'U.E.R. DE LUMINY

pour obtenir

LE GRADE DE DOCTEUR DE SPÉCIALITÉ EN INFORMATIQUE APPLIQUÉE

par

Robert PASERO

REPRÉSENTATION DU FRANÇAIS EN LOGIQUE DU PREMIER ORDRE EN VUE DE DIALOGUER AVEC UN ORDINATEUR

Soutenue le 28 mai 1973, devant le Jury,

MM. G. RAUZY — Président
J. C. BOUSSARD
A. COLMERAUER — Examinateurs
B. VAUQUOIS

à Evelyne et Frédérique

TABLE DES MATIERES

Je voudrais exprimer ici ma gratitude à toutes les personnes qui ont bien voulu m'apporter leur concours et leur soutien: à Messieurs les Professeurs B. VAUQUOIS et J.C. BOUSSARD qui ont bien voulu examiner ce travail; à Monsieur le Professeur G. RAUZY qui a accepté de présider le jury de cette thèse; particulièrement à Alain COLMERAUER, Maitre de Conférence d'Informatique qui, dans la direction de ce travail, a guidé et conseillé efficacement mes recherches au cours de nombreuses et fructueuses discussions que nous avons eues ensemble dans le cadre de son équipe; à Philippe ROUSSEL qui s'est montré un ami attentif auprès duquel j'ai toujours trouvé les conseils nécessaires sur tout ce qui a trait à la démonstration automatique; à Colette COURSAGET-COLMERAUER pour tous les entretiens que nous avons eus sur les problèmes se rapportant à la sémantique du langage naturel ; à tous mes amis du Groupe d'Intelligence Artificielle de Luminy auprès desquels j'ai toujours trouvé une aide précieuse et chaleureuse, sans oublier Brigitte JOUBERT qui a participé activement à la réalisation matérielle de cette thèse; et à Evelyne pour son aide précieuse dans la rédaction.

INTRODUCTION

Nous nous proposons, dans cette étude, de décrire un programme de traduction d'un texte français dans le langage de la logique du premier ordre.
Ce travail s'inscrit dans le cadre de la réalisation d'un système de communication homme-machine permettant de dialoguer avec l'ordinateur en langage naturel. (cf.réf. 1)
Ce système est maintenant opérationnel et voici un exemple de conversation que nous avons eue avec l'ordinateur:

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

REPONSE :

+SORT(*MARTIN) ;.

