THESE

présentée à

L'UNIVERSITE PARIS 7

par

Paul SABATIER

pour obtenir le grade de
DOCTEUR D'ETAT ES SCIENCES

CONTRIBUTION AU DEVELOPPEMENT D'INTERFACES EN LANGAGE NATUREL

soutenue le 8 Juillet 1987 devant le jury composé de

MM. M. Nivat Président

A. Colmerauer
M. Gross
F. Guenthner
D. Kayser
D. Perrin

THESE

présentée à

L'UNIVERSITE PARIS 7

par

Paul SABATIER

pour obtenir le grade de
DOCTEUR D'ETAT ES SCIENCES

CONTRIBUTION AU DEVELOPPEMENT D'INTERFACES EN LANGAGE NATUREL

soutenue le 8 Juillet 1987 devant le jury composé de

MM. M. Nivat Président

A. Colmerauer
M. Gross
F. Guenthner
D. Kayser
D. Perrin

R E S U M E

Cette thèse matérialise les sept années de recherche que nous avons menées, de 1981 à 1987, dans le domaine de la compréhension automatique du langage naturel appliquée à la conception d'interfaces pour des bases de connaissances.

Notre but était d'identifier, d'étudier et d'assembler de façon homogème les différents composants et connaissances d'une interface, en donnant à chacun le rôle indispensable qu'il a à jouer dans la construction.

Le premier chapitre de notre étude est consacré à la problématique générale des interfaces en langage naturel. L'intérêt, les limites et les fonctions de tels systèmes sont exposés. Nous spécifions les types de connaissances nécessaires à leur mise en oeuvre. Nous montrons comment l'impératif d'extensibilité et de portabilité d'une interface conduit à concevoir une architecture particulière.

Dans le second chapitre, on expose un ensemble de formalismes dérivés des Grammaires de Métamorphoses qui permettent de représenter les connaissances linguistiques nécessaires à l'analyse et à la synthèse de phrases, et d'associer à ces dernières les interprétations sémantiques considérées.

Dans le chapitre III, nous situons les différentes approches sémantiques du traitement du langage naturel. Nous exposons et justifions l'approche retenue et suivie au cours de nos recherches sur les interfaces : l'approche logique, celle de la sémantique formelle comme système de représentation et d'interprétation du langage naturel. Nous donnons les principes essentiels mis en ouvre dans les systèmes réalisés, en particulier pour la compréhension des phénomènes de quantification et de présupposition.

Nous présentons dans le chapitre IV une étude syntaxique et sémantique de phénomènes spécifiques du langage naturel dont l'usage se révèle vital pour une interface : les anaphores, proformes et ellipses. Nous montrons comment intégrer ces phénomènes dans une interface, et expliquons la façon dont ils sont traités dans les systèmes que nous avons réalisés.

La qualité d'une interface en langage naturel ne doit pas seulement être appréciée d'après l'étendue de sa couverture linguistique, mais aussi d'après la qualité des procédures qui sont

mises en oeuvre face aux phrases contenant des expressions non attendues par le système. Dans le chapitre V, nous exposons les différents types d'erreurs et de non-attendus qui peuvent se manifester dans une interface en langage naturel. Nous présentons les différentes techniques de traitement, et leurs incidences sur la conception de l'interface elle-même. Nous démontrons et illustrons l'intérêt d'une approche où l'aide à reformulation et l'assistance à la composition de phrases sont privilégiées.

Dans le chapitre VI, nous présentons les systèmes (bases de données, systèmes-experts et systèmes didactiques) et les interfaces réalisés au cours de nos recherches.

En annexe, nous donnons un ensemble de programmes Prolog commentés qui illustrent certaines des solutions techniques retenues dans ces applications.

Nos travaux permettent de montrer que des solutions peuvent être apportées au problème du dialogue en langage naturel avec des systèmes de bases de connaissances, et que des interfaces conviviales peuvent être construites.

Je remercie

Maurice GROSS, Professeur de Linguistique et d'Informatique à l'Université Paris 7, qui en 1982 m'a accueilli dans son laboratoire et a assuré attentivement la direction de ce travail. Grâce à lui j'ai pu apprendre maintes choses sur la syntaxe et le lexique de notre langue.

Alain COLMERAUER, Professeur d'Informatique à l'Université Aix-Marseille 2, qui en 1978 m'a donné les bases décisives pour me lancer dans cette entreprise, et n'a cessé depuis de me prodiguer idées et conseils.

Franz GUENTHNER, Professeur de Linguistique Informatique à l'Université de Tübingen, pour sa générosité scientifique et son aide amicale.

Daniel KAYSER, Professeur d'Informatique à l'Université Paris XIII, qui a accepté de se situer dans mon approche, et a examiné avec attention ce travail.

Maurice NIVAT et Dominique PERRIN, Professeurs d'Informatique à l'Université Paris 7, qui ont accepté de se distraire momentanément de préoccupations beaucoup plus théoriques pour me faire l'honneur d'être membres du jury.

T A B L E D E S M A T I E R E S

INTRODUCTION

Cette thèse matérialise les sept années de recherche que nous avons menées, de 1981 à 1987, dans le domaine de la compréhension automatique du langage naturel appliquée à la conception d'interfaces pour bases de connaissances.

Compte tenu de l'étendue et de la diversité de la problématique, il nous a fallu, pour obtenir quelques solutions, privilégier certains points dans notre étude. Nos efforts ont ainsi porté sur:

- La formalisation des connaissances lexicales, syntaxiques, sémantiques et logiques du langage naturel.

- L'intégration de ces connaissances et leurs traitements dans le cadre de la conception d'interfaces en langage naturel : stratégies d'analyse et de synthèse de phrases, évaluation de questions, déduction, etc.

- La modélisation du dialogue dans la communication homme-machine : détection et traitements des erreurs, aide à la (re)formulation.

- L'architecture des interfaces dans le cadre de la conception de systèmes extensibles et portables.

Au cours de ces sept années de recherche, nous avons réalisé différents systèmes d'interfaces, chacun d'eux illustrant les solutions proposées pour certains points de la problématique en question.

Cette thèse est ainsi organisée :

Le premier chapitre est consacré à la problématique générale des interfaces en langage naturel : fonctions, intérêts, limites et architectures de tels systèmes.

Dans le second chapitre, on expose un ensemble de formalismes dérivés des Grammaires de métamorphoses et leur mise en oeuvre pour représenter les connaissances linguistiques nécessaires à l'analyse et à la synthèse de phrases.

Le chapitre III est consacré à l'exposition et à la justification de l'approche sémantique que nous avons retenue et suivie au cours de nos recherches : l'approche logique, celle de la sémantique formelle comme système de représentation et d'interprétation du langage naturel.

Le chapitre IV est notre contribution linguistique à la compréhension du langage naturel dans le cadre d'interfaces. On y trouvera une analyse syntaxique et sémantique de phénomènes spécifiques du langage naturel dont l'usage se révèle vital pour une interface : les anaphores (proformes et ellipses).

Compte tenu des limites qu'elle présentera toujours, une interface se doit d'être conviviale. Le chapitre V est consacré à cette problématique et aux solutions concernant la détection et le traitement des erreurs (du "non-attendu") par l'interface.

Dans le chapitre VI, nous présentons les systèmes et interfaces associées que nous avons réalisés au cours de nos recherches.

Nous donnons en annexe un ensemble de programmes Prolog commentés qui illustrent certaines des solutions techniques retenues dans ces applications.

CHAPITRE I

INTERFACES EN LANGAGE NATUREL

Introduction

Fonctions, intérêts et limites des interfaces

Types de connaissances

Formalismes et procédures associées

Extensibilité et portabilité

Architecture d'une interface

Adaptation d'une interface portable

Conclusion

INTRODUCTION

Dans ce chapitre, nous spécifions les fonctions, les intérêts et les limites des interfaces en langage naturel (ILN, par la suite). Nous précisons les connaissances essentielles qu'une ILN doit mettre en oeuvre pour réaliser la tâche attendue. Nous montrons comment l'impératif d'extensibilité et de portabilité d'une ILN conduit à concevoir une architecture particulière. Nous précisons les aspects liés à l'adaptation d'une ILN portable à un quelconque domaine d'application.

FONCTIONS, INTERETS ET LIMITES DES INTERFACES

Une ILN est un système plus ou moins complexe ayant pour tâche de traduire des expressions (assertions, questions, ordres, etc.) d'une langue naturelle (français, anglais, etc.) dans celles d'un langage formel particulier.

Les domaines d'application des ILN sont nombreux et variés. Cela peut être une interface qui permet de traduire un ordre dans une commande propre à un système d'exploitation, comme dans l'exemple suivant :

> Copier dans le catalogue TEXTE tous les fichiers d'extension TXT qui se trouvent dans le catalogue courant et les effacer de ce dernier.

(Exemple de traduction)

```
COPY *.txt \texte\*.txt
DELETE *.txt
```

Cela peut être aussi une ILN pour traduire une question dans une requête formelle propre à un système de gestion de base de données (SGBD), comme dans l'exemple suivant :

> Quels sont les départements de plus de 2 millions d'habitants ?

(Exemple de traduction)

```
SELECT departement, population
FROM table1
WHERE population > 2000000
```

On devine l'intérêt des ILN. L'utilisateur est dispensé d'apprendre un langage formel : le langage utilisé est un sous-ensemble plus ou moins riche de sa langue. Nombre de logiciels

proposent des services quasi identiques mais nécessitent l'apprentisages de langages formels différents. Le langage naturel se présente alors comme un compromis pratique très recherché.

Plus important encore : l'utilisateur qui s'exprime en langage naturel n'a pas à connaître l'organisation et la structures des données pour formuler ses requêtes ou ses ordres. C'est par exemple toute la différence entre un langage de requête de type SGBD et un langage naturel. Le premier est procédural : l'utilisateur doit énumérer dans un certain ordre l'ensemble des opérations à effectuer pour parvenir à l'information recherchée. Le second est déclaratif et indépendant de toute forme de stockage des informations recherchées.

Les ILN s'adressent non seulement aux utilisateurs "occasionnels" mais aussi aux professionnels, qui, pour prendre des décisions, doivent pouvoir accéder facilement et rapidement à des informations stockées sous diverses formes et réparties parfois sur des matériels différents.

L'intérêt du langage naturel réside aussi dans sa puissance d'expression et de concision. Les proformes (pronoms, anaphores et ellipses) sont des phénomènes que l'on peut considérer comme caractéristiques d'un langage dit naturel. Ils assurent un dialogue vif, concis et spontané.

Il est des applications et des situations où une ILN se révèle dépourvue d'intérêt. Par exemple, les commandes d'un éditeur de texte ou d'un système d'exploitation sont avantageusement régies par un jeu de touches programmées, par l'usage d'un menu graphique doté d'une souris, ou encore par un ensemble restreint de commandes formelles. Le langage naturel retrouve cependant son intérêt comme instrument pour faciliter l'apprentissage de ces commandes formelles, comme dans l'exemple suivant :

> Comment renommer un fichier ?

– Utilisez la commande :

RENAME ancien-nom nouveau-nom

> Comment l'effacer ?

– Utilisez la commande :

DELETE nom-du-fichier

Comme substitut d'un langage formel ou comme moyen d'apprentissage de ce dernier, le langage naturel peut donc conduire à la conception de différents types d'interfaces pour

des logiciels.

La fonction première d'une ILN est d'analyser une expression naturelle pour la traduire dans une requête formelle. La production de réponses et de messages peut être réalisée directement par le système interfacé ou être confiée à l'interface elle-même. Dans ce dernier cas, au delà de sa fonction première d'analyse, l'interface aura aussi pour tâche d'interpréter et de synthétiser en langage naturel les réponses et messages émis par le système interfacé.

On trouvera dans le chapitre VI une présentation des ILN que nous avons réalisées dans le cadre des systèmes :

- MICROSIAL (Base de données) [Giraud, Pique et Sabatier 1980, Pique et Sabatier 1982].
- ORBI (Système-expert) [Oliveira, Pereira et Sabatier 1982].
- INTERFRANCE (Base de données) [Duchier, Sabatier 1982].
- INTERFACILE (Système didactique) [Mathieu, Sabatier 1986].