+SORT(OUI) ;.
```

Nous pouvons en avoir beaucoup d'autres puisque le système permet d'une façon générale:

- de décrire un certain "monde" à l'ordinateur, en français;
- de poser des questions toujours en français sur ce "monde".

Le programme général réalisant ce travail, peut en fait être divisé en trois parties, correspondant à trois tâches principales qui sont:

Un analyseur du français qui convertit un texte, écrit en français, en une arborescence faisant apparaitre les relations syntaxiques entre les élèments de base et mettant en évidence les liens sémantiques entre ces éléments. Nous appelons cette arborescence la "structure syntaxique" du texte.

Un traducteur qui traduit la structure précédente en un ensemble d'énoncés logiques simples, facilitant l'accès aux informations contenues dans le texte.

Un déducteur qui raisonne sur les énoncés logiques précédemment générés et produit les réponses aux différentes questions.

Ce programme est écrit en PROLOG (cf.appendice II) qui est à la base un démonstrateur automatique permettant de traiter aisemment les données formelles telles que arbres et listes. Chaque instruction n'est rien d'autre qu'un énoncé de logique du premier ordre, et exécuter un programme revient en fait à démontrer un théorème.

Nous allons nous intéresser essentiellement à la deuxième phase qui est la phase de traduction. A partir d'une étude sémantique du français nous mettrons en évidence les mécanismes généraux de traduction. Cela constituera l'essentiel des chapitres I et II qui auront de ce fait un aspect linguistique.
Nous étudierons dans le chapitre I le role des quantificateurs dans la phrase et comment, à partir de ces quantificateurs, nous pouvons représenter une phrase par une formule logique. Dans le chapitre II nous nous interesserons aux propositions enchassées (relatives appositives) et restrictives) et aux liens que créent les pronoms entre les phrases.
Le chapitre III constitue une description du programme PROLOG réalisant cette traduction. Nous donnerons au début de ce chapitre la syntaxe de la"structure syntaxique"avant de décrire les principales procédures utilisées. En fin de chapitre on trouvera le détail du programme constituant le "traducteur" ainsi que des exemples de traduction de phrases et également une description simple des principaux axiomes de déductions constituant le "déducteur" accompagnée d'exemples de déductions réalisés sur un texte.
Les appendices I et II donnent respectivement quelques rappels de logique, essentiellement sur la forme clausale d'une formule, et une description sommaire du langage de programmation PROLOG.

CHAPITRE I

MISE EN EVIDENCE DES MECANISMES GENERAUX DE TRADUCTION

A la lecture d'un texte nous dégageons un certain nombre d'informations, celles-ci pouvant être plus ou moins explicitement énoncées. Dès lors l'interprétation d'un texte peut paraitre quelque chose de subjectif.
Notre intention n'est pas de "lire entre les lignes", mais de n'extraire d'un texte que les informations nettement explicites. Cependant certaines ambigüités sont difficiles à supprimer sans une connaissance préalable des "objets du monde" et de leurs propriétés. Il s'agira alors de trouver une représentation assez générale permettant de traiter d'une façon adéquate le plus grand nombre possible de phrases.
Le système que nous nous proposons de décrire ne fait pas intervenir la sémantique des noms ou des verbes. La traduction d'un texte va se faire essentiellement à partir du type des arguments. Nous pourrons ainsi traiter un nombre important de phrases du fait même que nous ne sommes pas limités dans le choix des noms et des verbes.
Le but du présent chapitre est de mettre en évidence un certain nombre de mécanismes généraux de traduction à partir des quantificateurs associés aux différents types d'arguments.

I.1 IDEE D'UNE SYSTEMATISATION

Interpréter une phrase va consister à déterminer la formulation logique qui doit lui correspondre, cette représentation devant évidemment traduire les propriétés sémantiques énoncées par la phrase. Pour ce faire nous construirons la formule logique autour du verbe que nous considèrerons dès lors comme un prédicat:

ex(1) Pierre aime Marie.

s'interprète à l'aide de la formule

aime(Pierre,Marie)

Nous constatons qu'il n'est pas besoin d'expliciter davantage les arguments Pierre et Marie puisque nous sommes en présence de deux noms propres . Mais une phrase peut contenir d'autres arguments qui demandent à être interprétés du fait même qu'ils agissent sur la proposition énoncée.

Soit la phrase:

ex(2) Pierre aime une femme.

Il est tout à fait insuffisant et même incorrect de la traduire par

aimé(Pierre,femme)

puisque cette formulation ne rend pas compte du sens de la phrase. En effet cette dernière met en relation "Pierre" et "femme" mais, de plus, énonce une propriété sur "femme" essentielle et déterminante pour le sens de la proposition. Il faut donc que cette propriété apparaisse au niveau de la formulation logique. Nous interprétons donc la phrase par:

$$\exists x \quad femme(x) \wedge aime(Pierre,x)$$

En quelque sorte nous nous ramenons au cas de l'exemple (1) en introduisant une variable x que nous avons déterminée à l'aide du prédicat femme(x) et du quantificateur $\exists$.
Cependant il serait faux de croire que x correspond toujours à un individu ou à un objet précis. Soit la phrase:

ex(3) Pierre aime chaque femme.

cette phrase correspond à l'interprétation:

$$\forall x \quad femme(x) \supset aime(Pierre,x)$$

Le quantificateur change et dès lors toute la formulation logique. Le quantificateur $\exists$ correspond à l'emploi de l'article indéfini UN, le quantificateur $\forall$ à l'emploi de chaque. C'est parce que dans l'exemple (2) la variable est déterminée par $\exists$ que nous interprétons toute la phrase selon le schéma logique:

$$\exists x \quad \varphi(x) \wedge \beta(x)$$

C'est parce que dans l'exemple (3) la variable x est déterminée par le quantificateur universel que nous interprétons l'ensemble de la proposition par le schéma logique :

$$\forall x \quad \varphi(x) \supset \beta(x)$$

Or nous avons dans les deux cas associé au verbe aime le même prédicat aime(x,y) et au nom femme le même prédicat femme(x).

Nous pouvons donc constater que la différence d'interprétation de ces phrases repose sur la seule quantification des arguments puisque le verbe et les noms sont traités de la même manière. Cette quantification au niveau du langage naturel est marquée par l'emploi d'articles définis ou indéfinis, ainsi que par l'emploi de déterminateurs tels que chaque, aucun, etc...

Si nous voulons pousser plus avant l'analyse et savoir pourquoi nous choisissons d'interpréter une phrase par l'un ou l'autre des schémas logiques précédents, il apparaît que la notion de l'existence des objets décrits est déterminante. En effet quand nous choisissons d'interpréter une proposition par la formule

$$\exists x \quad \varphi(x) \wedge \beta(x)$$

nous voulons entre autre poser et établir l'existence d'un individu ou d'un objet ayant la propriété φ . Dans la phrase correspondante l'argument crée une référence à une "chose" du monde décrit par le locuteur et véhicule avec lui par là-même l'idée de son existence. Par contre la formulation

$$\forall x \quad \varphi(x) \supset \beta(x)$$

n'a pas pour intention d'établir l'existence d'une chose ayant la propriété φ. Son intention consiste plutôt à exprimer une proposition hypothètique de la forme: Si.....alors... sans chercher à déterminer l'existence ou la non existence d'objets ayant la propriété φ .
C'est pourquoi finalement l'interprétation d'une phrase repose sur la détermination des quantificateurs à partir des intentions explicites véhiculées par la phrase. Il s'agit en effet de déterminer dans quel cas tel argument fait référence à un objet du monde dont il faut présupposer l'existence et dans quel cas il faut traiter l'argument comme une référence universelle dont il n'est pas nécessaire de présupposer l'existence.

I.2 NOTION D'EXISTENCE ATTACHEE AUX ARGUMENTS

Quand nous disons que la notion d'existence est déterminante pour le choix de l'interprétation logique d'une phrase, et qu'il s'agit dès lors de la préciser pour chaque argument, il est bien évident qu'il n'est pas dans notre intention de faire oeuvre de physicien ou de métaphysicien, en choisissant d'attribuer ou non l'existence à certains objets.

-Un éléphant se pose sur la cime de l'arbre.

-Toute âme est une substance immortelle.

sont des assertions que nous traiterons pour elles-mêmes sans chercher à savoir si les objets décrits correspondent à quelque chose d'existant dans le monde et si les propositions énoncées sont justifiées par des faits réels.
La notion d'existence dont nous nous occupons doit s'attacher aux objets moins en fonction de la réalité du monde, qu'en fonction des arguments qui les représentent dans la phrase.
Certains arguments doivent être appréhendés comme nécessairemment existants pour que la phrase où ils apparaissent ait un sens, c'est à dire soit comprise dans tous ses éléments.
Remarquons qu'une phrase qui enonce une absurdité ou une impossibilité possède un sens à partir du moment où elle est néammoins comprise. Par conséquent lorsqu'un argument est dit présupposer l'existence de l'objet qu'il décrit, nous énonçons une règle formelle s'appliquant à la compréhension de la phrase où il apparait.
Nous verrons par ailleurs que certaines phrases n'exigent pas d'attacher à leurs arguments cette notion d'existence pour acquérir un sens. Notre intention consiste donc à déterminer dans quels cas certains arguments doivent présupposer l'existence des objets et dans quel cas ce n'est pas nécessaire.
Dans les exemples précédents nous avons vu la différence entre l'argument indéfini qui se traduit par le schéma $\exists x \ \varphi(x) \wedge \beta(x)$ et l'argument introduit par chaque dont le but n'est pas de poser une existence mais d'énoncer une proposition hypothètique. Cependant il n'est pas possible de s'appuyer uniquement sur la nature des arguments pour déterminer s'il est nécessaire ou non d'énoncer l'existence des objets décrits.

I.2.1 Arguments indéfinis

Considérons par exemple la phrase:

ex(1) Pierre n'est pas un linguiste.

Il est incorrect de dire ici que l'argument indéfini "un linguiste" fait référence à un objet précis du monde et qu'il présuppose l'existence d'un linguiste x que Pierre n'est pas. L'argument ne devra donc pas être interprété ici par la formule:

$\exists x \quad \varphi(x) \wedge$ Pierre n'est pas x

malgré la présence de l'indéfini.
En fait il s'agit ici d'une phrase à tournure négative qui peut

être interprétée en rejetant la négation de telle sorte qu'on nie la proposition contraire:

Il est faux que Pierre soit un linguiste.

ce qui donne alors la formule

$\forall x \quad \varphi(x) \supset$ Pierre n'est pas x

dans la quelle nous avons associé à l'argument "un linguiste" le schéma $\forall x \quad \varphi(x) \supset \beta(x)$
Nous verrons par la suite que la représentation des phrases à tournure négative pose de nombreux problèmes et que la solution proposée ici ne peut pas être considérée comme valable dans tous les cas.

D'autre part il existe également des phrases à tournures positives dans lesquelles l'argument indéfini ne fait pas référence à un existant. Ce sera bien sur le cas dans des phrases dont l'intention est définitionnelle, et où l'article indéfini à un sens générique.

ex(2) une chatte allaite ses petits.

Cette proposition veut s'appliquer à toutes les chattes et non pas à une chatte particulière x. C'est pourquoi, dans ce cas, nous ne devons pas poser l'existence d'une certaine chatte x.
Ce type de phrase a par conséquent une fonction logique différente.
L'interprétation est alors:

$\forall x \quad$ chatte$(x) \supset x$ allaite ses petits

ce qui correspond à la proposition: Toute chatte allaite ses petits.
L'emploi de l'indéfini au sens générique est cependant difficile à repérer. Il n'existe pas de critères apparent au niveau de la phrase permettant de conclure à l'emploi générique. En effet l'intention de la phrase n'est pas nettement explicitée. La confusion est d'ailleurs possible dans l'esprit de celui qui lit ou entend une telle phrase.
Il apparait par conséquent que d'autres facteurs interviennent dans la compréhension de ce type de phrases. On pourrait noter par exemple que dans un contexte narratif la fréquence des propositions à un sens générique est moins importante que dans un contexte théorique. Il semble aussi que la compréhension de telle phrase repose sur une connaissance préalable des objets du monde et de leurs propriétés.

-Un homme est l'auteur de sa destinée.

-Un homme est l'auteur de l'Iliade.

Ces deux phrases qui possèdent la même structure grammaticale sont cependant comprises différemment et ce, sans équivoque. Par conséquent, interviennent dans ce cas des critères extérieurs à la formulation des phrases qu'il n'est pas dans notre intention d'étudier.
C'est pourquoi nous restreindrons la fonction d'un argument de type indéfini, dans une phrase positive, à celle qui consiste à poser l'existence d'un certain objet, et qui correspond au schéma logique:

$$\exists x \quad \varphi(x) \wedge \beta(x)$$

En dehors de l'article indéfini, il existe d'autres déterminants ayant pour fonction d'introduire dans le discours de nouveaux objets qui s'interprétent selon le schéma existentiel. Il s'agit notamment du déterminant un certain ou certains. Ce dernier a sur l'indéfini l'avantage de ne jamais être équivoque. Alors que nous avons vu les problèmes d'interprétation que pose l'indéfini dans les phrases négatives ou dans les phrases à sens générique, avec ces nouveaux déterminants qui posent nettement et explicitement l'existence de l'objet qu'ils introduisent (et ce en le particularisant) ces problèmes n'apparaissent plus. En effet ils doivent être traités, aussi bien dans les phrases affirmatives que les phrases négatives, à l'aide du schéma existentiel.

-Pierre n'avoue pas certaines passions.

-Un certain coq chante tous les matins.

Ce type d'argument, parce qu'il pose sans ambigüité l'existence des objets qu'il décrit, semble par conséquent plus représentatif de ce que nous exprimons à l'aide du schéma logique existentiel. C'est pourquoi nous le considérons dès lors comme le modèle de référence de ce type de représentation.
Sur le même modèle, nous analyserons les déterminants quelques, divers, plusieurs dont la fonction peut être considérée comme équivalente à celle de l'indéfini pluriel des.

I.2.2 Arguments définis

L'argument défini, comme l'argument indéfini, semble toujours véhiculer avec lui une présupposition d'existence. En effet le défini fait toujours référence à un objet bien précis du monde, qu'il signale en le décrivant.
Cependant notons la différence qui existe entre les deux phrases:

ex(1)(a) Un homme vient.

(b) l'homme vient.

Dans la phrase (1)(a) nous ne connaissons pas l'homme dont on parle. Tout ce que cette phrase nous indique c'est qu'il existe un homme, et nous ne pouvons pas établir de relation entre cet homme mentionné ici et un quelconque individu déjà connu par exemple. Le but de la phrase n'est pas de faire en sorte qu'une telle relation s'établisse, mais simplement de "poser" l'existence de l'objet décrit.
Dans la phrase (1)(b) au contraire, le fait de dire "l'homme" présuppose la connaissance de l'objet décrit, et on nous demande d'établir une relation entre l'objet mentionné ici et celui que nous sommes supposés connaitre.
Le problème est donc de savoir comment cette relation s'établit.
Le défini introduit un objet, et sa mise en relation avec un autre objet du monde ne peut s'effectuer qu'à partir de la description qu'on en donne. En fait, quand nous disons "l'homme vient" -en excluant ici encore le sens générique- il ne peut y avoir d'ambigüité possible, et l'on est certain que dans le monde dans lequel nous nous situons, il y a qu'un seul objet vérifiant les propriétés énoncées.
Dans la phrase:

ex(2) La voiture de Paul est belle.

il est sous-entendu que Paul ne possède qu'une seule voiture, et que dans le monde décrit, il n'y a qu'un seul objet qui ait les propriétés "d'être voiture" et "d'appartenir à Paul".

Il y a donc dans l'emploi du défini une présupposition d'unicité, et c'est grâce à cette propriété que nous pouvons reconnaitre à la lecture d'un texte si l'argument défini décrit un objet déjà mentionné, ou pose l'existence d'un nouvel objet en précisant qu'il est unique.

En effet si nous considérons la phrase:

ex(3) le projet proposé par Pierre a été rejeté.

Il est bien clair que si c'est la première fois que nous entendons parler d'un projet proposé par Pierre, nous allons "enregistrer" que Pierre a proposé un projet, c'est à dire que celui-ci existe. Par contre si nous avons eu déjà connaissance d'un projet proposé par Pierre, nous établirons la relation entre ces deux objets, relation qui se résume à l'égalité.
Nous pouvons donc traiter l'existence d'un objet décrit par un argument défini à l'aide d'une formulation du genre

"il existe un et un seul x tel que..."

ce qui se traduit par la formule logique:

$$\exists x \quad (\varphi(x) \wedge \forall y \quad (\varphi(y) \supset x=y))$$

Dans la représentation logique des exemples qui suivent nous ne ferons pas toujours apparaitre la règle d'unicité associée à un argument défini si celle-ci n'est pas d'un intéret particulier. La règle d'unicité nous permet au niveau des déductions de résoudre les références

ex(3)(a) Un chien vient.

(b) le chien qui vient appartient à Pierre.

Nous représenterons ces deux phrases à l'aide des formules:

$$\exists x \ \text{chien}(x) \wedge \text{vient}(x)$$

$$\exists y \ (\text{chien}(y) \wedge \text{vient}(y) \wedge \forall z((\text{chien}(z) \wedge \text{vient}(z)) \supset y=z) \wedge \text{appartient}(y, \text{à Pierre}))$$

A l'aide de la "règle d'unicité" nous pouvons alors déduire que le chien dont l'existence est posée dans la première formule est le même que celui dont l'existence est posée dans la seconde formule.

REMARQUE I

Nous pouvons remarquer que le défini, quel que soit le contexte semble présupposer l'existence de l'objet décrit, et son interprétation ne parait pas être modifiée par l'emploi de la négation, contrairement à l'indéfini:

ex(4)(a) Pierre ne voit pas le chien.

(b) Aucun étudiant n'a passé l'examen.

Dans ces deux phrases il est clair que le défini désigne un "objet" bien précis et dont l'existence doit être posée:

$$\exists x\,(\text{chien}(x) \wedge \forall y(\text{chien}(y) \supset y=x) \wedge \text{voit}(\text{Pierre},x))$$

$$\exists x\,(\text{examen}(x) \wedge \forall y(\text{examen}(y) \supset y=x) \wedge \forall y(\text{étudiant}(y) \supset \neg\,\text{passe}(y,x)))$$

REMARQUE II

Le traitement du défini consiste à en faire un argument quantifié par un, en associant à la phrase une proposition qui exprime l'unicité. La phrase (3)(b) peut en fait être remplacée par l'énoncé:

un chien qui vient appartient à Pierre et chaque chien qui vient est ce chien.

I.2.3 Arguments quantifiés par chaque

Nous avons dit précédemment qu'un argument quantifié par chaque s'interprète à l'aide du schéma logique $\forall x \quad \varphi(x) \supset \beta(x)$.
Du point vu logique cela revient à considérer que l'argument x fait référence à n'importe quel individu d'une classe donnée φ mais sans présupposer bien entendu qu'il existe un seul élément dans cette classe. Dans la phrase:

ex(1) Chaque homme est mortel,

"chaque homme" fait référence à tous les individus ayant la propriété d'être homme.

Mais on peut se poser la question de savoir si l'on doit poser ou non l'existence de tels individus: c'est à dire si l'on doit traiter ce genre d'argument simplement par une quantification universelle, ou si l'on doit également poser l'existence d'individusayant la propriété "être homme". On peut en effet considérer que le fait d'énoncer une telle phrase présuppose l'existence d'au moins un homme.

En fait si nous nous plaçons dans l'optique des déductions futures ce qu'il est important de dégager d'une telle phrase c'est que, si x est un homme alors x est mortel, et non qu'il existe des hommes. C'est ce genre de formulation qui est utile pour les déductions.
Par ailleurs dans de telles phrases l'intention de poser une existence n'est pas nettement explicite. En effet, considérons la phrase:

chaque candidat doit se présenter.

On ne se préoccupe pas ici de savoir s'il existe ou non des individus ayant la propriété d'être candidat. Tout ce qui est dit se résume à:

si x est candidat alors x doit se présenter.

et il peut très bien n'y avoir aucun condidat.
Nous pouvons opposer à cette phrase la proposition:

chacun des candidats doit se présenter.

qui s'analyse comme

chaque un de les candidats doit se présenter.

et dans laquelle alors la présence du défini pose l'existence d'un certain ensemble A d'individus. La formulation de la phrase correspond à:

Si x appartient à A alors x doit se présenter.

Dans les deux cas nous avons donc le schéma logique:

$$\forall x \quad \varphi(x) \supset \beta(x)$$

mais $\varphi(x)$ dans le premier cas représente la propriété "êtrecandidat" dans le second la propriété "appartenir à A", soit l'appartenir à l'ensemble des candidats, ensemble qui est décrit par l'argument défini.
Avec l'emploi de chaque nous voulons déduire une certaine propriété β sur des objets ayant une propriété φ, mais sans présupposer que l'ensemble des x ayant la propriété φ "existe", par conséquent ce dernier peut en fait être vide. Avec l'emploi de chacun des au contraire, nous posons l'existence d'un certain ensemble (non vide) et nous voulons faire des déductions sur les éléments de cet ensemble.

La première formulation correspond donc simplement au schéma logique

$$\forall x \quad \varphi(x) \supset \beta(x)$$

la seconde formulation au schéma:

$$\exists x \quad \psi(x) \wedge \forall y \quad \varphi(y) \supset \beta(y)$$

La formule $\exists x \ \psi(x)$ étant introduite par le défini dans "chacun de les candidats".

Nous conclurons donc qu'un argument de type chaque ne véhicule pas avec lui une présupposition d'existence, et qu'il correspond simplement à la formulation hypothètique:

Si....alors....

Le quantificateur chaque sera donc pris comme modèle de référence de ce type de schéma logique: $\forall x \ \varphi(x) \supset \beta(x)$. Par ailleurs nous considérerons alors le quantificateur Tout (ou Toute) comme une variante de chaque:

ex(2) Tout candidat doit se présenter.

De même les expressions Tous les ou Toutes les doivent être considérées comme des variantes de chacun des, bien que le verbe soit alors au pluriel.

ex(3) Tous les candidats doivent se présenter.

La proposition (2) correspond à une maxime universelle, à un énoncé de droit, alors que la phrase (3) correspond à l'énoncé d'un fait.

REMARQUES

1°) Considérons la différence entre Tous les et les à l'aide des deux phrases:

ex(4)(a) Les ministres ont visité les installations.

(b) Tous les ministres ont visité Toutes les installations.

Dans la phrase (4)(a) est établie une relation entre deux ensembles x et y, et il est difficile de déduire des propriétés générales au

niveau des éléments de ces ensembles.
Par contre la phrase (4)(b) décrit également une relation entre les deux ensembles x et y, mais cette relation s'interprète entre chaque élément de ces ensembles par la proposition:

chaque ministre a visité chaque installation.

Nous verrons dans un paragraphe suivant comment nous interprètons la phrase (4)(a) au niveau des individus, pour pouvoir opérer des déductions.

2°) Par ailleurs le cas du singulier Tout le peut être considéré simplement comme une variante du défini le

Tout le pays est en effervescence.

En effet, considérer cette propriété au niveau des individus de l'ensemble décrit par le pays, cet ensemble ne contenant alors qu'un seul élément, est équivalent à énoncer simplement la proposition:

Le pays est en effervescence.

I.3 EFFETS DE LA NEGATION SUR LES ARGUMENTS

Les langues naturelles n'offrent pas la même précision dans l'emploi de la négation que le langage logique. L'usage consiste à insérer la négation dans le corps de la phrase, en la faisant généralement porter sur le verbe ou sur un argument. Il peut en résulter certaines ambigüités.

ex(1) Pierre ne remarque pas une faute.

Cette phrase peut donner lieu à deux interprétations différentes qui correspondent aux deux formules:

$\exists x \quad \text{faute}(x) \wedge \neg \text{remarque}(\text{Pierre}, x)$

$\forall x \quad \text{faute}(x) \supset \neg \text{remarque}(\text{Pierre}, x)$

Dans le premier cas, cela signifie "Il existe une faute que Pierre ne remarque pas", dans le second "Pierre ne remarque aucune faute". Cette ambigüité réside dans le fait que nous ne savons pas à priori sur quoi porte la négation dans cette phrase.
La première interprétation rejette la négation sur le verbe seul (négation marquée par la structure ne...pas); dès lors l'argument est introduit par l'article indéfini à valeur positive un que nous analysons à l'aide du déterminant un certain, et qui par conséquent pose une existence:

Pierre ne remarque pas une certaine faute.

La deuxième interprétation au contraire fait porter la négation sur l'argument qui suit le verbe. Cela revient en fait à considérer que cet argument est introduit par un indéfini à valeur négative ne...pas...un qui n'est alors qu'une variante de ne....aucun.

Pierre ne remarque aucune faute.

I.3.1 Arguments indéfinis

Nous avons ici deux interprétations possibles de l'indéfini,l'une à l'aide de un certain comme nous le faisons dans les phrases positives, l'autre à l'aide de aucun. Le choix de l'une ou de l'autre n'est pas aisé à déterminer car il semble qu'entrent en jeu de nombreux facteurs tels que le contexte, le type du verbe, l'usage idiomatique de certaines formules.

Afin de rendre plus explicite les deux genres d'interprétation il est possible de transformer la phrase négative en une phrase positive tout en conservant le sens primitif. Considérons la phrase:

ex(2)(a) Il n'aime pas un film de Fellini.

et interprétons ne pas aimer à l'aide du verbe détester.
Si nous interprétons cette phrase en considérant que l'indéfini pose une existence, pour nous ramener à une tournure positive il suffit simplement de remplacer le groupe verbal ne pas aimer par détester:

ex(2)(b) Il déteste un film de Fellini.

Par contre si on attribue à l'indéfini une valeur négative (il n'aime aucun film de Fellini) pour nous ramener à une tournure positive il est nécessaire d'effectuer une double transformation: aimer par détester et ne...pas...un par chaque.

ex(2)(c) Il déteste chaque film de Fellini.

Mais il faut à propos de cette dernière formulation faire la remarque suivante: dire que "ne pas aimer un film de Fellini" signifie "n'aimer aucun film de Fellini", c'est à dire "détester chaque film de Fellini" ne signifie pas que l'on accorde à un un sens générique mais plutôt une valeur négative correspondant

à la formulation:

Il est faux qu'il aime un film de Fellini.

qui s'interprète alors par:

$\neg(\exists x$ film de Fellini$(x) \wedge$ il aime $x)$

soit

$\forall x$ film de FEllini$(x) \supset$ il n'aime pas x

En fait pour appréhender correctement ce type de construction, il va falloir faire nettement la distinction entre l'indéfini un ayant un sens générique et l'indéfini un ayant une valeur négative. En effet ces deux sortes d'indéfinis conduisent à une même interprétation de la phrase mais en s'appuyant sur une analyse différente.
Considérons à présent une phrase ne comportant aucune ambigüité de ce type:

ex(3)(a) Pierre possède une voiture.

Cette phrase n'est pas équivoque et s'interprète parfaitement à l'aide de la formule:

$\exists x$ voiture$(x) \wedge$ possède(Pierre,x)

Ce qui signifie que un s'interprète ici comme un certain
Prenons maintenant cette même phrase en introduisant une négation sur le verbe:

ex(3)(b) Pierre ne possède pas une voiture.

Il existe, là non plus, aucune équivoque possible quant au sens, et l'interprétation de cette proposition est donnée par la formule

$\forall x$ voiture$(x) \supset \neg$ possède(Pierre,x)

soit

$\neg(\exists x$ voiture$(x) \wedge$ possède(Pierre,x))

Donc la phrase (3)(b) s'interprète sans équivoque par la phrase:

Pierre ne possède aucune voiture.

C'est à dire que la structure ne....pas...un est alors remplacée par ne...aucun .

Nous pouvons trouver ainsi un certain nombre de verbes pour lesquels la tournure négative n'est pas ambigüe:

ex(4)(a) Pierre n'est pas un linguiste.

(b) Le vers de terre ne devient pas un papillon.

(c) Paul ne ressemble pas à un intrus.

Nous mettons ainsi en évidence une certaine catégorie de verbes, qui semblent appartenir en fait à la classe des verbes d'état, et qui ont tous la particularité de ne pas introduire d'ambigüités dans les constructions négatives.
Le fait que ces verbes n'introduisent aucune ambigüité peut se vérifier de la façon suivante: s'il était possible d'interpréter l'article un à l'aide de un certain, il serait alors également possible de faire référence à l'argument ainsi déterminé à l'aide d'un argument défini. Mais on constate aisément que l'on obtient des formules absurdes

la voiture que ne possède pas Pierre.

le linguiste que n'est pas Pierre.

le papillon que ne devient pas le vers de terre.

l'intrus à qui ne ressemble pas Paul.

Par contre comme nous l'avons vu dans l'exemple (1), si nous considérons un verbe comme voir, il est alors possible de définir un objet par:

ex(5) la femme que ne voit pas Pierre.

Donc il semble que le traitement de ces phrases repose sur la nature de leur verbe. Il serait donc nécessaire d'effectuer une classification des différents verbes et de traiter la négation à l'aide de celle-ci. Cependant ce procédé resterait encore insuffisant puisque certaines ambigüités ne pourraient pas être supprimées (ex(1)).
Par ailleurs, même si l'objet décrit par l'argument indéfini semble être mieux déterminé, par l'introduction de relatives par exemple, ce type d'ambigüité subsiste. Il ne semble pas dès lors qu'il soit possible de s'appuyer sur cette détermination pour conclure que l'argument indéfini pose une existence.

ex(6)(a) Pierre ne possède pas une voiture qui roule vite.

(b) Paul n'aime pas une femme qui porte une perruque.

Il n'est pas suffisant également de considérer que l'argument indéfini qui sert d'antécédent à un pronom pose forcément une existence. En effet considérons la phrase:

ex(7) un psychiatre est une personne respectée.

Toutes les personnes qu'il analyse sont malades

Le pronom Il fait référence à l'argument un psychiatre qui ne pose pas ici une existence particulière.

Nous allons donc considérer d'une façon générale que les structures du type ne...pas...un doivent être interprétées comme ne...aucun. Cette façon de faire permet de donner une interprétation correcte pour les types de phrases construites autour d'un verbe d'état; pour les autres types, nous choisissons pour l'instant de les interpréter quelque soit le contexte toujours de la même façon. La seconde interprétation doit seulement être admise quand l'ambigüité est levée à l'aide d'une énonciation différente de la phrase, en déterminant l'argument d'une façon plus claire:

Jacques ne regarde pas une certaine femme.

Jacques ne regarde aucune femme.

Ces remarques ne s'appliquent pas uniquement à l'article indéfini singulier un, mais à toutes les formes d'articles indéfinis, ainsi qu'aux arguments introduits par un cardinal. La phrase:

ex(8) Pierre ne possède pas trois voitures.

doit être interprétée à l'aide d'une formule du genre:

$\forall x$ voiture(x) $\supset \neg$possède (Pierre,x)

dans laquelle x représente n'importe quel ensemble de trois voitures. Nous verrons en effet que pour pouvoir traiter les arguments ayant une marque de pluriel, nous allons devoir interpréter les arguments comme des ensembles d'individus.

Nous n'avons traité jusqu'à présent que des arguments indéfinis situés après le verbe, car il semble que ce soit les seuls dont la quantification soit modifiée par la négation.

Nous pouvons vérifier en effet que certaines phrases ambigües perdent leurs ambigüités si on modifie l'ordre des arguments (passivation), en plaçant l'objet en position sujet. Considérons pour cela la phrase:

ex(9)(a) Paul n'a pas réussi un examen.

Si nous l'énonçons sous la forme:

ex(9)(b) un examen n'a pas été réussi par Paul.

nous levons alors l'ambigüité. Il s'agit ici d'un certain examen. L'autre sens de la phrase (9)(a) correspond à l'énoncé:

ex(9)(c) aucun examen n'a été réussi par Paul.

De même certaines phrases ne peuvent être passivées directement:

ex(10)(a) Pierre ne possède pas une voiture.

Cette phrase n'est pas ambigüe et nous pouvons remarquer alors que l'énoncé "une voiture n'est pas possédée par Pierre" est incorrect car vide de sens. Cette phrase ne peut se passiver que sous la forme:

ex(10)(b) aucune voiture n'est possédée par Pierre.

ou

(c) pas une voiture n'est possédée par Pierre.

On peut donc considérer que les arguments indéfinis situés avant le verbe posent une existence, c'est à dire qu'ils doivent être interprétés sur le modèle du déterminant certain.
Nous pouvons résumer les transformations qui s'imposent sur les articles indéfinis en disant que les structures ne...pas...un doivent en fait être remplacée par ne...aucun.
Dans le cas d'argument situé avant le verbe, nous avons une structure équivalente, pas un....ne qui correspond également à aucun...ne.

I.3.2 Arguments quantifiés par aucun

Le quantificateur aucun introduit la négation au niveau de l'argument:

ex(1) Aucun homme ne vient.

On peut considérer au niveau de l'analyse que le verbe n'est pas négativé, la négation portant sur l'argument.
La phrase (1) peut s'énoncer à l'aide de la formulation hypothétique:

Si x est un homme alors x ne vient pas .

où la négation est alors rejetée sur le verbe.
Le schéma associé au quantificateur aucun est alors le même que celui qui est associé au quantificateur chaque, à la différence que l'on introduit une négation sur le verbe.
Considérons par exemple la phrase:

ex(2) Aucun homme ne voit un chien.

Cette phrase s'énonce alors à l'aide de la formulation:

Si x est homme alors x ne voit pas un chien.

La phrase "x ne voit pas un chien" s'interprétant comme "x ne voit aucun chien" c'est à dire:

Si y est un chien alors x ne voit pas y.

Ce qui se traduit par la formule:

$\forall x(\ (homme(x) \supset \forall y(chien(y) \supset \neg voit(x,y)))$

La phrase "x ne voit pas y" se traduisant par le prédicat négativé associé au verbe voir.

Considérons maintenant la phrase:

ex(3) aucun homme ne possède aucun ami.

qui signifie en fait: chaque homme possède au moins un ami.
Si nous appliquons les schémas précédents nous allons donc obtenir la formulation:

$\forall x$ (homme(x) $\supset$x ne possède pas aucun ami)

dans laquelle la phrase "x ne possède pas aucun ami" signifie "x possède un ami", c'est à dire qu'il nous faut ici aussi transformer les structures ne...pas...aucun en ...un auquel cas nous avons alors la formulation:

$\forall x$ $(homme(x) \supset \exists y(ami(y) \wedge possède(x,y)))$

Donc le quantificateur aucun se traite à l'aide d'un schéma universel, comme le quantificateur tout, la négation étant alors reportée sur le verbe. D'autre part, dans une phrase négative, les structures du type ne...pas...aucun doivent donc être remplacées par ...certain , c'est à dire qu'ici l'argumentse traite à partir d'un schéma existentiel.

I.3.3 Arguments quantifiés par chaque

Le quantificateur chaque et sa variante tout conduisent également à des interprétations différentes selon qu'ils apparaissent dans une phrase négative ou une phrase positive. Dans une phrase négative ils doivent être traduits à l'aide d'un schéma existentiel. Considérons pour cela les deux phrases suivantes:

ex(1)(a) Tout accord n'est pas maintenu.

(b) Pierre n'a pas résolu chaque problème.

Elles sont respectivement équivalentes aux phrases:

ex(2)(a) un certain accord n'est pas maintenu.

(b) Pierre n'a pas résolu un certain problème.

c'est à dire qu'elles se traduisent par les formules:

$\exists x$ accord(x) $\wedge \neg$maintenu(x)

$\exists x$ problème(x) $\wedge \neg$ a résolu(Pierre,x)

Par conséquent nous devons remplacer dans les phrases négatives le quantificateur chaque par certain, et traiter alors la phrase à l'aide du schéma existentiel.

REMARQUE I

On pourrait considérer que la négation porte sur toute la phrase (il est faux que...) et traiter alors celle-ci comme une phrase positive. Considérons en effet la phrase:

Tout accord n'est pas maintenu.

celle-ci peut s'énoncer par:

il est faux que tout accord est maintenu.

ce qui va se traduire par la formule

$$\neg(\forall x \quad accord(x) \supset maintenu(x))$$

qui est équivalente à:

$$\exists x \; accord(x) \wedge \neg maintenu(x)$$

Mais nous aurions également à faire des transformations sur les arguments, car comme nous l'avons vu, certains quantificateurs ne sont pas modifiés par la négation. Considérons la phrase:

un candidat n'a pas résolu chaque question.

Faire porter la négation sur toute la phrase aurait pour conséquence d'attribuer à "un étudiant" une quantification universelle. Nous aurions:

$$\neg(\exists x \; candidat(x) \wedge \forall y(question(y) \supset a\,résolu(x,y)))$$

soit

$$\forall x \quad (candidat(x) \supset \exists y(question(y) \wedge \neg\, a\,résolu(x,y)))$$

Ce qui ne traduit pas le sens de la phrase qui doit être rendu par la formule:

$$\exists x \quad (candidat(x) \wedge \exists y(question(y) \wedge \neg\, a\ résolu(x,y)))$$

REMARQUE II

Il faut remarquer d'autre part que les transformations que nous avons mis en évidence doivent s'effectuer selon l'ordre d'apparition des arguments dans la phrase. En effet considérons la phrase:

aucun homme n'est un idiot.

le quantificateur aucun est transformé en chaque en portant la négation sur le verbe:

chaque homme n'est pas un idiot.

le quantificateur un précédé de la négation est alors remplacé par chaque:

chaque homme n'est pas chaque idiot

Ce qui va se traduire par la formule:

$$\forall x \quad (homme(x) \supset \forall y(idiot(y) \supset \neg\, est(x,y)))$$

Le résultat ne serait pas le même si on commence par remplacer l'article indéfini un avant aucun.

REMARQUE III

Considérons la phrase:

Pierre n'a pas résolu chaque problème.

Nous avons vu que cette phrase signifiait en fait:

Pierre n'a pas résolu certains problèmes.

Il semble sous-entendu dans de telles phrases que l'action s'est réalisée dans certains cas. En effet cette phrase peut être traduite par:

Pierre a résolu certains problèmes

mais Pierre n'en a pas résolu d'autres.

La première assertion correspond au "sens implicite" de la phrase, alors que la seconde correspond au "sens explicite". Il est bien évident que seul le "sens explicite" doit être pris en considération, puisque c'est celui qui est mis en évidence et qui correspond à une réalité. Le "sens implicite" n'est dès lors qu'une supposition et ne peut être de ce fait pris en considération.
Nous pouvons énoncer la phrase:

Pierre n'a pas résolu chaque problème, puisqu'il n'a

pas résolu le problème suivant...

sans savoir si Pierre n'en a jamais résolu un seul.
La seule chose dont nous soyons certains c'est:

Il existe au moins un problème que Pierre n'a pas résolu.

I.4 INTERPRETATION ENSEMBLISTE DES ARGUMENTS

Nous n'avons traité jusqu'à présent que des arguments ayant la marque de singulier, c'est à dire que nous n'avons eu à traiter que des relations et propriétés se rapportant à des individus parfaitement identifiés. La phrase:

ex(1) Un homme vient.

se traduit parfaitement à l'aide de la formule

$$\exists x \quad \text{homme}(x) \wedge \text{vient}(x)$$

x désignant ici un individu particulier.

Si nous considérons maintenant un argument au pluriel, l'intention de la phrase ne peut être la même que pour un argument au singulier. Le singulier permet d'identifier l'individu dont on parle, en le particularisant. Le pluriel permet de créer un ensemble d'individus ayant des propriétés communes, sans donner d'indications permettant de les particulariser les uns des autres. Considérons la phrase:

ex(2) des hommes viennent.

Celle-ci ne peut pas se traduire directement au niveau des individus. En effet tout ce que nous savons c'est qu'il existe un certain nombre d'hommes qui viennent, mais nous ne pouvons pas les particulariser. L'argument "des hommes" décrit un ensemble et nous ne pouvons rien dire sur les individus sans les définir à partir de cet ensemble:

$$\exists x \quad \forall y \quad y \in x \supset y \text{ est homme} \wedge y \text{ vient}$$

où x représente l'ensemble décrit par "des hommes" et y les éléments de cet ensemble. Nous posons donc ici l'existence de l'ensemble x, et y est défini par le fait qu'il appartient à x.
Considérons également les phrases:

Des hommes viennent. L'un d'eux est grand.

"L'un d'eux" définit un individu par appartenance au groupe décrit précédemment. Il est évident que si nous ne posons pas l'existence de l'ensemble "des hommes", nous ne pourrons pas alors traiter ce genre de référence.

Revenons à l'exemple (2). Cette phrase nous donne les informations sémantiques suivantes:

- il existe un certain ensemble d'hommes.
- Cet ensemble possède plus d'un élément.
- Cet ensemble vient.

Il est clair que nous pouvons déduire de l'information "Cet ensemble vient" que "tout élément de cet ensemble vient", mais il ne s'agit là que d'une déduction. En fait la phrase énonce une propriété sur l'ensemble lui-même que nous traduisons par la formule:

$$\exists x \quad \text{homme}(x) \wedge \text{vient}(x)$$

dans laquelle x représente alors un ensemble.
Cette formule est cependant incomplète. En effet considérons la différence qui existe entre les deux phrases suivantes et l'exemple (2)

ex(3)(a) Trois hommes viennent.

(b) Un homme vient.

La seule différence qui existe entre ces trois phrases réside dans le nombre d'individus qui viennent, c'est à dire dans la cardinalité de l'ensemble décrit. D'où la nécessité de traduire l'information sur la cardinalité des ensembles. Nous la traduirons à l'aide du prédicat card.
Card(x,i) signifie "x possède i éléments".
Dans le cas du pluriel indéfini, nous noterons simplement:
card(x,plur) qui signifie "x possède plus d'un élément".
Par conséquent la phrase (2) se traduit alors par la formule:

$$\exists x \quad \text{card}(x,\text{plur}) \wedge \text{homme}(x) \wedge \text{vient}(x)$$

De même la phrase (3)(a) va se traduire par:

$$\exists x \quad \text{card}(x,3) \wedge \text{homme}(x) \wedge \text{vient}(x)$$

Par ailleurs si nous maintenons l'interprétation de la phrase (3)(b)

$$\exists x \quad \text{homme}(x) \wedge \text{vient}(x)$$

où x désigne un individu, nous nous trouvons devant deux genres de formules qui sont hétérogènes. Nous représenterons donc la phrase (3)(b) sur le modèle des autres en posant l'existence d'un

ensemble x de cardinalité 1, dont l'unique élément est alors l'individu défini par la formule précédente. Cela se traduit par:

$$\exists x \quad card(x,1) \wedge homme(x) \wedge vient(x)$$

Nous traitons par conséquent le singulier et le pluriel d'une manière homogène. La construction de la formule logique traduisant la phrase s'effectue toujours à l'aide des schémas logiques que nous avons mis en évidence ultérieurement mais en prenant soin d'y inclure la propriété sur le cardinal de l'ensemble qui devient ainsi une propriété descriptive de l'objet auquel l'argument fait référence. l'exemple (2) se traduit à l'aide du schéma existentiel:

$$\exists x \quad \varphi(x) \wedge \beta(x)$$

où $\varphi(x)$ représente les propriétés descriptives de l'ensemble x, c'est à dire:

- x est un ensemble d'hommes.
- x est de cardinalité supérieure à 1.

et $\beta(x)$ correspond à la propriété énoncée par le verbe de la phrase sur les arguments:

- x vient.

La nouveauté essentielle réside par conséquent dans le fait que l'argument est considéré comme faisant référence à un ensemble d'individus, et que relations et propriétés s'appliquent à des ensembles. Ces relations et propriétés se définissent au niveau des individus par l'équivalence suivante pour tout prédicat P associé à un verbe ou un nom, P ayant un argument:

(EI)

$$P(x) \equiv \forall y \quad y \in x \supset P(y)$$

Ainsi la formule:

$$\exists x \quad (card(x,plur) \wedge homme(x) \wedge vient(x))$$

est équivalente à:

$$\exists x \quad (card(x,plur) \wedge \forall y(y \in x \supset homme(y) \wedge vient(y)))$$

dans laquelle nous avons traduit les propriétés sur les ensembles par des propriétés sur les éléments de ces ensembles. Cependant cette opération ne doit être considérée que comme un second niveau d'interprétation et ne servira réellement qu'au niveau des déductions.

Considérons maintenant la phrase:

ex(4) Les écoliers sont en vacances.

L'argument défini ici fait référence à un ensemble d'individus ayant la propriété "être écolier". Mais, de plus, quand nous disons "les écoliers" nous nous adressons à tous les individus ayant la propriété "être écolier". Donc la présupposition d'unicité que nous avons vue dans le cas du singulier se traduit ici en terme d'inclusion. En effet "les écoliers" désigne le plus grand ensemble ayant la propriété "être écolier", c'est à dire que tout autre ensemble ayant cette propriété est inclus dans celui-ci. La propriété d'inclusion est exprimée à l'aide du prédicat Q:

Q(x,y) signifiant "x est inclus dans y"

La phrase (4) se traduit alors par la formule:

$$\exists x\ (\text{card}(x,\text{plur}) \wedge \text{écolier}(x) \wedge \text{être-en-vacances}(x) \wedge \forall y(\text{écolier}(y) \supset Q(y,x)))$$

où x et y représentent donc des ensembles d'individus.
Nous allons également exprimer la règle d'unicité dans le cas du singulier à l'aide du prédicat d'inclusion Q. Considérons la phrase:

ex(5) La voiture de Paul est accidentée.

Si nous considérons que "la voiture de Paul" désigne un ensemble de cardinalité un, dire que cet ensemble est unique est équivalent à dire qu'il est le plus grand ensemble possèdant de telles propriétés. Nous énoncerons donc la règle d'unicité en disant que tout ensemble qui a la propriété "être voiture de Paul" est inclus dans celui-ci. Ce qui nous permet de traduire la phrase par:

$$\exists x(\text{voiture de Paul}(x) \wedge \text{card}(x,1) \wedge \text{accidentée}(x) \wedge \forall y(\text{voiture de Paul}(y) \supset Q(y,x)))$$

Le traitement des arguments quantifiés par chaque ou aucun va être modifié dans le même sens. Les phrases:

ex(6)(a) Chaque homme est mortel.

(b) Aucun étudiant n'a réussi.

se traduisant donc par les formules:

$\forall x \quad ((\text{homme}(x) \wedge \text{card}(x,1)) \supset \text{mortel}(x))$

$\forall x \quad ((\text{étudiant}(x) \wedge \text{card}(x,1)) \supset \neg \text{réussi}(x))$

où la cardinalité des ensembles x est alors supposée égale à un puisque ces formulations ne s'appliquent qu'à des individus.

Revenons sur le cas l'argument défini pluriel. Nous avons vu qu'un tel argument fait référence à un ensemble d'individus ayant une propriété φ, et que, de plus, cet ensemble contient tous les individus ayant la propriété φ. Ainsi un tel argument peut-il alors être considéré comme jouant le même rôle qu'un argument introduit par chaque. Quand nous disons:

Les hommes sont mortels.

nous énonçons sur chaque homme la propriété d'être mortel.
On pourrait être tenté dès lors de traduire ce type d'argument à l'aide du schéma universel:

$\forall x \quad (\text{homme}(x) \wedge \text{card}(x,1)) \supset \text{mortel}(x)$

qui ne pose donc pas l'existence de l'ensemble "les hommes", mais exprime directement le propriété "être mortel" sur tout individu ayant la propriété "être homme". Du point de vu des déductions on trouve ici un avantage puisqu'alors l'implication est directement exprimée, le fait de ne pas poser l'existence de l'ensemble "les hommes" n'altèrant en rien les déductions.
Mais cela pose des problèmes au niveau de la recherche des antécédents des pronoms. Si la référence est globale, nous allons devoir alors la traiter comme une référence universelle également:

ex(7) Les bandits se sont enfuis. Ils ont volé une voiture.

se traduisant alors par:

$\forall x \ (\text{bandit}(x) \wedge \text{card}(x,1)) \supset \text{enfui}\ (x) \wedge x$ a volé une voiture

ce qui est alors équivalent à:

ex(8) chaque bandit s'est enfui et a volé une voiture

Or la phrase(8) ne traduit pas fidèlement la phrase (7). Dans cette dernière il est exprimé qu'une seule voiture a été volée par l'ensemble"les bandits", par conséquent qu'ils ont tous volé la même voiture. La phrase (8) au contraire fait dépendre l'argument "une voiture" de chaque bandit, c'est à dire que chaque bandit vole une voiture différente.
Considérons également le cas d'une référence à un individu particulier de l'ensemble:

Les bandits se sont enfuis.L'un d'eux possédait une arme.

Nous ne pouvons définir l'objet décrit par "l'un d'eux" que si nous posons l'existence de l'ensemble "les bandits", "l'un d'eux" s'analysent alors comme "un x qui appartient à eux". La phrase se traduit par:

$\exists x$ (bandit(x) $\wedge$ card(x,plur) $\wedge$ enfui(x)
$\wedge \exists$ y (Q(y,x) $\wedge$ card(y,1) $\wedge$ y possédait une arme))

Par ailleurs considérons la phrase:

ex(9) Les hommes aiment les femmes.

Traiter chaque argument défini à l'aide du schéma universel donne de cette phrase une interprétation qui est donc:

chaque homme aime chaque femme.

Ce qui semble être une déduction fallacieuse. En effet la phrase (9) exprime une relation entre deux ensembles x et y, et dès lors il est délicat de savoir ce qui se passe au niveau des individus. En effet si les propriétés unaires sur les ensembles permettent de déduire facilement des propriétés sur les individus, il semble que dans le cas de propriétés binaires, c'est à dire de relations entre deux ensembles, le problème soit plus délicat. Considérons la phrase:

ex(10) Les candidats flattent les électeurs.

Cette phrase va se traduire par la formule:

$\exists x \exists y$ candidat(x) $\wedge$ électeur (y)$\wedge$
card(x,plur) $\wedge$ card(y,plur)$\wedge$
Flatte(x,y)

Nous ne faisons pas intervenir la règle d'unicité pour ne pas alourdir la formule.
Les prédicats candidat(x) et électeur(x) s'interprètent au niveau des individus à l'aide de l' équivalence (E1) puisqu'il s'agit de prédicats unaires.
Le problème va être de déduire à partir de flatte(x,y) des propriétés sur les individus des ensembles x et y. Il est bien évident que ne pouvant différencier les éléments de x, les déductions que nous ferons s'appliqueront à chaque élément de x. Notre but va donc consister à trouver une interprétation au niveau des individus pouvant s'adapter à chaque verbe, et assez générale pour nous permettre de faire par la suite le maximun de déductions plausibles. Considérons la phrase (10). Si nous affirmons par ailleurs que

- Pierre est un candidat.

- Jacques est un électeur.

il ne nous est pas permis de déduire quoi que ce soit entre Pierre et Jacques.
Par contre considérons simplement l'affirmation:

- Pierre est un candidat.

Il semble assez juste de déduire alors que

Pierre flatte certains élécteurs.

De même si nous considérons la seule affirmation

- Jacques est un électeur.

Il est alors permis de penser que

certains candidats flattent Jacques.

"Certains" étant pris ici au sens de "au moins un".
Nous pouvons alors traduire le sens de la phrase (10) à l'aide des deux formules:

chacun des candidats flattent au moins un électeur

et

chacun des électeurs est flatté par au moins un candidat.

D'une manière générale, si P est une relation binaire entre les ensembles x et y, relation associée à un verbe à deux arguments, nous définirons P(x,y) au niveau des individus à l'aide de l'équi-

seule relation: {x} aime {y} signifiant donc que l'individu x aime l'individu y.

Il est bien évident que l'équivalence (EII) s'étend à des verbes comportant plus de deux éléments:

ex(12) Les instituteurs donnent des livres aux enfants.

cette phrase s'expriment alors à l'aide des trois formulations:

chaque instituteur donne au moins un livre à au moins un enfant

et

chacun des livres est donné par au moins un instituteur à au moins un enfant.

et

à chaque enfant est donné par au moins un instituteur, au moins un livre.

Cela va être exprimé par l'équivalence suivante, pour tout prédicat P associé à un verbe à trois arguments x,y,z:

$$(EIII) \qquad P(x,y,z) \equiv \begin{array}{l} \forall s\, \exists t\, \exists u \quad s \in x \supset t \in y \wedge u \in z \wedge P(s,t,u) \\ \qquad\qquad\qquad\qquad \wedge \\ \forall t\, \exists s\, \exists u \quad t \in y \supset s \in x \wedge u \in z \wedge P(s,t,u) \\ \qquad\qquad\qquad\qquad \wedge \\ \forall u\, \exists t\, \exists s \quad u \in z \supset t \in y \wedge s \in x \wedge P(s,t,u) \end{array}$$

Ceci s'étendant d'une manière analogue à des prédicats comportant plus de trois arguments.

REMARQUE

Un argument de type aucun se représente à l'aide du schéma suivant

$$\forall x \quad \varphi(x) \supset \beta(x)$$

la négation étant rejetée sur le verbe de $\beta(x)$, et $\varphi(x)$ représentant les propriétés associées au nom et à la cardinalité qui est supposée être égale à 1:

ex(13) aucun homme ne vient.

se traduit par:

$$\forall x \quad (\text{card}(x,1) \wedge \text{homme}(x)) \supset \neg \text{vient}(x)$$

Nous avons vu que certains arguments indéfinis dans des phrases négatives devaient être traités à partir de ce même schéma, et que nous les considérons comme quantifiés par aucun. Il est bien évident que la cardinalité est alors celle marquée avant cette transformation. La phrase:

ex(14) Pierre ne possède pas trois voitures.

se traduisant par:

$$\forall x \quad (\text{voiture}(x) \wedge \text{card}(x,3)) \supset \neg \text{possède}(\text{Pierre},x))$$

I.5 ANALYSE PRELIMINAIRE DES ARGUMENTS

Nous avons vu dans ce qui précède que nous traitons les propriétés énoncées par le nom et le nombre associés à un argument à l'aide d'un prédicat, de la même manière que les verbes.
Nous pouvons dès lors au niveau d'une première analyse traduire ces propriétés par des relatives dont le nom et le nombre jouent le rôle du verbe, l'argument étant signifié par un "indice" qui constitue une représentation formelle de l'objet décrit par l'argument. Ainsi donc nous n'avons plus au niveau de la phrase que deux types principaux de mots: les verbes et les quantificateurs.
La phrase:

ex(1) un homme vient.

sera anlysée de la façon suivante:

UN x (qui est un . qui est homme) vient

où le mot UN qui précède l'indice x ne doit plus être considéré comme l'article indéfini singulier mais comme un "quantificateur" que nous associerons donc à tout argument indéfini, qu'il soit singulier ou pluriel. Les phrases :

ex(2)(a) des hommes viennent.

(b) trois hommes viennent.

seront analysées par:

UN x (qui est pluriel . qui est homme) vient

UN x (qui est trois . qui est homme) vient

UN est donc le quantificateur associé à tout argument introduit par un article indéfini ou un cardinal. Nous avons vu que les déterminants quelques, divers, plusieurs pouvaient être considérés comme des variantes de des. Nous analyserons donc les arguments introduits par ces déterminants en les quantifiant par UN également, le nombre étant alors pluriel.

ex(3) plusieurs centrales ont arrêté le travail.

sera anlysée par:

UN x (qui est pluriel . qui est centrale) a arrêté le travail

Nous analyserons d'une manière sensiblement différente les arguments introduits par certain puisque comme nous l'avons vu celui-ci ne se traduit pas toujours comme l'indéfini. Le quantificateur associé à ce genre d'argument sera noté CERTAIN.
Pour la phrase:

ex(4) Pierre ne résoud pas certains problèmes.

le résultat de l'anlyse sera:

Pierre ne résoud pas CERTAIN x (qui est pluriel . qui est problème)

D'une manière analogue, le quantificateur associé à tout argument défini sera noté LE.

ex(5) les enfants jouent.

se représente alors par:

LE x (qui est pluriel . qui est enfant) joue

Nous représenterons à l'aide du même quantificateur LE les expressions Tout le...ou Toute la... .

Aux arguments introduits par Chaque ou Tout nous associerons un quantificateur que nous noterons CHAQUE .
La phrase:

ex(6) Tout homme est mortel.

se représentant alors par:

CHAQUE x (qui est un . qui est homme) est mortel.

Au niveau de cette analyse nous considérerons également le quantificateur aucun qui peut s'interpréter par tout si on introduit la négation sur le verbe, et qui correspond donc à l'emploi de aucun...ne ou ne...aucun. La phrase:

ex(7) aucune personne n'est venue.

se représentera par:

AUCUN x (qui est un . qui est personne) est venue.

Cette première analyse ne tient pas compte de la négation sur le verbe. Avant de représenter les phrases dans le langage logique, il va donc être nécessaire de modifier certains de ces quantificateurs en fonction de la négation comme nous l'avons vu auparavant. Ces modifications sont telles que nous n'aurons plus alors que trois types de quantificateurs, que nous noterons pour qu'il n'y ait pas d'ambigüités: DEF,INDEF,TOUT. Les schémas logiques associés à chacun d'eux étant alors:

pour DEF x $(\varphi(x))\ \beta(x)$ le schéma $\exists x\ \varphi(x) \wedge \forall y(\varphi(y) \supset Q(y,x)) \wedge \beta(x)$

pour INDEF x $(\varphi(x))\ \beta(x)$ le schéma $\exists x\ \varphi(x) \wedge \beta(x)$

pour TOUT x $(\varphi(x))\ \beta(x)$ le schéma $\forall x\ \varphi(x) \supset \beta(x)$

Les transformations à faire peuvent alors être décrites par les règles ordonnées suivantes:

...Chaque x φ(x)... ne v pas... ⇒ ...INDEF x φ(x)...ne v pas...

...ne v pas...CHAQUE x φ(x)... ⇒ ...ne v pas...INDEF x φ(x)...

...CHAQUE x φ(x)... ⇒ ...TOUT x φ(x)...

...ne v pas...UN x φ(x)... ⇒ ...ne v pas...TOUT x φ(x)...

...UN x φ(x)... ⇒ ...INDEF x φ(x)...

...CERTAIN x φ(x)... ⇒ ...INDEF x φ(x)...

...LE x φ(x)... ⇒ ...DEF x φ(x)...

...AUCUN x φ(x)...ne v pas... ⇒ ...TOUT x φ(x)... v...

...AUCUN x φ(x)... v... ⇒ ...TOUT x φ(x)...ne v pas...

...ne v pas...AUCUN x φ(x)... ⇒ ... v...INDEF x φ(x)...

... v...AUCUN x φ(x)... ⇒ ...ne v pas...TOUT x φ(x)...

dans lesquelles v désigne n'importe quel verbe, φ(x) l'ensemble des relatives sur x, et ... n'importe quelle suite de mots. Certaines règles suppriment la négation sur le verbe, d'autres au contraire introduisent la négation sur le verbe. Ces règles ne doivent pas être appliquées dans n'importe quel ordre mais selon l'ordre d'apparition des arguments dans la phrase. Laphrase:

ex(8) Tout homme ne possède pas un bateau.

est représentée par:

CHAQUE x φ(x) ne possède pas UN y ψ(y)

nous commençons par transformer le premier argument: CHAQUE x φ(x)

INDEF x φ(x) ne possède pas UN y ψ(y)

puis le second:

INDEF x φ(x) ne possède pas TOUT y ψ(y)

la traduction de la phrase étant alors obtenue en associant à chaque argument le schéma logique correspondant:

$$\exists x \quad (\varphi(x) \wedge \forall y(\psi(y) \supset \neg \text{possède}(x,y)))$$

Pour résoudre le problème des références posées par les pronoms nous allons bien entendu utiliser les "indices" associés à un argument. En effet, l'antécédent d'un pronom doit être trouvé au niveau de l'analyse syntaxique car les critères syntaxiques dans cette recherche sont plus importants que les critères sémantiques.
D'autre part, le pronom ne fait référence à un objet du monde décrit par le locuteur que par l'intermédiaire de l'argument qu'il pronominalise. Donc au niveau de l'analyse syntaxique nous considérons que les références sont résolues et qu'alors à un pronom est associé le même "indice" qu'à son antécédent.
Les phrases:

ex(9) Pierre possède une voiture.Elle est belle.

se représentent alors par:

Pierre possède Un x (x est un . x est voiture) . x est belle.

le pronom relatif étant lui aussi remplacé par l'indice associé à l'argument

Un nom propre joue un rôle particulier. En fait il est lui-même la représentation formelle de l'objet qu'il désigne, et peut jouer directement le rôle de l'indice que nous associons à un argument, celui-ci devant être considéré au niveau de la représentation logique comme une constante.
Un nom propre désigne toujours un individu, considéré comme un tout, une entité indivisible; donc sa cardinalité en tant que propriété sémantique est toujours égale à un, c'est à dire qu'un nom propre sera toujours considéré comme désignant un ensemble à un seul argument. La phrase:

Les Etats-Unis sont un pays riche.

se représentant alors par:

Les Etats-Unis (Les Etats-Unis est un) est un pays riche.

CHAPITRE II

TRADUCTION DES INFORMATIONS CONTENUES DANS UN TEXTE

Pour traduire un texte dans le langage logique, nous allons traduire chaque phrase à l'aide des schémas logiques associés aux arguments de la phrase, mais il va falloir également tenir compte des connecteurs entre les phrases. D'une façon générale on considère que les phrases sont connectées entre elles par le connecteur logique ET représenté par le point (.). Nous ne faisons pas intervenir d'autres connecteurs tels que ou, mais, etc...
Par ailleurs nous ne considérons le connecteur ET qu'entre deux phrases et non entre deux syntagmes nominaux ou deux verbes.
Il existe aussi entre les phrases d'autres types de liens, qui ont un rôle important au niveau de la traduction: ce sont les liens créés par les pronoms. Deux phrases indépendantes donnent naissance à deux formules disjointes A et B qui sont deux formules closes. Par contre, s'il existe entre ces deux phrases un lien dû à la présence d'un pronom, nous allons avoir une formule logique du type $\exists x\,(A(x) \wedge B(x))$, c'est à dire que les deux représentations vont se trouver gouvernées par le même quantificateur. La présence de pronom va donc modifier la portée des quantificateurs et englober à l'intérieur d'une même quantification toutes les phrases reliées entre elles par son intermédiaire.
Par ailleurs il ne suffira pas de déterminer la quantification associée à chaque argument. L'ordre de cette quantification est essentiel dans la mesure où le sens d'une formule peut changer totalement si on opère une modification de cet ordre.

II.1 ROLE DE L'ORDRE DES ARGUMENTS DANS LA TRADUCTION D'UNE PHRASE

L'ordre des arguments dans une phrase est important du point de vue de la sémantique de la phrase. La transformation actif-passif modifie parfois le sens d'une phrase parce qu'elle modifie l'ordre des arguments. Il est donc nécessaire de tenir compte de cet ordre au niveau de la formulation logique.
Il semble normal de considérer que l'ordre des arguments dans la phrase soit celui de la quantification. En appliquant ce principe la traduction d'une phrase peut alors être obtenue en associant à chaque argument, dans l'ordre où il apparaît, le schéma logique qui lui correspond, ce processus se réitérant ainsi sur chacun d'eux. Mais nous verrons que certains arguments définis n'obéissent pas à cette règle.
Dans les exemples qui suivent, nous ne ferons pas toujours intervenir la propriété de cardinalité, ni la règle d'unicité dans le cas du défini, ceci pour plus de clarté dans les formules.

Considérons la phrase:

ex(1) chaque enfant connaît une chanson.

La traduction de cette phrase va être obtenue de façon séquentielle en parcourant la phrase de gauche à droite. Nous allons donc commencer par traiter l'argument chaque enfant en lui associant le schéma logique correspondant:

$$\forall x \quad \text{enfant}(x) \supset x \text{ connaît une chanson}$$

L'argument chaque enfant une fois traité, le noyau de la phrase est alors représenté par:

x connaît une chanson.

où l'argument à traiter est une chanson, argument auquel nous allons associer le schéma logique existentiel:

$$\forall x \quad \text{enfant}(x) \supset \exists y(\text{chanson}(y) \wedge x \text{ connaît } y)$$

le noyau de la phrase est alors représenté par x connaît y dans laquelle chaque argument est maintenant défini. On traduit alors la proposition en considérant le verbe comme un prédicat:

$$\forall x \quad (\text{enfant}(x) \supset \exists y(\text{chanson}(y) \wedge \text{connaît}(x,y)))$$

cette formule logique exprime le fait que la chanson y dépend de l'enfant x choisi. En effet le quantificateur $\forall x$ associé à l'argument chaque enfant gouverne dans la formule le quantificateur $\exists y$ associé à une chanson. Donc nous traduisons par cette formule l'idée que chaque enfant connaît une chanson a priori différente pour chacun d'eux. En fait cette formule se représente par la clause suivante : (appendice I)

$$\neg\text{enfant}(x) \vee \text{chanson}(F(x))$$

$$\neg\text{enfant}(x) \vee \text{connaît}(x,F(x))$$

où F(x) désigne un objet qui dépend donc de x. Donc si on prend x et y différents, nous ne considérons pas a priori que F(x) et F(y) désignent le même objet. En fait il est possible à l'aide d'une règle d'égalité de mentionner par la suite que certains enfants connaissent la même chanson, mais cette idée n'est pas explicite dans la phrase (1).

Par contre considérons la phrase:

ex(2) une chanson est connue par chaque enfant.

Cette phrase semble indiquer au contraire qu'il existe une seule chanson que chaque enfant connaît. L'interprétation de cette phrase en tenant compte de l'ordre des arguments est alors:

$$\exists x \quad chanson(x) \wedge \forall y(enfant(y) \supset connait(y,x))$$

si l'on tient compte évidemment de l'équivalence:

$$est\ connu(x,y) \equiv connait(y,x)$$

La clause correspondante étant alors:

$$chanson(A)$$

$$\neg enfant(x) \vee connaît(x,A)$$

Donc nous vérifions ici que la transformation actif-passif ne conserve pas le même sens à la phrase. Cette différence est marquée au niveau de la formulation logique, par l'ordre différent des quantificateurs.
Condidérons d'autre part les phrases:

ex(3) un enfant chante une chanson.

une chanson est chantée par un enfant.

Ces deux phrases ont le même sens, la transformation actif-passif n'apportant pas ici de modifications quant au sens. Etudions donc leur deux représentations logiques. Pour la première phrase elle va être la suivante:

$$\exists x \quad (enfant(x) \wedge \exists y(chanson(y) \wedge chante(x,y)))$$

Pour la seconde:

$$\exists x \quad (chanson(x) \wedge \exists y(enfant(y) \wedge chante(y,x)))$$

Toutes les deux vont se représenter par les clauses:

$$enfant(A)$$

$$chanson(B)$$

$$connaît(A,B)$$

Donc dans ce cas le fait que la transformation actif-passif ne modifie pas le sens de la phrase se retrouve au niveau de la formulation logique.
Il en sera de même pour les phrases suivantes qui ont le même sens:

ex(4) chaque enfant chante chaque chanson.

chaque chanson est chantée par chaque enfant.

et qui se traduisent par la clause:

$\neg$ enfant(x) $\vee$ $\neg$chanson(y) $\vee$chante(x,y)

L'équivalence dans les exemples (3) et (4) est due au fait que l'on peut permuter dans une formule deux quantificateurs de même type:

$\exists x \exists y \; P(x,y) \equiv \exists y \exists x \; P(x,y)$

$\forall x \forall y \; P(x,y) \equiv \forall y \forall x \; P(x,y)$

Par conséquent l'ordre des arguments joue un rôle important, surtout quand la phrase introduit deux objets quantifiés différemment. Dans ce cas en effet la signification change si on modifie l'ordre. C'est donc le cas pour les phrases ayant deux arguments quantifiés l'un par Tout, l'autre par Un. La phrase:

ex Tout homme ne possède pas un bateau.

va se traduire, en tenant compte de la négation sur le verbe, par la formule:

$\exists x \; (homme(x) \wedge \forall y(bateau(y) \supset \neg possède(x,y)))$

soit par les clauses:

homme(A)

$\neg$bateau(x) $\vee$ $\neg$ possède(A,x)

D'autre part certains types d'arguments, en particulier les arguments définis, ne semblent pas dépendants de leur situation dans la phrase, et leur traitement repose sur d'autres critères:

ex(5) chaque homme aperçoit le chien.

Cette phrase si on tient compte de l'ordre des arguments va se traduire par:

$$\forall x \quad (\text{homme}(x) \supset \exists y(\text{chien}(y) \wedge \text{aperçoit}(x,y)))$$

Ce qui semble incorrect. En effet le défini pose l'existence d'un chien unique qui n'est pas dépendant de l'argument chaque homme. Pour traduire correctement cette phrase, il faut donc inverser l'ordre des arguments:

$$\exists x \quad (\text{chien}(x) \wedge \forall y(\text{homme}(y) \supset \text{aperçoit}(y,x)))$$

ce qui est aussi la traduction de la phrase:

le chien est aperçu par chaque homme.

Il apparait par conséquent que le défini n'est pas soumis à la règle portant sur l'ordre des arguments et qu'il doit être traité avant les autres arguments de la phrase. En fait le défini, le chien, dans l'exemple précédent pose l'existence d'un individu qui est chien, et dont l'existence n'est pas subordonnée à celle d'un autre objet. Mais considérons la phrase:

ex(6) chaque enfant défend le jouet qu'il possède.

L'argument défini,"le jouet qu'il possède",posant également l'existence d'un objet précis, son existence cette fois dépend par l'intermédiaire du pronom IL, de l'existence d'un autre objet.
Ainsi cette phrase doit être traduite par la formule:

$$\forall x \quad (\text{enfant}(x) \supset \exists y(\text{jouet}(y) \wedge \text{possède}(x,y) \wedge \text{défend}(x,y)))$$

où l'ordre des quantificateurs ici est le même que l'ordre des arguments. Les clauses correspondantes sont alors:

$$\neg\text{enfant}(x) \vee \text{jouet}(F(x))$$

$$\neg\text{enfant}(x) \vee \text{possède}(x,F(x))$$

$$\neg\text{enfant}(x) \vee \text{défend}(x,F(x))$$

Considérons la phrase:

x voit un chien.

L'indéfini un chien pose l'existence d'un individu mais cet individu dépend de x dans la mesure où c'est à partir de x que cet objet va se définir:

le chien que x voit.

La phrase précédente est alors équivalente à:

x voit le chien qu'il voit.

et ici on peut voir clairement que le chien qu'il voit ne se défini qu'à partir de x, ce qui signifie que son existence est liée à celle de x. L'argument défini ne dépend que des objets auxquels il fait référence dans sa définition. Si cette référence est marquée par un pronom, il est bien évident alors que sa définition ne pourra s'énoncer que dans une formule gouvernée par le quantificateur associé à l'antécédent de ce même pronom.

x voit le chien que y possède.

"le chien que y possède" ne dépend pas de x, mais seulement de la variable y. Donc cet argument ne peut être déterminé que si y est quantifié:

ex(7) chaque étudiant a traité chaque question sous la forme qu'il voulait.

"la forme qu'il voulait" dépend de l'argument chaque étudiant par l'intermédiaire du pronom il; donc cette phrase devra être traduite par:

$$\forall x\ (\text{étudiant}(x) \supset \exists y(\text{forme}(y) \wedge \text{veut}(x,y) \wedge \forall z(\text{question}(z) \supset \text{traite}(x,z,y))))$$

ce qui est aussi la traduction de:

chaque étudiant a traité, sous la forme qu'il voulait chaque question.

Si aucun pronom n'apparait dans sa définition, l'argument défini ne dépend alors d'aucun autre argument de la phrase, et son existence et ses propriétés peuvent alors être décrites en dehors de la phrase-elle-même:

B voit le monument que chaque touriste connait.

"Le monument que chaque touriste connaît" se définit parfaitement par lui-même et son existence et ses propriétés doivent être décrites indépendemment de tout autre argument de la phrase:

$$\exists y \ \text{monument}(y) \wedge \forall z(\text{touriste}(z) \supset \text{connait}(z,y)) \wedge \text{voit}(B,y)$$

ce qui se traduit par les clauses:

$$\text{monument}(A)$$

$$\neg\text{touriste}(x) \vee \text{connait}(x,A)$$

$$\text{voit}(B,A)$$

La référence à un argument de type chaque, peut-être marquée de façon indirecte:

ex(8) Chaque femme a donné à un enfant le jouet qu'il demandait.

"Le jouet qu'il demandait" dépend de un enfant qui lui-même dépend à son tour de chaque femme. Donc la définition de l'argument défini doit se faire dans la partie de la formule gouvernée par le quantificateur associé à "chaque femme" et à "un enfant":

$$\forall x \ (\text{femme}(x) \supset \exists y(\text{enfant}(y) \wedge \exists z(\text{jouet}(z) \wedge \text{demande}(y,z) \wedge \text{donne}(x,y,z))))$$

Ce qui va alors se traduire par les clauses:

$$\neg\text{femme}(x) \wedge \text{enfant}(F(x))$$

$$\neg\text{femme}(x) \wedge \text{jouet}(G(x))$$

$$\neg\text{femme}(x) \wedge \text{demande}(F(x),G(x))$$

$$\neg\text{femme}(x) \wedge \text{donne}(x,F(x),G(x))$$

Par conséquent une phrase peut-être traduite en associant à chaque argument le schéma logique qui lui correspond et que nous avons décrit. Cette opération doit être effectuée dans l'ordre d'apparition des arguments, à l'exception des arguments définis qui doivent être traités lorsqu'ils ne "possèdent" plus de pronom faisant référence à un argument non encore défini dans la formule.

Nous supposons bien entendu que l'antécédent d'un pronom précède toujours ce pronom dans la phrase. Cette hypothèse semble toujours se vérifier, à l'exception toutefois de certaines tournures stylistiques ou de certaines tournures d'insistances.

Il est venu tard, hier, Pierre.

Un tel résultat a pour avantage de ne pas avoir à s'occuper de la fonction particulière des arguments dans la phrase (sujet,objet,...) mais simplement de leurs situations.
Ainsi nous considèrons qu'un argument gouverne tous les autres arguments qui suivent dans la phrase à l'exception des arguments définis.
En fait, dans la construction des phrases les arguments du verbe occupent une place déterminée. Le sujet est dans la plupart des cas situé en première position et ainsi gouverne les autres arguments. Mais certains arguments semblent avoir un rôle particulier dans la phrase du fait même qu'ils peuvent occuper une place indéterminée. C'est le cas par exemple des circonstantiels de lieu ou de temps.

ex(9)(a) Chaque matin, Pierre boit un café.

(b) Pierre boit un café, chaque matin.

Dans cette phrase, le circonstantiel de temps "chaque matin", semble gouverner toute la phrase quelque soit sa position. Il est bien clair en effet que dans la phrase (9)(b) on ne veut pas signifier que Pierre boit le même café chaque matin. Mais certaines phrases peuvent être ambigües de ce point de vue là:

ex(10)(a) Chaque matin, Pierre rencontre un chien

(b) Pierre rencontre un chien, chaque matin.

L'une ou l'autre phrase ici ne nous permet pas a priori de déterminer s'il s'agit d'un chien différent chaque matin ou du même chien. En tenant compte de l'ordre des arguments nous traiterons ces deux phrases de façon différente. Mais il semble nécessaire de faire intervenir également en plus de l'ordre des arguments, la fonction de l'argument qui peut consister à situer dans le temps ou l'espace "l'action" exprimée par le verbe, ou de la décrire plus précisemment. Ce type d'argument circonstantiel en précisant l'action décrite agit sur les arguments directement liés au verbe.

II.2 TRADUCTION DES PHRASES LIEES ENTRE ELLES PAR UN PRONOM

Un pronom fait référence à un argument situé dans la même phrase ou dans une phrase différente. Nous ne faisons pas intervenir le cas de pronoms faisant référence à une phrase toute entière.
Nous avons vu qu'au niveau de l'analyse cette référence est résolue.
Le problème au niveau de la formulation logique de la phrase est donc d'associer au pronom la même variable qu'à son antécédent.
Cela suppose donc que le pronom apparaisse dans la partie de la formule où cette variable est définie.
Le cas d'un pronom faisant référence à un argument de la même phrase ne pose pas de problème puisque nous avons vu qu'un argument quantifie tout ce qui suit dans la phrase. Donc si on suppose, comme nous l'avons fait, qu'un pronom est toujours précédé de son antécédent, celui-ci sera au moment où il est traité déjà quantifié. Par exemple pour la phrase:

ex(1) Tout homme se nourrit.

nous commençons par traiter l'argument "Tout homme":

$$\forall x \quad (\text{homme}(x) \supset x \text{ se nourrit})$$

Il faut alors traiter l'expression "x se nourrit" où apparait l'argument signifié par le pronom se. Or nous travaillons alors sur une expression quantifiée par $\forall x$, c'est à dire que x est déterminé; donc nous devons remplacer directement le pronom se par la variable x associée à son antécédent:

$$\forall x \quad (\text{homme}(x) \supset x \text{ nourrit } x)$$

soit

$$\forall x \quad (\text{homme}(x) \supset \text{nourrit}(x,x))$$

REMARQUE

L'exemple (1) est interprété selon le schéma logique:

$$\forall x \quad \varphi(x) \supset \beta(x)$$

où $\varphi(x)$ représente les propriétés descriptives de l'argument, ici "x est homme" que nous traduisons par homme(x)
et $\beta(x)$ représente alors le noyau de la phrase dans laquelle nous avons simplement remplacé l'argument par la variable x. Dès lors dans un esprit de cohérence nous devons remplacer tous les représentants de l'argument, c'est à dire l'argument lui-même ainsi

que les pronoms correspondants, par cette même variable x.
Ainsi β(x) ne traduira pas

x se nourrit.

mais

x nourrit x.

De même si nous considérons la phrase:

Certains hommes n'aiment pas le métier qu'ils font.

cette dernière se traduira par la structure intermédiaire:

$\exists x$ (homme(x) $\wedge$ x n'aime pas le métier que x fait)

dans laquelle l'argument certains hommes et le pronom ils sont remplacés simultanément par la variable x définie par $\exists x$.

Considérons maintenant des pronoms faisant référence à un argument d'une autre phrase. Soit l'exemple:

ex(2) Pierre habite Marseille.

Il y possède un magasin.

Ces deux phrases sont liées entre elles par la présence des pronoms Il et y qui font référence respectivement aux deux noms propres Pierre et Marseille. Il est dans ce cas possible de représenter la deuxième phrase par: Pierre possède à Marseille un magasin, les noms propres jouant le rôle de constante et étant définis par eux-mêmes.
Mais si nous prenons la phrase:

ex(3) Pierre possède une voiture;

Il l'a achetée à Paul.

Nous pouvons remplacer directement dans la deuxième phrase le pronom il par Pierre mais nous ne pouvons pas remplacer directement le pronom la (l') par une voiture, celle-ci n'étant pas encore définie.
Si nous interprétons alors la première phrase nous obtenons:

$\exists x$ voiture(x) $\wedge$ possède(Pierre,x)

la seconde phrase, si on remplace "une voiture" par x devient alors:

Pierre a acheté x à Paul .

soit

a acheté(Pierre,x,à Paul)

Il est bien évident que cette formule ne traduit pas la phrase correctement puisque la variable x dans

a acheté(Pierre,x, à Paul)

n'est pas liée à la variable x de la formule précédente.
Les deux formules doivent être dominées par le même quantificateur $\exists$ x:

$$\exists x \ (\text{voiture}(x) \wedge \text{possède}(\text{Pierre},x) \wedge \text{a acheté}(\text{Pierre},x, \text{à Paul}))$$

Ainsi des phrases liées par un pronom doivent-elles être traitées globalement et non pas indépendamment les unes des autres. Par conséquent le quantificateur associé à un argument gouverne non seulement tout ce qui suit dans la phrase, mais également les phrases suivantes si elles sont liées par un ou plusieurs pronoms. Donc la présence d'un pronom va étendre la portée du quantificateur associé à l'antécédent de ce pronom.
Ainsi les phrases:

Chaque psychiatre est un médecin.

Toute personne qu'il analyse est malade.

vont se traduire globalement comme suit:

$$\forall x \ (\text{psychiatre}(x) \supset \exists y(\text{médecin}(y) \wedge \text{est}(x,y) \wedge \forall z((\text{personne}(z) \wedge \text{analyse}(x,z)) \supset \text{malade}(z))))$$

Ce qui se traduit alors par les clauses:

$$\neg \text{psychiatre}(x) \vee \text{médecin}(F(x))$$

$$\neg \text{psychiatre}(x) \vee \text{est}(x,F(x))$$

$$\neg \text{psychiatre}(x) \vee \neg \text{personne}(z) \vee \neg \text{analyse}(x,z) \vee \text{malade}(z)$$

REMARQUE

Il est bien évident que deux phrases indépendantes doivent être traitées indépendamment l'une de l'autre:

chaque homme est mortel.

Socrate est un homme.

se traduit par:

$\forall x \ (homme(x) \supset mortel(x)) \wedge \exists y(homme(y) \wedge est(socrate,y))$

Le connecteur entre les deux phrases se traduisant par le connecteur $\wedge$. Les clauses correspondantes sont alors:

$\neg homme(x) \vee mortel(x)$

$homme(A)$

$est\ (Socrate,A)$

Nous avons vu que l'argument défini devait être traité non pas en fonction de sa situation dans la phrase mais aussitôt que tous les objets auxquels il fait référence dans sa définition à l'aide de pronom ont été eux-mêmes définis. Cela va avoir pour conséquence de modifier certains liens entre les phrases:

ex(4) Chaque marin connait la mer. Elle est capricieuse.

Ces deux phrases sont à ce niveau liées par le pronom Elle. Mais le défini ne dépendant pas de chaque marin, la phrase "elle est capricieuse" ne devra pas être gouvernée par l'argument "chaque marin". Si nous traitons donc d'abord l'argument défini nous allons avoir:

$\exists x \ (mer(x) \wedge$ chaque marin connait x . x est capricieuse)

Les deux phrases "chaque marin connait x" et "x est capricieuse" doivent alors être considérées comme indépendantes à l'intérieur de la formule quantifiée par $\exists x$, et traitées indépendamment l'une de l'autre à ce niveau:

$\exists x \ (mer(x) \wedge \forall y(marin(y) \supset connait(y,x)) \wedge capricieuse(x))$

ce qui va se traduire alors par les clauses:

$$mer(A)$$

$$\neg marin(x) \vee connait(x,A)$$

$$capricieuse(A)$$

Pour obtenir ce résultat nous découperons un texte en paragraphes. Chaque paragraphe regroupe les phrases reliées entre elles par un pronom.
Le traitement des phrases d'un même paragraphe se fera alors globalement, c'est à dire qu'un quantificateur associé à un argument gouvernera tout ce qui suit dans le paragraphe, les arguments étant traités dans l'ordre où ils apparaissent.
Un argument de type existentiel sera alors représenté par une fonction de Skölem dépendant des variables universelles déjà rencontrées dans le paragraphe, à l'exception des arguments définis qui seront représentés par une fonction de Skölem dépendant des variables universelles apparaissant dans sa définition. Pour l'exemple précédent, nous obtiendrons alors l'expression suivante:

$$marin(x) \supset mer(A) \wedge connait(x,A) \wedge capricieuse(A)$$

à partir de laquelle nous génèrerons les clauses précédentes en générant à partir de $A \supset B$ la clause

$$\neg A \vee B$$

que si B est lié à A par une même variable, sinon nous ne générerons que la clause B.

II.3 TRAITEMENT DES RELATIVES

Si nous considérons le rôle joué par les relatives dans une phrase, il apparait que la relative décrit "l'objet" auquel elle se rapporte en apportant à son sujet des informations supplémentaires qui précisent sa définition dans le cas des relatives de type restrictif. Nous traiterons en effet ultérieurement des relatives appositives qui ne jouent pas le même rôle.
Nous avons vu que l'analyse d'un argument donne, pour l'argument "un homme" par exemple, la structure:

UN x (x est homme . x est un)

où x est homme est alors considéré comme une relative associée au symbole x, "un x qui est homme",dans laquelle nous avons substitué au pronom relatif qui l'indice x.
Considérons l'argument:

un homme qui vient

Si nous associons à cet argument le symbole x , nous allons avoir dans ce cas:

un x qui est homme et qui vient

Ce qui signifie que x est décrit à l'aide des deux relatives:

- x est homme
- x vient

L'analyse de cet argument donnera donc, en laissant de coté la pripriété sur la cardinalité de x:

UN x (x est homme . x vient)

La structure de ce qui est mis entre parenthèse correspond alors à un "paragraphe".
La représentation logique d'un argument de ce type s'obtient à l'aide du schéma:

$$\exists x \quad \varphi(x) \wedge \beta(x)$$

$\varphi(x)$ étant associé aux propriétés descriptives de x, $\beta(x)$ étant la phrase elle-même après avoir fait les substitutions nécessaires.

Dans $\varphi(x)$ nous allons donc représenter les propriétés énoncées par les relatives. La phrase:

ex(1) un homme que Pierre connait est venu.

va donc correspondre à l'analyse:

UN x (x est homme . Pierre connait x) est venu.

ce qui doit se traduire par la formule:

$\exists x$ homme(x) $\wedge$ connait(Pierre,x) $\wedge$ est venu(x)

qui est aussi la traduction des phrases:

Pierre connait un homme qui est venu.

Pierre connait un homme et cet homme est venu.

Le problème avec les relatives est donc d'extraire l'information pour l'énoncer à l'aide de plusieurs petites phrases simples. Nous avons donc là un travail qui consiste à désembriquer les relatives, pour les ramener à un niveau supérieur et les traduire comme des propositions principales. La phrase de l'exemple (1) est en fait désembriquée et énoncée sous la forme:

un x est homme et Pierre connait x et x est venu.

Considérons maintenant l'exemple suivant:

ex(2) Pierre lit un livre qu'un ami lui a donné

Le pronom relatif que fait référence à un livre et lui à Pierre. La particularité de cet exemple réside dans le fait que nous introduisons dans la relative un objet nouveau: un ami. Cette phrase va s'analyser à l'aide de la structure suivante:

Pierre lit UN x (x est livre . Un y(y est ami) a donné x à Pierre)