TYPES DE CONNAISSANCES

Une ILN met en jeu plusieurs types de connaissances dont :

- des connaissances sur la langue naturelle choisie pour communiquer;
- des connaissances sur le comportement humain en situation de dialogues;
- des connaissances sur le domaine d'application du système pour lequel l'interface est conçue;
- des connaissances sur le fonctionnement du type de système interfacé;

Les premières connaissances énumérées sont de type linguistique. Elles ont trait aux propriétés lexicales, syntaxiques, sémantiques et logiques de la langue choisie pour communiquer. La qualité et l'étendue de la couverture linguistique d'une interface dépendent de la richesse et de la finesse de cet ensemble de connaissances.

D'une façon générale, l'utilisateur d'une ILN s'attend à trouver

en face de lui un système coopératif et informatif. La résolution d'anaphores (ellipses, pronoms, etc.), l'aide à la (re) formulation, le traitement des erreurs sont des processus dont la mise en oeuvre relève du second ensemble de connaissances. C'est de ces dernières que dépend largement la convivialité d'une interface.

Les deux premiers types de connaissances sont relativement indépendants du domaine d'application et du fonctionnement du système interfacé.

Les connaissances du troisième type permettent de modéliser le domaine application et en sont donc largement dépendantes. Elles définissent la sémantique du domaine en termes d'objets, de relations, d'ensembles de règles et de contraintes.

Le dernier type de connaissances a trait à la structure et aux modes d'accès des informations (ou des services) pour lesquelles l'interface est conçue. Par exemple, dans le cas d'une interface à une base de données, la nature de ces connaissances variera selon que l'on a affaire à un SGBD relationnel, hiérarchique ou de type réseau.

FORMALISMES ET PROCEDURES ASSOCIEES

Les connaissances d'une interface sont représentées au moyen de divers types de formalismes.

Le lexique d'une interface se présente généralement sous la forme d'un dictionnaire. Chaque entrée est constituée par une forme à laquelle est associée une liste d'informations aussi appelées "traits".

Une entrée lexicale peut être une forme de base à partir de laquelle d'autres formes pourront être calculées. Par exemple, les mots:

départements, départemental, départementaux, départementale, départementales

pourront ne pas donner lieu à une entrée du lexique mais être obtenus à partir de l'entrée:

département

et au moyen d'un ensemble de règles de composition syntaxique et

morphologique associant racines, préfixes, suffixes, etc.

La difficulté majeure de cette approche réside dans l'élaboration des règles de composition morphologique. Les règles de flexions pour le genre (masculin vs féminin) et le nombre (singulier vs pluriel) apparemment simples, comportent en fait de nombreuses exceptions comme par exemple:

oeil / yeux (et non "oeils")

empereur / impératrice (et non "empereuse")

Les règles de composition morphologique opérant sur des racines, des préfixes et des suffixes sont difficilement contrôlables quant aux formes qu'elles peuvent engendrer. Par exemple, comment admettre

dépanner (dé + panne + er)
dépanneur (dé + panne + eur)
dépanneuse (dé + panne + euse)
dérailler (dé + rail + er)
dérailleur (dé + rail + eur)

et refuser:

dérailleuse

qui n'existe pas, sans avoir recours à une liste d'exceptions?

Une autre façon de représenter le lexique consiste à définir (en extension) la liste complète des mots et expressions attendus par l'ILN. Par exemple, le lexique peut être constitué par un ensemble de couples de la forme :

<mot ou expression> : <liste de traits>

Exemple :

```
fichier : catégorie(nom-commun),
          genre(masculin),
          nombre(singulier),
          type(fichier).
```

(Dans un formalisme proche de celui-ci, on trouvera dans le chapitre VI le lexique du système INTERFACILE).

Les structures de phrases attendues par une ILN peuvent être représentées par un ensemble de règles de réécriture constituant la GRAMMAIRE de l'interface.

Des procédures de réécriture mises en oeuvre dans le cadre

d'analyseurs (ou de synthétiseurs) sont associées aux modes de représentation des lexiques et des grammaires.

Divers formalismes peuvent être utilisés pour représenter les connaissances sur le domaine d'application du système interfacé. On peut citer :

- les réseaux sémantiques
- les cadres (ou "frames")
- les formalismes logiques.

Des procédures de parcours de graphe et de recherche sont associées aux deux premiers types de représentation. Les formalismes logiques font appel aux principes de raisonnement largement étudiés et expérimentés dans le cadre de la démonstration automatique.

EXTENSIBILITE ET PORTABILITE

La possibilité ou non d'introduire de nouvelles fonctionnalités dans un logiciel existant est révélateur de son caractère extensible ou non.

Réécrire un logiciel donné dans un langage de programmation différent ou l'installer sur diverses familles de matériel (mini, micros, etc.) sont des tâches qui conduisent à évaluer le logiciel comme facilement ou difficilement portable.

Il existe bien sûr divers degrès dans l'extensibilité et la portabilité d'un logiciel. Dans le cas d'une ILN ces notions revêtent cependant une signification et une importance particulières.

Pour ce type de logiciel, l'extensibilité peut être définie comme la possibilité de pouvoir (ou non) modifier le contenu des connaissances linguistiques manipulées par l'interface : lexique, constructions de phrases, etc.

La portabilité d'une interface en langage naturel peut être, elle, définie à différents niveaux. Cela peut être sa capacité (ou non) d'adaptation à :

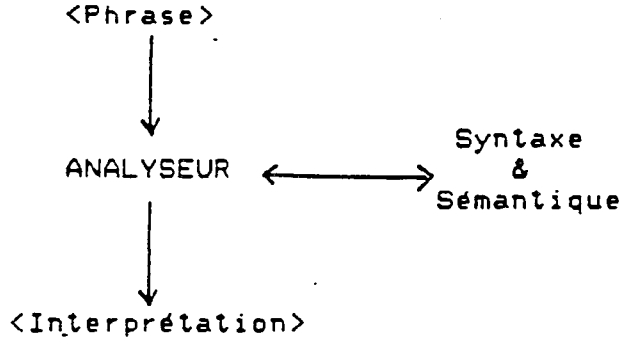
- différentes langues (français, anglais, etc.);

- différents domaines d'application;

- différents types de systèmes (différents types de SGBD par exemple).

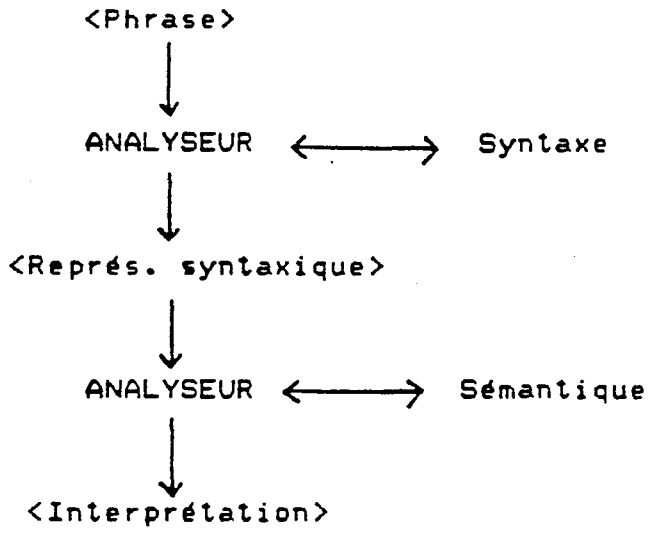
Le souci de faciliter l'adaptation à différents domaines d'application est un point clé dans la conception et le développement d'une interface.

Une interface où les connaissances syntaxiques et sémantiques sont imbriquées dans un seul et même composant présente l'inconvénient d'être difficilement portable vers d'autres domaines d'application. Le schéma suivant illustre une telle approche :



L'intégration des connaissances syntaxiques et sémantiques donne lieu à ce que l'on appelle des "grammaires sémantiques" [Burton 1976].

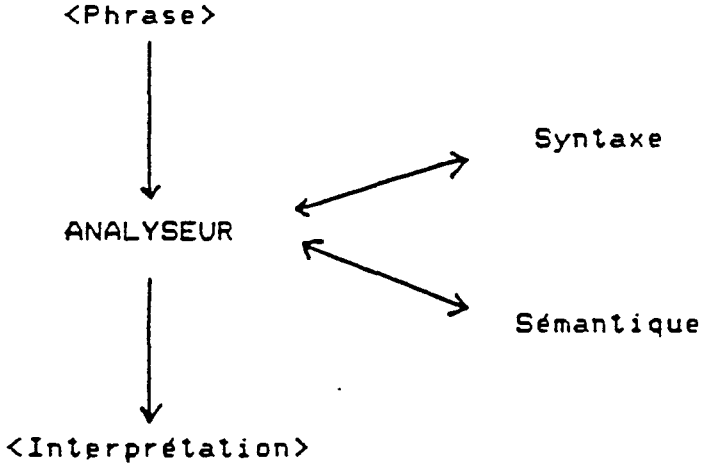
Afin de faciliter l'extensibilité et la portabilité, on est alors conduit à séparer les connaissances syntaxiques des connaissances sémantiques dans deux composants différents. On a alors le schéma suivant :



Dans ce cas, toute phrase est d'abord analysée du point de vue de sa syntaxe. La représentation syntaxique résultant de cette analyse est ensuite prise en compte par un analyseur sémantique qui en produit une interprétation. Cette approche est celle des interfaces des systèmes LUNAR [Woods et al. 1972], CHAT-80 [Pereira et Warren 1982], INTERFRANCE et INTERFACILE.

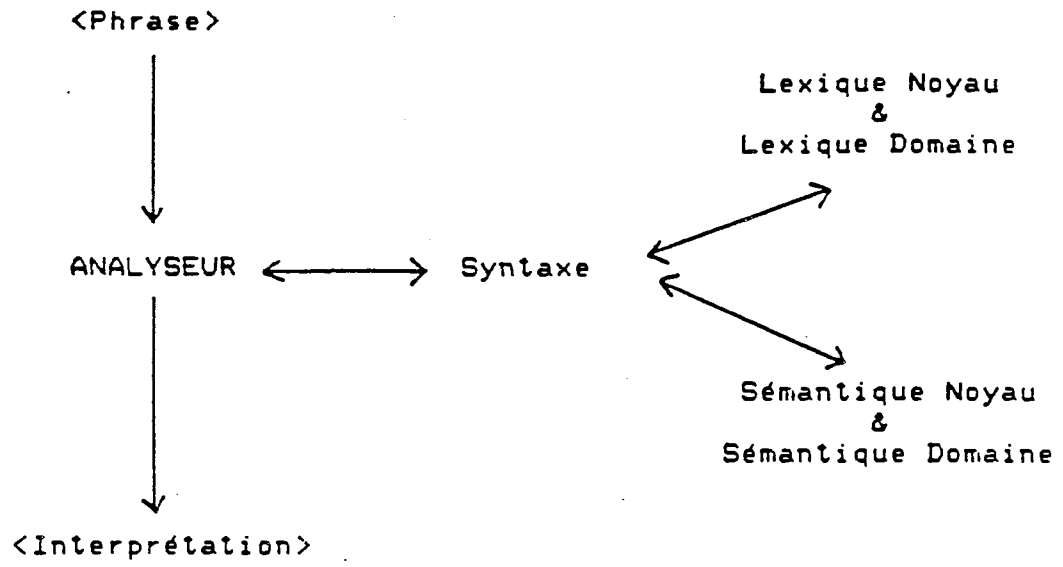
Si elle assure une facilité dans la portabilité et l'extensibilité de l'interface, cette approche présente cependant un inconvénient. Si la grammaire associée au premier analyseur est ambiguë, plusieurs analyses pourront être produites pour une même phrase. Il faudra attendre la fin de l'analyse sémantique pour que certaines ambiguïtés soient levées.