Les pronoms que et lui ayant été remplacés par les indices associés. Si nous traitons comme nous l'avons vu, de manière séquentielle les arguments apparaissent dans cette phrase, nous allons avoir successivement:

$\exists x$((x est livre . un y(y est ami) à donné x à Pierre) $\wedge$ Pierre lit x)

Les expressions x est livre et Pierre lit x se traduisent immédiatement par livre(x) et lit(Pierre,x) puisque tous leurs arguments sont quantifiés. Nous avons donc à traiter UN y(y est ami) et ceci à l'aide du schéma $\exists x \ \varphi(x) \wedge \beta(x)$ associé à ce type d'argument:

$$\exists x \ (\text{livre}(x) \wedge \exists y(\text{ami}(y) \wedge a \text{ donné}(y,x,a \text{ Pierre})) \wedge \text{lit}(\text{Pierre},x))$$

soit

$$\exists x \exists y \ \text{livre}(x) \wedge \text{ami}(y) \wedge a \text{ donné}(y,x, a \text{ Pierre}) \wedge \text{lit}(\text{Pierre},x)$$

Cette formule est également la traduction logique des phrases:

un ami a donné à Pierre un livre qu'il lit.

un livre que Pierre lit lui a été donné par un ami.

Pierre lit un livre. Ce livre lui a été donné par un ami.

Par conséquent le traitement des phrases relatives, quand elles sont associées à un argument se traitant à partir du schéma logique

$$\exists x \ \varphi(x) \wedge \beta(x)$$

consiste à énoncer la relative en la liant à la phrase principale à l'aide du connecteur et ou . , le pronom relatif étant alors remplacé par un pronom personnel. C'est ce qu'exprime le schéma $\exists x \ \varphi(x) \wedge \beta(x)$ où $\varphi(x)$ est la traduction des relatives, la formule obtenue étant liée par $\wedge$ à $\beta(x)$ représentant la traduction de la phrase principale où l'argument a été remplacé par x.

Considérons maintenant les relatives imbriquées dans les arguments de type chaque, c'est à dire dans les arguments se traitant à partir du schéma universel :

$$\forall x \ \varphi(x) \supset \beta(x)$$

ce qui est alors équivalent à:

$$\forall x \ \neg\varphi(x) \vee \beta(x)$$

$\varphi(x)$ est donc précédé d'une négation ($\neg$) que l'on va faire porter sur les prédicats contenus dans $\varphi(x)$.
Considérons en effet la phrase:

ex(3) Tout étudiant qui rate un examen doit redoubler.

Il est bien évident que "un examen" ici ne désigne pas un examen particulier, mais en fait n'importe quel examen. Considérons donc les diverses étapes de la formation de la formule logique.
Nous allons avoir successivement:

$$\forall x \quad (\text{étudiant}(x) \wedge x \text{ rate un examen}) \supset x \text{ doit redoubler}$$

L'argument un examen est l'argument qui doit alors être traité. Il se traite à l'intérieur de la parenthèse à l'aide du schéma existentiel:

$$\forall x \ (\text{étudiant}(x) \wedge \exists y(\text{examen}(y) \wedge \text{rate}(x,y))) \supset x \text{ doit redoubler}$$

ce qui pour obtenir la forme clausale se transforme en la formule:

$$\forall x \quad \text{étudiant}(x) \vee \forall y(\ \text{examen}(y) \vee \neg \text{rate}(x,y)) \vee x \text{ doit redoubler}$$

dans laquelle la quantification associée à y est $\forall y$, ce qui signifie que nous ne posons pas l'existence d'un examen particulier.
La phrase (3) est en fait équivalente à la formulation:

Si x est étudiant alors il est faux que x rate un examen où x doit redoubler.

soit

Si x est étudiant alors x ne rate pas un examen où x doit redoubler.

soit

x n'est pas étudiant où x ne rate pas un examen où x doit redoubler.

Donc dans le traitement d'un argument de type chaque, nous introduisons une négation sur $\varphi(x)$

$$\forall x \quad \neg\varphi(x) \vee \beta(x)$$

Cette négative va donc devoir être introduite sur les phrases relatives attachées à l'argument x, ce qui a pour effet de modifier la quantification associée aux arguments.
Dans le traitement des phrases pour obtenir la formule $\varphi(x)$, nous associerons à un argument le schéma logique qui lui correspond en tenant compte de la négation qui précède. Ainsi un schéma $\exists x \ \varphi(x) \wedge \beta(x)$ se traduira grossièrement par $\forall x \ \neg(\varphi(x) \wedge \beta(x))$ et le schéma $\forall x \ \varphi(x) \supset \beta(x)$ se traduira par $\exists x \ \neg(\varphi(x) \supset \beta(x))$.

Ainsi un argument de type chaque imbriqué dans un autre argument de type chaque va-t-il être quantifié par x.

ex(4) Toute mesure qui est admise par chaque personne fait l'unanimité.

Cette phrase va se traduire par la formule:

$$\forall x\ (\text{mesure}(x) \wedge \forall y(\text{personne}(y) \supset \text{admet}(y,x))) \supset \text{unanimité}(x)$$

qui pour obtenir la forme clausale se transforme en:

$$\forall x\ \ \text{mesure}(x) \vee \exists y(\text{personne}(y) \wedge \neg\text{admet}(y,x)) \vee \text{unanimité}(x)$$

les clauses correspondantes étant alors:

$$\neg\text{mesure}(x) \vee \text{personne}(F(x)) \vee \text{unanimité}(x)$$

$$\neg\text{mesure}(x) \vee \neg\text{admet}(F(x),x) \vee \text{unanimité}(x)$$

La fonction de Skölem F(x) joue ici un rôle particulier.
En fait F(x) représente, si x est une mesure qui ne fait pas l'unamité, une personne qui n'admet pas x. La formule précédente signifie:

x n'est pas une mesure

ou

il existe une personne qui n'admet pas x

ou

x fait l'unamité.

De même la phrase suivante:

ex(5) Tout médecin qui commet certaines erreurs de diagnostic n'est pas un bon praticien.

sera traduite en associant à "certaines erreurs de diagnostic" une variable quantifiée universellement.

Si x est médecin et si x commet certaines erreurs de diagnostic alors x n'est pas un bon praticien.

soit:

$$\forall x \forall y\ (\text{médecin}(x) \wedge \text{erreur de diagnostic}(y) \wedge \text{commet}(x,y)) \supset \neg\text{bon praticien}(x)$$

Considérons maintenant le cas des arguments de type défini. Comme nous l'avons vu déjà, ceux-ci ne dépendent que des arguments auxquels ils font référence par un pronom dans leur définition. Etudions la phrase suivante:

ex(6) Toute personne qui aborde le problème reste perplexe.

L'argument défini "le problème" renvoie à un objet existant qui ne dépend pas de l'argument "toute personne qui aborde le problème". Nous devons donc au préalable poser son existence, ce qui nous donne la formulation intermédiaire:

$\exists x$ (problème(x) $\wedge$ toute personne qui aborde x reste perplexe)

La phrase se traduisant alors par la formule:

$\exists x$ problème(x) $\wedge$ $\forall y$((personne(y) $\wedge$ aborde(y,x)) $\supset$ perplexe(y))

Il est bien évident que si nous traitions l'argument défini après avoir traité l'argument quantifié par chaque nous aurions une formulation totalement différente, le quantificateur associé à l'argument défini étant alors un quantificateur universel. C'est ce qui se produit dans la phrase suivante:

ex(7) chaque homme qui épouse la femme qu'il aime est heureux.

Nous ne pouvons pas ici poser l'existence de "la femme qu'il aime" avant d'avoir défini l'argument auquel "il" fait référence. Par conséquent le quantificateur associé à cet argument va apparaitre dans une formule précédée d'une négation et va devenir alors un quantificateur universel. Nous allons obtenir la formule:

$\forall x$ (homme(x) $\wedge$ $\exists y$(femme(y) $\wedge$ aime(x,y) $\wedge$ épouse(x,y))) $\supset$ heureux(x)

ce qui est alors équivalent à:

$\forall x \forall y$ (homme(x) $\wedge$ femme(y) $\wedge$ aime(x,y) $\wedge$ épouse(x,y)) $\supset$ heureux(x)

Cette traduction ne pose donc pas l'existence d'une certaine femme que x aime. Cette existence est hypothètique. Il semble bien que ce soit là l'intention de la phrase. Nous ne devons pas considérer à partir d'une telle phrase, en effet que, quelque soit l'homme que nous prenons, la femme qu'il aime existe. Ce n'est que dans le cas où cette dernière existe et qu'il l'épouse que nous pouvons conclure que l'homme est heureux.

La règle selon laquelle, un argument défini qui ne fait pas référence dans sa définition à un autre argument doit être traité avant les autres arguments de la phrase, reste donc valable dans le cas où un tel argument apparait dans une relative.
En fait ici il s'agit de poser son existence et de le décrire en dehors de la formule sur laquelle la négation va porter, c'est à dire en dehors de $\varphi(x)$ si nous avons un schéma universel:

$$\forall x \quad \varphi(x) \supset \beta(x)$$

Dans le cas d'un schéma $\exists x \quad \varphi(x) \supset \beta(x)$ cela ne sera en fait pas nécessaire, les formules obtenues dans les deux cas étant équivalentes:

ex(8) La femme qui est assise sur le divan est belle.

La formule obtenue étant alors:

$$\exists x \quad (\text{femme}(x) \wedge \exists y(\text{divan}(y) \wedge \text{est assise}(x, \text{sur } y)) \wedge \text{belle}(x))$$

qui est équivalente à :

$$\exists y \exists x \quad \text{divan}(y) \wedge \text{femme}(x) \wedge \text{est assise}(x, \text{sur } y) \wedge \text{belle}(x)$$

Donc il faudra surtout prendre des précautions dans le cas d'arguments se traitant à l'aide du schéma universel:

$$\forall x \quad \varphi(x) \supset \beta(x)$$

Avant de traduire une phrase à partir d'un tel schéma, il sera donc nécessaire de poser l'existence des objets décrits par un argument de type défini si celui-ci ne contient pas dans sa définition de pronom faisant référence à un argument non encore déterminé.

Revenons sur l'exemple (8), en considérant la règle d'unicité associée à tout argument de type défini. Nous avons vu que cette règle d'unicité paut être rendue par la formulation suivante de la phrase

> une femme qui est assise sur le divan est belle et chaque femme qui est assise sur le divan est cette femme.

A partir d'une telle formulation nous allons donc définir deux fois l'objet décrit par "le divan" en précisant dans les deux cas qu'il est unique. Cela peut être évité si on définit au préalable l'objet décrit par le divan:

$$\exists x\ (\text{divan}(x) \wedge \forall y(\text{divan}(y) \supset \ldots(y,x)) \wedge \text{la femme qui est assise sur x est belle})$$

la phrase "la femme qui est assise sur x est belle" pouvant alors se formuler sous la forme:

une femme qui est assise sur x est belle et chaque femme qui est assise sur x est cette femme.

Cela suppose donc qu'avant de traiter un argument défini à l'aide du schéma logique associé, nous traitions au préalable les arguments définis pouvant apparaitre dans sa définition, si ceux-ci ne sont définis qu'à partir d'objets déjà quantifiés.
En effet dans la phrase:

ex(9) la femme qui regarde le miroir qu'elle tient est belle.

nous ne pouvons pas poser l'existence de "le miroir qu'elle tient" avant d'avoir déterminé l'objet auquel "elle" fait référence.
Nous dirons donc dans ce cas:

une femme qui regarde le miroir qu'elle tient est belle

et

toute femme qui regarde le miroir qu'elle tient est celle-ci

La méthode qui consiste à traduire une phrase en associant à chaque argument suivant le type de son quantificateur, soit le schéma logique $\exists x\ \varphi(x) \wedge \beta(x)$
soit le schéma

$$\forall x\ \varphi(x) \supset \beta(x)$$

implique donc que la quantificateur associé à un argument gouverne $\varphi(x)$ et $\beta(x)$, $\varphi(x)$ traduisant les propriétés énoncées par les relatives associées à l'argument, $\beta(x)$ traduisant la phrase principale. De plus un argument apparaissant dans une proposition enchassée si l'on s'en tient à ces schémas ne va gouverner que les arguments qui suivent dans la proposition enchassée. Considérons par exemple la phrase:

ex(10) Tout homme qui élève un enfant a des responsabilités.

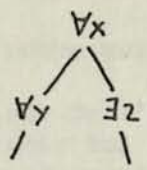
que l'on traduira par la formule:

$$\forall x((\text{homme}(x) \wedge \exists y(\text{enfant}(y) \wedge \text{élève}(x,y))) \supset \exists z(\text{responsabilité}(z) \wedge a(x,z)))$$

Le quantificateur associé à x gouverne toutes les autres variables et en particulier z, par contre y ne gouverne pas la variable z. La formule précédente est donc une formule du type:

$$\forall x \left[\forall y\ P(x,y)\right] \vee \exists z\ R(x,z)$$

si on supprime le signe $\supset$ en ne faisant porter la négation que sur les prédicats (voir appendice I). Nous pouvons alors représenter la structure de dépendance entre les variables par l'arbre:



Dans cette formule la variable z ne dépend donc que de x et les clauses correspondantes sont alors:

$$\neg\text{homme}(x) \vee \neg\text{enfant}(y) \vee \neg\text{élève}(x,y) \vee \text{responsabilité}(F(x))$$

$$\neg\text{homme}(x) \vee \neg\text{enfant}(y) \vee \neg\text{élève}(x,y) \vee a(x,F(x))$$

En fait si l'on considère la formule précédente, elle est équivalente à la formule:

$$\forall x\forall y\ ((\text{homme}(x) \wedge \text{élève}(x,y) \wedge \text{enfant}(y)) \supset \exists z(\text{responsabilité}(z) \wedge a(x,z)))$$

d'après laquelle la fonction de Skölem associée à z dépend de x et y:

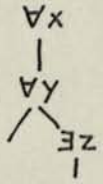
$$\neg\text{homme}(x) \vee \neg\text{enfant}(y) \vee \neg\text{élève}(x,y) \vee \text{responsabilité}(F(x,y))$$

$$\neg\text{homme}(x) \vee \neg\text{enfant}(y) \vee \neg\text{élève}(x,y) \vee a(x,F(x,y))$$

La formule est alors du type:

$$\forall x\forall y\ \left[P(x,y) \vee \exists z\ R(x,z)\right]$$

et"l'arbre de dépendance" entre les variables est:



Traduire une phrase à l'aide de la seconde formule c'est considérer en fait qu'un argument B enchassé dans un argument A gouverne tous les arguments qui suivent B dans la proposition enchassée et tous les arguments qui suivent A dans la proposition principale.
Mais considérons la phrase donnée par l'exemple (10). Il semble fallacieux dans une telle phrase de considérer que l'arguement "des responsabilités" dépend de "un enfant"; en fait on doit considérer qu'il ne dépend que de l'homme choisi. Une telle phrase n'a pas pour intention d'établir un lien entre l'ensemble des responsabilités R qu'aurait un homme H élevant un enfant y, et cet enfant y lui-même, mais simplement de définir un ensemble R de responsabilités pour tout homme H qui a la propriété d'élever un enfant. Par exemple dans la phrase:

ex(11) Tout homme qui fait un travail perçoit un salaire.

le "salaire" ne dépend que de l'homme choisi et non du travail qu'il fait; ce qu'il faut retenir ici c'est la propriété "faire un travail" quel que soit en fait le travail dont il est question. Cette phrase est équivalente à:

Tout homme qui travaille perçoit un salaire.

Par conséquent la fonction F(x,y), dans la deuxième formulation de l'exemple (10), fonction qui représente "les responsabilités qu'a l'homme x qui élève l'enfant y" doit être considérée comme constante par rapport à y, ce qui fait intervenir l'égalité:

$$\forall x \forall y \forall z \quad F(x,y) = F(x,z)$$

Par contre dans la première formulation, la fonction de Skölem F(x) représente "les responsabilités qu'a l'homme x" cet homme x ayant la propriété d'élever un enfant (au moins un); par conséquent elle ne dépend donc que de x.
Nous préférons cette dernière formulation qui ne fait intervenir dans les fonctions de Skölem que les variables dont dépend réellement la fonction et qui ne fait dépendre les fonctions de Skölem que du minimum de variables universelles.

Mais il va se trouver des cas où nous ne pourrons pas limiter la portée des quantificateurs associés aux arguments d'une proposition enchassée à cette seule proposition enchassée,et, où nous allons donc devoir "modifier" la portée des quantificateurs.
Considérons eneffet la phrase:

ex(12) Chaque homme qui possède une voiture l'entretient

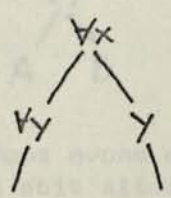
Si nous interprétons cette phrase par les schémas habituels nous allons obtenir la formule:

$$\forall x \left[\text{homme}(x) \wedge \exists y(\text{voiture}(y) \wedge \text{possède}(x,y))\right] \supset \text{entretient}(x,y)$$

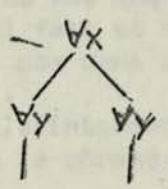
Il est bien évident que cette formule est incorrecte car la variable y dans entretient(x,y) n'est pas quantifiée par $\exists y$.
La phrase doit être traduite ici par:

$$\forall x \forall y \left[\text{homme}(x) \wedge \text{voiture}(y) \wedge \text{possède}(x,y)\right] \supset \text{entretient}(x,y)$$

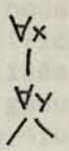
Ces modifications seront réalisées en ajoutant dans la liste des variables qui gouvernent la partie droite de l'implication, la variable y quantifiée par $\forall$ à partir de l'occurence du pronom faisant référence à y, si y n'apparait pas déjà dans cette liste.
Nous pouvons par exemple représenter la structure de dépendance obtenue entre les variables au moment où nous rencontrons le pronom par:



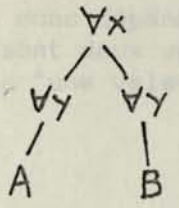
Le symbole y signifie que nous avons à ce niveau un pronom, c'est à dire une occurence de la variable y, ne se trouvant pas dans la portée du quantificateur $\forall y$.
Nous remplaçons alors y par $\forall y$ pour obtenir l'arbre:



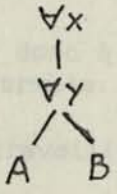
qui sera considéré comme équivalent à :



puisque le fait qu'il ait le même symbole Y signifie que nous avons la même variable. Les variables différentes sont toujours représentées par des symboles différents.
Une structure de dépendance du type:



signifie en fait que A est gouverné par ∀x et ∀y et que B est gouverné par ∀x et ∀y; ce qui est alors équivalent à la structure:



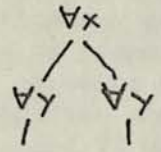
En opérant de la sorte, nous avons donc "déplacé" le quantificateur ∀y de telle manière qu'il soit situé au même niveau que ∀x.
Considérons la phrase:

ex(13) Tout homme qui fait un travail lui accorde une valeur.

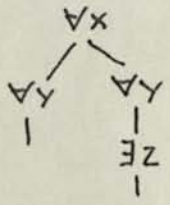
Il est bien évident dans ce cas que "~~une~~ valeur" dépend de l'homme choisi et du travail qu'il fait et ceci à cause de la présence du pronom lui. On ne peut pas dans cette phrase remplacer "faire un travail" par "travailler".
Considérons donc la formule intermédiaire suivante, obtenue au cours de la traduction de la phrase:

$$\forall x\ (\text{homme}(x) \wedge \exists y(\text{travail}(y) \wedge \text{fait}(x,y))) \supset x \text{ lui accorde une valeur.}$$

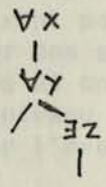
Le fait que l'on trouve le pronom "lui" référant à y dans la partie droite de l'implication, va nous obliger à déplacer lequantificateur ∃y (qui est en fait un quantificateur ∀y si on introduit la négation). Cette opération va être réalisée en réintroduisant la variable y dans la liste des variables gouvernant cette partie de la formule. Ainsi nous allons avoir à ce niveau "l'arbre de dépendance":



L'argument qui suit, va donc dépendre, puisqu'il s'agit d'un existentiel, de x et y qui sont deux variables universelles gouvernant la formule dans laquelle "une valeur" apparait. "L'arbre de dépendance "est alors:



soit



La formule précédente est donc à la suite du traitement du pronom "lui" transformée en la formule:

$$\forall x \forall y \ (\text{homme}(x) \wedge \text{travail}(y) \wedge \text{fait}(x,y)) \supset x \text{ accorde à } y \text{ une valeur}$$

pour donner finalement:

$$\forall x \forall y \ (\text{homme}(x) \wedge \text{travail}(y) \wedge \text{fait}(x,y)) \supset \exists z(\text{valeur}(z) \wedge \text{accorde}(x,z,\text{à } y)$$

ce que l'on va traduire par les clauses:

$$\neg\text{homme}(x) \vee \neg\text{travail}(y) \vee \neg\text{fait}(x,y) \vee \text{valeur}(F(x,y))$$

$$\neg\text{homme}(x) \vee \neg\text{travail}(y) \vee \neg\text{fait}(x,y) \vee \text{accorde}(x,F(x,y),y)$$

La fonction de Skölem F(x,y) désigne ici "la valeur qu'accorde l'homme x au travail y qu'il fait".

La présence d'un pronom faisant référence à un argument imbriqué dans une relative modifie la portée du quantificateur associé à cet argument. Dans une phrase l'argument est quantifié au moment où il apparait. Le quantificateur d'un argument est déterminé par la première occurence de l'argument lui-même. La portée de cette quantification n'est pas clairement précisée et le lien entre les variables se fait par l'utilisation de pronom. Nous considérons que ce quantificateur gouverne au minimum tous les arguments qui suivent dans la proposition. Mais cette portée va pouvoir être modifiée par la suite à cause de la présence de pronom. Si nous voulons représenter une phrase par une formule logique il faut faire en sorte que le quantificateur associé à un argument soit situé de telle manière qu'il gouverne toutes les occurences des pronoms faisant référence à cet argument. On peut alors pour représenter une phrase utiliser une écriture, proche de l'écriture logique, mais dans laquelle la notion de variables liées correspond à la représentation des variables par un même symbole. C'est à dire que les variables qui ne sont pas définies par le même quantificateur sont représentées par des symboles différents. Une telle écriture n'est pas ambigüe et on peut alors introduire la quantification d'une variable au niveau de la première occurence de la variable. Par exemple pour l'exemple (13) on peut alors écrire avec ces conventions:

$$\forall x\ (\text{homme}(x) \wedge \exists y(\text{travail}(y) \wedge \text{fait}(x,y))) \supset \exists z(\text{valeur}(z) \wedge \text{accorde}(x,z,\text{à } y))$$

Pour obtenir l'écriture logique il sera donc nécessaire de rétablir la portée normale des quantificateurs pour que ceux-ci gouvernent toutes les occurences de la variable qu'ils définissent. Les règles permettant de"repousser" les quantificateurs sont celles du formalisme logique

$$(Qx\ P(x)) \wedge R \quad \Rightarrow \quad Qx(P(x) \wedge R)$$

$$(Qx\ P(x)) \supset R \quad \Rightarrow \quad \overline{Qx}(P(x) \supset R)$$

$$P \wedge QxR(x) \quad \Rightarrow \quad Qx(P \wedge R(x))$$

$$P \supset QxR(x) \quad \Rightarrow \quad Qx(P \supset R(x))$$

$$\neg QxP \quad \searrow\Rightarrow \quad \overline{Qx}\neg P$$

Qx étant mis pour $\exists x$ (ou $\forall x$), $\overline{Qx}$ signifie alors $\forall x$(ou $\exists x$).

Ces transformations sont réalisées sur "l'arbre de dépendance" précédent pour rétablir la portée normale des quantificateurs.

Il est bien évident que l'on peut prévoir des schémas beaucoup plus complexes que ceux que nous avons dans les exemples précédents. En fait il y a problème quand nous avons des quantificateurs universels. Si l'on ne prévoit que des variables existentielles, et c'est sourtout ce type de variables qui apparait dans un discours, il n'y a pas en fait de restrictions à apporter au système. Considérons par exemple la phrase:

ex(14) un homme qui portait une valise a demandé un renseignement à Paul.

Cette phrase peut se traduire par la formule:

$$\exists x \ (\text{homme}(x) \wedge \exists y(\text{valise}(y) \wedge \text{portait}(x,y)) \wedge$$

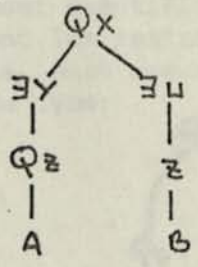
$$\exists z(\text{renseignement}(z) \wedge \text{a demandé}(x,z, \text{à Paul})))$$

qui est équivalente à la formule:

$$\exists x \exists y \exists z \ \text{homme}(x) \wedge \text{valise}(y) \wedge \text{portait}(x,y) \wedge$$

$$\text{renseignement}(z) \wedge \text{a demandé}(x,z, \text{à Paul})$$

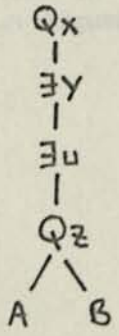
Ces deux formules sont équivalentes et ne modifient pas l'écriture des fonctions de Skölem. Donc dans ce cas, si nous avons un schéma du type:



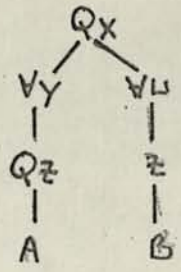
Qx étant mis pour ∃x ou ∀x

Qz étant mis pour ∃z ou ∀z

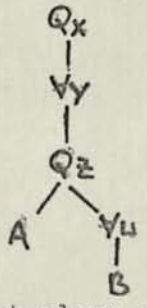
il est parfaitement correct et cela ne change rien à la construction des clauses de le transformer en:



Par contre supposons que nous ayons "l'arbre de dépendance" suivant:



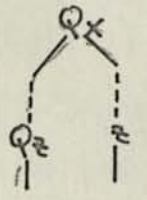
qui traduit le fait que A dépend de z,y,x et B de z,u et x. Dans ce cas il semble que le schéma de dépendance doive être modifié pour obtenir:



Les transformations sont alors plus complexes. Mais il est rare de trouver des phrases donnant de tels schémas:

-d'une part parce que le quantificateur "chaque" est peu employé dans un récit.
-d'autre part parce qu'un pronom ne réfère que très rarement à un argument quantifié par "chaque".

Nous faisons donc les restrictions nécessaires, quant à la structure des phrases, nous assurant que si nous obtenons un schéma de dépendance du type:



nous supposons qu'entre Qx et Qz, et Qx et z il n'y a pas de noeud représentant un argument quantifié universellement.

REMARQUE

Les phrases, où un pronom fait référence à un argument apparaissant dans une relative, semblent pouvoir toujours s'énoncer d'une façon telle que l'argument auquel le pronom fait référence soit alors un argument de la phrase principale. L'exemple(13) peut en effet s'énoncer par:

Tout homme accorde à tout travail qu'il fait une valeur.

Par ailleurs on peut remarquer qu'un énoncé qui éloigne le pronom de l'argument auquel il se réfère n'est pas habituel, et est surtout utilisé pour marquer une idée d'insistance:

Tout homme qui fait un travail accorde une valeur à ce travail.

Considérons maintenant le cas des relatives appositives. Nous considèrons une relative comme appositive quand elle n'a pas pour but de définir l'objet auquel elle fait référence mais simplement d'apporter à son sujet des informations supplémentaires. Dans tous les cas on pourra en fait les "désembriquer" et les lier à la phrase principale à l'aide du connecteur et en remplaçant le pronom relatif par un pronom d'un autre type.
En effet considérons la phrase:

ex(15) Pierre, que j'ai rencontré hier, doit faire un voyage.

celle-ci peut s'énoncer sous la forme:

Pierre doit faire un voyage. Je l'ai rencontré hier.

Nous avons vu que dans le cas d'argument se traitant à partir du schéma $\exists x \ \varphi(x) \wedge \beta(x)$, c'est déjà ce qui est réalisé sur les relatives restrictives. Il n'est pas nécessaire alors de faire la différence.
Celle-ci sera par contre nécessaire pour les arguments traduits à l'aide du schéma universel $\forall x \ \varphi(x) \supset \beta(x)$. En effet dans $\varphi(x)$ ne doit intervenir que les propriétés qui permettent de définir x, c'est à dire les propriétés qui ont un caractère restrictif.
Considérons la phrase:

ex(16) Chaque projet, qui pourtant est bien préparé, est refusé.

La relative n'a pas pour role de définir l'argument "chaque projet".

$$\forall x \quad \text{projet}(x) \supset (x \text{ est bien préparé} \wedge x \text{ est refusé})$$

Ce qui correspond à la traduction de l'énoncé:

chaque projet est bien préparé. Il est refusé.

De même considérons l'argument défini dans la phrase:

ex(17) le ministre, qui connaissait les revendications, les ignora cependant.

Nous ne devons pas faire intervenir dans la définition de la règle d'unicité la relative "qui connaissait les revendications". En effet cette phrase exprime l'existence "d'un et d'un seul ministre et"ce ministre connait les revendications".C'est à dire donc que la règle d'unicité ne doit faire intervenir ici que la propriété d'être ministre. Par conséquent cela se traduit par la formule:

$$\exists x \ \text{ministre}(x) \wedge \forall y(\text{ministre}(y) \supset Q(y,x)) \wedge$$
$$x \text{ connaissait les revendications} \wedge$$
$$x \text{ les ignora cependant}$$

Ce qui est aussi la traduction de la périphrase:

le ministre connaissait les revendications et les ignora cependant.

Il est donc nécessaire pour certains arguments de faire intervenir le fait qu'une relative est restrictive ou appositive; cela sera surtout utile dans la construction des schémas universaux

$$\forall x \quad \varphi(x) \supset \beta(x)$$

où alors la traduction logique des relatives restrictives doit apparaitre dans $\varphi(x)$,celle des relatives appositives dans $\beta(x)$. Pour les arguments se traitant à partir des schémas existentiels il n'est pas nécessaire de faire la différence, toutes les relatives pouvant être en fait considérées comme appositives.

II.4 TYPE ET TRAITEMENT DES QUESTIONS

Un texte est en fait composé d'un ensemble de phrases descriptives qui permettent au locuteur de décrire un certain "monde", et de questions auxquelles nous répondons à partir des informations contenues dans le texte. Les questions seront traitées comme les autres phrases, c'est à dire à partir des mêmes schémas logiques, leur représentations sous forme logique introduisant une partie "réponse" que l'on peut traduire par:

Si... alors la réponse est...

Avant de voir plus en détail comment nous traitons les questions, considérons d'abord les différents types de questions auxquels nous allons nous intéresser. Ils sont au nombre de deux.

A) les questions portant sur un argument d'une phrase, celui-ci étant alors signifié par un pronom interrogatif. En général cet argument est alors placé en tête de phrase.

Qui est venu?

Où travaille Jacques?

Qui est ce qui a fait ce travail?

Quand est ce que Paul arrive?

Quel film as-tu vu?

En fait ces phrases ont une structure identique à celle des autres phrases (affirmatives), si ce n'est l'ordre des arguments qui est modifié, et la présence d'un argument particulier qui est signalé par un pronom interrogatif. Le traitement effectif de la question se fera à partir de cet argument.

B) Les questions en Est-ce que..., qui demandent une réponse en <u>oui</u> ou en <u>non</u>.

Est-ce que Pierre est venu?

Jacques possède-t-il ce livre?

Dans ce cas il n'y a pas de pronom interrogatif. L'interrogation porte sur un fait, sur la réalisation d'un évènement.

Considérons d'abord les questions du type (A). Une question de ce type se présente donc comme une phrase affirmative, la seule différence étant marquée par la présence d'un argument interrogatif. C'est le traitement de cet argument qui va nous permettre de formuler la question.
Soit donc la question:

ex(1) Que voit Pierre?

Cette question peut se formuler à l'aide de l'énoncé:

Si Pierre voit x alors une réponse est x.

Ce qui se traduit donc par la formule:

$\forall x$ voit(Pierre,x) $\supset$ Réponse(x)

où Réponse(x) est un prédicat particulier signifiant "la réponse est x".
Cela peut alors être obtenu en associant à l'argument signifié par un pronom interrogatif le schéma suivant:

$\forall x$ $(\varphi(x) \wedge \beta(x)) \supset$ Réponse(x)

où $\varphi(x)$ représente les propriétés descriptives de l'argument, $\beta(x)$ la phrase après avoir substitué à l'argument la variable x.
Ainsi par exemple la phrase:

ex(2) Quel homme est venu?

se traduit par:

$\forall x$ (homme(x) $\wedge$ est venu(x)) $\supset$ Réponse(x)

où $\varphi(x)$ est alors représenté par homme(x) et $\beta(x)$ par est venu(x).

Au niveau de l'analyse le pronom interrogatif va avoir une structure analogue à celles des autres arguments, mais nous lui associerons un quantificateur spécial noté INTERO.
Ainsi la phrase (2) est représentée par:

INTERO x(x est homme . x est un) est venu.

c'est à dire que nous avons donc la structure:

INTERO x($\varphi(x)$) $\beta(x)$

que l'on va traduire en logique par:

$$\forall x\ [\varphi(x) \wedge \beta(x)] \supset \text{Réponse}(x)$$

ce qui signifie donc que les arguments apparaissant dans $\varphi(x)$ et $\beta(x)$ vont être traités en tenant compte de la négation introduite par l'implication.

ex(3) Quelle personne possède une voiture?

Cette phrase est représentée par:

INTERO x (x est personne) possède UN y (y est voiture)

Il est bien clair que une voiture ne pose pas ici l'existence d'une voiture particulière. Considérons donc la construction de la formule:

$$\forall x\ (\text{personne}(x) \wedge \text{"x possède un y(y est voiture)"}) \supset \text{Réponse}(x)$$

La phrase "x possède un y(y est voiture)" est alors traduite, ce qui donne:

$$\forall x\ (\text{personne}(x) \wedge \exists y(\text{voiture}(y) \wedge \text{possède}(x,y))) \supset \text{Réponse}(x)$$

soit

$$\forall x \forall y\ (\text{personne}(x) \wedge \text{voiture}(y) \wedge \text{possède}(x,y)) \supset \text{Réponse}(x)$$

Ce qui va alors se traduire par la clause:

$$\neg\text{homme}(x) \vee \neg\text{voiture}(y) \vee \neg\text{possède}(x,y) \vee \text{Réponse}(x)$$

D'une façon analogue, dans la question:

Quel homme possède chaque document.

Le quantificateur associé à chaque document va être un quantificateur existentiel:

$$\forall x\ (\text{homme}(x) \wedge \text{"}_x \text{ possède chaque document"}) \supset \text{Réponse}(x)$$

$$\forall x\ (\text{homme}(x) \wedge \forall y(\text{document}(y) \supset \text{possède}(x,y))) \supset \text{Réponse}(x)$$

$$\forall x \exists y\ (\text{homme}(x) \wedge (\text{document}(y) \supset \text{possède}(x,y))) \supset \text{Réponse}(x)$$

ce qui va alors se traduire par les clauses:

$$\neg\text{homme}(x) \vee \text{document}(F(x)) \vee \text{Réponse}(x).$$

$$\neg\text{homme}(x) \vee \neg\text{possède}(x,F(x)) \vee \text{Réponse}(x).$$

L'argument défini dans une question se comporte comme nous l'avons déjà vu. On doit en effet considérer qu'il pose une existence et il doit de ce fait être traité avant les autres arguments:

ex(4) Qui a acheté la maison de Paul?

Cette question nous donne l'assurance que la maison de Paul existe, et cela doit être traduit en posant l'existence et l'unicité de l'objet décrit. Cette information peut-être dans certains cas redondante ou bien contraire aux informations que nous avons déjà. Il s'agit là d'un problème général qui concerne toute nouvelle information donnée par le locuteur.
On peut considérer que la question n'introduit pas d'objet nouveau et que dans ce cas la maison de Paul n'est qu'une référence vers un objet déjà mentionné dont on sait alors qu'il est unique. On peut alors traiter la question comme:

Qui a acheté une maison de Paul?

et la traduire par:

$\forall x \forall y$ (maison de Paul(y) $\wedge$ a acheté(x,y) $\supset$ Réponse(x)

Mais considérons les deux questions suivantes:

Qui a réussi l'examen?

Qui l'a raté?

Ces deux questions sont liées entre elles par le pronom personnel "le". Mais la réponse à la deuxième question peut être obtenue quelque soit la réponse à la première. C'est à dire qu'il n'est pas nécessaire ici d'avoir répondu à la première question pour pouvoir répondre à la seconde. Or si nous traitons "l'examen" à partir d'un schéma universel, nous allons donc devoir écrire.

$\forall x \forall y$ (examen(y) $\wedge$ réussi(x,y)) $\supset$ Réponse(x).

$\forall x \forall y$ (examen(y) $\wedge$ raté(x,y)) $\supset$ Réponse(x).

C'est à dire que nous n'avons donc pas traité le pronom le simplement en le remplaçant par la variable associée à l'argument auquel il fait référence, mais en fait comme si nous avions énoncé la deuxième question sous la forme:

Qui a raté un examen?

ce qui n'est pas conforme au traitement habituel des pronoms.
Par contre si nous posons l'existence de cet examen avant de traiter les autres arguments de la phrase, comme c'est le cas pour le défini nous allons avoir successivement:

$$\exists x\ \text{examen}(x) \wedge \text{"Qui a réussi } x \text{ . Qui a raté } x\text{"}$$

$$\exists x\ \text{examen}(x) \wedge \forall y(\text{réussi}(y,x) \supset \text{Réponse}(y))) \wedge \forall z(\text{rate}(z,x) \supset \text{Réponse}(z))$$

ce qui est alors une représentation correcte des deux questions.
Par ailleurs nous pouvons remarquer qu'après avoir déterminé la variable associée à l'argument défini les deux questions sont alors indépendantes et constituent deux formules closes. En fait nous ne faisons intervenir dans la partie droite d'une implication associée à une question que le prédicat Réponse(x) et non ce qui suit dans le paragraphe. C'est à dire qu'une phrase ne peut être liée à une question que si elle fait référence à un argument défini de la question (ou à un nom propre). En effet considérons l'exemple suivant:

Qui a réussi un examen?

Qui l'a raté?

La deuxième question parait incorrecte. Le pronom "le" ne semble pas pouvoir être utilisé pour faire référence à un "examen".
Cette double question doit se formuler par:

Qui a réussi un examen?

Qui a raté un examen? (ou Qui en a raté un?)

où les deux questions sont alors indépendantes.
Considérons également la question:

ex(5) Quel homme possède la voiture qu'il aime?

Le défini dépend de "homme" auquel il fait référence par le pronom "il" et son existence ne peut donc être posée avant que l'argument "Quel homme" ait été traité. La formulation logique va alors être:

$$\forall x\ (\text{homme}(x) \wedge \exists y(\text{voiture}(y) \wedge \text{aime}(x,y) \wedge \text{possède}(x,y))) \supset \text{Réponse}(x)$$

soit

$$\forall x \forall y\ (\text{homme}(x) \wedge \text{voiture}(y) \wedge \text{aime}(x,y) \wedge \text{possède}(x,y)) \supset \text{Réponse}(x)$$

dans laquelle donc le quantificateur associé à l'argument défini est un quantificateur universel.

REMARQUE I

Le fait de sortir des définis quand ils posent une existence permet de rendre la similitude entre les deux questions:

Qui a réussi l'examen?

L'examen a été réussi par qui?

Ces deux questions se traduisent alors par la même formule.
Par contre les deux questions:

Qui a réussi un examen?

un examen a été réussi par qui?

seront traitées différenment. On peut tout de même considérer que la deuxième forme est inusitée et même incorrecte car elle peut être ambigüe.

REMARQUE II

Nous avons vu que nous considérons qu'un pronom ne peut faire référence à un argument d'une question que s'il fait référence à un argument défini ou à un nom propre.
En fait nous éliminons en faisant cela les questions du genre:

Quel homme est venu?

Que voulait-il?

qui se traduit par la formule:

$$\forall x((\text{homme}(x) \wedge \text{venu}(x)) \supset (\text{Réponse}(x) \wedge \text{"Que voulait x"}))$$

c'est à dire que la réponse à la deuxième question implique que l'on ait répondu à la première.

Les questions en est-ce que vont se traduire par une double formulation, l'une amenant la réponse OUI, l'autre la réponse NON.
La question:

Est-ce que Pierre a lu ce livre?

va se traduire à partir des deux énoncés:

Si Pierre a lu ce livre alors la réponse est oui.

Si Pierre n'a pas lu ce livre alors la réponse est non.

C'est à dire qu'une phrase ψ de ce type va alors être représentée par:

$$(\psi \supset \text{Réponse(oui)}) \wedge (\neg\psi \supset \text{Réponse(non)})$$

Considérons l'exemple:

ex(6) Est-ce que Pierre possède une voiture?

Cela va donc s'énoncer sous la forme:

$$(\text{(Pierre possède une voiture)} \supset \text{Réponse(oui)}) \wedge$$
$$(\neg\text{(Pierre possède une voiture)} \supset \text{Réponse(non)})$$

La phrase "Pierre possède une voiture" se traduisant alors comme nous l'avons vu par:

$$\exists x \ \text{voiture}(x) \wedge \text{possède}(\text{Pierre}, x)$$

Ce qui donne donc la formule:

$$((\exists x \ \text{voiture}(x) \wedge \text{possède}(\text{Pierre}, x)) \supset \text{Réponse(oui)}) \wedge$$
$$(\neg(\exists x \ \text{voiture}(x) \wedge \text{possède}(\text{Pierre}, x)) \supset \text{Réponse(non)})$$

Ce qui est alors équivalent à:

$$\forall x \ (\text{voiture}(x) \wedge \text{possède}(\text{Pierre}, x)) \supset \text{Réponse(oui)}$$

$$\exists x \ (\neg\text{voiture}(x) \vee \neg\text{possède}(\text{Pierre}, x)) \supset \text{Réponse(non)}$$

et se traduit par les clauses:

$\neg$voiture(x) $\vee \neg$ possède(Pierre,x) $\vee$ Réponse(oui)

voiture(A) $\vee$ Réponse(non)

possède(Pierre,A) $\vee$ Réponse(non)

Nous allons devoir ici aussi, avant de formuler la question sous cette forme, affirmer l'existence des objets décrits par les arguments définis si ceux-ci ne possèdent pas dans leur définition de pronom faisant référence à un objet non encore défini.
La question:

Est-ce que Pierre connait le résultat?

se traduit par:

$$\exists x \text{ résultat}(x) \wedge (\text{connait}(\text{Pierre},x) \supset \text{Réponse}(\text{oui})) \wedge (\neg \text{connait}(\text{Pierre},x) \supset \text{Réponse}(\text{non}))$$

ce qui signifie donc:

Il existe un résultat (et un seul) et si Pierre le connait alors la réponse est oui

et si Pierre ne le connait pas alors la réponse est non.

II.5 TRADUCTION DU VERBE ETRE

Nous avons traité le verbe être comme un verbe quelconque en traduisant une relation

x est y

par le prédicat

est(x,y)

La phrase: Pierre est un homme.

se traduit par:

$\exists x$ homme(x) $\wedge$ est(Pierre,x)

Considérons alors les deux phrases:

Pierre est médecin

Pierre est un médecin.

Ces deux phrases demandent une analyse différente. Dans le premier cas le verbe est "est médecin", et la phrase énonce une propriété sur Pierre qui est "être médecin". La phrase va alors se traduire par:

est médecin(Pierre)

C'est également le cas des phrases:

Pierre est venu.

Pierre est malade.

La seconde phrase au contraire met en relation deux individus x et y, l'un signifié par "Pierre", l'autre par "un médecin".
On pourrait traduire cela en considérant que le verbe être énonce toujours une propriété sur Pierre et affirmer donc simplement:

est médecin(Pierre)

Cette façon de faire ne rend pas exactemnt compte de l'intention de la phrase. En effet quand nous disons:

Pierre est un médecin.

nous introduisons en fait un nouvel individu qui a la propriété d'être médecin et qui est mis en relation avec Pierre. Considérons pour cela les phrases suivantes:

ex(1) La Chine est un pays surpeuplé.

Ce pays possède actuellement...

Nous avons dans la deuxième phrase l'argument Ce pays qui fait donc référence à "un pays surpeuplé". Cela s'analyse donc de la manière suivante:

La Chine est un x(x est pays surpeuplé) . x possède actuellement...

La traduction de l'exemple (1) étant alors:

$$\exists x \quad \text{pays surpeuplé}(x) \wedge \text{est}(\text{la Chine}, x) \wedge \text{possède}(x, \ldots)$$

en considérant que la relation est(x,y) est une relation d'égalité entre les deux ensembles x et y. Les propriétés de l'égalité permettant alors de déduire:

$$\text{pays surpeuplé}(\text{la Chine}) \wedge \text{possède}(\text{la Chine}, \ldots)$$

Nous avons donc ainsi un traitement du verbe être qui est le même que pour les autres verbes, et les déductions qui s'imposent avec ce genre de phrase seront obtenues en traduisant la relation est (x,y) par la relation d'égalité.

CHAPITRE III

DESCRIPTION DU PROGRAMME

Nous appelerons parfois procédures un certain ensemble de clauses commençant par le même littéral. En effet la démonstration d'un littéral sur ces clauses peut être considérée comme analogue à l'exécution d'une procédure.

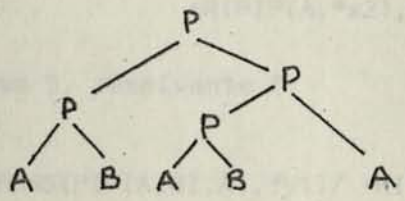
Avant de décrire les principales procédures utilisées dans le programme PROLOG, nous allons donner un exemple d'un type de prédicat fondammental qui montre comment réaliser, dans certains cas, les transformations sur un arbre.

III.1 EXEMPLES DE PROCEDURE DE TRANSFORMATION

1°) Considérons par exemple la grammaire Context-Free suivante:

<arbre> :: = P(<arbre>,<arbre>) | A | B

Nous voulons alors transformer un arbre de ce type, soit:



en substituant à chaque occurence du symbole B le symbole C.
Pour cela nous allons définir un prédicat TRANS(x,y) qui signifie que "le transformé" de x est y. Il est défini de la façon suivante:

```
1  +TRANS(P(*x,*y),P(*x1,*y1)) -TRANS(*x,*x1)
                               -TRANS(*y,*y1) ..

2  + TRANS (A,A) ..

3  + TRANS (B,C) ..
```

Ces trois clauses définissent la procédure TRANS. Résoudre la clause suivante va correspondre à appeler cette procédure:

```
4   - TRANS (P(P(A,B),P(P(A,B),A)),*x)/ +R(*x) ..
```

Cette règle signifie: si le transformé de l'arbre est *x alors la réponse est *x. La résolution du littéral -TRANS(x,y) va en fait consister à unifier y avec le transformé de x, ce qui à la"sortie"

de la procédure TRANS va nous permettre de récupérer dans y la valeur du transformé de x. Examinons le "déroulement" du programme qui est ici déterministe.

4 s'unifie avec 1: résolvante 5:

5: -TRANS(P(A,B),*x1) -TRANS(P(P(A,B),A),*Y1)/
+R(P(*x1,*y1)) ..

5 s'unifie avec 1, résolvante 6:

6: -TRANS(A,*x1) -TRANS(B,*x2) -TRANS(P(P(A,B),A),*y1)/
+R(P(P(*x1,*x2),*y1)) ..

6 s'unifie avec 2, résolvante 7:

7: -TRANS(B,*x2) -TRANS(P(P(A,B),A),*y1)/
+R(P(P(A,*x2),*y1)) ..

7 s'unifie avec 3, résolvante 8:

8: -TRANS(P(P(A,B),A),*y1)/ +R(P(P(A,C),*y1))..

8 s'unifie avec 1, résolvante 9:

9: -TRANS(P(A,B),*x1) -TRANS(A,*x2)/ +R(P(P(A,C),P(*x1,*x2)))..

9 s'unifie avec 1, résolvante 10:

10: -TRANS(A,*z1) -TRANS(B,*z2) -TRANS(A,*x2)/
+R(P(P(A,C),P(P(*z1,*z2),*x2)))..

10 s'unifie avec 2, résolvante 11:

11: -TRANS(B,*z2) -TRANS(A,*x2)/ +R(P(P(A,C),P(P(A,*z2),*x2)))..

11 s'unifie avec 3, résolvante 12:

12: -TRANS(A,*x2)/ +R(P(P(A,C),P(P(A,C),*x2)))..

12 s'unifie avec 2, résolvante 13:

13: +R(P(P(A,C),P(P(A,C),A)))..

Dans cet exemple nous avons remplacé le symbole B par un autre symbole C. Il est bien clair que A,B et C peuvent en fait être des arbres d'un autre type, et qu'on peut également prévoir l'appel à d'autres procédures réalisant des transformations secondaires sur ces arbres.

2°) Supposons que nous voulions faire la transformation précédente mais que, au lieu de remplacer B par C, nous voulions remplacer B par un terme F(I),I étant le nombre de noeuds P rencontrés en balayant l'arbre.
La procédure va s'écrire alors en utilisant un "compteur" permettant de compter le nombre de noeuds P rencontré. Cela va être réalisé en utilisant l'opérateur successeur noté ' (apostrophe), défini comme opérateur unaire gauche-droite. Le successeur d'un nombre I est alors noté I', l'initialisation de ce nombre se faisant à 0:

1 +TRANS(P(*x,*y),P(*x1,*y1),*I,*II) -TRANS(*x,*x1,*I',*I1)
-TRANS(*y,*y1,*I1,*II) ..

2 +TRANS (A,A,*I,*I) ..

3 +TRANS (B,F(*I),*I,*I) ..

si nous résolvons sur ces trois règles la clause:

4 -TRANS (P(P(A,B),P(P(A,B),A)),*x,0,*I)/ +R(*x) ..

nous récupérons en réponse:

+R (P(P(A,F(0")),P(P(A,F(0"")),A))) ..

Le prédicat TRANS(x,y,I,J) défini ici signifie donc que y est le transformé de x et J le transformé de I.
x et I doivent être considérés comme les valeurs initiales, les valeurs à l'appel, et y et J comme les valeurs à la sortie de la procédure. En fait on peut considérer que la procédure ne possède que deux arguments, et utiliser un opérateur binaire / permettant de

regrouper en un seul argument les deux variables désignant l'arbre à appel et l'arbre à la sortie. Nous écrirons alors:

TRANS (x/y,I/J)

L'utilisation de ces couples de variables, permet en fait de lier les littéraux des clauses générées, au cours de la démonstration, par une variable PROLOG.

3°) Considérons maintenant le problème précédent, mais en substituant à B un terme F(I), I étant le nombre de noeuds P dominant B.
La procédure va donc se définir ainsi:

1 + TRANS (P(*x,*y),P(*x_1,*y_1),*I) -TRANS(*x,*x_1,*I')
-TRANS(*y,*y_1,*I') ..

2 + TRANS (A,A,*I) ..

3 + TRANS (B,F(*I),*I) ..

Si nous résolvons sur ces axiomes la clause:

-TRANS (P(P(A,B),P(P(A,B),A)),*x,0) / +R(*x) ..

nous allons récupérer la réponse:

+R (P(P(A,F(0')),P(P(A,F(0''')),A))) ..

Nous utilisons, pour traduire un texte en un ensemble d'énoncées logiques, ce type de prédicat "transformé". Le texte est représenté par une arborescence type structure profonde de Chomsky que nous appelons structure syntaxique du texte, et nous représenterons les énoncés logiques également à l'aide d'une arborescence, nommée structure clausale. Nous allons décrire dans ce qui suit la syntaxe de ces deux structures.
Le programme décrit par la suite aura pour but de réaliser des transformations sur la structure syntaxique d'un texte afin d'obtenir la structure clausale. Une seconde étape consistera à générer à partir de cette dernière structure les clauses PROLOG correspondantes.

III.2 STRUCTURE SYNTAXIQUE D'UN TEXTE

Cette structure doit nous faciliter l'accès aux informations nécessaires aux transformations futures, tout en permettant la représentation d'un ensemble assez large de phrases.
Nous donnons ici les règles BNF définissant la syntaxe de cette structure en explicitant les règles et en dégageant leur contenu sémantique.

Texte - Paragraphe

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

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

Un texte est une liste de prargraphes. La notion de paragraphe a pour but de grouper une liste de phrases reliées entre-elles par des pronoms. Nous avons vu en effet que cette liaison jouait un role important au niveau de la formation des formules logiques. Ainsi, la portée des quantificateurs sera limitée à un paragraphe et non étendue à toutes les phrases du texte.
Les paragraphes sont construits de la façon suivante:

- -Une phrase contenant un pronom faisant référence à un argument d'une phrase précédente est placée dans le même paragraphe que cette dernière.
- -Une phrase ne contenant pas de pronom faisant référence à un argument d'une autre phrase débute un nouveau prargraphe.

Nous avons vu par ailleurs que l'ordre des phrases devait être respecté, cet ordre intervenant dans la formation des formules. Cette règle, alliée à la construction des paragraphes, suppose donc que nous n'avons pas de structures du genre:

$$\ldots P_1 . P_2 \ldots$$

où P_1 et P_2 sont deux phrases successives, P_2 ayant une référence vers une phrase qui précède P_1, et P_1 n'ayant pas de référence vers une phrase précédente.

Phrase

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

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

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

La phrase est l'élément principal du paragraphe. Les phrases se divisent en deux groupes:

- -les phrases déclaratives pour lesquelles le type phrase est alors AFF.
- -les phrases interrogatives que l'on divise en deux types:
  - -INTERO pour les interrogatives dont la question porte sur un argument, c'est à dire dont un des arguments est un pronom interrogatif.
  - -OUINON pour les interrogatives dont la réponse est oui ou non.

Les types REST et APOS correspondent aux propositions relatives. Le premier trait signale que la relative est restrictive, le second que la relative est une appositive.
Les syntagmes prépositionnels sont représentés sous forme d'une liste. Cela permet d'avoir un nombre quelconque d'arguments tout en conservant leur ordre.
Un des éléments de la liste peut être noté VERBE. Il s'agit là d'une marque permettant de diviser la liste en deux:

- -les syntagmes prépositionnels situés avant le verbe. Ce sont ceux qui précèdent la marque VERBE dans la liste.
- -les syntagmes prépositionnels situés après le verbe. Ce sont ceux qui suivent la marque VERBE dans la liste.

Si la liste ne contient pas cette marque, les arguments sont alors tous supposés être avant le verbe.

Syntagme prépositionnel

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

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

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

Le syntagme prépositionnel permet d'associer au syntagme nominal la préposition qui l'introduit dans la phrase, ce qui nous permet de définir la fonction de l'argument dans la phrase. Donc <prep> désigne la préposition qui introduit l'argument. Dans le cas où l'argument ne comporte pas de préposition, c'est le cas du sujet, de l'objet et de certains arguments de temps, nous noterons cette préposition respectivement SUJ,OBJ ou TEMPS.
Nous avons prévu une liste de prépositions pour exprimer certaines ambigüités. En effet une question du type:

où est Paul?

ne précise pas la préposition qui introduit la réponse, mais simplement le type de préposition, qui doit introduire un "lieu".

Nous noterons alors dans <liste prep> la liste des prépositions pouvant introduire un argument de type lieu. Cette liste dans ce cas est: A,APRES,AVANT,DANS,EN,SOUS,VERS.
De même pour les arguments de type temps, la liste des prépositions pouvant les introduire est: A,APRES,AVANT,DANS,DE,DURANT,ET,PENDANT, TEMPS,VERS.

Syntagme nominal

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

<nom prop.ou indice> ::=ASTER.<nom propre>.<genre>.<nombres>|
ASTER.<nom propre>|<variable>

<genre> ::=MAS | FEM | <variable>

<nombre> ::=PLU ! 1

<nom propre> ::="Tout nom propre"

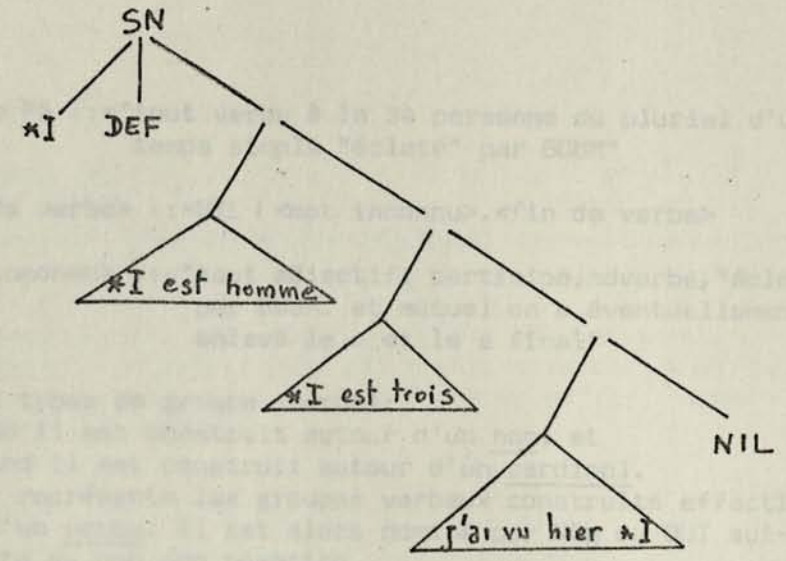
<quantif.> ::=DEF | INDEF | CERTAIN | CHAQUE | AUCUN | PROP|
PRON | INTERO

<variable> ::="Toute variable telle qu'elle est définie en
PROLOG,c'est à dire grossièrement* suivi d'un mot"

Le syntagme nominal est l'élement principal de la phrase en ce sens que c'est à partir de lui que vont se faire les transformations.
Le <nom prop.ou indice> correspond à l'indice, ou nom, associé à chaque argument: ce nom est une variable PROLOG *I ou un nom propre, précédé de ASTER et éventuellement suivi de genre et du nombre de l'article défini qui l'ontroduit. *G marque un genre indéterminé (LES).
Nous avons déjà vu le rôle du quantificateur <quantif> . Nous en avons ici deux autres: PROP et PRON, qui sont utilisés pour avoir une structure homogène à celles des autres arguments, dans le cas d'arguments signifiés par des noms propres ou des pronoms.
Le troisième argument du SN, <paragraphe>, represente la liste des propositions relatives. Nous avons une structure de paragraphe car les phrases sont toutes liées entre elles par la présence du pronom relatif. L'argument:

Les trois hommes que j'ai vu hier

sera représenté par l'arbre:



le type de chacune des propositions relatives étant comme nous l'avons vu soit REST soit APOS. La relative associée au nom (*I est homme) est évidemment une relative de type restrictif et sera de type REST. La relative associée au cardinal (*I est trois) semble devoir être considérée, de par son traitement, comme une appositive. Nous lui associerons donc le type APOS.

Groupe verbal

<groupe verbal> ::=<oui>{(<verbe S>.<verbe P>).<fin de verbe>}|
NOM(<genre>.<nom commun S>.<nom commun P>)|
CARD(<cardinal>)

<oui> ::=OUI | NON

<genre> ::=FEM | MAS |<variable>

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

<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 3è personne du singulier d'un temps simple "éclaté" par BOUM"

<verbe P> ::="tout verbe à la 3è 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 BOUM, et auquel on a éventuellement enlevé le e et le s final"

Il y a donc trois types de groupe verbal:
le type NOM, quand il est construit autour d'un nom, et
le type CARD, quand il est construit autour d'un cardinal.
Le troisième type représente les groupes verbaux construits effectivement à partir d'un verbe. Il est alors dominé par NON ou OUI suivant qu'il comporte ou non une négation.
Il est bien clair que dans les deux premiers cas, la phrase n'a alors qu'un seul argument, le sujet, et que le verbe ne peut être négativé.
Nous n'acceptons que les phrases comportant des arguments à la troisième personne. Les pronoms je, tu, nous, vous ne sont pas permis. De même les verbes doivent être à la troisième personne. De plus les groupes verbaux du troisième type,sont formés du verbe et de tout ce qui suit le verbe s'il ne s'agit pas d'un argument.

ex Pierre est médecin.

Pierre court vite.

D'une manière générale ce qui joue le rôle de verbe est composé d'un verbe suivi d'infinitif, d'adverbes, participes, adjectifs ou noms qui ne forment pas un argument.
Nous avons introduit au niveau du groupe verbal une structure faisant apparaitre les mots au singulier et ces mêmes mots au pluriel.
Cela est nécessaire pour établir la relation entre un verbe (ou un nom) au pluriel et ce même verbe (ou ce nom) au singulier dans le cadre des déductions à faire par la suite sur le texte.
Cette manière de faire nous permet actuellement d'avoir un minimum de règles morphologiques, ces règles étant alors assez simples car elles ne font que donner le pluriel des verbes ou des noms communs, à partir du singulier ou inversement.
Il est bien évident qu'un système plus complet devra avoir des règles morphologiques plus élaborées faisant intervenir un dictionnaire.
Nous ne faisons pas de différence, quant à la structure entre les deux phrases:

ex Pierre est médecin.

Pierre est venu.

car nous ne pouvons pas déterminer que "médecin" est un nom et "venu" un participe. Il est nécessaire pour cela d'avoir accès à un dictionnaire.
Nous disposons actuellement d'un analyseur écrit en PROLOG qui transforme un texte écrit en français dans cette structure dite structure syntaxique. Les règles de morphologie sur les noms et les verbes sont également écrites en PROLOG.

REMARQUE

La fonction de l'argument est signalée par la préposition qui l'introduit. En fait nous avons adopté pour l'instant cette façon de faire car il est souvent complexe sans dictionnaire de déterminer la fonction d'un argument et il peut y avoir des ambigüités. Il serait nécessaire d'avoir des traits du type humain, non humain, abstrait, etc... Nous pensons qu'avec l'aide de ces traits, la fonction de l'argument pouvant être obtenue de façon plus précise, il est alors préférable de la signaler explicitement à l'aide du type de la préposition, par exemple avec une structure du genre:

SP(<type de prep>,<prep>,<synt.mominal>)

La préposition serait alors NIL dans le cas du sujet, de l'objet ou de certains arguments de temps, la marque sujet, objet, ou temps étant alors introduit dans <type de prep>.
Dans le cas d'une question comme:

où est Paul?

le type de la préposition introduisant l'argument réponse est alors Lieu, et, la préposition étant alors indeterminée serait représentée par une variable PROLOG *P.

III.3 STRUCTURE CLAUSALE D'UN TEXTE

Pour représenter un texte sous forme d'un ensemble de clauses, nous allons donc devoir définir un certain nombre de prédicats nous permettant de traduire les propriétés et relations énoncées par le texte.
Nous allons, avant d'engendrer les clauses PROLOG, représenter le texte à l'aide d'une arborescence qui nous permet de représenter les clauses par un arbre en tenant compte du fait que la construction de cet arbre se fait à partir des schémas:

$A \wedge B$

ou

$A \supset B$.

Nous allons donc décrire la syntaxe de cette structure à l'aide des règles BNF en précisant par la suite la signification des "prédicats".

<texte> ::=<formule> ET <formule>|<formule>IMP<formule>

<formule> ::=NO<formule> | (<texte>) | <prédicat>

<prédicat> ::=<P.phrase>|<P.inclusion>|<P.égalité>|<P.dans>|
<P.réponse>|<P.être>|<P.cardinalité>|
<P.nb.de question>| NIL

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

<P.inclusion> ::=Q(<nom>,<nom>)

<P.égalité> ::=EG(<nom>,<nom>)

<P.être> ::=ETRE(<nom>,<genre>,<nom commun S>.<nom commun P>)

<P.dans> ::=DS(<prep.>,<liste de prep.>)

<P.réponse> ::=R(<indice>,<réponse>)

<P.cardinalité> ::=CARD(<nom>,<cardinalité>)

<P.nb.de question> ::=NQ(<indice>)

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

<SP> ::=SP(<prep>,<nom>)

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

<nom> ::=<variable>|ASTER.<nom propre>.<genre>.<nombre>|
ASTER.<nom propre>|F(<indice>,<liste de nom>)

<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.>, <variable>)

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

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

<genre> ::=FEM | MAS |<variable>

<nombre> ::=PLU | 1

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

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

<verbe P> ::="tout verbe à la 3è 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 éven-
tuellement enlevé le e et le s final"

<variable> ::="Toute variable telle qu'elle est définie en
PROLOG,c'est à dire grossièrement *suivi
d'un mot".

Nous avons là une représentation des formules logiques à partir des opérateurs ET,IMP,NO qui sont respectivement la conjonction, l'implication et la négation. Les prédicats représentés dans cette structure se définissent comme suit.

Prédicat phrase

Ce prédicat est un prédicat associé au verbe d'une phrase. Considérons en effet la phrase:

Pierre aime Marie.

cela se traduit si on considère le verbe aime comme un prédicat par:

aime(Pierre,Marie)

Nous allons utiliser un prédicat universel P et considérer le verbe comme un argument de ce prédicat:

P(Pierre.Marie.NIL,oui(aime))

les arguments étant alors représentés par une liste en définissant le point (.) comme un opérateur.
L'écriture des arguments se faisant dans n'importe quel ordre, nous allons utiliser également la notion de syntagme prépositionnel pour définir leurs fonctions:

P(SP(suj,Pierre).SP(obj,Marie).NIL,oui(aime))

Le <nom> associé aux arguments est ici Pierre et Marie car il s'agit de noms propres. Pour d'autres types d'arguments nous aurions un <nom> qui serait:

- une variable PROLOG *N dans le cas d'argument représenté par une variable universelle.
- un terme F(<indice>,<liste de nom>) dans le cas d'argument représenté par une variable existentielle. Il s'agit alors d'une représentation des fonctions de Skölem à l'aide d'une fonction universelle F que l'on définit alors par:

F(I,x) =I(x)

où I est effectivement la fonction de Skölem.
I est un nombre associé à chaque variable existentielle, nombre construit à l'aide de l'opérateur successeur '.
La <liste de nom> x désigne la liste des variables universelles dont dépend la fonction de Skölem. Ce sera donc une liste de <nom> du type *N.

Prédicat inclusion et égalité

Le prédicat Q est utilisé pour exprimer l'inclusion entre ensembles, le prédicat EG pour exprimer l'égalité entre ensembles.

Q(x,y) signifie "x est inclus dans y"

$EG(x,y) \equiv Q(x,y) \wedge Q(y,x)$

Le prédicat EG est utilisé pour traduire les phrases dont le verbe est le verbe être. Nous avons en fait l'équivalence:

P(SP(suj,x).SP(obj,y).NIL,oui(être)) ≡ EG(x,y)

De même si on considère est dans comme un verbe spécial exprimant l'inclusion, on peut énoncer l'équivalence:

P(SP(suj,x).SP(obj,y).NIL,oui(est dans)) ≡ Q(x,y)

Prédicat être

Ce prédicat va être utilisé pour traduire les phrases dont le verbe est un nom. Nous avons vu que, par exemple, la phrase:

Un homme vient.

se traduit par la formule:

$homme(x) \wedge vient(x)$

Pour représenter la propriété homme(x) nous allons donc utiliser le prédicat ETRE

ETRE(x,homme) ≡ P(SP(suj,x).NIL,NOM(homme))

Prédicat cardinalité

Comme le prédicat ETRE, le prédicat exprimant la cardinalité d'un ensemble va permettre de traduire les phrases dont le verbe est du type card.
Ainsi la phrase:

P(SP(suj,x).NIL,card(I))

s'exprimera à l'aide de:

CARD(x,I)

En fait celui-ci s'énonce sous deux formes:

CARD(x,E(I)) qui signifie "la cardinalité de x est I"

CARD(x,S(I)) qui signifie "la cardinalité est supérieure à I".

Cette deuxième forme est utilisée pour traduire la cardinalité des ensembles décrits par un argument ayant une marque de pluriel indéfini.

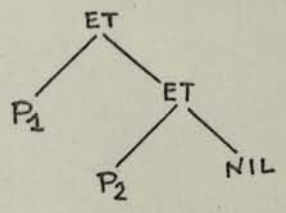
Prédicat DS,NQ et R

Le prédicat DS(x,L) signifie que x est un élément de la liste L.
Il est utilisé dans la structure clausale pour exprimer qu'une préposition P doit appartenir à un ensemble donné de prépositions.
Le prédicat NQ(I) signifie que le nombre de phrases interrogatives apparaissant dans le texte est I-1. Il s'agit là d'un prédicat particulier permettant de répondre aux questions dans l'ordre où elles sont posées en arrêtant la recherche de réponse quand toutes les questions ont été traitées.
Le prédicat R(I,x) est un prédicat signifiant "la réponse à la Iè question est x".

La réponse x peut être de deux types:
- -oui ou non pour les questions du type "est ce que"... qui interrogent sur la réalisation d'un fait.
- -un argument SP(<prep>,*x) pour les autres types de question, qui interrogent sur un argument.

REMARQUE

Nous avons mentionné un prédicat particulier NIL. Cela correspond à un prédicat à 0 arguments, c'est à dire à une valeur logique qui est ici la valeur VRAIE. Son utilisation permet de conserver à la structure clausale une structure de liste.
Ainsi:



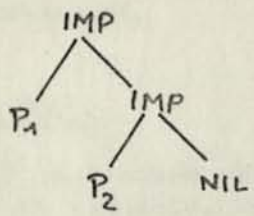
correspond à la formule:

$$P_1 \wedge P_2 \wedge NIL$$

où P_1 et P_2 représentent des prédicats et NIL la valeur VRAIE, ce qui est alors équivalent à:

$$P_1 \wedge P_2$$

De même



représente donc la clause:

$$P_1 \supset (P_2 \supset NIL)$$

soit

$$\overline{P_1} \vee \overline{P_2} \vee NIL$$

Cette clause ayant la valeur VRAIE, n'est pas générée et le système obtenu est équivalent.
En fait l'opérateur ET et l'opérateur . jouent le même rôle, et nous pouvons passer de ET en le remplaçant par.
C'est ce qui est fait dans l'écriture des règles PROLOG constituant le programme.

III.4 DESCRIPTION DES PROCEDURES PRINCIPALES

Le programme utilisé est construit principalement autour de trois procédures récursives qui ont pour nom TTEXTE, TPARAGRAPHE et TPHRASE et qui réalisent respectivement la transformation d'un texte, d'un paragraphe et d'une phrase. Ces trois procédures sont des procédures réalisant des transformations sur la structure syntaxique d'un texte pour obtenir la structure clausale de ce même texte, et vont par conséquent s'écrire d'une façon similaire à la procédure TRANS décrite précédemment.

III.4.1 Procédure TTEXTE

La procédure TTEXTE va transformer la structure syntaxique d'un texte en sa structure clausale. Nous allons pour définir cette procédure utiliser le prédicat TTEXTE suivant:

TTEXTE (I/I1,J/J1,T/P)

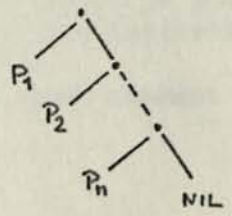
dans lequel / est un opérateur binaire droite-gauche, et où les "couples" d'arguments désignent respectivement:

- -I et I1 des termes construits à l'aide de l'opérateur successeur ' (unaire gauche-droite) et de la constante 0, permettant de compter le nombre de phrases interrogatives.
- -J et J1 sont des termes ayant la même structure et permettant de compter les variables existentielles.
- -T désigne la structure syntaxique d'un texte et P la structure clausale d'un texte.

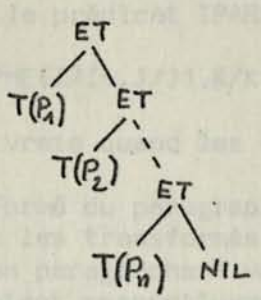
Le prédicat TTEXTE précédent prend la valeur vraie si les transformés de I,J,et T sont respectivement I1,J1 et P. C'est à dire en fait si P est le transformé du texte T dans lequel il y a I1-I questions et J1-J arguments quantifiés existentiellement.
Nous allons définir ce prédicat à partir du prédicat TPARAGRAPHE qui donne le transformé d'un paragraphe. Nous avons vu qu'un texte est en fait une liste de paragraphes:

$P_1.P_2...P_n.$ NIL

que nous représentons par la structure de "peigne" suivante en ayant défini "." comme opérateur binaire droite-gauche:



Chaque paragraphe correspond à une formule close en ce sens que chaque argument apparaissant dans un paragraphe n'est "défini" que dans ce paragraphe. Tout pronom dans un paragraphe ne fait référence qu'à un argument du paragraphe. Les paragraphes vont alors être traités indépendamment les uns des autres. Si nous appelons $T(P_i)$ le transformé du paragraphe P_i (pour tout i), la structure clausale du texte précédent est alors:



que nous écrirons:

$$T(P_1)\ ET\ T(P_2)\ ET \ldots T(P_n)\ ET\ NIL$$

en définissant ET comme opérateur binaire droite-gauche. La procédure TTEXTE est alors décrite par les clauses PROLOG suivantes:

```
+TTEXTE(*I/*II,*J/*JJ,*P.*T/*TP ET *TT)

        -TPARAGRAPHE(*I/*I1,*J/*J1, NIL/*K1,0/NIL,*P/*TP)

        -TTEXTE(*I1/*II,*J1/*JJ,*T/*TT) ..

+TTEXTE(*I/*I,*J/*J,NIL/NIL) ..
```

La première règle signifie que le transformé d'un texte *P.*T, où *P est le premier paragraphe du texte, est *TP ET *TT si *TP est le transformé du paragraphe *P et,*TT le transformé du texte *T. Le nombre de question est alors *II-*I si le nombre de questions dans le paragraphe *P est *I1-*I et si le nombre de question dans le texte *T est *II-*I1. De même pour le nombre de variables existentielles. La deuxième règle est une règle terminale: le transformé d'un texte vide (NIL) est NIL, le nombre de questions étant alors 0=*I-*I et le nombre de variables existentielles apparaissant dans le texte étant également 0=*J-*J.
Nous allons voir maintenant comment est définie la procédure TPARAGRAPHE.

III.4.2 Procédure TPARAGRAPHE

Cette procédure transforme la structure syntaxique d'un paragraphe en sa structure clausale. Comme nous l'avons vu un paragraphe est une liste de phrases reliées entre elles par un ou plusieurs pronoms. Nous allons donc définir le transformé d'un paragraphe à partir du transformé d'une phrase.
Définissons d'abord le prédicat TPARAGRAPHE,suivant:

TPARAGRAPHE(I/I1,J/J1,K/K1,E/LV,P/TP)

qui prend la valeur vraie quand les conditions suivantes sont vérifiées:

-TP est le transformé du paragraphe P.
-I1,J1 et K1 sont les transformés des arbres I,J,K.

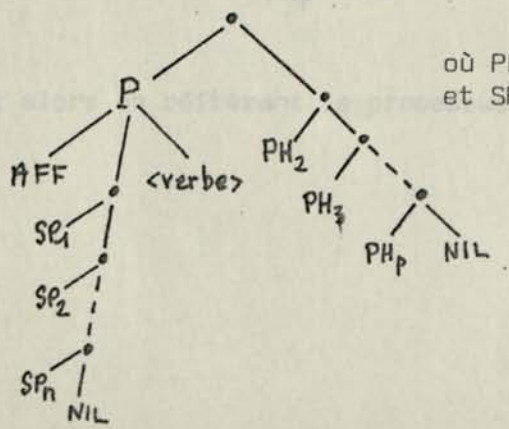
Le transformé TP d'un paragraphe P va se définir à partir des termes E et LV qui représentent respectivement:

E: l'état de la négation. La formule que nous allons générer peut être sous la portée d'une négation. Le transformé TP de P doit être obtenu en tenant compte de cette négation. Nous signalerons la présence de cette dernière en donnant à E la valeur 1. Dans le cas contraire E aura la valeur 0.

LV: la liste des variables universelles qui dominent la partie de la formule à construire. Cette liste est nécessaire pour déterminer les fonctions de Skölem.

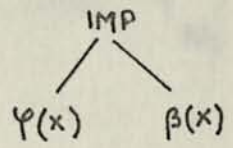
Nous avons déjà indiqué dans TTEXTE ce que représentent les couples (I,I1) et (J,J1), qui sont le "compteur de questions" et le "compteur de variables existentielles". I1 est le transformé de I si le nombre de questions du paragraphe P est I1-I.J1 est le transformé de J si le nombre de variables existentielles rencontrées dans P est J1-J.
Le terme K représente une liste de variables universelles. Le transformé K1 s'obtient de la manière suivante: chaque fois que nous rencontrons une variable universelle V dans la construction de TP, K se transforme en V.K. Donc K1 est en fait la concatenation de K et de la liste des variables universelles rencontrées dans P.
Le transformé d'un paragraphe P va être obtenu en transformant la première phrase du paragraphe, de la façon suivante:

-la première phrase du paragraphe est du type AFF

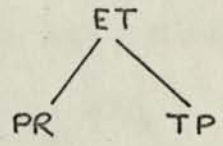


où PH_i représente une phrase
et SP^i_i un syntagme prépositionnel

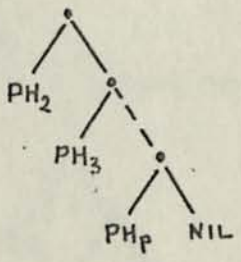
Nous allons alors transformer la première phrase à partir de ses arguments SP_i en associant à chacun d'eux un ou le schémas logique qui lui correspond. Supposons que le schéma logique associé à SP1 soit le schéma universel $\forall x \ \varphi(x) \supset \beta(x)$, le transformé de la structure précédente va être:



où $\varphi(x)$ est alors le transformé des relatives associées à SP1, et, $\beta(x)$ le transformé de la structure précédente à partir de SP2, dans laquelle chaque occurence de l'indice I associé à l'argument SP1 a été remplacé par une représentation de la variable universelle x.
Quand tous les arguments SP_i de la phrase ont été ainsi traités c'est à dire quand ils ont tous été quantifiés, le transformé de la structure précédente est alors:

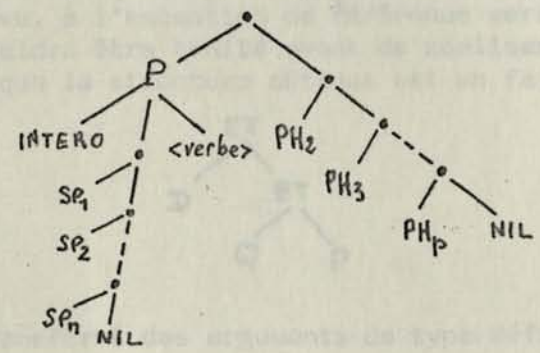


où PR est le prédicat traduisant la phrase en fonction du verbe, et TP le transformé du paragraphe:

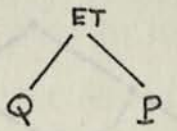


qui s'obtient alors en réitérant le processus.

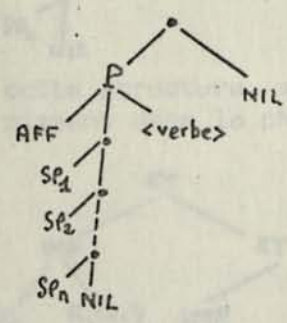
-La première phrase du paragraphe est du type INTERO



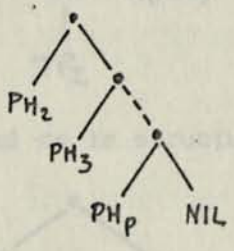
Le transformé de cette structure est alors:



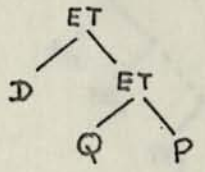
où Q est le transformé de la structure de paragraphe:



et P le transformé de la structure de paragraphe:

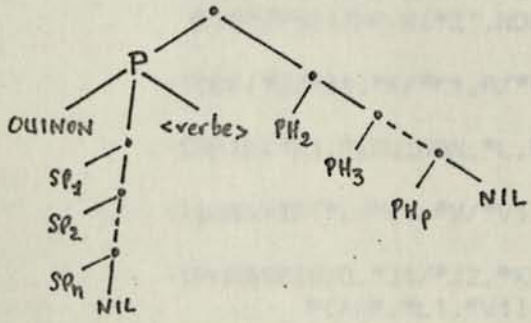


En effet, il n'y a pas, dans ce cas, de pronom dans $P_{H_2}, P_{H_3}, \ldots P_{H_p}$ faisant référence à un argument SP_i de la phrase interrogative comme nous l'avons vu, à l'exception de référence vers un argument de type DEF qui doit alors être traité avant de réaliser cette transformation. C'est à dire que la structure obtenue est en fait:

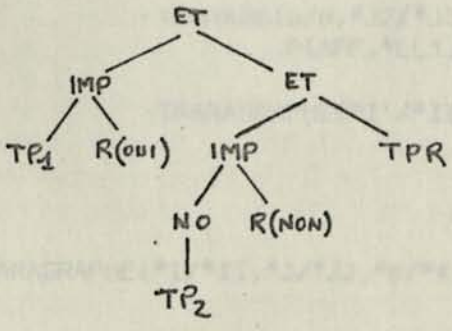


D étant le transformé des arguments de type défini apparaissant dans la phrase interrogative;

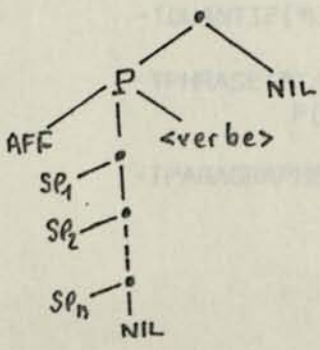
-La première phrase du paragraphe est du type OUINON



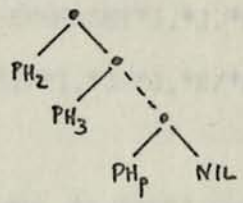
Le transformé de cette structure, après avoir traité les arguments de type DEF apparaissant dans la phrase interrogative, va être la structure:



où TP_1 est le transformé de la structure de paragraphe:



et TP_2 le transformé de cette même structure, mais en tenant compte que la formule générée est alors sous la portée d'une négation, ce qui va modifier la quantification des arguments. TPR est alors le transformé de la structure:



La procédure TPARAGRAPHE est décrite par les règles PROLOG suivantes:

```
+TPARAGRAPHE(*I/*II,*J/*JJ,*K/*KK,O/*VL,P(OUINON,*L,*V).*P/
              *DEF ET(*TPH1 IMP R(*I',OUI)) ET
              ((NO*TPH2)IMP R(*I',NON)) ET*TP)

-TDEF(*J/*J1,*K/*K1,O/*VL,*L/*DEF)

-COPIE(*K1.P(OUINON,*L,*V),*K1.P(OUINON,*LL,*V))

-TQUANTIF(*L/*L1,*V/*V1)

-TPHRASE(O/O,*J1/*J2,*K1/*K2,1/*VL,*L1,
         P(AFF,*L1,*V1).NIL/*TPH1)

-TQUANTIF(*LL/*LL1,*V/*V1)

-TPHRASE(O/O,*J2/*J3,*K2/*K3,O/*VL,*LL1,
         P(AFF,*LL1,*V1).NIL/*TPH2)

-TPARAGRAPHE(*I'/*II,*J3/*JJ,*K3/*KK,O/*VL,*P/*TP) ..


+TPARAGRAPHE(*I/*II,*J/*JJ,*K/*KK,O/*VL,P(INTERO,*L,*V).
                                    *P/*DEF ET*TPH ET*TP)

-TDEF(*J/*J1,*K/*K1,O/*VL,*L/*DEF)

-TQUANTIF(*L/*L1,*V/*V1)

-TPHRASE(*I/*I',*J1/*J2,*K1/*K2,O/*VL,*L1,
         P(AFF,*L1,*V1).NIL/*TPH)

-TPARAGRAPHE(*I'/*II,*J2/*JJ,*K2/*KK,O/*VL,*P/*TP) ..
```

```
+TPARAGRAPHE(*I,*J,*K,*E,P(*T,*L,*V).*P/*TP)

                -TQUANTIF(*L/*L1,*V/*V1)

                -TPHRASE(*I,*J,*K,*E,*L1,P(*T,*L1,*V1).*P/*TP) ..

+TPARAGRAPHE(*I/*I,*J/*J,*K/*K,*E,NIL/NIL) ..
```

dans lesquelles:

1°) TDEF(*J/*J1,*K/*K1,0/*VL,*L/*DEF) est un prédicat qui prend la valeur vraie quand les conditions suivantes sont vérifiées:

- *DEF est le transformé des arguments définis apparaissant dans *L. Cette transformation est fonction de "l'état de la négation" (ici elle est marquée par 0) et de la liste des variables universelles *VL
- *J1 et *K1 sont les transformés de *J et *K, c'est à dire si le nombre de variables existentielles créees est *J1-*J et si la liste *K1 est la liste des variables universelles générées, concaténée à *K.

*DEF correspond en fait à la traduction des relatives associées à l'argument de type défini, avec la propriété d'unicité. S'il y a plusieurs arguments définis, *DEF est la conjonction des formules traduisant les relatives associées à ces arguments. S'il n'y a pas d'arguments définis, *DEF prend alors la valeur NIL.

2°)COPIE est un prédicat évaluable décrit dans l'appendice II. IL nous permet ici de faire une copie de la phrase P tout en conservant les liens avec la liste *K des variables déjà générées.

3°)TQUANTIF(*L/*L1,*V/*V1) est un prédicat qui prend la valeur vraie quand les transformés de *L et *V sont *L1 et *V1. Ces transformations ont été définies dans le chapitre I et consistent à modifier les quantificateurs des arguments en fonction de la négation sur le verbe.

En fait nous n'appelons pas directement à partir de TPARAGRAPHE la procédure TPHRASE mais une procédure TPHRASE1 qui réalise certaines transformations sur la phrase si son verbe est le verbe être, et appelle elle-même la procédure TPHRASE.

III.4.3 Procédure TPHRASE

La procédure TPHRASE réalise les transformations sur une phrase à partir de la liste d'arguments de cette phrase. Cette procédure est décrite à l'aide du prédicat TPHRASE suivant:

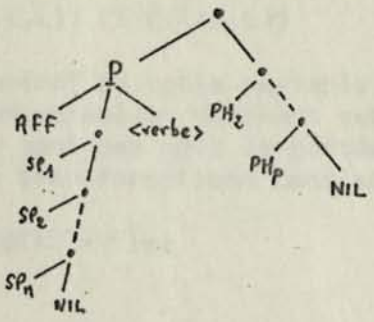
TPHRASE(I/I1,J/J1,K/K1,E/LV,LSP,PH.P/TP)

qui prend la valeur vraie quand les conditions suivantes sont vérifiées:

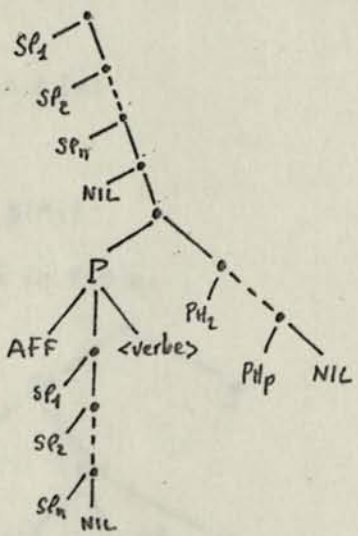
-TP est le transformé de la phrase PH suivi du paragraphe P. Le transformé de la phrase PH est obtenu à partir de la liste d'arguments LSP, et, est fonction de E et LV qui représentent l'état de la négation et de la liste des variables universelles.

-I1,J1 et K1 sont les transformés de I,J et K. Nous avons déjà défini le transformé de ces termes.

Le transformé d'une phrase à partir de sa liste d'arguments se définit comme suit: Considérons une phrasePH,première phrase d'un paragraphe PH.P:



Le transformé de cette structure va s'obtenir en balayant l'arbre suivant,de haut en bas et de gauche à droite:



Ceci revient à traiter les arguments de la phrase dans l'ordre où ils apparaissent. Les transformations associées aux SP_i vont consister à introduire les schémas logiques que nous avons vus dans les chapitres précédents en introduisant immédiatement les fonctions de Skölem C'est à dire que le schéma

$$\forall x \quad \varphi(x) \supset \beta(x)$$

va se représenter par:

$$\varphi(*x) \text{ IMP } \beta(*x)$$

et le schéma

$$\exists x \quad \varphi(x) \wedge \beta(x)$$

va s'écrire;

$$\varphi(F(i,L)) \text{ ET } \beta(F(i,L))$$

si i est le "numéro" de cette variable existentielle et L la liste des variables universelles dominant cette formule, dans le cas où ces formules ne sont pas sous la portée d'une négation.
Sinon , les transformations consistent à réécrire le schéma:

$$\forall x \quad \varphi(x) \supset \beta(x)$$

sous la forme:

$$\varphi(F(i,L)) \text{ IMP } \beta(F(i,L))$$

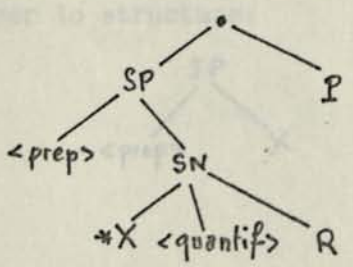
et le schéma:

$$\exists x \quad \varphi(x) \wedge \beta(x)$$

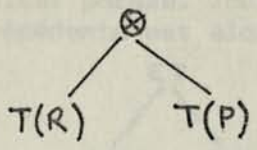
sous la forme:

$$\varphi(*x) \text{ et } \beta(*x)$$

Donc chaque noeud de la forme:



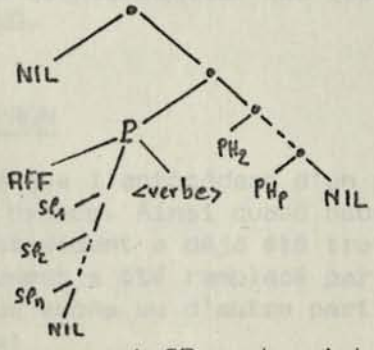
va se transformer en la structure suivante:



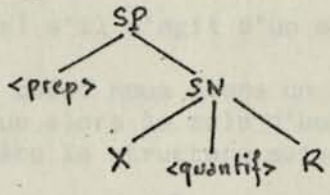
où -⊗ désigne, soit ET, soit IMP
-T(R) est le transformé du prargraphe R des relatives associées à l'argument
-T(P) le transformé de la structure P.

Cette transformation s'accompagne d'une substitution dans R et dans P de toute occurence de * par la réprésentation de la variable associée en fonction de la quantification: soit une variable PROLOG *V soit un terme F(*I,*L).

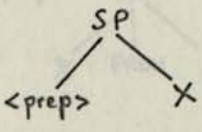
Quand tous les arguments SP_i sont ainsi transformés, T(P) représente alors le transformé de l'arbre:



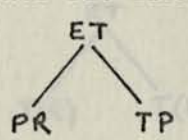
dans lequel chaque argument SP_i est maintenant "quantifié", c'est à dire que si l'on regarde la structure d'un SP_i



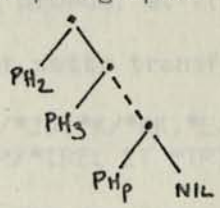
x est maintenant la représentation de la variable associée au SN, soit la représentation formelle de l'argument que l'on peut simplement représenter par la structure:



La phase peut être représentée par le prédicat PR associé au verbe de la phrase (prédicat phrase, inclusion, être...). Le transformé de la structure précédente est alors:



où TP est le transformé du paragraphe:



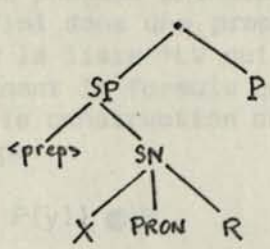
Etudions donc les transformations propres à chaque type d'argument en fonction de son quantificateur qui peut être: PRON,PROP,DEF, INDEF,CHAQUE,INTERO.

Argument de type PRON

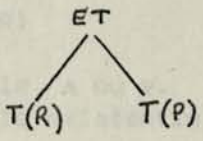
Nous avons supposé que l'antécédent d'un pronom prècède toujours l'occurence de ce pronom. Ainsi quand nous rencontrons un argument de type PRON, l'antédédent a déjà été traité, ce qui signifie que le nom N de l'argument a été remplacé par la représentation logique de l'argument. Nous avons vu d'autre part que cette représentation est de trois types:

- -une variable PROLOG *V s'il s'agit d'un argument quantifié universellement
- -un terme F(I,x) s'il s'agit d'un argument quantifié existentiellement.
- -un nom propre quand nous avons un argument signifié par un nom propre qui joue alors le role d'une constante logique.

Donc si on considère la structure suivante:



x est la variable associée à l'argument auquel le pronom fait référence et le transformé de cette structure va être simplement:



T(R) étant le transformé du paragraphe R représentant les relatives éventuelles associées au pronom, et T(P) le transformé de la structure P.
La clause PROLOG réalisant cette transformation est:

```
+TPHRASE(*I,*J/*JJ,*K/*KK,*E/*LV,SP(*Prep,SN(*N,PRON,*REL)).
         *SP,*P/*TREL ET *TP)

-TRLIST(*N,*LV/*LV1)

-TPARAGRAPHE(0/0,*J/*J1,*K/*K1,*E/*LV1,*REL/*TREL)

-TPHRASE(*I,*J1/*JJ,*K1/*KK,*E/*LV1,*SP,*P/*TP) ..
```

où le prédicat TRLIST(*N,*LV/*LV1) est vrai si *LV1 est le transformé de *LV. ce transformé est obtenu en fonction de *N de la façon suivante:

- -Si *N représente un argument de la phrase quantifié existentiellement ou une constante alors *LV1 est égal à *LV.
- -Si *N représente un argument de la phrase quantifié universellement alors:
  Si *N est un élément de la liste *LV, *LV1 est égal à *LV.
  Si *N n'est pas un élément de la liste *LV, alors
  *LV1 = *N.*LV.

Cette procédure TRLIST modifie en fait, dans certains cas, la portée du quantificateur associé à la variable représentée par *N. Ce sera le cas de certaines phrases principales où un pronom fait référence à un argument défini dans une proposition enchassée. Il est nécessaire alors de modifier la liste *LV qui représente la liste des variables universelles dominant la formule que nous construisons, cette liste intervenant dans la construction des fonctions de Skölem.
Une formule du type:

$$Qx\ (\forall y\ P(y))\ \otimes\ R$$

est transformée dans ce cas en:

$$Qx\ \forall y(P(y) \otimes R)$$

$\otimes$ représentant le symbole $\wedge$ ou $\vee$.
Dans le cas d'une variable existentielle, une formule du type:

$$QX\ (\exists y\ P(y)) \otimes R$$

peut toujours être transformée en

$$Qx\ \exists y\ (P(y) \otimes R)$$

ce qui ne modifie en rien la construction des fonctions de Skölem pouvant apparaitre dans R, puisque celles-ci ne dépendent que des variables universelles.

Argument de type PROP

Les arguments de type PROP correspondent à des arguments que nous représentons formellement par le nom propre lui-même. La transformation dans ce cas est décrite par la règle PROLOG suivante:

```
+TPHRASE(*J,*J/*JJ,*K/*KK,*E/*LV,SP(*Prep,SN(*N,PROP,*REL)).
         *SP,*P/*TREL ET *TP)

-TPARAGRAPHE(0/0,*J/*J1,*K/*K1,*E/*LV,*REL/*TREL)

-TPHRASE(*I,*J1/*JJ,*K1/*KK,*E/*LV,*SP,*P/*TP) ..
```

Argument de type DEF

Les arguments de type DEF se traitent à l'aide du schéma existentiel avec une règle d'unicité:

$$\exists x\ (\varphi(x) \wedge \forall y(\varphi(y) \supset Q(y,x)) \wedge \beta(x))$$

Considérons la phrase:

L'homme que j'ai vu hier est le frère de Jean.

Nous allons la traduire à l'aide de la formulation:

un homme que j'ai vu hier est le frère de Jean
et chaque homme que j'ai vu hier est cet homme.

soit

UN x (x est homme.x est un.j'ai vu hier x) est le frère de Jean

et CHAQUE y (y est homme.j'ai vu hier y) est dans x

Le verbe est dans étant ici un verbe spécial que nous traduirons à l'aide du prédicat inclusion Q;
Nous allons donc faire une recopie des relatives restrictives pour construire la phrase "CHAQUE y (φ(y)) est dans x" qui va se traduire par la règle d'unicité:

$$\forall y \quad \varphi(y) \supset Q(y,x)$$

Par ailleurs nous avons vu qu'un argument défini apparaissant dans la portée d'une négation, c'est à dire se traitent par une quantification universelle, ne doit pas faire intervenir la règle d'unicité. L'argument défini se traitera alors simplement comme argument de type INDEF. C'est le cas de l'argument défini dans la phrase:

ex(1) chaque homme qui voit le chien qu'il aime...

qui va alors se traiter comme:

chaque homme qui voit un chien qu'il aime...

et se traduire par la formule:

$$\forall x \forall y \ (\text{homme}(x) \wedge \text{chien}(y) \wedge \text{aime}(x,y) \wedge \text{voit}(x,y)) \supset \ldots.$$

Il faudra tenir compte aussi du fait que certains arguments définis ont été déjà traités, puisque ceux-ci ne sont pas toujours examinés dans l'ordre où ils apparaissent. Nous nous en rendons compte en étudiant le nom *N associé à l'argument qui est alors un terme de la forme F(I,x).
Si l'argument défini se traite à partir du schéma existentiel,

$$\exists x \quad \varphi(x) \wedge \beta(x)$$

la variable x qui le représente va être remplacée par une fonction de Skölem F(I,z), où z représente la liste des variables universelles dont x dépend. Nous avons vu qu'un argument défini ne dépend que des arguments auxquels il fait référence (à l'aide d'un pronom) dans sa description.

Dans la phrase:

ex(2) Tout homme possède la voiture qu'il aime.

si nous associons à "Tout homme" la variable x, "la voiture qu'il aime" dépend de x par l'intermédiaire du pronom IL. On obtient donc la clause:

-homme(x) +voiture(f(x1)

-homme(x) +aime(x,f(x))

-homme(x) +possède(x,f(x))

où f(x) représente "la voiture que x aime".
Mais dans la phrase:

ex(3) Tout homme connait le problème.

"le problème" ne dépend pas de "tout homme" et sera alors représenté par une constante de Skölem A. Cela va se représenter en structure clausale par:

homme(x)IMPproblème(A) ET connait(x,A)

qui est une représentation de la formule:

$\exists x$ (problème(x) $\wedge$ $\forall$y(homme(y) $\supset$connait(y,x)))

Pour obtenir la clause correspondant à cette formule, nous n'engendrerons, à partir de la structure clausale, que la partie droite de l'implication quand celle-ci n'est pas liée par une variable à la partie gauche: ainsi la structure P(x)IMP R(A), si R(A) n'est pas liée à P(x) par une variable, se traduit par la clause:

+ R(A)

Par exemple, pour la phrase précédente, la structure clausale:

homme(x) IMP porblème(A) ET connait(x,A)

va engendrer les clauses:

+ problème(A)

-homme(x) +connait(x,A)

Nous n'avons pas fait intervenir dans la traduction des exemples (2) et (3) la règle d'unicité.

REMARQUE

Avant d'énoncer la règle d'unicité sous la forme décrite précédemment il est nécessaire de traiter les arguments définis pouvant apparaitre dans les relatives restrictives:

ex(4) La fille qui est sur le divan est belle

nous avons vu que cette phrase se traduit par:

une fille qui est sur le divan est belle
et chaque fille qui est sur le divan est cette fille

Il est bien clair qu'avant de faire la copie de la relative "qui est sur le divan" il est préférable de définir d'abord l'argument "le divan" pour ne pas avoir à le définir deux fois. Cela va se traduire en fait par:

Il y a un divan
et chaque divan est ce divan
et une fille qui est assise sur ce divan est belle
et chaque fille qui est assise sur ce divan est cette fille.

La phrase "il y a un divan" se traduisant par:

$\exists x \quad divan(x) \wedge \beta(x)$

$\beta(x)$ correspond à la traduction de la phrase

il y a x

qui a une valeur toujours vraie, l'expression "il y a " jouant le role de verbe. Nous obtenons donc simplement:

$\exists x \quad divan(x)$

Les transformations sur les arguments de type DEF sont décrites par les clauses PROLOG suivantes:

```
+TPHRASE(*I,*J,*K,*E,*SP(*Prep,SN(*N,DEF,*REL)).*SP,*P/*TP)

        +COPIE(*N,BIDON)

        -TPHRASE(*I,*J,*K,*E,*SP,*P/*TP) ..

+TPHRASE(*I,*J,*K,1/*LV,SP(*Prep,SN(*N,DEF,*REL)).*SP,*P/*TP)

        -TPHRASE(*I,*J,*K,1/*LV,SP(*Prep,SN(*N,INDEF,*REL)).
                 *SP,*P/*TP) ..

+TPHRASE(*I,*J/*JJ,*K/*KK,0/*LV,SP(*Prep,SN(*N,DEF,*REL)).
         *SP,*P/*DEF ET *TREL ET *UNI ET *TP)

        -DIV(*REL,*REST1,*APOS1)

        -TDEF(*J/*J1,*K/*K1,0/*LV,*REST1/*DEF)

        -DEPLIST(*REST1,*K/*KPI,NIL/*L)

        -COPIE(*K1.SN(*N,CHAQUE,*REST1),*K1.*SN)

        -DEF(*J1/*J2,*K1/*K2,0/*L,*N,*E1,*E2)

        -TPARAGRAPHE(0/0,*J2/*J3,*K2/*K3,0/*LV,*REL/*TREL)

        -TPARAGRAPHE(0/0,*J3/*J4,*K3/*K4,0/*LV,
                     P(AFF,SP(SUJ.NIL,*SN).
                     SP(OBJ.NIL,SN(*N,PRON,NIL)).NIL,OUI(DANS)).
                     NIL/*UNI)

        -TPHRASE(*I,*J4/*JJ,*K4/*KK,0/*LV,*SP,*P/*TP) ..
```

où les prédicats:

-DIV(x,y,z) est vraie si y est la liste des relatives de type REST élément de la liste x, et z la liste des relatives de type APOS élément de la liste y.
-DEPLIST(L,x/y,z/u) est vraie si, u' étant la liste des variables PROLOG *N ayant une occurence commune dans l'arbre L et la liste x, u est le concaténé de u' avec z; la liste y étant alors la liste x à laquelle on a supprimé les éléments de u'.
-DEF est vraie pour les valeurs suivantes:

+DEF(*J/*J',*K/*K,0/*L,F(*J,*L),0/*L,1/*L)

+DEF(*J/*J,*K/*V.*K,1/*L,*V,1/*V.*L,0/*V.*L)

Ces prédicats sont définis dans la liste des clauses composants le programme que nous donnons en fin de chapitre.
Le prédicat DIV(x,y,z) nous permet de définir une procédure qui nous donne à partir de la liste x des relatives associées à un argument, la liste y des restrictives et la liste z des appositives.
Le prédicat DEPLIST(L,x/y,z/u) définit une procédure permettant d'obtenir en u la liste des variables universelles ayant une occurence dans L et qui sont élément de la liste x. Cette procédure est utilisée, L étant l'arbre des relatives associées à un argument défini, et,x la liste des variables universelles générées,pour construire la liste des variables universelles dont dépend l'argument défini.
Le prédicat DEF est utilisé pour obtenir, en fonction de l'état de la négation et de la liste des variables universelles, la représentation du terme associé à l'argument, c'est à dire soit un terme F(I,x), soit une variable *V. Cela est obtenu par simple unification tout en transformant les "compteurs" et les listes que nous avons à gérer.

Argument de type INDEF

Les arguments de type INDEF se traitent, s'ils ne sont pas dans la portée d'une négation, à partir du schéma logique existentiel

$$\exists x \quad \varphi(x) \wedge \beta(x)$$

La variable associée à l'argument est alors une variable existentielle. Si nous sommes dans la portée d'une négation, cette dernière étant introduite au niveau des prédicats, la variable associée à l'argument sera alors une variable universelle.

$$\neg(\exists x \quad \varphi(x) \wedge \beta(x)) \equiv \forall x \neg(\varphi(x) \wedge \beta(x)).$$

La clause PROLOG réalisant les transformations sur ce type d'argument est:

```
+TPHRASE(*I,*J/*JJ,*K/*KK,*E,SP(*Prep,SN(*N,INDEF,*REL)).
         *SP,*P/*TREL ET *TP)

-DEF(*J/*J1,*K/*K1,*E,*N,*E1,*E2)

-TPARAGRAPHE(0/0,*J1/*J2,*K1/*K2,*E,*REL/*TREL)

-TPHRASE(*I,*J2/*JJ,*K2/*KK,*E,*SP,*P/*TP) ..
```

Argument de type CHAQUE

Les arguments de type chaque se traitent à l'aide du schéma logique universel:

$$\forall x \quad \varphi(x) \supset \beta(x)$$

La variable associée est alors une variable universelle, qui devient une variable existentielle si nous sommes dans la portée d'une négation:

$$\neg(\forall x \quad \varphi(x) \supset \beta(x)) \equiv \exists x \ \neg(\varphi(x) \supset \beta(x))$$

Par ailleurs l'implication $\varphi(x) \supset \beta(x)$, qui va se transformer pour obtenir la forme clausale en $\varphi(x) \wedge \beta(x)$, introduit une négation sur $\varphi(x)$. L'appel de la procédure générant la structure $\varphi(x)$ va donc devoir se faire en modifiant l'état précédent de la négation.
L'instruction PROLOG qui réalise le travail sur les arguments de type chaque est la suivante:

```
+TPHRASE(*I,*J/*JJ,*K/*KK,*E,SP(*Prep,SN(*N,CHAQUE,*REL)).
         *SP,*P/*DEF ET *REST IMP *APOS ET *TP)

-SCARD(*REL,*REL1)

-DIV(*REL1,*REST1,*APOS1)

-TDEF(*J/*J1,*K/*K1,*E,*REST1/*DEF)

-CHAQ(*J1/*J2,*K1/*K2,*E,*N,*E1,*E2)

-TPARAGRAPHE(0/0,*J2/*J3,*K2/*K3,*E2,*REST1/*REST)

-TPARAGRAPHE(0/0,*J3/*J4,*K3/*K4,*E1,*APOS1/*APOS)

-TPHRASE(*I,*J4/*JJ,*K4/*KK,*E1,*SP,*P/*TP) ..
```

où:

-le prédicat SCARD(x,y) prend la valeur vraie si y est l'arbre x dans lequel on a remplacé éventuellement le type APOS de la relative exprimant la cardinalité de l'argument, par le type REST.

-le prédicat CHAQ est défini d'une manière analogue au prédicat DEF. Il est utilisé pour obtenir la représentation de la variable associée à un argument de type chaque, et est défini de la manière suivante:

+CHAQ(*J/*J,*K/*V.*K,0/*LV,*V,0/*V.*LV,1/*V.*LV)

+CHAQ(*J/*J',*K/*K,1/*LV,F(*J,*LV),1/*LV,0/*LV)

Argument de type INTERO

Les arguments de type INTERO représentent les arguments signifiés par un pronom interrogatif. Nous avons vu que si nous avons la structure de phrase suivante:

INTERO x $(\varphi(x))\ \beta(x)$

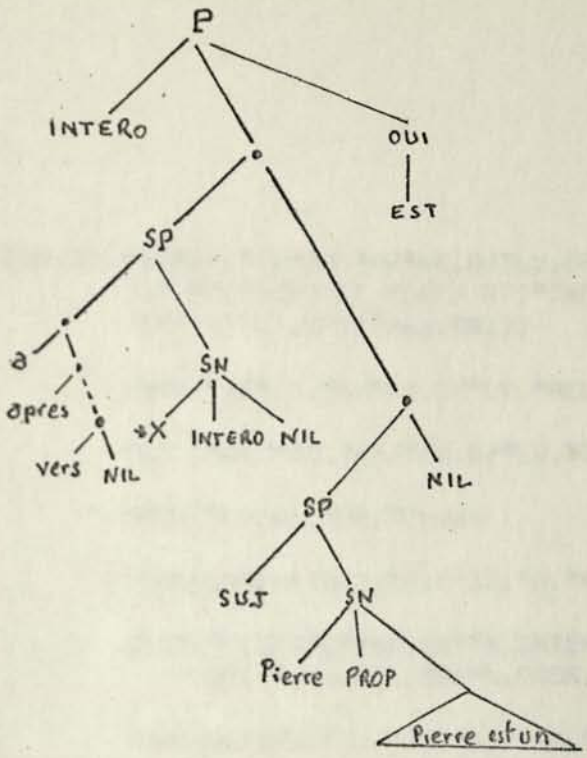
La représentation logique est alors:

$\forall x\ [\varphi(x) \wedge \beta(x)] \supset$Réponse(x)

Par ailleurs pour les arguments de type INTERO nous pouvons avoir une liste de prépositions. La réponse doit alors être recherchée parmi les arguments introduits par une des prépositions contenues dans la liste:
Par exemple la question:

où est Pierre?

qui se représente par l'arbre:



va se traduire par l'ensemble des clauses suivantes:

```
-P(SP(A,*x).SP(SUJ,Pierre).NIL,oui(EST)) +R(SP(A,*x)) .;

-P(SP(Après,*x).SP(SUJ,Pierre).NIL,oui(EST)) +R(SP(Après,*x)) .;

etc.................

-P(SP(vers,*x).SP(SUJ,Pierre).NIL,oui(EST)) +R(SP(vers,*x)) .;
```

que l'on peut en fait résumer en une seule clause en utilisant le prédicat DS(x,L) décrit dans la syntaxe:

```
-P(SP(*prep,*x).SP(SUJ,Pierre).NIL,oui(EST)) -DS(*prep,A.Après...vers.NIL)
                                              +R(SP(*prep,*x)) .;
```

La règle PROLOG qui réalise les transformations à partir d'arguments de type INTERO, est la suivante:

```
+TPHRASE(*I/*I',*J/*JJ,*K/*KK,O/*LV,SP(*LPrep,SN(*N,INTERO,*REL)).
        *SP,*P/*DEF1 ET *DEF2 ET(*TREL ET *TP ET *PR)
         IMP R(*I',SP(*Prep,*N)))

        -TDEF(*J/*J1,*K/*K1,O/*LV,*REL/*DEF1)

        -TDEF(*J1/*J2,*K1/*K2,O/*LV,*SP/*DEF2)

        -PRED(*LPrep,*PR,*Prep)

        -TPARAGRAPHE(O/O,*J2/*J3,*N.*K2/*K3,1/*N.*LV,*REL/*TREL)

        -SBSP(*P,SP(*LPrep,SN(*N,INTERO,*REL)),
              SP(*Prep.NIL,SN(*N,PRON,NIL)),*P1)

        -TPHRASE(O/O,*J3/*JJ,*K3/*KK,1/*N.*LV,*SP,*P1/*TP) ..
```

Le prédicat PRED est défini par:

```
+PRED(*x.*y,DS(*V,x.*y),*V)     si y est différent de nil

+PRED(*x.NIL,NIL,*x)
```

Ce prédicat permet de générer au niveau de la structure clausale le prédicat DS(x,y) que nous avons vu précédemment. Quand nous avons une liste de prépositions se réduisant à un seul élément,au lieu de générer:

```
-P(SP(*prep,*x).*y,*V) -DS(*prep,A.NIL)...
```

nous générons simplement:

```
-P(SP(A,*x).*y,*V) ....
```

Le prédicat SBSP(x,y,z,t) est vrai quand l'arbre t est l'arbre x dans lequel on a substitué chaque occurence de y par z.

Règle terminale

Cette règle va s'appliquer quand la liste d'argument est vide (NIL). Il faut alors traduire la phrase par le prédicat associé au verbe, puisque tous les arguments sont maintenant quantifiés.

La règle PROLOG est la suivante:

```
+TPHRASE(*I,*J,*K,*E,NIL,*PH.*P/*TPH ET *TP)

        -REPRE(*PH,*TPH)

        -TPARAGRAPHE(*I,*J,*K,*E,*P/*TP) ..
```

—> le prédicat REPRE(x,y) étant vrai si y est le transformé de x, cette transformation consistant à transformer la phrase représentée par x en la représentation du prédicat associé au verbe de cette phrase. Le prédicat REPRE est décrit dans le détail du programme que nous donnons en fin de chapitre.

III.4.4 Appel de la procédure TTEXTE:

Le programme lui-même ne va consister qu'en une seule règle PROLOG qui est en fait un appel de la procédure TTEXTE.
Cette règle est:

```
-STRUCTURE SYNTAXIQUE(*x) -TTEXTE(0/*I,0/*J,*x/*y)

                          +ONA(NQ(*I') ET *y) ..
```

Elle signifie: Si *x est la structure syntaxique d'un texte et si le transformé de *x est *y, le nombre de questions et de variables existentielles étant *I et *J, alors on a la structure clausale

NQ(*I') ET *y.

La donnée du programme est alors constituée par la seule clause:

```
+STRUCTURESYNTAXIQUE (x)
```

qui signifie que x est la structure syntaxique d'un texte.
Pour générer les clauses PROLOG correspondantes nous allons donc "désembriquer" l'arbre *x obtenu dans le prédicat ONA(*x).

III.4.5. Générations des clauses PROLOG à partir de la structure clausale

Pour générer les clauses PROLOG nous allons utiliser des règles comportant une partie réponse.
Nous avons représenté l'ensemble des clauses par un terme où chaque prédicat est une fonction. Le principe qui permet de générer les clauses va consister à "désembriquer" l'arbre représentant les clauses pour atteindre les termes associés aux prédicats, ces prédicats constituant alors la réponse.
Nous avons par exemple les règles suivantes:

```
-ONA(*x ET*y) +ONA(*x) ;;

-ONA(*x ET*y) +ONA(*y) ..

-ONA(*x IMP*y) +ONA(*y) +ONA(NO*x) ..
```

qui "désembriquent" les termes construits à partir des opérateurs ET ou IMP. De même pour la négation:

```
-ONA(NO(*x ET*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) ..
```

Les règles terminales sont celles qui vont comporter une partie réponse et qui vont s'appliquer chaque fois que nous aurons un terme construit à partir d'un symbole fonctionnel associé a un symbole de prédicat. (En fait nous utilisons le même symbole puisqu'il ne peut y avoir ambigüités).
Ainsi pour le symbole de prédicat Q nous aurons les règles:

```
-ONA(Q(*x,*y))/ +Q(*x,*y) ..

-ONA(NO Q(*x,*y))/ -Q(*x,*y) ..
```

Nous avons ainsi deux règles pour chacun des symboles fonctionnels associés à un prédicat.
Nous avons vu que NIL correspond à la valeur logique VRAIE, ce qui va se traduire par la règle:

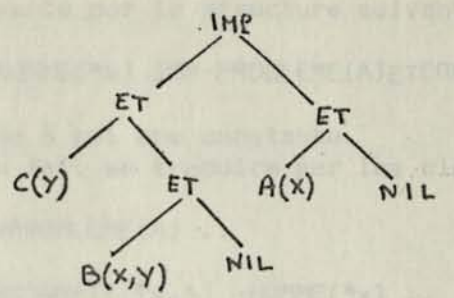
```
-ONA(NO NIL) ..
```

EXEMPLE

Considérons la clause suivante:

+A(x) -B(x,y) -C(y) ..

qui se traduit par la structure clausale:



Considérons donc que nous avons la clause:

+ONA((C(y) ET B(x,y) ET NIL) IMP A(x) ET NIL)

Nous allons donc obtenir successivement à partir de cette clause les résolvantes:

+ONA(NO(C(y) ET B(x,y) ET NIL)) +ONA(A(x) ET NIL)

+ONA(NO C(y)) +ONA(NO(B(x,y) ET NIL)) +ONA(A(x) ET NIL)

+ONA(NO(B(x,y) ET NIL) +ONA(A(x) ET NIL)/ -C(y)

+ONA(NO B(x,y)) +ONA(NO NIL) +ONA(A(x) ET NIL)/ -C(y)

+ONA(NO NIL) +ONA(A(x) ET NIL)/ -B(x,y) -C(y)

+ONA(A(x) ET NIL)/ -B(x,y) -C(y)

+ONA(A(x))/ -B(x,y) -C(y)

/+A(x) -B(x,y) -C(y)

La clause +A(x) -B(x,y) -C(y) est alors envoyée dans le fichier réponse.

Nous allons en fait générer les clauses d'une façon sensiblement différente. Cette modification est nécessaire pour le traitement des arguments définis.
Donnons un exemple:

ex Tout homme connait le problème

va être présenté par la structure suivante:

HOMME(*x) IMP PROBLEME(A)ETCONNAIT(*x,A)

dans laquelle A est une constante.
Cela doit en fait se traduire par les clauses:

+PROBLEME(A) ..

+CONNAIT(*x,A) -HOMME(*x) ..

c'est à dire que PROBLEME(A) ne se trouve pas dans l'implication.
Ce résultat sera obtenu en ne générant à partir d'une structure A IMP B la clause +B -A que dans le cas où B est liée à A par une variable commune; sinon on génère simplement la clause +B.
Nous ferons une exception à cette règle dans le cas où la partie droite de l'implication est une réponse représentée par un terme R(I,x).
La règle ne va s'appliquer en fait que pour les arguments définis qui ne dépendent pas forcément des arguments qui les précèdent, mais simplement des arguments à partir desquels ils sont définis.
En fait la règle précédente permet de poser l'existence des arguments définis indépendamment de tout autre argument auquel il n'est pas lié par une variable.
On considère donc que lorsqu'on représente un individu par une fonction de Skölem, on pose l'existence de cet individu.
Si cette fonction de Skölem dépend d'une variable x, c'est que l'existence de cet individu est liée à l'existence de x. Par contre si cette fonction de Skölem est une constante, alors cet individu existe et son existence n'est pas liée à l'existence d'un autre individu.
Considérons la structure suivante:

R(*x) IMP P(A)

où *x est une variable PROLOG (universelle) et A une constante de Skölem.
Cela doit en fait s'interpréter par la formule:

$$\exists x \, (P(x) \wedge \forall y (R(y) \supset P(x))$$

qui est équivalent à:

$$\exists x \; P(x)$$

soit à la clause:

+P(A)

Ces transformations sur les schémas du type A IMP B sont réalisées grace aux instructions PROLOG suivantes:

```
-ONA(*x IMP*y ET*z) +ONA(*x IMP*Y) ;;

-ONA(*x IMP*y ET*z) +ONA(*x IMP*z) ..

-ONA(*x IMP*y IMP*z) -T(*y IMP*z,*u) +ONA(*x IMP*u) ..

-ONA(*x IMP R(*I,*y)) +ONA(NO*x)+ONA(R(*I,*y)) ..

-ONA(*x IMP*y) -DEPEND(*y,*x) +ONA(NO*x) +ONA(*y) .;

-ONA(*x IMP*y) +ONA(*y) ..
```

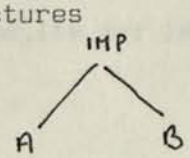
où T est défini par les règles:

```
+T(*x IMP*y ET*z,(*x IMP*y) ET(*x IMP*z)) ..

+T(*x IMP*y IMP*z,*x IMP*u) -T(*y IMP*z,*u) ..

+T(*x IMP R(*I,*y),NO(*x ET NO R(*I,*y))) ..

+T(*x IMP*y, NO(*x ET NO*y)) -DEPEND(*y,*x) .;

+T(*x IMP*y,*y) ..
```

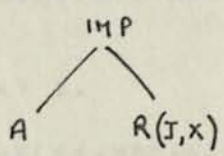
et où le prédicat DEPEND(x,y) prend la valeur vraie si y et x sont liés entre eux par une variable PROLOG commune.

REMARQUE

Considérons les structures



Elles ne peuvent être générées que dans deux cas
- -dans le cas d'une phrase interrogative de type OUINON , ou d'un argument de type INTERO, auquel cas nous avons alors:



et où nous supposons que R(I,x) dépend toujours de A.
-dans le cas d'un argument de type CHAQUE à l'aide du schéma:

$$\forall x \quad \varphi(x) \supset \beta(x)$$

qui nous assure alors que tout argument apparaissant dans $\beta(x)$ dépend de x, à l'exception donc de certains arguments définis qui auraient dus être en fait traités avant la construction de ce schéma.
Donc si dans $\beta(x)$, il existe un argument n'étant pas lié à $\varphi(x)$ par une variable, c'est à dire ne dépendant pas de x, c'est que celui-ci est un argument de type défini et que son existence doit alors être posée en dehors de l'implication.

III.4.6 Exemples de traduction de phrases

Nous donnons ici quelques exemples de phrases simples traitées par le programme. On trouvera la phrase en Français et les énoncés logiques engendrés sous forme de clauses.
Certains prédicats décrits dans la syntaxe ont un argument supplémenteire représenté par une variable PROLOG *I. Cet argument permet de cpntroler la stratégie lors de la phase de démonstration pour obtenir les réponses aux questions posées.

Exemple 1

TEXTE D' ENTREE:

LA FEMME QUI EST ASSISE SUR LE DIVAN, EST BELLE.

cette phrase est traduite par les énoncés logiques suivants:

```
+P(SP(SUJ,F(O',NIL)).SP(SUR,F(O,NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).
NIL-A-S-S-I-S.NIL),*X) ;.

+P(SP(SUJ,F(O',NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL-B-E-L-L.NIL),
*X) ;.

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

+Q(*X,F(O,NIL),*Y)-ETRE(*X,MAS.NIL-D-I-V-A-N.NIL-D-I-V-A-N-S,*Z)-DIF(*
X,F(O,NIL)) ;.

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

```
+Q(*X,F(O',NIL),*Y)-P(SP(SUJ,*X).SP(SUR,F(O,NIL)).NIL,OUI((NIL-E-S-T.N
IL-S-O-N-T).NIL-A-S-S-I-S.NIL),*Z)-ETRE(*X,FEM.NIL-F-E-M-M-E.NIL-F-E-M
-M-E-S,*T)-DIF(*X,F(O',NIL)) ;.

+NQ(O') ;.

+ETRE(F(O,NIL),MAS.NIL-D-I-V-A-N.NIL-D-I-V-A-N-S,*X) ;.

+ETRE(F(O',NIL),FEM.NIL-F-E-M-M-E.NIL-F-E-M-M-E-S,*X) ;.
```

Exemple 2

TEXTE D' ENTREE:

TOUT SOUSCRIPTEUR RECOIT UNE PRIME QUI EST EGALE AUX INTERETS QU'IL A ACQUIS SUR LE LIVRET QU'IL POSSEDE.CETTE PRIME EST EXONEREE DE TOUT IMPOT.

La traduction de ces phrases donne:

```
+P(SP(OBJ,F(O',*X.NIL)).SP(SUJ,*X).NIL,OUI((NIL-P-O-S-S-E-D-E.NIL-P-O-
S-S-E-D-E-N-T).NIL),*Y)-ETRE(*X,MAS.NIL-S-O-U-S-C-R-I-P-T-E-U-R.NIL-S-
O-U-S-C-R-I-P-T-E-U-R-S,*Z)-CARD(*X,E(1),*T) ;.

+P(SP(OBJ,F(O'',*X.NIL)).SP(SUJ,*X).SP(SUR,F(O',*X.NIL)).NIL,OUI((NIL-
A.NIL-O-N-T).NIL-A-C-Q-U-I.NIL),*Y)-ETRE(*X,MAS.NIL-S-O-U-S-C-R-I-P-T-
E-U-R.NIL-S-O-U-S-C-R-I-P-T-E-U-R-S,*Z)-CARD(*X,E(1),*T) ;.

+P(SP(SUJ,F(O,*X.NIL)).SP(A,F(O'',*X.NIL)).NIL,OUI((NIL-E-S-T.NIL-S-O-
N-T).NIL-E-C-A-L.NIL),*Y)-ETRE(*X,MAS.NIL-S-O-U-S-C-R-I-P-T-E-U-R.NIL-
S-O-U-S-C-R-I-P-T-E-U-R-S,*Z)-CARD(*X,E(1),*T) ;.

+P(SP(SUJ,*X).SP(OBJ,F(O,*X.NIL)).NIL,OUI((NIL-R-E-C-O-I-T.NIL-R-E-C-O
-I-V-E-N-T).NIL),*Y)-ETRE(*X,MAS.NIL-S-O-U-S-C-R-I-P-T-E-U-R.NIL-S-O-U
-S-C-R-I-P-T-E-U-R-S,*Z)-CARD(*X,E(1),*T) ;.

+P(SP(SUJ,F(O,*X.NIL)).SP(DE,*Y).NIL,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL-E
-X-O-N-E-R-E.NIL),*Z)-ETRE(*Y,MAS.NIL-I-M-P-O-T.NIL-I-M-P-O-T-S,*T)-CA
RD(*Y,E(1),*U)-ETRE(*X,MAS.NIL-S-O-U-S-C-R-I-P-T-E-U-R.NIL-S-O-U-S-C-R
-I-P-T-E-U-R-S,*V)-CARD(*X,E(1),*W) ;.

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

+CARD(F(O',*X.NIL),E(1),*Y)-ETRE(*X,MAS.NIL-S-O-U-S-C-R-I-P-T-E-U-R.NI
L-S-O-U-S-C-R-I-P-T-E-U-R-S,*Z)-CARD(*X,E(1),*T) ;.
```

```
+Q(*X,F(O',*Y.NIL),*Z)-P(SP(OBJ,*X).SP(SUJ,*Y).NIL,OUI((NIL-P-O-S-S-E-
D-E.NIL-P-O-S-S-E-D-E-N-T).NIL),*T)-ETRE(*X,MAS.NIL-L-I-V-R-E-T.NIL-L-
I-V-R-E-T-S,*U)-ETRE(*Y,MAS.NIL-S-O-U-S-C-R-I-P-T-E-U-R.NIL-S-O-U-S-C-
R-I-P-T-E-U-R-S,*V)-CARD(*Y,E(1),*W)-DIF(*X,F(O',*Y.NIL)) ;.

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

+Q(*X,F(O'',*Y.NIL),*Z)-P(SP(OBJ,*X).SP(SUJ,*Y).SP(SUR,F(O',*Y.NIL)).N
IL,OUI((NIL-A.NIL-O-N-T).NIL-A-C-Q-U-I.NIL),*T)-ETRE(*X,*U.NIL-I-N-T-E
-R-E-T.NIL-I-N-T-E-R-E-T-S,*V)-ETRE(*Y,MAS.NIL-S-O-U-S-C-R-I-P-T-E-U-R
.NIL-S-O-U-S-C-R-I-P-T-E-U-R-S,*W)-CARD(*Y,E(1),*X1)-DIF(*X,F(O'',*Y.N
IL)) ;.

+NQ(O') ;.

+ETRE(F(O,*X.NIL),FEM.NIL-P-R-I-M-É.NIL-P-R-I-M-E-S,*Y)-ETRE(*X,MAS.NI
L-S-O-U-S-C-R-I-P-T-E-U-R.NIL-S-O-U-S-C-R-I-P-T-E-U-R-S,*Z)-CARD(*X,E(
1),*T) ;.

+ETRE(F(O',*X.NIL),MAS.NIL-L-I-V-R-E-T.NIL-L-I-V-R-E-T-S,*Y)-ETRE(*X,M
AS.NIL-S-O-U-S-C-R-I-P-T-E-U-R.NIL-S-O-U-S-C-R-I-P-T-E-U-R-S,*Z)-CARD(
*X,E(1),*T) ;.

+ETRE(F(O'',*X.NIL),*Y.NIL-I-N-T-E-R-E-T.NIL-I-N-T-E-R-E-T-S,*Z)-ETRE(
*X,MAS.NIL-S-O-U-S-C-R-I-P-T-E-U-R.NIL-S-O-U-S-C-R-I-P-T-E-U-R-S,*T)-C
ARD(*X,E(1),*U) ;.
```

III.5 AXIOMES GENERAUX DE DEDUCTIONS - EXEMPLES DE DEDUCTIONS

Pour pouvoir faire des déductions sur le texte à partir des clauses engendrées il est nécessaire de rajouter quelques règles de déductions générales. En particulier, il est nécessaire d'exprimer les propriétés de l'inclusion et de l'égalité entre ensembles. Il sera nécessaire également de traduire des règles d'équivalence que nous avons données dans le chapitre I, règles permettant de passer d'une propriété entre ensemble à une propriété sur les sous-ensembles.

III.5.1 Règles sur les ensembles

-Transitivité de l'inclusion et de l'égalité

Un axiome important est l'axiome de transitivité de l'inclusion (et de l'égalité).

$$\forall x \forall y \forall z \quad (Q(x,y) \wedge Q(y,z)) \supset Q(x,z)$$

que l'on traduit par la clause:

```
-Q(x,y) -Q(y,z) +Q(x,z)
```

De même pour l'égalité:

$$-EG(x,y)\ -EG(y,z)\ +EG(x,z)$$

-Equivalence entre inclusion et égalité

La formule traduisant l'équivalence est la suivante:

$$\forall x \forall y\ \ (Q(x,y) \wedge Q(y,x)) \equiv EG(x,y)$$

cela se traduit par les trois clauses:

$$-Q(x,y)\ -Q(y,x)\ +EG(x,y)$$

$$-EG(x,y)\ +Q(x,y)$$

$$-EG(x,y)\ +Q(y,x)$$

-Symétrie et réflexivité de l'égalité

$$\forall x \forall y\ \ EG(x,y) \supset EG(y,x)$$

$$\forall x\ \ \ EG(x,x)$$

-Propriété sur la cardinalité des ensembles

Nous allons traduire simplement le fait que deux ensembles égaux ont même cardinalité par la clause:

$$-EG(x,y)\ -CARD(x,i)\ +CARD(y,i)$$

Par ailleurs nous avons vu que nous générons deux types de propriété sur la cardinalité. L'une correspond à l'égalité (CARD(x,E(i))), l'autre à une cardinalité supérieure à i (CARD(x,S(i))). Dans les axiomes de déductions nous exprimerons le fait qu'un ensemble est de cardinalité "égale ou supérieure à i" par CARD(x,ES(i)) que l'on définit par les clauses:

$$-CARD(x,E(i))\ +CARD(x,ES(i))$$

$$-CARD(x,S(i))\ +CARD(x,ES(i))$$

III.5.2 Axiomes de déductions sur les phrases:

-Permutation des arguments

Nous avons vu que les arguments d'une phrase sont réprésentés par une liste, et ce, dans un ordre quelconque. Donc si une phrase P est vraie pour une liste d'arguments L

+P(L,V)

(V représentant le groupe verbal)
la phrase P' construite sur le même modèle avec une liste d'arguments L', L' étant une permutation de L, est encore vraie:

P(L,V) ⊃P(L',V) si L' est une permutation de L.

En fait on peut généraliser encore plus en considérant que L' est une sous-liste de L, à une permutation près, ce que l'on traduit par la clause:

-P(L,V) -DS1(T,L) +P(T,V)

où DS1(T,L),est un prédicat qui prend la valeur vraie si T est une sous-liste à une permutation près de L. Il se définit à partir du prédicat DS(x,L) qui est vrai si x est un élément de la liste L.

-Substitution d'un argument par un argument "égal"

Cela va consister à traduire le fait que deux phrases sont équivalentes si leurs listes d'arguments L et L' vérifient les conditions suivantes: la liste L est identique à la liste L' dans laquelle toute occurence d'un argument A a été remplacée par un argument B qui lui est égal.
La clause traduisant cela est la suivante:

-P(L,V) -SBS(L,y,x,T) -EG(y,x) +P(T,V)

Le prédicat SBS(L,y,x,T) est vrai si la liste T est identique à la liste L dans laquelle les occurences de y ont été remplacées par x.