L'inconvénient précédent peut être écarté en conduisant en parallèle les analyses syntaxique et sémantique. Cette interaction peut être illustrée par le schéma suivant :



Cette technique est appliquée en particulier dans les interfaces des systèmes RUS [Bates et Bobrow 1983] et DATALOG [Hafner et Godden 1985].

Pour faciliter leur maintenance, leur portabilité et leur extensibilité, on a eu recours pour les interfaces des systèmes MICROSIAL et ORBI à la configuration suivante :



Dans cette approche, l'analyse est dirigée par le composant syntaxique (Syntaxe). Ce composant est une grammaire formée d'un ensemble de règles de réécriture opérant sur des symboles de catégories générales (phrase, groupe nominal, article, etc.). A certaines règles sont associées :

- des contraintes sémantiques à vérifier,
- des règles pour la composition de l'interprétation et de la traduction.

Les contraintes sémantiques sont générales. Elles portent sur des accords de types. Elles vérifient par exemple le caractère bien formé de la relation sémantique d'un verbe et de ses arguments au niveau des types du sujet et des compléments. La vérification des contraintes est formulée au niveau des règles syntaxiques. La nature de ces contraintes reste propre au domaine d'application et au vocabulaire associé. Ces contraintes sont spécifiées au niveau des mots et des expressions contenus dans le lexique dépendant du domaine ("Lexique Domaine") et de la partie dépendante du domaine du composant sémantique ("Sémantique Domaine"). Le lexique noyau contient les mots et les expressions indépendants du domaine (articles, prépositions, pronoms interrogatifs, etc.).

Concevoir une interface portable conduit à définir un langage intermédiaire (doté d'une syntaxe et d'une sémantique) dans lequel toute phrase en langage naturel sera traduite avant d'être réécrite dans une expression propre au langage formel du système à interfacer. Un sous-ensemble de la logique est un candidat possible pour un tel langage intermédiaire.

Ainsi, pour les systèmes MICROSIAL et ORBI , dans le schéma précédent, la partie indépendante du composant sémantique (Sémantique Noyau) contient en particulier les règles de quantification et de composition de la formule exprimant l'interprétation logique d'une phrase.

Le système TEAM [Martin et al 1983] d'interface à une base de données présente une approche similaire.

ARCHITECTURE D'UNE INTERFACE

Dans le cas où la fonction de l'analyseur n'est pas de produire directement l'expression du langage formel propre au système interfacé (une requête SQL, par exemple), l'ILN comprend aussi un traducteur. Ce dernier a pour fonction de prendre l'interprétation fournie par l'analyseur pour la traduire dans en une expression formelle.

L'analyseur auquel sont associées les connaissances lexicales, syntaxiques et sémantiques, et le traducteur constituent les composants fondamentaux de l'architecture d'une ILN.

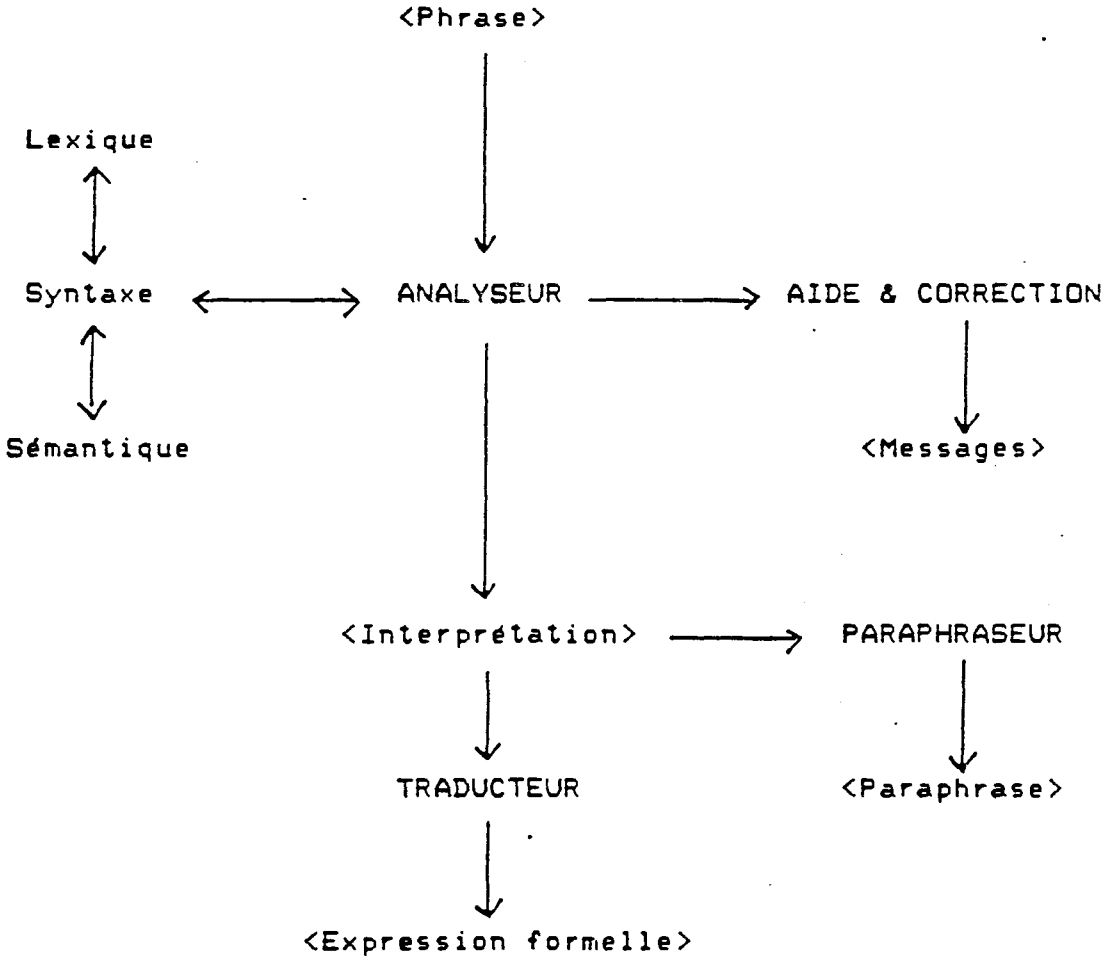
Bien que secondaires, d'autres composants assurent des fonctions importantes dans le cas où une phrase contient des expressions ou des constructions non attendues ou ambiguës. C'est sur ces composants que repose en grande partie la convivialité d'une ILN. Ces composants sont :

- un paraphraseur
- un système pour le traitement des erreurs

Le paraphraseur a pour fonction de pouvoir reformuler en langage naturel toute phrase d'un utilisateur. Cette fonction est nécessaire en cas d'ambiguïté; c'est à dire lorsque l'analyseur attribue à une phrase plusieurs interprétations. Le paraphraseur opère à partir de la (ou des) interprétations(s) fournie(s) par l'analyseur et d'un ensemble de règles de paraphrases.

La requête d'un utilisateur peut contenir des mots mal orthographiés ou inconnus, des constructions non attendues par l'interface ou des erreurs sémantiques sur le domaine d'application. Une ILN conviviale doit être dotée d'un système pour traiter de tels phénomènes et proposer une aide à la (re)formulation.

Une organisation possible de ces composants peut conduire à concevoir une ILN avec l'architecture suivante :



ADAPTATION D'UNE INTERFACE PORTABLE

Pour être utilisée sur un domaine d'application donné, une ILN portable nécessite une phase préliminaire d'adaptation ("customization" en anglais).

Une telle adaptation met en jeu trois ensembles de connaissances à maîtriser :

- des connaissances sur le domaine d'application ("Ce dont il question");
- des connaissances sur le codage des données qui est propre au système interfacé ("Comment est codé ce dont il est question");
- des connaissances linguistiques ("Comment parler sur ce dont il est question").

Potentiellement, la tâche d'adaptation peut être confiée à un des futurs utilisateurs (l'"utilisateur-adaptateur") de l'ILN, au concepteur de cette dernière, ou à l'administrateur du système interfacé.

S'il veut réaliser l'adaptation de l'ILN, le concepteur aura à acquérir et à maîtriser les trois ensembles de connaissances.

Le premier ensemble de connaissances est connu au moins en partie par l'utilisateur-adaptateur de l'ILN. Il sait par exemple qu'il aura à faire avec la base de données du personnel de son entreprise. Il voudra alors demander les noms des employés, leurs dates de naissance, leurs salaires, leurs adresses, leurs âges, etc. D'une façon générale, l'utilisateur-adaptateur ignore (ou veut ignorer) tout ce qui concerne le second ensemble de connaissances propres au codage.

Les deux premiers ensembles de connaissances sont maîtrisés par la personne chargée d'administrer le système pour lequel l'ILN doit être adaptée. Dans le cas d'une base de données, c'est l'administrateur même du SGBD à interfacer. L'administrateur n'est pas forcément un futur utilisateur de l'ILN. En reprenant l'exemple précédent, l'administrateur saura où se trouve l'information relative à la date de naissance d'un employé. Par contre, si l'information concernant l'âge d'un employé n'est pas spécifiée dans la base, il pourra très bien ne pas imaginer qu'une question sur ce sujet soit posée par l'utilisateur de l'interface alors que cette information peut être déduite si on connait et la date de naissance des employés, et la date courante.

Comparées à celles de l'ILN, les connaissances linguistiques requises par la personne chargée de l'adaptation restent minimales. Cette dernière devra définir le lexique dépendant du domaine d'application : noms, verbes, et adjectifs en particulier.

L'administrateur du système interfacé reste la personne la mieux placée pour réaliser les opérations de base de la phase d'adaptation de l'ILN. Il est souhaitable que tout utilisateur puisse lui aussi définir des adaptations qu'il juge nécessaires, comme la définition de synonymes ou d'expressions formulant de nouvelles relations déduites de celles existantes (comme l'âge d'un employé dans l'exemple précédent). Dans tous les cas, pour réaliser ces adaptations, des formalismes et des outils simples et puissants doivent être mis à la disposition de l'administrateur et de l'utilisateur. Ces aspects ont fait l'objet d'une attention particulière dans les ILN des systèmes TEAM [Martin et al 1983], DATALOG [Hafner et Godden 1985] et TELI [Ballard 1985].

CONCLUSION

Nous avons présenté dans ce chapitre quelques uns des multiples aspects des interfaces en langage naturel. Nous avons insisté sur ceux qui nous paraissent fondamentaux parce que conduisant à une approche rationnelle du problème et à la conception de systèmes facilement portables et conviviaux.

Les chapitres qui suivent sont consacrés à la problématique sous-jacente à ces aspects, et aux solutions apportées.

CHAPITRE II

GRAMMAIRES DE METAMORPHOSES ET FORMALISMES DERIVES

Introduction

Grammaires
Dependances et Généralisations
Traductions

Grammaires de Métamorphoses

Formalismes orientés vers l'expression des dépendances
Grammaires d'Extraposition
Grammaires Discontinues
Grammaires de Puzzle

Formalismes orientés vers l'expression des traductions
"Modifier Structure Grammars"
"Definite Clause Translation Grammars"

Stratégies d'analyse et de synthèse : programmation
en Prolog

Applications

Conclusion

INTRODUCTION

Comment traduire des chaînes d'un langage dans celle d'un autre langage?

Telle est la question générale que l'on est amené à se poser lorsqu'on a à concevoir un système, plus ou moins complexe, comme une interface, par exemple, ayant pour tâche de traduire une question en langage naturel dans une expression d'un langage de requête formel.

Si l'ensemble des chaînes des deux langages est fini, il suffit de les énumérer et de les mettre en correspondance.

Si l'ensemble est infini ou de cardinalité élevée (c'est au moins le cas si l'un des langages est naturel), une question préliminaire à celle précédemment énoncée se pose:

Comment définir, reconnaître et produire les chaînes d'un langage dont l'ensemble est en pratique infini?