-Axiomes de déductions à partir de l'inclusion

Nous allons traduire ici les règles permettant de passer du pluriel au singulier.
Nous avons donné dans le chapitre I, une règle d'équivalence

$$V(x,y) \equiv \forall U \;\; U \in x \supset \exists t \;\; t \in y \wedge V(U,t)$$
$$\wedge$$
$$\forall t \;\; t \in y \supset \exists U \;\; U \in x \wedge V(U,t)$$

Nous allons traduire l'implication qui permet de passer de la propriété générale V(x,y) à une propriété sur les individus.
En fait nous allons traduire cette implication en utilisant non pas la propriété d'appartenance mais celle d'inclusion:

$$V(x,y) \supset \forall U \;\; Q(U,x) \supset \exists t \;\; Q(t,y) \wedge V(U,t)$$
$$\wedge$$
$$\forall t \;\; Q(t,y) \supset \exists U \;\; Q(U,x) \wedge V(U,t)$$

Cette implication est écrite pour le cas d'une phrase à deux arguments x et y. Il nous faut la traduire pour un nombre quelconque d'arguments.
Nous pouvons alors l'exprimer pour toute liste $L=x_1.x_2....x_n$

$$\forall L \forall V \forall x \forall y \forall T \left(P(L,V) \wedge DS(x,L) \wedge Q(y,x) \wedge SBT(L,x,y,T) \right) \supset P(T,V)$$

où -DS(x,L) est vrai si x est un élément de la liste L.
-SBT(L,x,y,T) est vrai si T est la liste obtenue à partir de la liste L de la manière suivante:
Toute occurence de x dans L est remplacée par y
Tout élément z de L (z différent de x) est remplacé par h(L,V,Y,z)

Les termes h vont représenter des ensembles qui ont les propriétés:

+Q(h(L,V,y,z),z) ..

+CARD(h(L,V,y,z),ES(1)) ..

puisque ces ensembles sont supposés non vides.

REMARQUE

Au lieu d'écrire h(L,V,x,z) et SBT(L,y,x,T) nous écrirons respectivement:

h(P(L,V),x,z)

SBT(P(L,V),L,y,x,T)

où P est ici un symbole fonctionnel différent donc du symbole de prédicat P (nous le notons de la même façon puisque aucune confusion n'est possible).

La clause correspond à l'implication précédente est alors:

-P(L,V) -DS(SP(z,x),L) -Q(y,x) -SBT(P(L,V),L,x,y,T) +P(T,V).

Considérons la phrase:

les hommes aiment les femmes.

qui se traduit par:

+P(SP(SUJ,x).SP(OBJ,y).NIL,aime)

où x représente l'ensemble "les hommes", et y l'ensemble "les femmes". Considérons d'autre part l'assertion "Pierre est un homme", de laquelle nous pouvons déduire:

+Q(Pierre,x)

De ces deux affirmations nous allons pouvoir déduire:

Pierre aime des femmes.

en remplaçant dans la phrase précédente:

x par Pierre

y par h(P(SP(SUJ,x).SP(OBJ,y).NIL,aime),Pierre,y)

pour obtenir donc:

+P(SP(SUJ,Pierre).SP(OBJ,h(P(-),Pierre,y))).NIL,aime)

le terme h(P(-),Pierre,y) désignant "les femmes que Pierre aime" et ayant les propriétés:

Q(h(P(-),Pierre,y),y) ..

CARD(h(P(-),Pierre,y),ES(1)) ..

Nous ne considérons l'implication que dans le sens permettant de passer du pluriel au singulier. En sens inverse nous aurions une règle permettant de déduire: si une propriété est vraie pour tous les individus d'un ensemble, alors elle est vraie sur l'ensemble. En fait l'implication dans ce sens est moins utile dans les déductions et peut être omise dans une première étape.

III.5.3 Contraintes - Controle de l'exécution du programme

-Ordre entre les clauses

Une première méthode pour établir des contraintes à l'exécution consiste à ordonner les clauses, en utilisant la ponctuation de chacunes d'elles.
D'une manière générale nous placerons en tête les clauses qui contiennent le moins de variables pures.
Le Fichier AXIOME est constitué de la façon suivante: les clauses générées à partir du texte suivies des clauses traduisant les axiomes généraux de déductions. En effet les clauses générées à partir du texte sont moins générales et contiennent plus d'arguments "ground" que les axiomes généraux décrits précédemment.
D'autre part entre les clauses du texte et les axiomes généraux nous intercalons des clauses spéciales permettant d'empêcher l'unification, sur les axiomes généraux, d'un littéral qui ne contient que des variables pures.
Considérons la transitivité de l'inclusion:

+Q(*x,*y) -Q(*x,*z) -Q(*z,*y) ;.

si nous voulons résoudre sur cette clause le littéral -Q(*U,*V) nous allons alors obtenir la résolvante:

-Q(*U,*z) -Q(*z,*V)

où le premier littéral est identique au littéral qui nous a permis d'obtenir cette résolvante.

Permettre de telles unifications peut dans certaines démonstrations être utile. Cependant le fait semble assez rare, et, d'autre part on prend le risque de se lancer dans des démonstrations sans fin. Pour éviter cela nous ferons précéder la règle sur la transivité de Q d'une clause en ..

```
+Q(bidon1 , bidon2 ) +Couic ..
```

sur laquelle donc un littéral -Q(x,y) ne pourra s'unifier que si x et y sont des variables pures, +Couic étant alors un littéral incancélable. La règle étant en .. d'autres résolutions du littéral -Q(x,y) ne seront pas tentées.
Nous ferons donc précéder la liste des axiomes généraux de déductions de clauses de ce type pour chacun des littéraux se trouvant en première position dans une clause.

-Ordre des littéraux dans une clause

Nous avons vu que l'ordre des littéraux dans une clause est important puisque une"instruction"PROLOG est une clause ordonnée. De ce fait une clause +A +B si l'on veut qu'elle serve aussi bien dans un sens que dans l'autre, devra s'écrire à l'aide des deux règles

```
+A +B ..

+B +A ..
```

ex la règle de transivité de l'inclusion

```
-Q(x,y) -Q(y,z) +Q(x,z)
```

s'écrira en PROLOG à l'aide des trois règles ordonnées:

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

-Q(*y,*z) -Q(*x,*y) +Q(*x,*z) ;.

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

Il n'est pas nécessaire pour avoir un système complet d'écrire les 3 autres règles qui consistent à permuter les deux derniers littéraux de chacune des règles précédentes. Ce qui est important c'est d'avoir en première position chacun des littéraux de la clause, car le premier littéral d'une clause PROLOG est en fait le nom d'une procédure et sert en même temps à faire des tests sur la forme des arguments. Les autres littéraux de la clause

peuvent alors être énoncés dans n'importe quel ordre. Nous les écrirons donc dans l'ordre qui nous parait être le plus adapté aux déductions futures.
D'une manière générale seront placés en tête les littéraux ayant le moins de chance de s'unifier: c'est le cas des littéraux P(L,V) où certains traits dans la liste L et le verbe V réduiront les unifications possibles. Les littéraux de contrôle sont alors placés en fin de clause: c'est le cas des littéraux du type DS(x,y) et même parfois Q(x,y) ou EG(x,y).

ex Reprenons la clause:

-P(L,V) -DS(SP(z,x),L) -Q(y,x) -SBT(P(L,v),L,x,y,T) +P(T,V)

Cette clause va se traduire pour réaliser les déductions qui nous intéressent, par les deux règles:

+P(*T,*V)-P(*L,*V) -SBT(P(*L,*V),*L,*x,*y,*T) -DS(SP(*z,*x),*L)
-Q(*y,*x) ;.

-P(*L,*V) +P(*T,*V) -SBT(P(*L,*V),*L,*x,*y,*T) -DS(SP(*z,*x),*L)
-Q(*y,*x) ;.

Ces deux règles permettent de montrer, pour la première, qu'une phrase P(*T,*V) est vraie si la phrase P(*L,*V) est vraie, *T et *L vérifiant les conditions exprimées dans les littéraux qui suivent. La deuxième inversement permet de montrer que P(*L,*V) est faux si P(*T,*V) est faux, *L et *T vérifiant les mêmes conditions que précédemment.

-Autres types de contraintes

D'autres types de contraintes vont être introduits pour nous permettre de mieux contrôler l'exécution.

-Contrôler si un terme est une variable pure

Nous serons souvent amenés à ne permettre l'unification que si un des arguments du littéral n'est pas une variable pure.
ex Considérons la transivité de l'inclusion:

+Q(*x,*y) -Q(*x,*z) -Q(*z,*y) ;.

Nous avons vu déjà que nous empêchons l'unification si les arguments *x et *y sont des variables pures. De même la résolution de cette clause doit affecter à *z une certaine valeur A, c'est à dire qu'après avoir résolu le deuxième littéral -Q(*x,*z), nous allons demander que *z ne soit plus une variable pure mais soit en quelque sorte ground ou un terme construit à partir d'un symbole fonctionnel. Pour cela nous utiliserons le prédicat évaluable Copie:

```
+Copie(*z,bidon1)
```

prend la valeur vraie si une copie de *z peut s'unifier avec bidon1, ce qui n'est possible que si *z est une variable pure: la clause précédente s'écrira alors:

```
+Q(*x,*y) -Q(*x,*z) +Copie(*z,bidon1) -Q(*z,*y) ;.
```

-Utilisation du prédicat DIF(x,y)

Un autre contrôle va être réalisé à l'aide du prédicat DIF(x,y) qui vérifie que deux termes x et y sont formellement différents.
ex Considérons la symétrie et la réflexivité de l'égalité:

```
+EG(*x,*y) -EG(*y,*x) ;.

+EG(*x,*x) ;.
```

La première règle ne doit donc servir que si *x est formellement différent de *y, ce que nous réaliserons en utilisant le prédicat DIF:

```
+EG(*x,*y) -EG(*y,*x) -DIF(*x,*y) ;.

+EG(*x,*x) ;.
```

De même pour la transivité de l'inclusion nous allons écrire la clause en posant comme condition que *x, *y et *z soient différents formellement:

```
+Q(*x,*y) -Q(*x,*z) +Copie(*z,bidon) -Q(*z,*y)
          -DIF(*x,*y) -DIF(*x,*z) -DIF(*y,*z) ;.
```

-Etablissement de contraintes à l'aide du prédicat DAF.

Un autre type de contrainte va être réalisé en utilisant le prédicat DAF. Pour cela nous allons introduire dans chacun des

des prédicats P,Q,EG, CARD,ETRE, une variable supplémentaire i:

P(L,V,i)

Q(x,y;i)

EG(x,y,i)

CARD(x,i,J)

ETRE(x,y,i)

Cette variable sera alors testée à l'aide du prédicat DAF, pour empêcher certaines unifications. Considérons par exemple la symétrie de l'égalité:

+EG(*x,*y) -EG(*y,*x) -DIF(*x,*y) ;.

cette règle peut s'appliquer sans cesse sur elle-même, c'est à dire que si nous résolvons

-EG(A,B) sur cette clause, la résolvante

-EG(B,A) peut encore être résolue sur cette clause.

Pour empêcher cela nous allons donc utiliser le prédicat DAF, en introduisant dans EG une variable supplémentaire:

+EG(*x,*y,*i) -DAF(*i,1) -EG(*y,*x,1) -DIF(*x,*y) ;.

Cette clause sera donc utilisée pour démontrer un littéral

EG(*x,*y,k)

que si k est différent formellement de 1. K doit alors être un "nombre" quelconque différent de 1, ou une variable pure *i. Nous appelerons cette variable une "variable de controle". Elle nous permet de diviser l'ensemble des clauses permettant de démontrer un certain littéral P de manière à ce que ce littéral ne puisse être cancelé que sur une partie de ces clauses. Supposons que nous ayons n clauses permettant de résoudre $P:P_1P_2\ldots P_n$. Nous pouvons par exemple contrôler l'exécution en ne permettant l'unification sur P_i que si la variable de contrôle est différente de i. Ainsi le littéral P, si la variable de contrôle est égale à i, ne pourra-t-il être résolu sur $P_1\ldots P_{i-1}, P_{i+1},\ldots P_n$. Le littéral P, si la variable de contrôle est $*^i$ (une variable pure) ou un

nombre k différent de 1,2,...n, pourra alors s'unifier sur $P_1P_2..P_n$
Donc en fait, une variable de contrôle représentée par une variable pure *i constitue une contrainte nulle.
Les clauses générées à partir du texte seront toutes constituées de littéraux avec une contrainte nulle, c'est à dire une variable de contrôle *i. Ces littéraux pourront donc être résolus sur n'importe quelle clause, et inversement, tout littéral, quel que soit sa contrainte pourra être résolu sur une de ces clauses.
ex Considérons les règles permettant de démontrer qu'une phrase P est vraie.

-nous avons premièrement les clauses du texte, qui ont donc une contrainte nulle.
-nous avons la clause P_1 (permutation) qui va s'écrire:

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

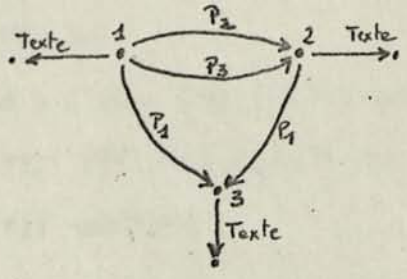
-nous avons la clause P_2 (égalité) qui va s'écrire:

```
+p(*T,*V,*i) -DAF(*i,1) -DAF(*i,2) -P(*L,*V,2) -SBS(*L,*y,*x,*T)
             -DS(SP(*z,*y),*L) -EG(*x,*y,*J) -DIF(*x,*y) ;.
```

-nous avons la clause P_3 (inclusion) qui va s'écrire:

```
+P(*T,*V,*i) -DAF(*i,1) -DAF(*i,2) -P(*L,*V,2)
             -SBT(P(*L,*V),*L,*y,*x,*T) -DS(SP(*z,*y),*L)
             -Q(*x,*y,1) -DIF(*x,*y) ;.
```

La stratégie utilisée pour démontrer une phrase P(L,V,i) peut alors être décrite à l'aide du graphe suivant:



au noeud 1, nous avons une contrainte nulle, la phrase peut être résolue sur toutes les clauses
au noeud 2, après avoir déjà utilisé P_2 ou P_3, la contrainte est telle qu'elle ne permet d'utiliser que P_1 ou le texte.
au noeud 3, nous ne pouvons utiliser que les clauses du texte.

D'une façon générale le Texte ne contenant pas de contrainte pourra toujours être utilisé.

III.5.4 Appel de ces procédures de déductions

On peut considérer que les règles décrites précédenmment constituent un ensemble de procédures permettant de démontrer les littéraux construits à l'aide des prédicats P, EG,Q,ETRE,CARD.
En fait il faut ajouter à cet ensemble de clauses, les clauses du texte lui-même. Nous avons donc un ensemble de clauses ordonnées, les clauses du texte précédant les clauses correspondant aux axiomes généraux de déductions.
Parmi les clauses générées nous avons des clauses spéciales, du type:

```
+R(*i,*x) .....
```

qui signifient "la *ième réponse est *x si..." et qui constituent donc une procédure permettant de trouver la réponse à la *iè question. L'exécution du programme va consister à "appeler" successivement ces procédures dans l'ordre des questions. C'est à dire d'abord +R(1,*x)..., puis +R(2,*x)..., jusqu'à ce que le numéro i de la question soit égal au nombre maximum de questions, nombre qui est donné par la clause:

```
+NQ(*I) ;.
```

Ceci est réalisé à l'aide des clauses suivantes:

```
+NQ(*I) -R(*I,*x) +Rep(*I,*x) ;;

+NQ(*I) -NQ(*I') ..
```

Le corps du programme lui-même est alors simplement composé de:

```
-NQ(0') ..
```

cette clause constituant à elle-seule le fichier DONNEES.
Les clauses étant ordonnées, nous allons avoir dans l'ordre dans le fichier axiomes:

```
-
-
-
+NQ("MAX")..                          } ensemble de clauses
-                                     } générées.
-
-
+NQ(*I) -R(*I,*x) +rep(*I,*x) ;;      } axiomes généraux
+NQ(*I) -NQ(*I') ..                   }
-
-
-
```

Donc résoudre la clause

```
-NQ(O')
```

va consister:
1°) à vérifier que O' n'est pas égal à "MAX", qui est le nombre maximum de questions; sinon l'exécution est terminée.
2°) à démontrer alors la clause résolvante suivante, après avoir unifié O' avec *J.

```
-R(O',*x) +Rep(O',*x) ;;
```

qui signifie en fait: "si la réponse à la 1ere question est *x alors la réponse est *x"
3°) à unifier le littéral -NQ(O') avec la 3ème clause pour récupérer la résolvante:

```
-NQ(O'')
```

sur laquelle nous allons recommencer le même travail pour trouver la réponse à la 2ème question si elle existe.
Une des dernières clauses de l'ensemble d'axiomes va être la clause:

```
+R(*I,je.ne.sais.pas) ..
```

qui signifie alors "la réponse à la *Ième question est je ne sais pas", et ce quelque soit *I. En fait comme nous ne cherchons qu'une seule réponse par question, cette règle ne s'appliquera que si nous n'avons pu obtenir de réponses à la question.

III.5.5 Exemples de déductions faites sur un texte:

Nous donnons ici des exemples de réponses obtenues sur des textes, en donnant simplement le texte d'entrée en français tel qu'il est introduit dans le système, et les réponses obtenues en français également. En fait les réponses sont données sous la forme:

+SORT(x)

où x est effectivement la réponse en français. Le prédicat SORT(x) signifie simplement "x est une réponse" et est utilisé pour obtenir les réponses sur le fichier réponse PROLOG. Nous avons pour générer les réponses en français quelques axiomes supplémentaires décrivant un mini-synthétiseur du français.

Exemples 1

```
TEXTE D' ENTREE:

TOUT HOMME QUI EST CELIBATAIRE, PEUT CONVOITER TOUTE
FEMME QUI EST CELIBATAIRE.TOUT HOMME QUI EST RICHE,PEUT EPOUSER
TOUTE FEMME QU'IL PEUT CONVOITER.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) ;.
```

Exemple 2

TEXTE D' ENTREE:

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

REPONSE :

+SORT(A*MARSEILLE) ;.

+SORT(JE NE SAIS PAS) ;.

Exemple 3

TEXTE D' ENTREE:

*AMEDEE HABITE AU *CANADA.
*JULES Y POSSEDE UNE VOITURE.
OU HABITE *AMEDEE?
QUE POSSEDE *JULES?
QUI HABITE AU *CANADA?

REPONSE :

+SORT(AU*CANADA) ;.

+SORT(UNE CERTAINE VOITURE) ;.

+SORT(*AMEDEE) ;.

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

LIRE CEKONINSERICI
AMEN

LIRE TRADUCTION-
```

on insère ici les données qui sont constituées par la clause +STRUCTURESYNTAXIQUE(X), X étant la structure syntaxique du texte.

```
** APPEL DE LA PROCEDURE TTEXTE ..

-STRUCTURESYNTAXIQUE(*X) -TTEXTE(O/*I,O/*J,*X/*Y) +ONA(NO(*I').*Y) ..

** PROCEDURE TTEXTE.TRANSFORMATION DE TEXTE ..

+TTEXTE(*I/*II,*K/*K1,*X.*Y/*XX.*YY)
    -TPARAGRAPHE(*I/*I2,*K/*K2,NIL/*Z,O/NIL,*X/*XX)
    -TTEXTE(*I2/*II,*K2/*K1,*Y/*YY) ..

+TTEXTE(*I/*I,*K/*K,NIL/NIL) ..

** PROCEDURE TPARAGRAPHE.TRANSFORMATION DE PARAGRAPHE ..

** REGLE TERMINALE..

+TPARAGRAPHE(*I/*I,*J/*J,*K/*K*X,NIL/NIL) ..

** PHRASE DE TYPE OUINON..

+TPARAGRAPHE(*I/*II,*J/*JJ,*K/*KK,O/*X,P(OUINON,*L,*V).*Z/*D.(*Y IMP
      R(*I',OUI)).((NO *Y1) IMP R(*I',NON)).*Z1)
     -TDEF(*J/*JO1,*K/*KO1,O/*X,*L/*D)
     -COPIE(*KO1.P(OUINON,*L,*V),*KO1.P(OUINON,*LL,*V))
     -TQUANTIF(*L/*L1,*V/*V1)
     -TPHRASE1(O/O,*JO1/*J1,*KO1/*K1,1/*X,*L1,P(AFF,*L1,*V1).NIL/*Y)
     -TQUANTIF(*LL/*LL2,*V/*V2)
     -TPHRASE1(O/O,*J1/*J2,*K1/*K2,O/*X,*LL2,P(AFF,*LL2,*V2).NIL/*Y1)
     -TPARAGRAPHE(*I'/*II,*J2/*JJ,*K2/*KK,O/*X,*Z/*Z1) ..

** PHRASE DE TYPE INTERO..

+TPARAGRAPHE(*I/*II,*J/*JJ,*K/*KK,O/*X,P(INTERO,*L,*V).*Z/*D.*W1.*Z1)
      -TDEF(*J/*JO1,*K/*KO1,O/*X,*L/*D)
      -TQUANTIF(*L/*L1,*V/*V1)
      -TPHRASE1(*I/*I',*JO1/*J1,*KO1/*K1,O/*X,*L1,P(AFF,*L1,*V1).NIL/
      *W1)
      -TPARAGRAPHE(*I'/*II,*J1/*JJ,*K1/*KK,O/*X,*Z/*Z1) ..

** PHRASE DE TYPE AFF..

+TPARAGRAPHE(*I,*J,*K,*X,P(*Y,*L,*V).*Z/*W)
      -TQUANTIF(*L/*L1,*V/*V1)
      -TPHRASE1(*I,*J,*K,*X,*L1,P(*Y,*L1,*V1).*Z/*W) ..
```

```
** PROCEDURE TPHRASE.TRANSFORMATION DE PHRASE ..

** REGLE TERMINALE ..

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

** SN PRON ..

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

** SN PROPRE ..

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

** SN INDEF ..

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

** SN DEF ..

+TPHRASE(*I,*J,*K,*X,SP(*Z,SN(*A,DEF,*U)).*V,*W)
    +COPIE(*A,BIDON)
    -TPHRASE(*I,*J,*K,*X,*V,*W) .;

+TPHRASE(*I,*J,*K,1/*X,SP(*Z,SN(*A,DEF,*U)).*V,*W/*W1)
   -TPHRASE(*I,*J,*K,1/*X,SP(*Z,SN(*A,INDEF,*U)).*V,*W/*W1)..

+TPHRASE(*I,*J0/*JJ,*K0/*KK,0/*X,SP(*Z,SN(*A,DEF,*U)).*V,*W/*D.*U1.*S1
   .*W1)
   -DIV(*U,*REST1,*APOS1)
   -TDEF(*J0/*J,*K0/*K,0/*X,*REST1/*D)
   -DEPLIST(*REST1,*K0/*API,NIL/*L)
   -COPIE(*K.SN(*A,CHAQUE,*REST1),*K.*SN)
   -DEF(*J/*J1,*K/*K1,0/*L,*A,*B,*C)
   -TPARAGRAPHE(0/0,*J1/*J2,*K1/*K2,0/*X,*U/*U1)
   -TPARAGRAPHE(0/0,*J2/*J3,*K2/*K3,0/*X,P(AFF,SP(SUJ.NIL,*SN).SP(OBJ.
   NIL,SN(*A,PRON,NIL)).NIL,OUI(DANS)).NIL/*S1)
   -TPHRASE(*I,*J3/*JJ,*K3/*KK,0/*X,*V,*W/*W1) ..

** SN CHAQUE ..

+TPHRASE(*I,*J0/*JJ,*K0/*KK,*X,SP(*Z,SN(*A,CHAQUE,*U)).*V,*W/*D.*REST
    IMP *APOS.*W1)
    -SCARD(*U,*UU)
    -DIV(*UU,*REST1,*APOS1)
    -TDEF(*J0/*J,*K0/*K,*A,*REST1/*D)
    -CHAQ(*J/*J1,*K/*K1,*A,*A,*B,*C)
```

```
    -TPARAGRAPHE(O/O,*J1/*J2,*K1/*K2,*C,*REST1/*REST)
    -TPARAGRAPHE(O/O,*J2/*J3,*K2/*K3,*B,*APOS1/*APOS)
    -TPHRASE(*I,*J3/*JJ,*K3/*KK,*B,*V,*W/*W1) ..

** SN INTERO ..

+TPHRASE(*I/*I',*J/*JJ,*K/*KK,O/*Y,SP(*Z,SN(*A,INTERO,*U)).*V,*W/*D.
    (*U1.*W1.*PR) IMP R(*I',SP(*Z1,*A)))
    -TDEF(*J/*J01,*K/*K01,O/*Y,*U/*D1)
    -TDEF(*J01/*J02,*K01/*K02,O/*Y,*V/*D2)
    -UNI(*D1.*D2,*D)
    -PRED(*Z,*PR,*Z1)
    -TPARAGRAPHE(O/O,*J02/*J1,*A.*K02/*K1,1/*A.*Y,*U/*U1)
    -SBSP(*W,SP(*Z,SN(*A,INTERO,*U)),SP(*Z1.NIL,SN(*A,PRON,NIL)),*W3)
    -TPHRASE(O/O,*J1/*JJ,*K1/*KK,1/*A.*Y,*V,*W3/*W1) ..

** RECHERCHE EVENTUELLE ET TRAITEMENT DES SYNTAGMES DEFINIS ..

+TDEF(*J/*J,*K/*K,1/*Z,*X/NIL)
+TDEF(*I,*K,*X,VERBE.*SP/*Z) -TDEF(*I,*K,*X,*SP/*Z) ..

+TDEF(*J/*JJ,*K/*KK,O/*X,P(*I,*L,*V).*Z/*R.*T)
   -TDEF(*J/*J1,*K/*K1,O/*X,*L/*R)
   -TDEF(*J1/*JJ,*K1/*KK,O/*X,*Z/*T) ..

+TDEF(*J/*JJ,*K/*KK,*X,SP(*Z,SN(*A,DEF,*V)).*U/*W.*S)
   -COPIE(*A,BIDON) -CONST(*V,*A.*K/*API)
   -TPHRASE(O/O,*J/*J1,*K/*K1,*X,SP(*Z,SN(*A,DEF,*V)).NIL,NIL.NIL/*W)
   -TDEF(*J1/*JJ,*K1/*KK,*X,*U/*S) .;

+TDEF(*J/*JJ,*K/*KK,*X,SP(*Z,SN(*A,*T,*V)).*U/*W.*S)
   -TDEF(*J/*J1,*K/*K1,*X,*V/*W)
   -TDEF(*J1/*JJ,*K1/*KK,*X,*U/*S) ..

+TDEF(*J/*J,*K/*K,*X,NIL/NIL) ..

+CONST(P(*X,*L,*V).*P,*Y/*YY) ..
   -CONST(*L,*Y/*Y1) -CONST(*P,*Y1/*YY) ..
+CONST(SP(*X,SN(*Y,PRON,*U)).*Z,*V/*VV)
   +COPIE(*V.*Y,*P.*P)
   -CONST(*U,*V/*V1) -CONST(*Z,*V1/*VV) ..
+CONST(SP(*X,SN(*Y,PROP,*U)).*Z,*V/*VV)
   -CONST(*U,*V/*V1) -CONST(*Z,*V1/*VV) ..
+CONST(SP(*X,SN(*Y,*T,*U)).*Z,*V/*VV)
   -COPIE(*Y,BIDON)
   -CONST(*U,*V/*V1) -CONST(*Z,*V1/*VV) .;
+CONST(*X.*Z,*V) -CONST(*Z,*V) ..
+CONST(NIL,*X/*X) ..

** CONSTRUCTION DE LA LISTE DE VARIABLES DONT DEPEND LE DEFINI ..

+DEPLIST(*X,NIL/NIL,*Y/*Y) ..
+DEPLIST(VERBE.*SP,*X,*Y) -DEPLIST(*SP,*X,*Y)..
+DEPLIST(P(*I,*L,*V).*Z,*X/*XX,*Y/*YY)
   -DEPLIST(*L,*X/*X1,*Y/*Y1)
   -DEPLIST(*Z,*X1/*XX,*Y1/*YY) ..
+DEPLIST(SP(*V,SN(*A,PRON,*U)).*Z,*X/*XX,*Y/*YY)
   -COPIE(*A,BIDON) +COPIE(*X.*A,*P.*P)
```

```
   -MOINS(*X,*A.NIL,*X1)
   -DEPLIST(*Z,*X1/*XX,*A.*Y/*YY) .;
+DEPLIST(SP(*V,SN(*B,PRON,*U)).*Z,*X/*XX,*Y/*YY)
   +COPIE(*B,BIDON) -UNI(*B,F(*I,*A))
   -MOINS(*X,*A,*X1) -CONCT(*A,*Y,*Y1)
   -DEPLIST(*Z,*X1/*XX,*Y1/*YY) .;
+DEPLIST(SP(*V,SN(*A,*T,*U)).*Z,*X/*XX,*Y/*YY)
   -DEPLIST(*U,*X/*X1,*Y/*Y1)
   -DEPLIST(*Z,*X1/*XX,*Y1/*YY) ..
+DEPLIST(NIL,*X/*X,*Y/*Y) ..

+MOINS(*X,*A.*B,*Y) -MOINSIMP(*X,*A,*Z) -MOINS(*Z,*B,*Y) ..
+MOINS(*X,NIL,*X) ..

+MOINSIMP(*X.*Y,*Z,*X.*YY) -DAF(*X,*Z) -MOINSIMP(*Y,*Z,*YY) .;
+MOINSIMP(*X.*Y,*X,*Y) ..
+MOINSIMP(NIL,*X,NIL)..

+CONCT(*X.*Y,*Z,*X.*YY) -CONCT(*Y,*Z,*YY) ..
+CONCT(NIL,*Z,*Z) ..

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

+TPHRASE1(*I,*J,*K,*X,*L,P(*Q,*L,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL))
   .*Z/*W)
   -TSP(NIL,*L,P(*Q,*L,OUI((NIL-E-S-T.NIL-S-O-N-T).NIL)),*N)
   -CONC(*N,*Z,*M) -LIST(*M,*L1)
   -TPHRASE(*I,*J,*K,*X,*L1,*M/*W) ..

+TPHRASE1(*I,*J,*K,*X,*L,P(*Q,*L,NON((NIL-E-S-T.NIL-S-O-N-T).NIL))
   .*Z/*W)
   -TSP(NIL,*L,P(*Q,*L,NON((NIL-E-S-T.NIL-S-O-N-T).NIL)),*N)
   -CONC(*N,*Z,*M) -LIST(*M,*L1)
   -TPHRASE(*I,*J,*K,*X,*L1,*M/*W) ..

+TPHRASE1(*I,*J,*K,*X,*L,P(*T,*L,R(*V)).*P/*W)
   -TPHRASE(*I,*J,*K,*X,*L,P(*T,*L,*V).*P/*W) ..

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

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

+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,R(*V)))
   -VERB(*V,*V1) -OS(*L,*LL) ..

+LIST(P(*X,*L,*V).*Z,*L) ..

** CREATION EVENTUELLE DU PREDICAT DS ..

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

```
** CONSTRUCTIONS DES VARIABLES ..

+DEF(*J/*J',*K/*K,0/*X,F(*J,*X),0/*X,1/*X) ..
+DEF(*J/*J,*K/*V.*K,1/*X,*V,1/*V.*X,0/*V.*X) ..

+CHAQ(*J/*J,*K/*V.*K,0/*X,*V,0/*V.*X,1/*V.*X) ..
+CHAQ(*J/*J',*K/*K,1/*X,F(*J,*X),1/*X,0/*X) ..

** REGLES SUR TRLIST , UNI , CONC , VERB , OS ET DS ..

+TRLIST(*N,*I/*Y,*I/*N.*Y) -COPIE(*N,BIDON) -COPIE(*N.*Y,*V.*V) .;
+TRLIST(*N,*X,*X) ..

+UNI(*X,*X) ..

+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 DE TRANSFORMATION DU QUANTIFICATEUR ..

+TQUANTIF(*L/*L,NOM(*V)/NOM(*V)) ..
+TQUANTIF(*L/*L,CARD(*V)/CARD(*V)) ..
+TQUANTIF(*L/*L,R(*V)/R(*V)) ..

+TQUANTIF(SP(*P,SN(*N,CHAQUE,*R)).*SP/SP(*P,SN(*N,INDEF,*R)).*SP1,
   NON(*V)/*V1)
   -TQUANTIF(*SP/*SP1,NON(*V)/*V1) ..

+TQUANTIF(SP(*P,SN(*N,CERTAIN,*R)).*SP/SP(*P,SN(*N,INDEF,*R)).*SP1,
   *V/*V1)
   -TQUANTIF(*SP/*SP1,*V/*V1) ..

+TQUANTIF(SP(*P,SN(*N,AUCUN,*R)).*SP/SP(*P,SN(*N,CHAQUE,*R)).*SP1,
   NON(*V)/*V1)
   -TQUANTIF(*SP/*SP1,OUI(*V)/*V1) ..

+TQUANTIF(SP(*P,SN(*N,AUCUN,*R)).*SP/SP(*P,SN(*N,CHAQUE,*R)).*SP1,
   OUI(*V)/*V1)
   -TQUANTIF(*SP/*SP1,NON(*V)/*V1) ..

+TQUANTIF(VERBE.*SP/*SP1,*V/*V1)
   -TTQUANTIF(*SP/*SP1,*V/*V1) ..

+TQUANTIF(NIL/NIL,*V/*V) ..
```

```
+TQUANTIF(*SP.*LSP/*SP.*LSP1,*V/*V1) -TQUANTIF(*LSP/*LSP1,*V/*V1)..

+TTQUANTIF(SP(*P,SN(*N,CHAQUE,*R)).*SP/SP(*P,SN(*N,INDEF,*R)).*SP1,
   NON(*V)/*V1)
   -TTQUANTIF(*SP/*SP1,NON(*V)/*V1)..

+TTQUANTIF(SP(*P,SN(*N,CERTAIN,*R)).*SP/SP(*P,SN(*N,INDEF,*R)).*SP1,
   *V/*V1)
   -TTQUANTIF(*SP/*SP1,*V/*V1)..

+TTQUANTIF(SP(*P,SN(*N,INDEF,*R)).*SP/SP(*P,SN(*N,CHAQUE,*R)).*SP1,
   NON(*V)/*V1)
   -TTQUANTIF(*SP/*SP1,NON(*V)/*V1)..

+TTQUANTIF(SP(*P,SN(*N,AUCUN,*R)).*SP/SP(*P,SN(*N,CHAQUE,*R)).*SP1,
   OUI(*V)/*V1)
   -TTQUANTIF(*SP/*SP1,NON(*V)/*V1)..

+TTQUANTIF(SP(*P,SN(*N,AUCUN,*R)).*SP/SP(*P,SN(*N,INDEF,*R)).*SP1,
   NON(*V)/*V1)
   -TTQUANTIF(*SP/*SP1,OUI(*V)/*V1)..

+TTQUANTIF(*SP.*LSP/*SP.*LSP1,*V/*V1) -TTQUANTIF(*LSP/*LSP1,*V/*V1)..

+TTQUANTIF(NIL/NIL,*V/*V)..

** REGLES SUR SBSP ..

+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) ..
+SBSP(NIL,*X,*Y,NIL) ..


** 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.*Z) +ONA(*X IMP *Y) ;;
-ONA(*X IMP *Y.*Z) +ONA(*X IMP *Z) ..
-ONA(*X IMP *Y IMP *Z) -T(*Y IMP *Z,*U) +ONA(*X IMP *U) ..
-ONA(*X IMP R(*I,*Y)) +ONA(NO *X) +ONA(R(*I,*Y)) ..
-ONA(*X IMP *Y) -DAF(*Y,NIL)
  -DEPEND(*Y,*X)
  +ONA(NO *X) +ONA(*Y) .;
-ONA(*X IMP *Y) +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,*I) ..
-ONA(NO ETRE(*X,*Y)) /-ETRE(*X,*Y,*I) ..
-ONA(CARD(*X,*Y)) /+CARD(*X,*Y,*I) ..
-ONA(NO CARD(*X,*Y)) /-CARD(*X,*Y,*I) ..
-ONA(NO NIL) ..
-ONA(DS(*X,*Y)) /+DS(*X,*Y) ..
-ONA(NO DS(*X,*Y)) /-DS(*X,*Y) ..
-ONA(NQ(*X)) /+NQ(*X) ..

+T(*X IMP *Y.*Z,(*X IMP *Y).(*X IMP *Z)) ..
+T(*X IMP *Y IMP *Z,*X IMP *U) -T(*Y IMP *Z,*U) ..
+T(*X IMP R(*I,*Y),NO(*X.NO R(*I,*Y))) ..
+T(*X IMP *Y,NO(*X.NO *Y)) -DAF(*Y,NIL) -DEPEND(*Y,*X) .;
+T(*X IMP *Y,*Y) ..

+DEPEND(*X,*Y) -COPIE(*X,BIDON) -COPIE(*Y,BIDON) +DAF(*X,*Y) .;
+DEPEND(BIDON1,BIDON2) +COUIC ..
+DEPEND(*X,*Y) -COPIE(*X,BIDON) -DEPEND(*Y,*X).;
+DEPEND(BIDON,*Y) +COUIC ..
+DEPEND(*X.*Y,*Z) -DEPEND(*X,*Z) .;
+DEPEND(*X.*Y,*Z) -DEPEND(*Y,*Z) ..
+DEPEND(*X IMP *Y,*Z) -DEPEND(*X ,*Z) .;
+DEPEND(*X IMP *Y,*Z) -DEPEND(*Y,*Z) ..
+DEPEND(NO *X,*Y) -DEPEND(*X,*Y) ..
+DEPEND(P(*X,*Y),*Z) -DEPEND(*X,*Z) ..
+DEPEND(EG(*X,*Y),*Z) -DEPEND(*X.*Y,*Z) ..
+DEPEND(Q(*X,*Y),*Z) -DEPEND(*X.*Y,*Z) ..
+DEPEND(ETRE(*X,*Y),*Z) -DEPEND(*X,*Z) ..
+DEPEND(CARD(*I,*X),*Z) -DEPEND(*I,*Z) ..
+DEPEND(R(*I,*X),*Z) ..
+DEPEND(DS(*X,*Y),*Z) -DEPEND(*X,*Z) ..
+DEPEND(SP(*X,*Y),*A) -DEPEND(*Y,*Z) ..
+DEPEND(F(*I,*X),*Z) -DEPEND(*X,*Z) ..

**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)),NO P(*YY,OUI(*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(TRADUCTION,(EXONINSERICI,SORTIE,OCCUR)
ECRIRE(SORTIE)


AMEN
```

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

LIRE CEKONINSERICI
** REGLES DE DEDUCTIONS SUR LES PHRASES ..

** PERMUTATION DES ARGUMENTS ..

+P(*T,*V,*I) -DAF(*I,1) -ECRIT(1LISTE(*T)) -P(*L,*V,1)
   -DS1(*T,*L)
   -DIF(*T,*L) ;.
-P(*L,*V,*I) -DAF(*I,1) -ECRIT(2LISTE(*L)) +P(*T,*V,1)
   -DS1(*T,*L)
   -DIF(*T,*L) ;.

** SUBSTITUTION D'UN ARGUMENT PAR UN ARGUMENT EGAL ..

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

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

** AXIOMES DE DEDUCTIONS A PARTIR DE L'INCLUSION ..

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

** 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,*J) +COUIC ..
-CARD(BIDON,*I,*J) +COUIC ..
+ETRE(BIDON1,BIDON2,*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) ;.
```

on insère ici les énoncés logiques.

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

+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,*J) -DAF(*J,1) -EG(*X,*Y,2) -CARD(*X,*I,1) -DIF(*X,*Y) ;.
+CARD(*X,ES(*I),*J) -CARD(*X,E(*I),*J) ;.
+CARD(*X,ES(*I),*J) -CARD(*X,S(*I),*J) ;.

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

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

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

+ETRE(*X,*Y,*I) -DAF(*I,1) +COPIE(*X,BIDON) -ETRE(*X,*Y,1) .;
+ETRE(*X,*Y,*I) -DAF(*I,1) +COPIE(*X,BIDON)
   -ETRE(*Z,*Y,1) -EG(*X,*Z,2) -DIF(*Z,*X) .;
+ETRE(*X,*Y,*I) -DAF(*I,1) +COPIE(*X,BIDON)
   -ETRE(*Z,*Y,1) -ETRE(*R,*T,1) +COPIE(*Y,*T)
   -Q(*X,*R,1) -Q(*R,*Z,1)
   -DIF(*X,*Z) -DIF(*X,*R) -DIF(*Z,*R) ..
```

```
** REGLES SUR CARDL ..

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

** SYNTHESE DES REPONSES ..

 -REP(*I,*J) -ECRIT(*J) ;;

** REPONSE EN OUI, NON, JE NE SAIS PAS ..

-REP(*I,JE.NE.SAIS.PAS)-ECRIT(JE.NE.SAIS.PAS)/+SORT(JE.NE.SAIS.PAS)..
-REP(*I,OUI) -ECRIT(OUI) /+SORT(OUI) ..
-REP(*I,NON) -ECRIT(NON) /+SORT(NON) ..

** ELIMINATION DES PREP OBJ, SUJ, TEMPS ..

-REP(*I,SP(*Z,*U)) -DS(*Z,OBJ.SUJ.TEMPS.NIL) +REP1(*I,SP(NIL,*U)) .;
-REP(*I,*J) -ECRIT(3.*J) +REP1(*I,*J) ..

** NOMS PROPRES ..

-REP1(*I,SP(*Z,INV.*Y)) -SYNTH(*Z.*Y,*T) -ECRIT(*T) /+SORT(*T) ..
-REP1(*I,SP(*Z,ASTER.*U.INC.1)) -SYNTH(*Z.L.APOST.ASTER.*U,*T)
   -ECRIT(*T) /+SORT(*T) ..
```

```
-REP1(*I,SP(*Z,ASTER.*U.MAS.1)) -ECRIT(4.*U) -SYNTH(*Z.LE.ASTER.*U,*T)
   -ECRIT(*T)
   /+SORT(*T) ..
-REP1(*I,SP(*Z,ASTER.*U.FEM.1)) -SYNTH(*Z.LA.ASTER.*U,*T) -ECRIT(*T)
   /+SORT(*T) ..
-REP1(*I,SP(*Z,ASTER.*U.*G.PLU)) -SYNTH(*Z.LES.ASTER.*U,*T) -ECRIT(*T)
   /+SORT(*T) ..
-REP1(*I,SP(*Z,ASTER.*U)) -SYNTH(*Z.ASTER.*U,*T) -ECRIT(*T)
   /+SORT(*T) ..

** CAS OU LA REPONSE EST UN NOM COMMUN ..

-REP1(*I,SP(*Z,*X)) -ETRE(*X,INC.*S.*P,1) -CARD(*X,E(1),*J)
   -BOUM(*SS,*S) -SYNTH(*Z.1.*SS,*T) -ECRIT(*T) /+SORT(*T) .;
-REP1(*I,SP(*Z,*X)) -ETRE(*X,MAS.*S.*P,1) -CARD(*X,E(1),*J)
   -BOUM(*SS,*S) -SYNTH(*Z.UN.CERTAIN.*SS,*T) -ECRIT(*T)
   /+SORT(*T) .;
-REP1(*I,SP(*Z,*X)) -ECRIT(8.*X) -ETRE(*X,FEM.*S.*P,1)
   -CARD(*X,E(1),*J)
  -ECRIT(9.*S.*P)
   -BOUM(*SS,*S) -SYNTH(*Z.UNE.CERTAINE.*SS,*T) -ECRIT(*T)
   /+SORT(*T) .;
-REP1(*I,SP(*Z,*X)) -ETRE(*X,*G.*S.*P,1) -CARD(*X,E(*J),*K)
   -BOUM(*PP,*P) -SYNTH(*Z.*J.*PP,*T) -ECRIT(*T) /+SORT(*T) .;
-REP1(*I,SP(*Z,*X)) -ETRE(*X,*G.*S.*P,1) -CARD(*X,S(*J),*K)
   -BOUM(*PP,*P) -SYNTH(*Z.DES.*PP,*T) -ECRIT(*T)
   /+SORT(*T) .;
-REP1(*I,SP(*Z,*X)) -ETRE(*X,*G.*S.*P,1) -CARD(*X,ES(*J),*K)
   -BOUM(*PP,*P) -SYNTH(*Z.AU.MOINS.*J.*PP,*T) -ECRIT(*T)
   /+SORT(*T) ..

** ELISION ET CONTRACTION ..

+SYNTH(NIL.*X,*Y) -SYNTH(*X,*Y) ..
+SYNTH(*1.*L.*X,*1.L.APOST.*Y) -DS(*1,A.DE.NIL) -DS(*L,LE.LA.NIL)
   -V(*X) -SYNTH(*X,*Y) .;
+SYNTH(*1.*X,*2.APOST.*Y) -DS(*1,LE.LA.DE.NIL) -V(*X) -ELIS(*1,*2)
   -SYNTH(*X,*Y) .;
+SYNTH(A.LA.ASTER.*X,EN.ASTER.*Y) -SYNTH(*X,*Y) ..
+SYNTH(A.L.APOST.ASTER.*X,EN.*Y) -SYNTH(*X,*Y) ..
+SYNTH(A.LE.*X,AU.*Y) -ECRIT(5.*X) -SYNTH(*X,*Y) ..
+SYNTH(A.LES.*X,AUX.*Y) -SYNTH(*X,*Y) ..
+SYNTH(DE.LE.*X,DU.*Y) -SYNTH(*X,*Y) ..
+SYNTH(DE.LES.*X,DES.*Y) -SYNTH(*X,*Y) ..
+SYNTH(DE.DES.*X,DE.*Y) -SYNTH(*X,*Y) ..
+SYNTH(*X.*Y,*X.*Z) -SYNTH(*Y,*Z) ..
+SYNTH(*X,*X) ..

+V(ASTER.*X) -V(*X) ..
+V(*X.*Y) -V(*X) ..
+V(*X) -BOUM(*X,*XX) -VV(*XX) ..
+VV(NIL-H) ..
+VV(NIL-A) .. +VV(NIL-E) .. +VV(NIL-I) ..
+VV(NIL-O) .. +VV(NIL-U) .. +VV(NIL-Y) ..
+VV(*X-*Y) -VV(*X) ..

+ELIS(*1,*2) -BOUM(*1,*11-*111) -BOUM(*2,*11) ..
```

```
AMEN

LIRE SUPPORT
-NQ(0') ..
AMEN

DEMONTRER(CEKONINSERICI,SUPPORT,RESULT)
INTERFACE SORTIE
ASTER=* APOST='
AMEN

OPERATEURS SORTIE
CONCATENATEUR DC  ..
AMEN

ECRIRE(RESULT)

AMEN
```

CONCLUSION

En abordant l'étude de la traduction d'un langage naturel dans celui de la logique du premier ordre nous avons été amenés à opérer un certain nombre de simplifications pour obtenir un premier système opérationnel. Bien que l'ordinateur n'accepte qu'un sous-ensemble du français, ce sous-ensemble est très vaste et, du fait qu'il comprend notamment tous les noms et tous les verbes il nous permet d'avoir avec l'ordinateur des "conversations" portant sur n'importe quel sujet. C'est une des caractéristiques importantes du système.
Dans une première étape, l'extension du système, peut se concevoir assez simplement sur les points suivants:

- sur l'ensemble du français accepté, qui peut être rendu plus large, en permettant l'emploi des épithètes, des complétives et de certaines structures de phrases comme:

  Il y a un chien qui vient

  ou

  C'est un chien qui vient

  Par ailleurs on peut également faire le lien entre un même verbe à différents temps et permettre l'emploi de la première et seconde personne du singulier et du pluriel.

- sur les déductions réalisées, qui peuvent être amméliorées par un meilleur contrôle à l'exécution de la stratégie bien entendu, mais également par certaines modifications au niveau de la représentation logique et de la traduction des phrases. Ainsi il semble préférable de donner aux arguments définis une représentation proche de celle des noms propres, c'est à dire de les représenter par un terme qui soit en quelque sorte "l'arbre syntaxique" de l'argument. Nous pourrions ainsi écrire la règle d'unicité et les propriétés des définis d'une manière générale au niveau des axiomes généraux de déductions. L'avantage immédiat est que deux arguments définis sans relatives sont alors immédiatement identifiés s'ils portent le même nom. Nous pensons par ailleurs que la quantification des phrases peut être réalisée en donnant un rôle particulier au sujet et à certains circonstantiels, comme le lieu et le temps, en adoptant une représentation des phrases fixant un ordre entre les arguments, ce qui ammélioreraît considérablement les déductions.

L'introduction d'un dictionnaire semble s'imposer pour certaines de ces modifications, sa consultation permettant d'acquérir des informations supplémentaires sur les mots. Mais cette consultation doit être conçue de telle sorte qu'un mot n'apparaissant pas dans le dictionnaire puisse être analysé sans que le système soit "bloqué". Une gestion dynamique est alors nécessaire, des informations pouvant être ajoutées ou supprimées en cours d'exécution.
Un projet plus ambitieux consiste à rendre le système conversationnel. A l'heure actuelle le texte comprenant les questions est traité d'un seul bloc, le système répondant alors à toutes les questions. Dans un cadre conversationnel, qui implique que les trois procédures correspondant aux trois tâches principales soient récursives, les phrases pourraient alors être traitées une à une. Cela permettrait également de vérifier qu'une nouvelle phrase n'apporte pas des informations redondantes ou contradictoires à celles déjà en notre possession.
Par ailleurs l'ordinateur pourrait alors lui-même poser des questions à l'interlocuteur dans certains cas d'ambigüités,par exemple pour obtenir des informations supplémentaires.
Nous pensons ainsi, tout en contribuant au développement du nouveau langage de programmation que représente PROLOG,mettre au point un système général permettant effectivement de dialoguer avec l'ordinateur en français.

APPENDICE I

RAPPEL DE LOGIQUE - FORME CLAUSALE

Considérons une formule logique construite en utilisant les connecteurs logiques $\wedge$(et), $\vee$(ou), $\supset$(implique), ainsi que la négation $\neg$(non) et les quantificateurs $\forall$ et $\exists$.

$$\forall x\ ((P(x) \wedge \exists y(Q(y,x) \wedge \forall z\ R(x,y,z))) \supset \exists U\ S(x,U))$$

Nous adoptons une représentation standard des variables où les variables sont représentées par des symboles différents quand il s'agit de variables non déterminées par le même quantificateur.
Nous pouvons faire sur cette formule un ensemble de transformations permettant de l'écrire sous une forme particulière qui est la forme clausale (cf.réf. 4).

Elimination du signe implication ($\supset$)

Ceci s'effectue en utilisant l'équivalence:

$$A \supset B \equiv \neg A \vee B$$

pour toutes les occurences du signe $\supset$
La formule précédente devient donc:

$$\forall x\ (\neg(P(x) \wedge \exists y(Q(y,x) \wedge \forall z\ R(x,y,z))) \vee \exists U\ S(x,U))$$

Réduction de la portée de la négation ($\neg$)

La portée de la négation peut être réduite de telle sorte qu'elle ne s'applique plus que sur un prédicat. Ceci se réalise à l'aide des équivalences suivantes:

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

$$\neg\neg A \equiv A$$

$$\neg\forall x\ A \equiv \exists x\ \neg A$$

$$\neg\exists x\ A \equiv \forall x\ \neg A$$

L'exemple précédent se récrit donc:

$$\forall x\ (\neg P(x) \vee \forall y(\neg Q(y,x) \vee \exists z \neg R(x,y,z)) \vee \exists U\ S(x,U))$$

Elimination des quantificateurs existentiels.

L'algorithme pour supprimer les quantificateurs existentiels va utiliser les "fonctions de Skölem".
Le principe est de supprimer tout quantificateur existentiel $\exists x$, en remplaçant chaque occurence de la variable x définie par $\exists x$, par la "fonction de Skölem" associée à cette variable.
Cette fonction dépend de toutes les variables universelles dominant la variable existentielle éliminée.
L'exemple précédent s'écrit donc:

$$\forall x\ (\neg P(x) \vee \forall y(\neg Q(y,x) \vee \neg R(x,y,f(x,y))) \vee S(x,g(x)))$$

où f et g sont deux nouveaux symboles fonctionnels qui n'apparaissaient pas dans la formule. La fonction de Skölem f associée à z dépend donc de x et y, la fonction de Skölem g associée à u ne dépend que de x.
Si la variable existentielle éliminée n'est dominée par aucun quantificateur universel, la fonction de Skölem génèrée n'aura alors aucun argument, et sera donc une constante.
Nous remplaçons alors chaque occurence de la variable par un symbole n'ayant aucune occurence dans la formule.
Ainsi $\exists x\ P(x)$ s'écrit simplement $P(a)$.
Pour éliminer tous les quantificateurs existentiels il suffit donc de répéter cet algorithme.

Forme prénexe.

Une formule F est dite sous forme prénexe si elle est de la forme PM où P est une suite de quantificateurs, et M une formule sans aucun quantificateur.
P est alors appelé le préfixe de la formule et M la matrice. Pour obtenir la forme prénexe à partir de la formule précédente il suffit donc d'utiliser les règles suivantes:

$$\forall x\ P(x) \otimes R \Rightarrow \forall x\ (P(x) \otimes R)$$

$$R \otimes \forall x\ P(x) \Rightarrow \forall x\ (R \otimes P(x))$$

où le symbole ⊛ est mis pour ∧ ou ∨.
La forme prénexe de la formule précédente est donc:

$$\forall x \forall y \neg P(x) \vee \neg Q(y,x) \vee \neg R(x,y,f(x,y)) \vee S(x,g(x))$$

Forme normale prénexe conjonctive

Une formule normale prenexe conjonctive F est une formule prénexe PM où la matrice M est une conjonction de formules qui sont elles-mêmes des disjonctions de littéraux (un littéral étant un prédicat précédé ou non du signe ¬ de la négation).

Nous pouvons mettre toute matrice M, d'une formule prenexe, sous cette forme en utilisant les règles:

$$A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C)$$

$$(A \wedge B) \vee C \Rightarrow (A \vee C) \wedge (B \vee C)$$

ex

$$\forall x \forall y \forall z \neg (P(x) \vee R(x,z)) \wedge (\neg Q(x) \vee T(f(x))) \wedge S(x,y,z)$$

Nous pouvons alors distribuer le préfixe sur la conjonction puisque $\forall x \ (A \wedge B)$ est équivalent à $(\forall x \ A) \wedge (\forall x \ B)$.
Nous obtenons donc ainsi des formules qui ne sont plus que des disjonctions de littéraux quantifiés universellement. De telles formules sont alors appelées des Clauses.
L'exemple précédent se traduit donc par les clauses:

$$\forall x \forall y \forall z \ \ P(x) \vee R(x,z)$$

$$\forall x \forall y \forall z \ \ \neg Q(x) \vee T(f(x))$$

$$\forall x \forall y \forall z \ \ S(x,y,z)$$

Représentation des clauses

Pour représenter les clauses nous allons supprimer ces quantificateurs universels puisque nous savons que toutes les variables ayant une occurence dans la formule possède une telle quantification.
Par ailleurs le connecteur entre les littéraux d'une même clause étant toujours le connecteur ∨ , nous pouvons également l'omettre par convention.

Nous ferons précéder l'écriture d'un prédicat du signe + (qui sera parfois omis) si ce prédicat n'est pas négativé, du signe - si le prédicat est négativé.
L'exemple précédent s'écrit donc:

$$+P(x) + R(x,z)$$

$$-Q(x) + T(f(x))$$

$$+S(x,y,z)$$

REMARQUE

Considérons la formule suivante:

$$\forall x\ (P(x) \vee \forall y\ R(x,y) \vee \exists z\ Q(x,z))$$

une telle formule se représente par la clause:

$$+ P(x) + R(x,y) + Q(x,F(x))$$

où la fonction de Skölem associée à la variable z est F(x) et ne dépend donc que de x.
Mais cette formule est équivalente à:

$$\forall x \forall y\ (P(x) \vee R(x,y) \vee \exists z\ Q(x,z))$$

ce qui se traduit alors par la clause:

$$+ P(x) + R(x,y) + Q(x,F(x,y))$$

où la fonction de Skölem dépend alors de x et y.
Ceci montre l'intérêt, si nous ne voulons pas faire dépendre les fonctions de Skölem de trop de variables, de limiter la portée des quantificateurs et de ne pas passer immédiatement à la forme prénexe.

APPENDICE II

LE LANGAGE DE PROGRAMMATION PROLOG

Terminologie et principe de base.

Les concepts de base propre à PROLOG sont les suivants: clause, unification et résolution. (cf.Réf. 3)

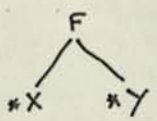
Clauses

Une clause comme nous l'avons vu représente une formule logique du premier ordre. Chaque argument d'un prédicat est un terme.Ces termes sont soit des variables, soit des arbres construits à partir de symboles fonctionnels et d'autres termes.
Les variables d'une clause sont précédées de * pour les distinguer des autres symboles.

ex *x est une variable.

F(*x,*y) est un terme construit à partir du symbole fonctionnel F et des variables *x et *y.

Les termes ont une structure d'arbre, et nous les appelerons aussi bien arbre que terme:



Les littéraux d'une clause PROLOG sont ordonnés (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 dans un sens.

ex l'implication:

$$P(x) \supset Q(x)$$

est équivalente à:

$$\neg P(x) \vee Q(x)$$

qui peut s'écrire en PROLOG de deux façons

- P(*x) + Q(*x)

+ Q(*x) - P(*x)

La première clause permettant alors de démontrer que P(*x) est faux si l'on n'a pas Q(*x).
La deuxième clause au contraire permet de montrer que l'on a Q(*x) si l'on démontre P(*x).
Si l'on veut pouvoir faire les démonstrations dans les deux sens il faut alors écrire les deux clauses.
Un ensemble de clauses est une conjonction de clauses. 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, les clauses sont ordonnées entre elles, et cet ordre est important dans l'exécution d'un fichier, chaque clause pouvant être assimilée en quelque sorte à une suite d'appels de procédures.

Substitutions - Instances

Une substitution est une suite de couples:

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

Appliquer une substitution S sur une expression E (terme, littéral, formule atomique ou clause) consiste à remplacer dans E toutes les occurences des variables figurant dans S par le terme associé.
L'expression obtenue sera notée par E . S et on dira que c'est une instance de E.
Ainsi la substitution S= (*x→A,*y→B) appliquée sur F(*x,*y) donne F(A,B).

Unification

L'unification est l'algorithme de base de toute la démonstration automatique. Nous verrons que dans certains cas il s'apparente à un appel de procédure, mais il est en fait beaucoup plus puissant puisque les données que nous manions ne sont pas toujours constances, mais peuvent comporter des variables.
Soient E_1 et E_2 deux expressions (termes ou formules atomiques).
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 $E_1 . S = E_2 . S$

ex F(*x,H(A,B)) et F(c,*y) sont unifiables par:

S= (*x→C,*y→H(A,B))

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'exécuter" un fichier de clauses.
Le principe est le suivant: Etant donné deux clauses dont leurs premiers littéraux sont opposés

+A+C

-A+D

nous pouvons alors résoudre ces deux clauses pour obtenir, si on résoud la première sur la seconde:

+D+C

Pour déduire cela il suffit en fait de considérer la transitivité de l'implication.
En effet +A+C est équivalent à:

-A ⊃ +C

et -A+D est équivalent à:

-D ⊃ -A

la transitivité nous permet alors de déduire:

-D ⊃ +C

ce qui est équivalent à la clause:

+D+C

Il est bien évident que à la place de C et D nous pouvons avoir une suite de littéraux, la résolution se fera de la même façon. Nous avons dans l'exemple pris deux prédicats égaux de signes opposés (-A et +A), c'est à dire en fait deux littéraux sans variables. Le principe reste le même pour les clauses avec variables, la seule différence résidant dans le fait que les deux prédicats doivent être unifiables (au lieu d'être égaux)

ex +P(*x) +Q(*x)

-P(F(*y)) +R(*y)

donne après résolution de la première clause sur la seconde:

+R(*y) +Q(F(*y))

En fait une clause PROLOG est constituée d'une partie principale et d'une partie réponse qui sert à stocker certaines informations et à constituer les clauses de sortie résultant de l'exécution 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 controler l'exécution du fichier.
Dès qu'une clause ayant sa partie principale vide est générée elle est ajoutée au fichier de sortie.

Description du langage PROLOG

Un programme écrit en PROLOG consiste en un ensemble de commandes qui gèrent des fichiers de clauses.
Exécuter un programme consiste alors à résoudre un fichier de clause clauses A (Données) sur un fichier de clauses B (Axiomes), les parties réponses des clauses générées étant envoyées dans un fichier C (Réponse).
On peut en fait considérer que chaque instruction est un énoncé logique de premier ordre et exécuter un programme revient à démontrer un théorème.

Commande LIRE.

Chaque fichier utilisé porte un nom. La lecture de ces fichiers est alors effectuée grace à la commande LIRE suivi du nom du fichier et de l'ensemble constituant le fichier. La fin d'une commande PROLOG est signalé par AMEN.

ex LIRE TEST

+P(*x) -Q(*x) ..

+R(F(*x,*y),*z) ..

AMEN

La commande LIRE crée un fichier portant le nom TEST composé des deux clauses qui suivent.
PROLOG dispose également de deux commandes, INTERFACE et OPERATEURS qui permettent une écriture plus simple des arguments des littéraux.

Commande INTERFACE.

Cette commande permet en fait de redéfinir les unités dans la lecture ou l'écriture future des fichiers de clauses.
Ainsi la commande suivante:

```
INTERFACE  DEF

DEBUT= (   FIN=  )   ONA= +   STOP= :

AMEN
```

exécutée pour la lecture de la clause suivante:

```
ONA  P  DEBUT  F(*x)  FIN  STOP.
```

donnera en mémoire:

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

En fait cette commande permet de redéfinir des unités à l'aide de l'écriture

```
A=B
```

et dans toute la portée de la commande INTERFACE, chaque unité A sera remplacée en mémoire par B pour la lecture, ou se récrira B pour l'écriture.
L'interface reste valable tant qu'un autre interface n'est pas exécuté.

Commande OPERATEURS

La commande opérateurs permet de définir des unités comme opérateurs unaires ou binaires. En outre elle permet de définir un "concatenateur".
Par exemple considérons l'arbre:

```
ET(A ET(B,C))
```

Cette écriture peut être simplifiée en définissant ET comme opérateur par la commande:

```
OPERATEUR  OPER

BINAIRE  DG ET .

AMEN
```

et en écrivant alors simplement:

```
A ET B ET C
```

Par ailleurs si nous définissons ET comme concatenateur DG par la commande:

```
OPERATEUR  OPER

CONCATENATEUR DG ET .

AMEN
```

Il sera alors suffisant d'écrire l'expression sous la forme:

```
A  B  C
```

Par ailleurs il existe également trois commandes permettant d'éditer de copier ou de supprimer des fichiers.

ECRIRE (N_1,N_2,...N_p) qui effectue l'écriture des fichiers nommés par N_1,N_2,...N_p. Cette écriture se fait en accord avec l'interface et les conventions d'opérateurs.

CONCATENER(N,N_1,N_2,...N_p); cette commande fait une copie des fichiers N_1,N_2,...et N_p qui sont concatené pour créer un nouveau fichier N.

SUPPRIMER(N_1,N_2,...N_p) qui a pour effet de supprimer N_1,N_2...et N_p de la mémoire.

Commande DEMONTRER

C'est évidemment la commande la plus importante. C'est elle qui permet d'exécuter un fichier de clause à partir de la méthode de résolution. Cette commande se présente de la façon suivante:

```
DEMONTRER (N1,N2,N3)
```

ou N_1, N_2, N_3 sont les noms de trois fichiers de clauses.

N_1 étant le fichier AXIOME

N_2 le fichier DONNEES

N_3 le fichier REPONSE

Le fichier DONNEES constitue l'ensemble des données du programme constitué par le fichier AXIOMES. Les résultats sont envoyé dans le fichier Réponse.

Démonstration

Les clauses du fichier données sont traitées successivement par le fichier axiome. Le traitement d'une clause C par le fichier axiome s'effectue ainsi: le fichier axiome est parcouru du début à la fin jusqu'à trouver une clause qui puisse se résoudre sur C. Si la clause générée a une partie principale vide, elle est alors envoyé sur le fichier réponse, sinon elle est à nouveau traitée par le fichier axiome.

Déductions - Backtracking

Une déduction est constituée de la suite des clauses générées dans la démonstration d'une clause C. Cela consiste essentiellement à "supprimer" successivement chacun des littéraux $L_1, L_2, \ldots L_n$ de la clause C. Supposons que nous ayons "supprimer" les littéraux $L_1, L_2, \ldots L_i$, il reste donc à supprimer $L_{i+1}, \ldots L_n$. S'il n'est alors pas possible de supprimer L_{i+1} c'est que nous sommes dans une impasse. Il va donc falloir Backtracker pour essayer de canceler L_i d'une autre façon sous certaines conditions de retour.

Conditions de retour

Les conditions de retour sont fixées par la ponctuation des clauses utilisées en axiomes. Il y a quatre types de ponctuations possibles qui ont la signification suivante:

Si une clause A a la ponctuation .. cela signifie que A est une "règle obligatoire" en ce sens que si une clause Cn est résolue avec A pour générer C_{n+1}, lors du backtracking, C_n n'est alors résolu avec aucun autre axiome. On remonte directement à C_{n-1}.

Si une clause A a la ponctuation .; (règle semi-obligatoire) cela va avoir pour conséquence que 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 n'ont pu être supprimés.

Si une clause A a la ponctuation ;. (règle semi-optionnelle) C_n n'est résolu avec d'autres axiomes que si aucune réponse n'a pu être générée dans les déductions sur C_{n+1}, c'est à dire que si nous n'avons pas pu rendre la partie principale de C_{n+1} vide.

Si une clause A a la ponctuation ;; (optionnelle) nous allons avoir alors un backtracking classique en ce sens que C_n sera alors résolu sur les axiomes qui suivent A.

Ces quatre options permettent de controler l'exécution du backtracking et de programmer en quelque sorte des stratégies. 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 un seul axiome. Si les clauses sont ponctuées par .; cela signifie que pour supprimer un littéral d'une clause, un seul parmi les axiomes pourra servir dans le premier pas des déductions. La différence avec .. est que si la clause en .; est utilisée pour tenter de supprimer un littéral, une autre tentative pourra être faite avec une autre clause si la suppression n' a pu être obtenue.
Lorsqu'un ensemble d'axiomes est ponctué par ;. cela signifie qu'une seule réponse peut être obtenue.
Les clauses en ;; signifient que toutes les résolutions possibles sur une clause seront tentées.
Considérons les deux clauses suivantes:

```
ex(1)  -P(A) +R(B) ..

       -P(*y) +R(C) ..
```

cela s'interprète comme:

Si P(A) alors R(B) sinon R(C)

En effet si nous essayons de résoudre le littéral +P(E), nous appliquerons la deuxième règle qui nous donne alors R(C) et il n'y aura pas d'autre façon de résoudre P(E). Par contre si nous voulons résoudre +P(A) nous appliquerons dans l'ordre d'abord la première règle qui produit R(B) et nous n'essaierons pas d'autres résolutions.

ex(2) -ONA(ET(x, y)) +ONA(x) ;;

-ONA(ET(x, y)) +ONA(y) ..

Ces deux règles signifient que si ONA(et(x,y)) alors ONA(x) et ONA(y) . La première règle doit alors être en ;; pour pouvoir de toute manière lors du backtracking tenter la résolution sur la seconde règle.

Littéraux évaluables

Ces littéraux ont tous la particularité de ne jamais être résolus avec les axiomes, c'est à dire qu'ils ne sont pas unifiés avec les axiomes, mais sont évalués par une procédure PROLOG. Ces littéraux sont au nombre de trois: DAF,COPIE,ECRIT et sont définis comme suit:

ECRIT(x) où x est une expression,est toujours vraie, mais son évaluation s'accompagne de l'impression de x.

DAF(x,y) où x et y sont deux expressions, prend la valeur vraie à l'évaluation si x n'est pas formellement égal à y, sinon il prend la valeur faux.

ex DAF(*x,*x) est faux

DAF(*x,*y) est vraie

COPIE(x,y) où x et y sont deux expressions, s'évalue ainsi: x est recopié pour donner l'arbre x'. x' est alors unifié avec y. Si cette unification est possible, le résultat de l'évaluation est vraie, sinon le résultat est faux. Si l'unification est possible celle-ci est appliquée sur toute la clause.

Rappelons que la copie de l'arbre x est un arbre x' ayant la même structure, la seule différence résidant dans le fait que les variables de x sont recopiées en de nouvelles variables qui ne figurent pas dans la clause où se trouve le littéral ± COPIE.

ex une copie de F(*x,H(A,*y),*x)

est l'arbre F(*v,H(A,*u),*v)

Ces trois types de prédicat évaluables sont dits non translatables en ce sens qu'ils occupent une place bien déterminée dans la clause et ne sont évalués que quand ils se trouvent être le premier littéral de la clause à résoudre.
Mais il existe deux autres prédicats évaluables dits translatables, DIF et BOUM, qui sont alors constamment évalués quelque soit leurs positions. Il y a en fait trois possibilités à l'évaluation de tels prédicats:

-l'évaluation est impossible à faire, auquel cas le littéral est conservé et sera évalué ultérieurement quand cela deviendra possible;
-l'évaluation donne vraie, auquel cas en aboutit donc à une impasse et on stoppe la résolution de la clause en backtrackant.
-l'évaluation donne faux, auquel cas le littéral est supprimé et la résolution de la clause se poursuit normalement.

Ces deux littéraux sont définis comme suit:
DIF(x,y) où x et y sont deux expressions, s'évalue comme suit:

FAUX si x et y égaux formellement

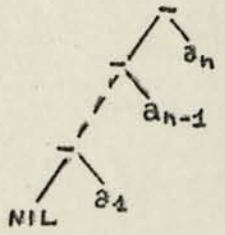
évaluation impossible si x et y unifiables mais non égaux

VRAI si x et y non unifiables.

BOUM(x,y) prend la valeur vraie si les expressions x et y sont sans variables et si y est "l'éclatée" de x ie si les conditions suivantes sont réalisées:

-x est une constante de la forme $a_1 \ldots a_n$ (a_i caractère)

-y est le "peigne" formé à partir de ces caractères comme suit



qui si - est défini comme opérateur binaire gauche-droite s'écrit alors:

$$nil\text{-}a_1\text{-}a_2\text{-}\ldots\text{-}a_{n-1}\text{-}a_n$$

Ce dernier résultat est surtout utilisé dans les règles morphologiques pour pouvoir tester la terminaison de certains mots (plûriel, singulier).

Analogies entre démonstrations et appels de procédures

Lorsqu'une clause C de la forme $\overline{L}_1\overline{L}_2\ldots\overline{L}_n/R_1\ldots R_p$ est traitée par un ensemble d'axiomes, PROLOG tente de supprimer de cette clause la partie principale pour pouvoir déduire de cette clause la partie réponse.
Donc on tente successivement de supprimer $\overline{L}_1, \overline{L}_2\ldots$et $\overline{L}_n$ ($\overline{L}_i$ est le littéral L_i changé de signe), c'est à dire qu'on essaie de démontrer chacun des littéraux L_i à partir des axiomes.
Pour pouvoir démontrer L_1 il faut alors utiliser un axiome de la forme: $L_1\overline{M}_1\ldots\overline{M}_k/S_1\ldots S_1$ après quoi il faut démontrer $M_1\ldots M_k$.
Démontrer un littéral L_1 revient en fait à appeler une procédure avec comme argument la liste d'argument de L_1. Le texte de la procédure est alors constitué de tous les axiomes (ordonnés) dont le premier littéral est identique en signe et en symbole de prédicat.
Chaque axiome représente alors une suite d'appels à d'autres (ou à la même) procédure. Ainsi $L_1\overline{M}_1\ldots\overline{M}_k/S_1\ldots S_1$ représente des appels successifs aux procédures $M_1\ldots$et M_k.

BIBLIOGRAPHIE

1 CHOMSKY,N.,
Syntactic Structures (Mouton,The Hague,1957).

2 COLES,L.S.,
An On-Line Question-Answering System with Natural Language and Pictorial Imput
Proc. ACM 23rd Nat1. Conf., pp. 157-167 (1968).

3 COLMERAUER,A., KANOUI,H.,PASERO,R., ROUSSEL,Ph.,
Un système de communication homme-machine en français
Rapport préliminaire- Octobre 1972 - Université d'Aix-Marseille

4 NILSSON,N.,
Problem-Solving Methods in Artificial Intelligence
Mc Graw-Hill Computer Science Series.

5 REICHENBACH,H.,
Elements of symbolic logic
Mac Millan, New York, 1967

6 ROBINSON,J.A.,
A Machine-oriented Logic Based on the Resolution Principle
J. ACM, Vol.12,N°1,pp.23-41 (1965).

7 ROUSSEL,Ph.,
Définition et traitement de l'égalité formelle.
Thèse de troisième cycle,Mai 1972,Université d'Aix-Marseille.

8 SIMMONS,R.F.,
Natural Language Question-Answering Systems:1969
C.ACM,Vol.13,N°1,pp. 15-30 (Janvier 1970a)