La réponse à cette question peut être confiée à un système de réécriture, plus ou moins complexe, associant grammaire(s) et stratégie(s) de réécriture.

Au moyen d'un ensemble fini de règles, une grammaire peut définir en compréhension l'ensemble infini des chaînes bien formées d'un langage.

Une stratégie de réécriture formule au moyen d'un algorithme la démarche à suivre pour reconnaître (ou "analyser") et produire (ou "synthétiser") à partir d'une grammaire donnée toutes les chaînes définies.

Le concept de "chaîne bien formée" relève de la grammaire, celui d' "efficacité" de la stratégie. Pouvoir modifier facilement une grammaire sans avoir à toucher à la stratégie; tester et comparer différentes stratégies, et choisir la plus efficace, tel est l'intérêt de vouloir conserver au niveau pratique la séparation conceptuelle entre grammaire et stratégie.

Dans ce chapitre, nous rappelons d'abord les caractéristiques essentielles des grammaires comme systèmes de réécriture. Puis nous précisons les notions de dépendance (et de généralisation) et de traduction. Nous rappelons ensuite le principe des Grammaires de Métamorphoses (GM) qui permettent d'exprimer dans un

formalisme unique les dépendances et les traductions associées aux chaînes d'un langage. Nous présentons une étude des caractéristiques essentielles des formalismes dérivés des GM: ceux orientés vers l'expression des dépendances contextuelles, et ceux orientés vers l'expression des traductions. Nous donnons un aperçu des applications des GM et de ses dérivés. Nous rappelons les principales stratégies d'analyse et de synthèse et leur programmation en Prolog. Enfin, nous concluons sur l'intérêt de concevoir ou non des formalismes dérivés des GM.

GRAMMAIRES

Formellement, au sens de Chomsky [Chomsky 1959], une grammaire G définissant les chaînes d'un langage L est un 4-uplet (T,N,A,R) où:

(1) T est un ensemble fini de symboles appelés symboles TERMINAUX. Les chaînes (ou "phrases") de L sont des suites finies de ces symboles.

(2) N est un ensemble fini de symboles appelés symboles NON-TERMINAUX (ou "symboles auxiliaires").

(3) A est un symbole de N appelé symbole INITIAL (ou "axiome").

(4) R est un ensemble fini de REGLES DE REECRITURE (ou "de production") de la forme:

U -> V

où :

- U est une suite non vide de symboles contenant au moins un symbole non-terminal.

- V est une suite (éventuellement vide) de symboles terminaux et/ou de symboles non-terminaux.

La règle peut se lire:

La suite de symboles U se réécrit dans (ou "peut être remplacée par") la suite de symboles V.

On dira que dans G, une suite de symboles C est une chaîne de type F si :

- F est un symbole non-terminal (F appartient à N),

- F peut se réécrire dans la chaîne C au moyen d'un nombre fini d'applications des règles R de G.

Dans les règles de grammaire, toute expression entre simples apostrophes désigne un symbole terminal. Exemples de symboles terminaux :

'a' 'b' 'Max' 'qui' 'connaît'

Toute expression commençant par au moins deux lettres désigne un

symbole non-terminal. Exemples de symboles non-terminaux :

ss sa phrase gn relative

Par la suite, nous noterons la suite vide par l'expression "<>"

Si pour une des règles, U contient plusieurs symboles, alors la grammaire est dite DEPENDANTE DU CONTEXTE. Le langage formé des chaînes ayant n symboles 'a' suivis de n symboles 'b' suivis de n symboles 'c', c'est-à-dire:

$$a^n b^n c^n \qquad (n > 0)$$

peut être défini à partir des grammaires dépendantes du contexte G1 ou G2.

```
ss -> 'a' ss sa sb          'a' sa -> 'a' 'b'
ss -> 'a' sa sb             'b' sa -> 'b' 'b'
ss sa -> sb sa1             'b' sb -> 'b' 'c'
sb sa1 -> sa sa1            'c' sb -> 'c' 'c'
sa sa1 -> sa sa
```

<Grammaire G1>

```
ss -> 'a' sb sc              'b' sb -> 'b' 'b'
ss -> 'a' ss sb sc           'b' sc -> 'b' 'c'
sc sb -> sb sc               'c' sc -> 'c' 'c'
```

<Grammaire G2>

Si pour toutes les règles, U est un symbole non-terminal, la grammaire est dite INDEPENDANTE DU CONTEXTE. Voici un exemple de ce type de grammaire.

```
phrase -> sn verbe sn
sn -> 'Max'
sn -> 'Luc'
sn -> det nc relative
det -> la
det -> chaque
nc -> 'personne'
verbe -> 'apprécie'
verbe -> 'connaît'
relative -> <>
```

```
relative  ->  'qui'  verbe  sn
relative  ->  'que'  sn  verbe
```

<Grammaire G3>

Deux grammaires sont dites équivalentes si elles définissent le même ensemble de chaînes.

Une grammaire définit les chaînes d'un langage dans les termes de chaînes de constituants et de sous-constituants. Après la réécriture complète des différents symboles, la définition qu'une grammaire donne d'une chaîne a la forme d'une structure. Si toutes les règles qui ont été appliquées sont indépendantes du contexte, la structure assignée à cette chaîne est un arbre qu'on peut représenter soit par une expression parenthésée, soit par un schéma.

DEPENDANCES ET GENERALISATIONS

Une grammaire exprime les dépendances d'occurrence entre les symboles des chaînes bien formées. Les dépendances peuvent être formulées au moyen de règles indépendantes ou dépendantes du contexte, et donneront lieu à deux types de grammaire.

Si une grammaire dépendante du contexte est nécessaire pour définir un langage dépendant du contexte, le principal intérêt de définir au moyen d'une grammaire dépendante du contexte un langage indépendant du contexte est de permettre l'expression de généralisations dans la définition des chaînes du langage. Le recours à des généralisations peut faciliter l'application d'autres traitements.

Par exemple, si on veut définir une relative comme constituée d'un pronom relatif suivi d'une phrase dans laquelle le sujet est absent (si le pronom relatif est "qui") ou l'objet est absent (si le pronom relatif est "que"), et cela en utilisant le symbole "phrase" (employé pour définir une phrase où le sujet et l'objet sont présents), on peut écrire la grammaire G4 dépendante du contexte suivante:

```
phrase  ->  sn  verbe  sn
sn  ->  'Max'
sn  ->  'Luc'
sn  ->  det  nc  relative
det  ->  'la'
det  ->  'chaque'
nc  ->  'personne'
verbe  ->  'apprécie'
verbe  ->  'connaît'
relative  ->  <>
relative  ->  'qui'  phrase
relative  ->  'que'  phrase
'qui'  sn  verbe  sn  ->  'qui'  xx  verbe  sn
'que'  sn  verbe  sn  ->  'que'  sn  verbe  xx
xx  ->  <>
```

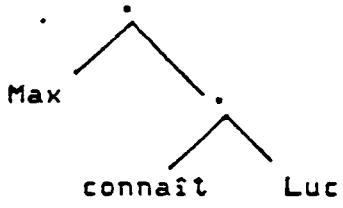
<Grammaire G4>

TRADUCTIONS

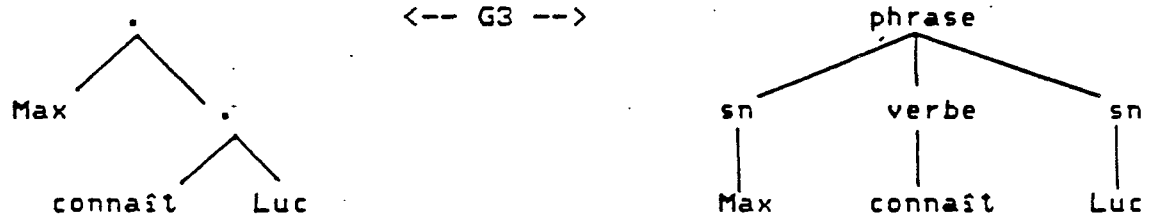
Une grammaire peut être vue comme un formalisme mettant en relation un ensemble de chaînes et un ensemble d'arbres. Une chaîne n'est rien d'autre qu'un type particulier d'arbre (qu'on appelle aussi LISTE). C'est un arbre binaire en cascade. Par exemple, la chaîne :

Max connaît Luc

peut être représentée par l'arbre binaire en cascade:



où le noeud "." peut se lire "est suivi de". Ainsi, par exemple, G3 met en relation les deux arbres suivants:



Lorsqu'elle fonctionne en analyse, une grammaire peut être vue comme un système traduisant un type particulier d'arbres (les "chaînes" ou "listes") dans d'autres arbres. L'arbre d'analyse (ou "de dérivation") d'une chaîne peut être vu comme la trace finale de l'ensemble des règles qui en s'appliquant ont conduit à la reconnaissance de la chaîne. L'arbre de dérivation (ou "structure") fournie par la grammaire pour une chaîne donnée est une traduction particulière de cette chaîne.

Cette traduction peut servir de représentation intermédiaire pour la production de traductions plus complexes. Si on veut associer à la structure d'une chaîne une traduction autre que le simple arbre de dérivation (comme une formule sémantique par exemple), deux méthodes sont possibles:

(1) soit, on conçoit un autre système qui aura pour tâche de produire la traduction choisie à partir de la structure produite par la grammaire, après que la structure soit produite, ou au fur et à mesure qu'elle se construit.

(2) soit, on introduit dans la grammaire les éléments

nécessaires de façon que la traduction choisie soit directement associée à la structure de chaque chaîne.

GRAMMAIRES DE METAMORPHOSES

C'est Alain Colmerauer [1975] qui a conçu et formalisé le principe des Grammaires de Métamorphoses (GM) dont les Systèmes-Q [Colmerauer 1970] peuvent être considérés comme les ancêtres.

En quoi le formalisme des GM se distingue-t-il de celui des grammaires classiques?

Dans les grammaires classiques, les symboles terminaux et non-terminaux sont des symboles simples. Ce sont des symboles fonctionnels sans argument aussi appelés ATOMES. Dans les GM, les symboles peuvent être plus complexes. Ce sont des TERMES. Est appelé terme:

- une variable,
- un symbole fonctionnel sans argument (ou "atome"),
- un symbole fonctionnel dont les arguments sont des termes.

Une variable est ici notée par une seule lettre suivie ou non d'une entier. Exemples de variables :

```
x   y   z   g1   g2
```

Les symboles des grammaires de métamorphoses sont des termes. Voici des exemples de tels symboles :

```
sn(x)

det(masculin,singulier,le)

phrase(x,y)

phrase(sans-sujet,avec-objet)
```

Voici un exemple de règle de GM:

```
phrase(x,y)  ->  sn(x)  verbe  sn(y)
```

Dans une même règle, les variables de même nom désignent le même objet (un terme) connu ou inconnu.

Comme dans ce qui précède, les symboles non-terminaux sont ici formés d'au moins deux lettres. Les symboles terminaux sont notés entre apostrophes simples.

Les symboles étant des termes, leur réécriture est précédée par

une opération d'UNIFICATION [Robinson 1965]. L'unification de deux termes consiste à les rendre égaux en leur appliquant une substitution minimale (si une telle substitution existe).

Dans une GM, des CONDITIONS peuvent être formulées dans la partie droite de chaque règle. Comme les symboles, elles ont la forme de termes. A la différence des symboles, ce sont des appels à des procédures sortant du cadre de l'opération de réécriture. Les conditions expriment en général des contraintes sur la valeur des variables apparaissant dans certains symboles ou dans d'autres conditions de la règle. Le recours aux conditions peut faciliter l'expression de certaines contraintes en rendant l'écriture de la grammaire plus claire.

Voici par exemple une règle de GM contenant une telle condition (notée entre accolades):

```
sn  ->  determinant(x)  nom(y)   { accord-genre-nombre(x,y) }
```

La puissance des GM tient au fait que les symboles peuvent contenir:

- les traductions que l'on souhaite intégrer dans les structures des chaînes,

- les restrictions contextuelles affectant la réécriture des symboles concernés.

De symbole en symbole, la propagation des traductions et des restrictions est réalisée par le simple jeu de l'unification et des variables.

Dans la GM suivante, les arguments des symboles sont utilisés pour associer une traduction aux chaînes du langage :

```
phrase(t)  ->  sn(t,x,r)  verbe(r,x)

sn(r,x,r)  ->  np(x)

sn(t,x,r)  ->  det(t,x,v,r)  nc(v,x)

np(Max)  ->  'Max'

det(existe(x,v,r),x,v,r)  ->  'un'

det(quelque-soit(x,v,r),x,v,r)  ->  'chaque'

nc(exemple(x),x)  ->  'exemple'

nc(exercice(x),x)  ->  'exercice'

verbe(stupide(x),x)  ->  'est'  'stupide'
```

```
verbe(intéressant(x),x)    ->  'est'  'intéressant'
```

<Grammaire GM1>

Avec GM1, la chaîne:

chaque exemple est intéressant

se réécrit dans:

phrase(t)

avec :

t = quelque-soit(x,exemple(x),intéressant(x))

Les dépendances contextuelles peuvent être formulées dans une GM sans avoir recours à des règles de forme dépendantes du contexte. Le jeu des variables des symboles permet toujours une formulation indépendante du contexte. Un exemple intéressant: voici une grammaire de métamorphoses rendant compte du langage formel dépendant du contexte :

$$a^n \; b^n \; c^n \qquad (n > 0)$$

```
ss  ->  sa(n)  sb(n)  sc(n)

sa(1)  ->  'a'

sa(successeur(i))  ->  'a'  sa(i)

sb(1)  ->  'b'

sb(successeur(i))  ->  'b'  sb(i)

sc(1)  ->  'c'

sc(successeur(i))  ->  'c'  sc(i)
```

<Grammaire GM2>

GM2 est équivalente à G1 et à G2.

En désignant dans la première règle par un même nom "n" la variable des symboles, la dépendance est imposée. On a choisi arbitrairement les termes "1" et "successeur(i)" pour exprimer la dépendance au sein des symboles. Par exemple pour la chaîne :

a a a a a b b b b b c c c c c

on aura dans la première règle:

```
n = successeur(successeur(successeur(successeur(1))))
```

Un autre exemple. La grammaire de métamorphose suivante est équivalente à la grammaire dépendante du contexte G4.

La généralisation quant à la définition des relatives (en termes de "phrase") y est cependant beaucoup plus claire que dans G4.

```
phrase(x,y)  ->  sn(x)  sv(y)

sn(present)  ->  'Max'

sn(present)  ->  'Luc'

sn(present)  ->  det  nc  relative

sn(absent)   ->  <>

det  ->  'la'

det  ->  'chaque'

nc  ->  'personne'

sv(x)  ->  verbe  sn(x)

verbe  ->  'apprécie'

verbe  ->  'connaît'

relative     ->  <>

relative     ->  'qui'  phrase(absent,present)

relative.    ->  'que'  phrase(present,absent)
```

<Grammaire GM3>

Le premier argument du symbole "phrase(-,-)" spécifie ici la nature (absent vs present) du sujet; le second, celle de l'objet.

Si pour formuler les dépendances contextuelles, on souhaite ne pas avoir recours à des symboles complexes contenant des variables les GM offrent un formalisme pour procéder ainsi.

Dans les GM, la partie gauche d'une règle peut être constituée d'un symbole non-terminal suivi de symboles terminaux. (Cette contrainte est en fait nécessaire pour programmer et exécuter efficacement l'opération de réécriture).

La forme générale d'une règle de GM est la suivante :

N U -> V

où :

- N est un symbole non-terminal
- U est une suite éventuellement vide de symboles terminaux.
- V est une suite (éventuellement vide) de symboles et de conditions.

A titre d'exemple, GM4 est une grammaire de métamorphoses sans variable (et dont tous les symboles sont des atomes). Comme GM3, GM4 décrit les relatives au moyen du symbole général "phrase".

```
phrase -> sujet verbe objet

sujet -> sn

sujet -> 'sujet-absent'

sujet 'objet-absent' -> 'objet-absent' sujet

objet -> sn

objet -> 'objet-absent'

sn -> 'Max'

sn -> 'Luc'

sn -> det nc relative

det -> 'la'

det -> 'chaque'

nc -> 'personne'

verbe -> 'apprécie'

verbe -> 'connaît'

verbe 'objet-absent' -> 'objet-absent' verbe

relative -> <>

relative -> relatif phrase

relatif 'sujet-absent' -> 'qui'

relatif 'objet-absent' -> 'que'
```

<Grammaire GM4>

Un autre exemple: GM5 sans variable (et dont tous les symboles sont des atomes) est équivalente à G1, à G2 et à GM2 :

```
ss -> 'a' 'b' 'c'

ss -> 'a' sb ss 'c'

sb 'a' -> 'a' sb

sb 'b' -> 'b' 'b'
```

<Grammaire GM5>

Seuls des symboles terminaux peuvent être introduits dans la partie gauche d'une règle, et uniquement à la droite du symbole qui se réécrit. Cette contrainte qui permet de programmer efficacement l'opération de réécriture peut avoir dans certains cas des effets secondaires non négligeables. Dans GM4, les symboles terminaux "sujet-absent" et "objet-absent" ont été créés et introduit afin de formuler les dépendances contextuelles ("qui" et "sujet-absent" dans "phrase", "que" et "objet-absent" dans "phrase"). La contre-partie de la création de ces symboles terminaux est que GM4 reconnaîtra ou produira, non seulement la chaîne:

la personne que Max apprécie connaît Luc

mais aussi la chaîne:

la personne que Max apprécie objet-absent connaît Luc

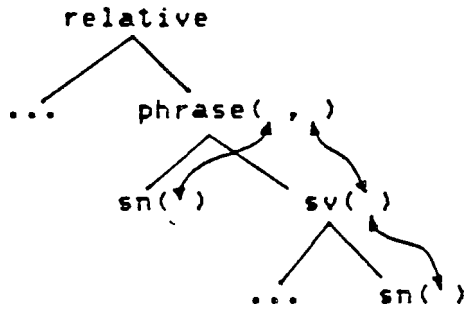
Dans le cadre d'un dialogue homme-machine, il y a peu de chance qu'un utilisateur émette la seconde phrase même s'il a connaissance des symboles terminaux du type "sujet-absent" ou "objet-absent" (que l'on appelle "pseudo-terminaux") utilisés dans la grammaire. Par contre, dans le cas où la grammaire est utilisée en synthèse, il faudra contrôler les chaînes produites pour y retirer les symboles pseudo-terminaux.

Par contre, le langage reconnu ou produit par GM5 est celui "attendu", dans la mesure où aucun symbole pseudo-terminal n'est créé.

Une GM dont chaque partie gauche des règles est constituée par un seul symbole non-terminal est aussi appelée "Definite Clause Grammars" (DCG) [Pereira et Warren 1980]. GM1, GM2 et GM3 sont des grammaires du type DCG.

Le principe de réécriture est un formalisme très puissant pour définir les chaînes d'un langage. Ce principe se révèle encore plus puissant en opérant sur des symboles qui sont des termes, comme avec les Grammaires de Métamorphoses.

Dans une GM, il arrive que l'on ait recours aux arguments d'un symbole seulement pour transmettre des contraintes ou des traductions entre deux symboles distants. C'est par exemple le cas dans GM3 pour les symboles "phrase(-,-)" et "sv(-)" qui transmettent la contrainte entre le pronom relatif ("qui" vs "que") et les "sn(-)" sujet et objet de la phrase constituant la relative. Le schéma suivant illustre ce phénomène.



Cette méthode est très puissante. Pour certains les règles perdent de leur clarté avec la multiplication des arguments et des variables servant uniquement à transmettre les contraintes et les traductions entre symboles très distants, comme c'est généralement le cas dans des grammaires non triviales.

Plusieurs formalismes dérivés des GM ont été proposés. Nous allons maintenant en présenter les caractéristiques essentielles. Ces dérivés ne visent pas à augmenter la capacité générative des GM (qui sont équivalentes aux grammaires de type 0) mais à augmenter leur clarté en ce qui concerne l'expression des dépendances contextuelles et des traductions.

FORMALISMES ORIENTES VERS L'EXPRESSION DES DEPENDANCES

Dans le cadre du principe des GM, plusieurs formalismes ont été proposés afin d'exprimer plus clairement encore les dépendances contextuelles. Nous présentons trois de ces formalismes:

- les Grammaires d'Extraposition
- les Grammaires Discontinues
- les Grammaires de Puzzle

GRAMMAIRES D'EXTRAPOSITION

Le premier formalisme proposé dérivé des GM est celui des Grammaires d'Extraposition (GX) ("Extraposition Grammars") de Fernando Pereira [1981]. Le but de ce formalisme est de faciliter l'expression des dépendances contextuelles, en diminuant le nombre des variables servant à spécifier et à transmettre les dépendances, et en évitant la création de symboles "pseudo-terminaux".

Dans ce but, les GX permettent l'écriture de règles de la forme:

```
N  S1  ...  S2  --- Sn-1  ...  Sn   ->   V
```

La règle s'interprète comme si on avait écrit:

```
N  S1  ...  S2  --- Sn-1  ...  Sn   ->   V   ...  ...
      (a)                (b)                (a)  (b)
```

où:

- N est un symbole non-terminal
- "..." est mis pour une suite non spécifiée de symboles non-terminaux. "..." est aussi appelé TROU ("gap" en anglais).
- S1, S2, Sn-1 et Sn sont des symboles terminaux ou non-terminaux.
- V est une suite de symboles (terminaux et non-terminaux) et de conditions, comme dans une GM ou une DCG.
- Les trous apparaissant dans la partie gauche d'une règle sont replacés après V, et dans le même ordre.

Dans une expression comme:

```
Si  ...  Sj
```

le trou "..." se comporte en fait comme une variable particulière qui s'unifie avec la suite de symboles (éventuellement vide) située entre les symboles Si et Sj.

Voici par exemple la grammaire GX1 équivalente aux grammaires de métamorphoses GM3 et GM4 :

```
phrase  ->  sn(sujet)  verbe  sn(objet)

sn(x)  ->  'Luc'

sn(x)  ->  'Max'
```

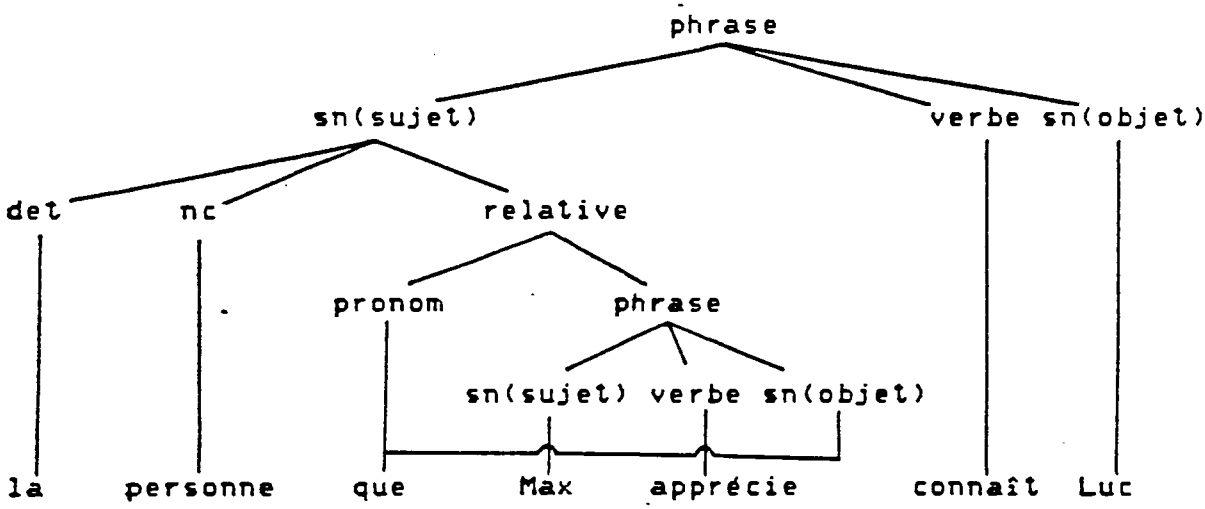
```
sn(x)  ->  det  nc  relative

det  ->  'la'

det  ->  'chaque'

nc  ->  'personne'

relative  ->  pronom  phrase

relative  ->  <>

verbe  ->  'apprécie'

verbe  ->  'connaît'

pronom  ...  sn(sujet)  ->  'qui'

pronom  ...  sn(objet)  ->  'que'
```

<Grammaire GX1>

La chaîne:

la personne que Max apprécie connaît Luc

recevra la structure :



Quant au langage formel reconnu ou produit par GM2 et GM5, voici une grammaire d'extraposition équivalente:

```
ss  ->  sa  sb  sc

sa  ->  'a'

sa  ...  sxb  ->  'a'  sa
```

```
sb  ->  'b'

sb  ...  sxc  ->  sxb  'b'  sb

sc  ->  'c'

sc  ->  sxc  'c'  sc
```

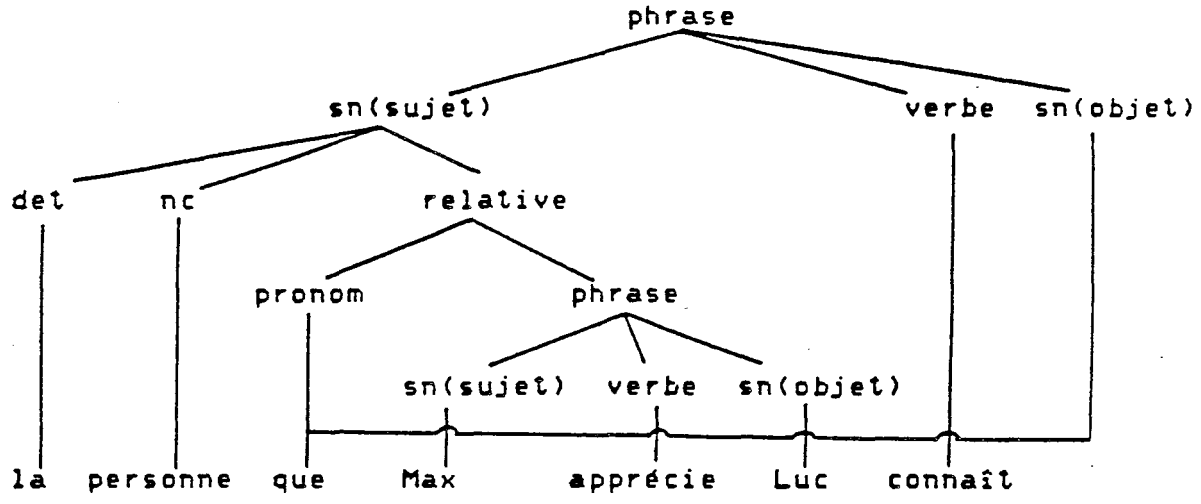
<Grammaire GX2>

sa , sb, sc, sxb, sxc sont les symboles non-terminaux de GX2.

Dans une règle de GX, les trous doivent être manipulés avec précaution. A partir de GX1, la chaîne:

la personne que Max apprécie Luc connaît

qui est incorrecte sera pourtant reconnue et recevra la structure suivante:



Pourquoi? Parce que dans la règle:

```
pronom  ...  sn(objet)  ->  'que'
```

le trou "..." désigne (ou "s'unifie avec") la suite de symboles située entre "pronom" et "sn(objet)". Dans ce cas, rien n'empêche que la suite soit celle située entre "pronom" et la deuxième occurrence de "sn(objet)".

Pour éviter cela, il faut restreindre la dépendance entre le pronom de la relative et l'objet de la phrase formant la relative. En d'autres termes, il faut que le trou "..." s'unifie avec la liste située entre "pronom" et la première occurrence de "sn(objet)".

Pour régler le problème d'une façon générale, Pereira a recours à deux symboles non-terminaux quelconques, "debut" et "fin" par exemple, marquant le début et la fin de la suite de symboles où la dépendance peut être réalisée. On introduit alors ces symboles dans la règle des relatives. Au lieu de la règle:

relative -> pronom phrase

on a la règle:

relative -> debut pronom phrase fin

Et on ajoute la règle générale :

debut ... fin -> <>

La chaîne:

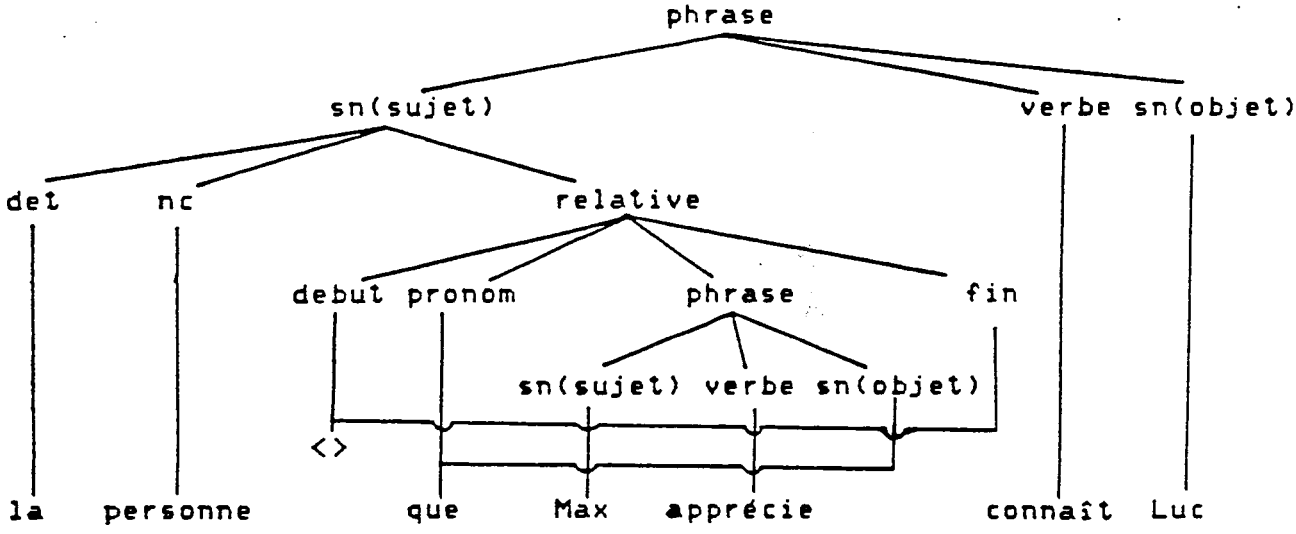
la personne que Max apprécie Luc connaît

ne sera ni reconnue ni produite.

Par contre la chaîne:

la personne que Max apprécie connaît Luc

recevra maintenant la structure :



Voici un autre exemple de règles de GX:

sn(x) ... prep sn(y) 'de' sn(x) -> sn(x) prep sn(y) dequel(x).

avec pour le symbole "dequel(x)", les règles:

dequel(x) -> 'duquel' {masculin(x) singulier(x)}

dequel(x) -> 'de' 'laquelle' {feminin(x) singulier(x)}

dequel(x) -> 'desquels' {masculin(x) pluriel(x)}

dequel(x) -> 'desquelles' {feminin(x) pluriel(x)}

Dans une phrase comme:

la maison sur les murs de laquelle Max peint est à vendre

les règles GX précédentes permettent d'analyser la chaîne:

la maison sur les murs de laquelle Max peint

à partir de la suite:

la maison	Max peint	sur	les murs	de	la maison
sn(x)	...	prep	sn(y)	de	sn(x)

Les trous apparaissant dans la partie gauche d'une règle de GX sont toujours replacés à la fin de la partie droite de la règle, et toujours dans le même ordre.

GRAMMAIRES DISCONTINUES

Les Grammaires Discontinues (GD) (ou "Gapping Grammars") [Dahl et Abramson 1984, Dahl 1984] sont une généralisation des GX. Dans une GD, les trous apparaissant dans la partie gauche de la règle peuvent être réordonnés dans la partie droite.

Les règles de GD sont de la forme:

```
N  S1  trou(t1) --- Sn  trou(tn)  Sn+1  ->  V
```

où:

- N est un symbole non-terminal,
- Si est un symbole terminal ou non-terminal,
- trou(ti) désigne une suite de symboles non précisés,
- la suite "S1 trou(t1) --- Sn trou(tn) Sn+1 " est éventuellement vide,
- V est une suite de symboles et de trous réordonnés ou non.

En fait, dans une expression comme:

```
Si  trou(ti)  Si+1
```

il est dit, d'une façon déclarative, que trou(ti) se réécrit dans la suite de symboles (éventuellement vide) située entre Si et Si+1; ti désignant (ou "s'unifiant avec") cette suite.

Voici par exemple une règle GD pour l'anglais, avec une inversion des trous:

```
sn(x)  trou(y)  prep  det  trou(z)  'of'  sn(x)  ->
sn(x)  prep  'whose'  trou(z)  trou(y)
```

Une telle règle permet, si on le souhaite, de décrire la chaîne:

```
the house on whose walls Max is painting
```

comme dérivée de:

the house	Max is painting	on	the	walls	of	the house
sn(x)	trou(y)	prep	det	trou(z)	of	sn(x)

GRAMMAIRES DE PUZZLE

Nous avons proposé [Sabatier 1983] un premier formalisme dérivé des GM afin d'exprimer les dépendances contextuelles au moyen de conditions spécifiques apparaissant dans la partie droite des règles. L'idée sous-jacente était de réserver l'usage des arguments des symboles à l'expression des traductions que l'on souhaite associer aux chaînes du langage.

Voici un exemple simple d'une telle grammaire :

```
phrase  ->  sn  verbe  sn
sn  ->  <>  { 'qui' # verbe }
sn  ->  <>  { 'que' sn verbe # }
sn  ->  'Max'
sn  ->  'Luc'
sn  ->  det  nc  relative
det  ->  'la'
det  ->  'chaque'
nc  -> 'personne'
verbe  ->  'apprécie'
verbe  ->  'connaît'
relative  ->  <>
relative  ->  'qui'  phrase
relative  ->  'que'  phrase
```

Dans les deux premières règles de réécriture de sn, le contexte dans lequel sn se réécrit vide (<>) est spécifié par les conditions :

```
{ 'qui' # verbe }

{ 'que' sn verbe # }
```

où le symbole # est mis pour le symbole sn se réécrivant.

Ces expressions sont à lire comme les conditions pouvant apparaître dans la partie droite des règles de GM. Ces conditions sont ici d'un type particulier: elles formulent des contraintes sur la nature des coupes dans l'arbre de dérivation syntaxique implicitement construit.

Cette façon de spécifier les contraintes en termes de coupes nous a conduit à développer un autre formalisme dans lequel, une grammaire est spécifiée par un ensemble d'arbres, les Grammaires de Puzzle (GZ) [Sabatier 1984], et dont nous rappelons maintenant les caractéristiques.

Définir un langage par un ensemble d'arbres (ou "forêt"), peut être un moyen plus simple, plus clair et parfois plus puissant

que d'avoir recours à des règles de réécriture.

A toute règle indépendante du contexte d'une grammaire classique (où les symboles sont des atomes):

A -> B1 B2 ... Bn

où:

- A est un symbole non-terminal
- B1 B2 ... Bn est une suite de symboles terminaux ou non-terminaux

on peut faire correspondre un arbre où:

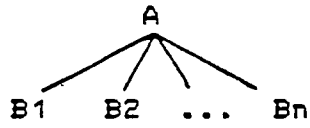
- A est la racine
- B1, B2, ... et Bn sont les feuilles.

Et inversement, à un tel arbre, on peut faire correspondre une règle indépendante du contexte.

On peut représenter l'arbre soit par l'expression parenthésée:

A(B1,B...,Bn)

soit par le schéma:



Si un langage est défini par une grammaire indépendante du contexte formée de n règles, ce même langage, avec les mêmes généralisations, peut être défini par une forêt de n arbres. Et inversement.

Dans ce cas, la forêt et la grammaire ne sont que des variantes notationnelles d'un même langage.

Cependant la définition d'un langage au moyen d'arbres peut dépasser le simple jeu des variations notationnelles. Le recours à des arbres plutôt qu'à des règles de réécriture pour définir les chaînes d'un langage et spécifier des généralisations peut être des plus puissants.

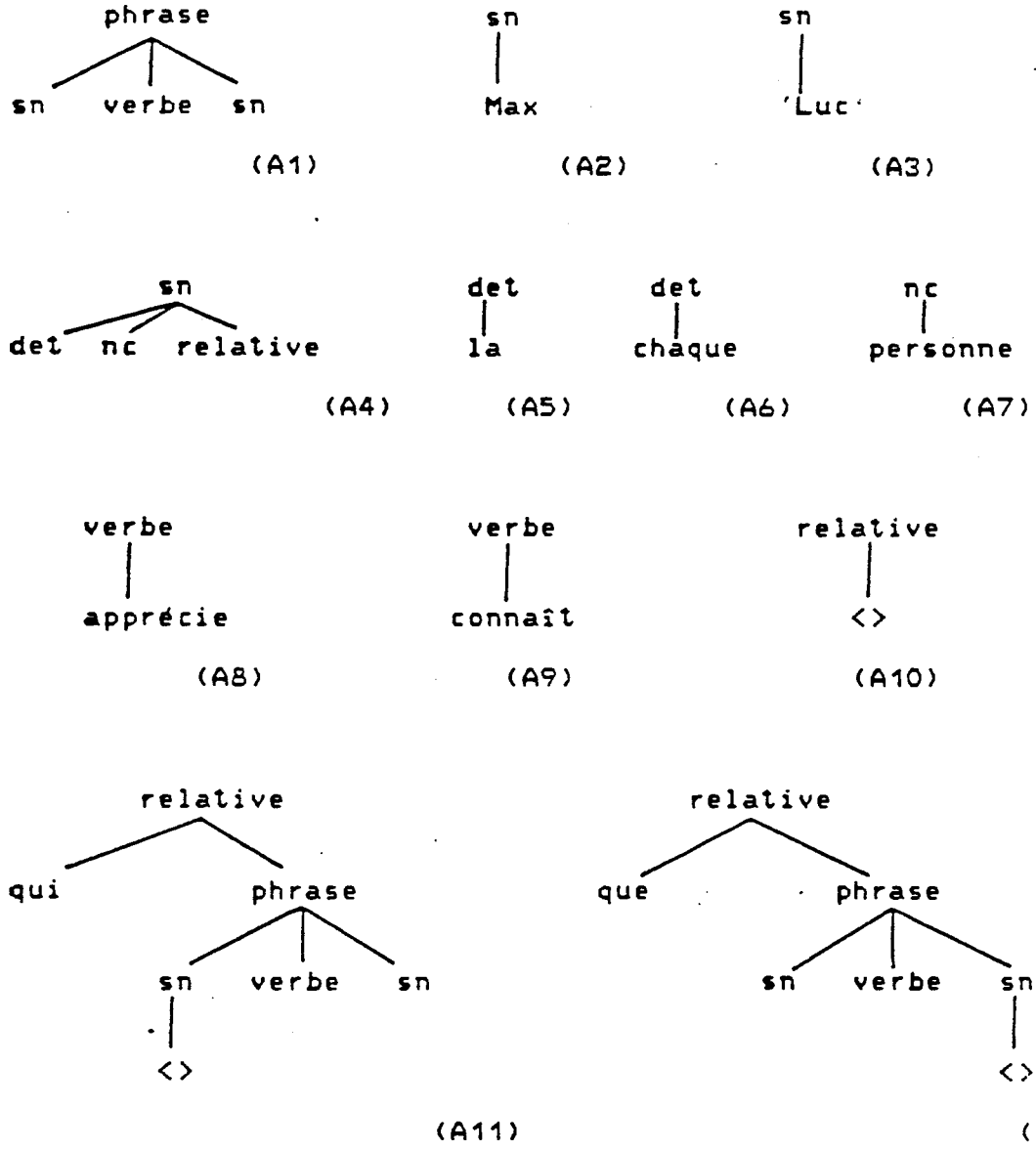
A titre d'exemple, reprenons les règles de la grammaire

dépendante du contexte G4:

```
phrase  ->  sn  verbe  sn                          (R1)
sn  ->  'Max'                                      (R2)
sn  ->  'Luc'                                      (R3)
sn  ->  det  nc   relative                         (R4)
det  ->  'la'                                      (R5)
det  ->  'chaque'                                  (R6)
nc  ->  'personne'                                 (R7)
verbe  ->  'apprécie'                              (R8)
verbe  ->  'connaît'                               (R9)
relative  ->  <>                                   (R10)
relative  ->  'qui'  phrase                        (R11)
relative  ->  'que'  phrase                        (R12)
'qui'  sn  verbe  sn  ->  'qui'  xx  verbe  sn     (R13)
'que'  sn  verbe  sn  ->  'que'  sn  verbe  xx     (R14)
xx  ->  <>                                         (R15)
```

<Grammaire G4>

Les mêmes chaînes avec les mêmes généralisations peuvent être définies par la forêt formée des 12 arbres schématisés suivants:



<Forêt F1>

Le langage défini en 15 règles par G4 est défini en 12 arbres par F1. Les arbres A1 à A10 sont de simples variantes notationnelles des règles R1 à R10. Par contre, l'arbre A11 correspond à la seule combinaison possible des règles R11, R13 et R15; et l'arbre A12, à la seule combinaison possible des règles R12, R14 et R15. (On en a profité pour éliminer le symbole non-terminal ad-hoc xx introduit dans les règles R13 et R14 pour conserver le statut de règles dépendantes du contexte).

A11 et A12 expriment clairement la définition des relatives en terme de phrase et la formulation de la dépendance entre la nature du pronom ("qui" vs "que") et du sn (sujet vs objet) qui

se réécrit dans la chaîne vide <>.

Les arbres A11 et A12, à la différence des autres arbres qui composent F1 sont des arbres complexes.

Un arbre est dit SIMPLE (ou "de profondeur 1") si et seulement si toutes ses feuilles sont directement dominées par la racine de l'arbre. Dans le cas contraire, l'arbre est dit COMPLEXE.

A un arbre complexe correspondent plusieurs règles de réécriture, et au moins une de ces règles est indépendante du contexte.

Autrement dit, un arbre complexe formule une généralisation dans la définition d'une chaîne. Dans une grammaire classique, cette généralisation sera formulée au moyen d'une ou plusieurs règles indépendantes du contexte et d'une ou plusieurs règles dépendantes du contexte.

Dans F1, les symboles formant les racines, les feuilles et les noeuds intermédiaires sont des symboles simples (comme dans les grammaires classiques).

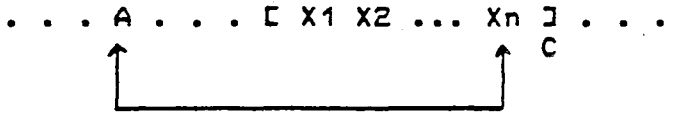
Il est nécessaire que les racines et les feuilles des arbres puissent être des symboles complexes pour pouvoir exprimer toute relation de dépendance entre:

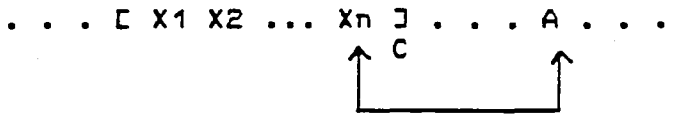
un symbole Xn défini comme le dernier élément d'une sous-chaîne de type C dont la longueur est variable

et

un symbole A situé hors de cette sous-chaîne de type C.

Selon que A est situé à gauche ou à droite de la sous-chaîne C, on a les deux schémas suivants:



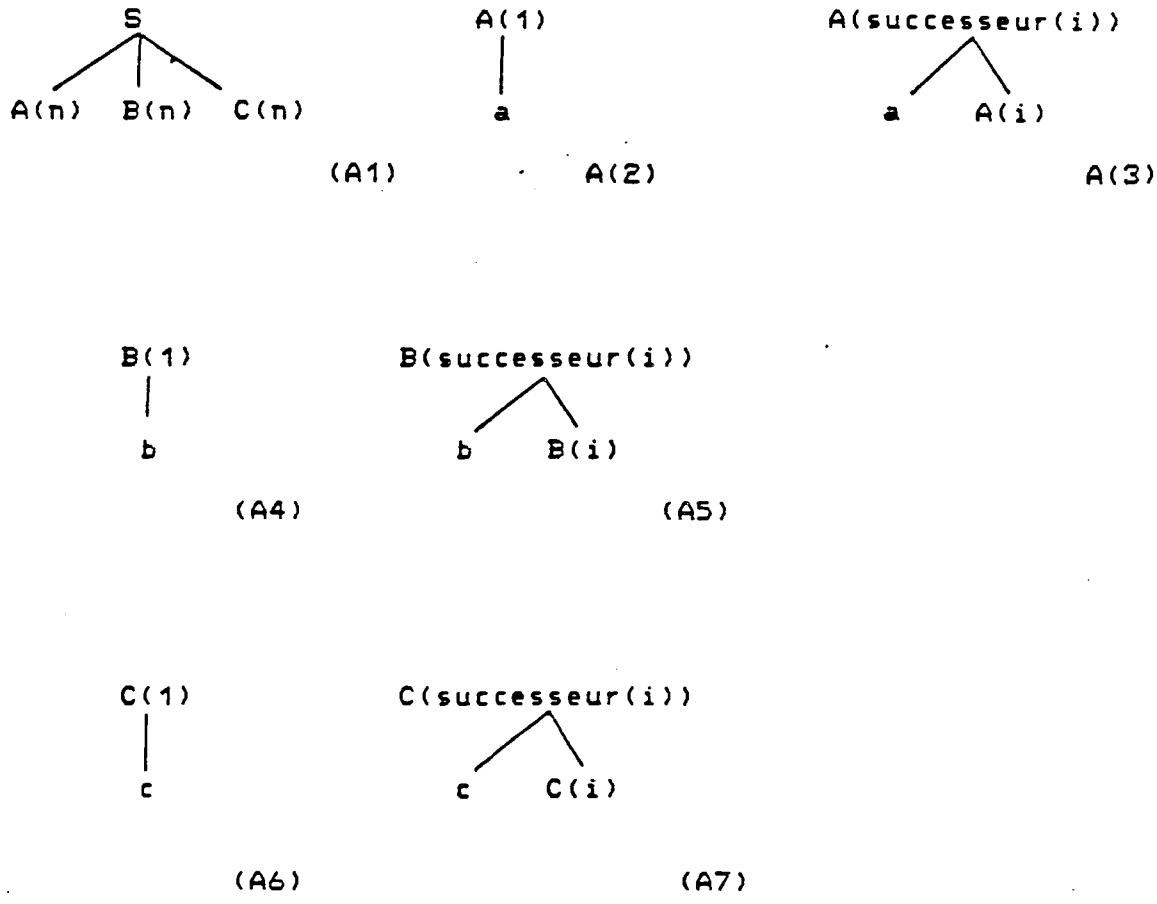


On retrouve en particulier ces schémas imbriqués les uns dans les autres dans les langages dits dépendants du contexte.

En ayant recours à des symboles complexes, le langage formel dépendant du contexte

$$a^n\ b^n\ c^n \qquad (n > 0)$$

peut être défini par une forêt, comme F2, par exemple :



<Forêt F2>

Ici chaque arbre de F2 n'est qu'une variante notationnelle d'une règle correspondante de la grammaire de métamorphoses GM2 présentée plus haut. Et inversement.

Appelons DCG Simple, toute grammaire de métamorphose dont chaque règle:

- n'a qu'un symbole dans sa partie gauche,
- et ne contient pas de conditions dans sa partie droite.

Tout langage dépendant du contexte défini par une DCG simple de n règles peut être défini par une forêt de n arbres où chaque arbre n'est qu'une variante notationnelle d'une règle de la grammaire.

Pour définir un langage dépendant du contexte par une forêt, le recours à des symboles complexes est donc nécessaire.

On peut illustrer les schémas de dépendance précédents dans le

cas d'un langage indépendant du contexte. Soit par exemple le langage formel formé par l'ensemble infini des chaînes:

$$a\ z^{n}\ b \qquad (n > 0)$$

$$b\ z^{n}\ a$$

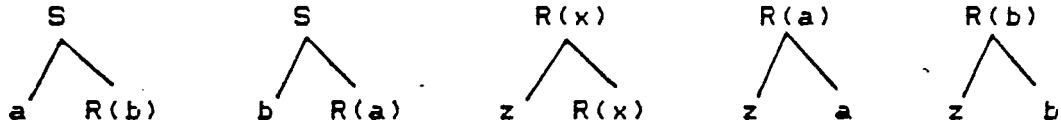
Ce langage peut être défini par la forêt F3 suivante:

<Forêt F3>

Mais si on décide que le dernier élément (b ou a) doit être défini comme dernier élément de la sous-chaîne de longueur variable R, à savoir:

$$z^{n}\ b \quad \text{ou} \quad z^{n}\ a$$

le recours au symbole R(-) est nécessaire. La forêt F4 définissant le langage en ces termes sera alors:

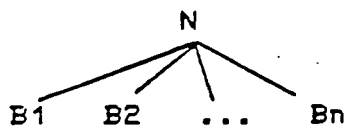


<Forêt F4>

Dans le cadre d'un langage indépendant du contexte, l'utilisation de symboles complexes permet aussi dans certains cas de réduire le nombre d'arbres d'une forêt tout en exprimant des généralisations sur la définition des chaînes.

Appelons ARBRE-CLE, tout arbre K de la forme:

(K)



où:

- N est un symbole simple ou complexe, N est appelé la RACINE de K,

- Bi est soit:

  - un symbole simple ou complexe, Bi est appelé une FEUILLE de K,

  - la chaîne vide (notée "<>"),

  - un arbre-clé, la racine de Bi est appelé un NOEUD de K.

Si chaque Bi est une feuille de K, K est un arbre-clé SIMPLE.

Si au moins un des Bi est un arbre-clé, alors K est un arbre-clé COMPLEXE.

A tout arbre-clé K est associé un ensemble R, éventuellement vide, de restrictions. Nous noterons cet ensemble entre accolades. {} dénote un ensemble vide de restrictions.

Les restrictions ne sont rien d'autres que les fameuses conditions pouvant apparaître dans la partie droite des règles des grammaires de métamorphoses. Ici, les restrictions formulent des conditions sur la nature des variables apparaissant dans l'arbre-clé concerné.

On aura recours à l'usage des restrictions pour formuler plus clairement et plus rapidement des contraintes générales, et qu'il serait fastidieux d'intégrer dans des arbres-clés en compliquant ceux existants ou en en créant d'autres.

Une grammaire de puzzle GZ est un ensemble de couples:

GZ = { (K1,R1), (K2,R2), ... , (Kn,Rn) } (n>0)

où :

- Ki est un arbre-clé appelé arbre-clé de base

- Ri est un ensemble, éventuellement vide, de restrictions associé à Ki.

- Chaque couple (Ki,Ri) est appelé une PIECE de GZ.

Deux arbres-clés K1 et K2 sont assemblables si, à la fois:

- K1 et K2 sont assemblables par ancrage.

- les ensembles de restrictions R1 et R2 respectivement associés à K1 et K2 sont vérifiés.

Deux arbres-clés K1 et K2 sont assemblables par ancrage si à la fois:

- S1 est une feuille de K1

- S2 est la racine de K2

- S1 et S2 sont unifiables.

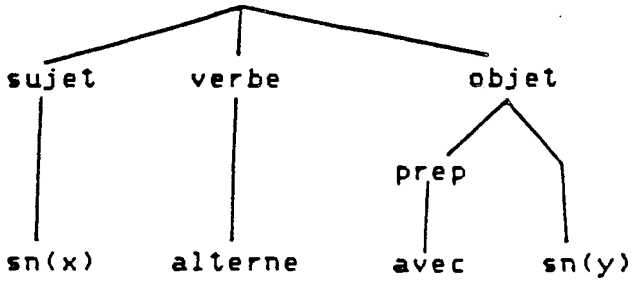
Après unification de S1 et de S2 et de l'affectation des variables de K1 et de K2, le résultat de l'assemblage est un nouvel arbre-clé K3 constitué de l'arbre-clé K1 dont la feuille S2 a été remplacée par K2.

On dira alors qu'une chaîne C fait partie du langage défini par une grammaire de puzzle GZ si il existe un arbre-clé K dont la suite des feuilles est C. K est un arbre-clé de base. Ou bien K est obtenu après assemblages successifs d'arbres-clés.

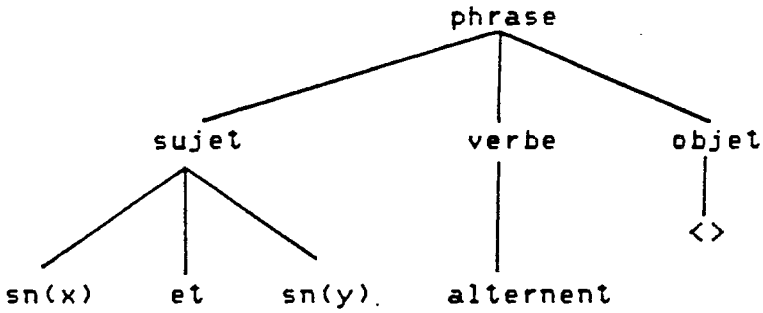
C'est parce que l'assemblage des arbres-clés rappelle celui des pièces d'un puzzle, que nous avons appelé ce type de grammaire "grammaire de puzzle".

Voici un exemple simple de grammaire de puzzle:

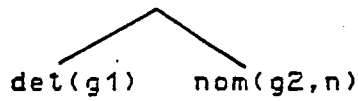
(P1)



(P2)



(P3)



(P4) det(mas) {}

le

(P5) det(fem) {}

la

(P6) nom(mas,jour) {}

jour

(P7) nom(fem,nuit) {}

nuit

<Grammaire GZ1>

GZ1 est formée de 7 pièces, P1 à P7.

Dans P1, la restriction different(-,-) exclura les chaînes:

le jour alterne avec le jour
la nuit alterne avec la nuit

Dans P2, cette même restriction exclura les chaînes:

le jour et le jour alternent
la nuit et la nuit alternent

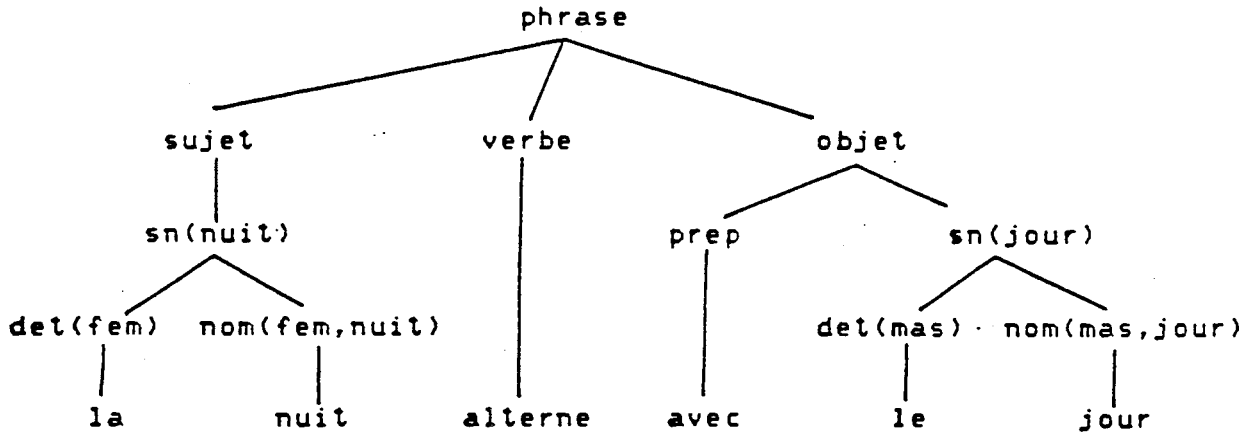
Dans P3, la restriction genre(-,-) interdira les chaînes (ou sous-chaînes):

le nuit
la jour

Par exemple, la chaîne:

la nuit alterne avec le jour

est bien définie dans GZ1. Et GZ1, par assemblages successifs d'arbres-clés, en donnera la structure suivante, à savoir l'arbre-clé:



Rappelons que le principal intérêt d'avoir recours dans une GZ à des arbres-clés de base complexes est de permettre l'expression directe de généralisations qui faciliteront la production de traduction réalisée en parallèle ou ultérieurement.

Comme dans les grammaires de métamorphoses, on peut avoir recours à des symboles complexes pour spécifier directement les traductions que l'on souhaite associer aux chaînes du langage. Dans ce cas, l'emploi d'arbres-clés de base complexes est dépourvu d'intérêt. Des arbres-clés simples suffisent.

FORMALISMES ORIENTES VERS L'EXPRESSION DES TRADUCTIONS

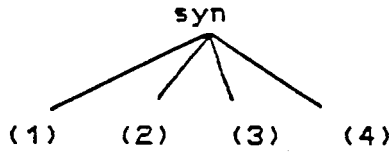
Le souci de pouvoir formuler clairement des traductions plus complexes que les arbres de dérivation purement syntaxique a conduit au développement de plusieurs formalismes dans le cadre du principe des GM. Nous présentons deux de ces formalismes:

- les "Modifier Structure Grammars"
- les "Definite Clause Translation Grammars"

"MODIFIER STRUCTURE GRAMMARS"

Parmi le type de traductions recherchées, celui concernant l'interprétation logique des phrases d'un langage naturel a fait l'objet de nombreuses applications. (l'exemple donné précédemment avec GM1 est une illustration très simple de ce type de traductions). Plusieurs tentatives de normalisation dans le codage de telles traductions ont été proposées.

En particulier, il faut citer le formalisme des "Modifier Structure Grammars " (MSG) qui a son origine dans [McCord 1981, 1982], et qui a été généralisé dans [Dahl et McCord 1983]. L'intérêt des MSG est de nous fournir un formalisme permettant d'uniformiser les données de l'analyse syntaxique d'une phrase en vue de son interprétation sémantique (logique). McCord propose d'associer à chaque chaîne un item syntaxique de la forme:



où:

(1) est une liste d'informations syntaxiques et morphologiques concernant la chaîne décrite par l'item: la catégorie (phrase, sn, verbe, etc.), le genre (masculin, pluriel), le nombre (singulier, pluriel), etc.

(2) est un terme précisant la façon dont l'item se comporte comme modificateur d'autres items,

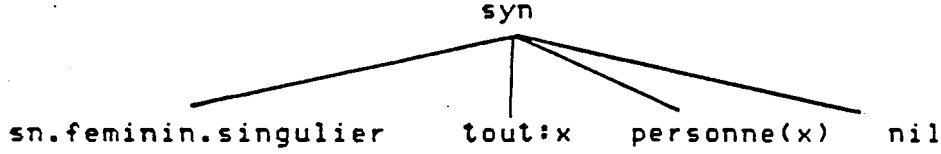
(3) est l'expression prédicative associée à l'item,

(4) est la liste de tous les modificateurs de l'item concerné tels qu'ils apparaissent dans la chaîne ("ordre de surface"); chaque modificateur étant lui-même un item.

Pour la chaîne:

chaque personne

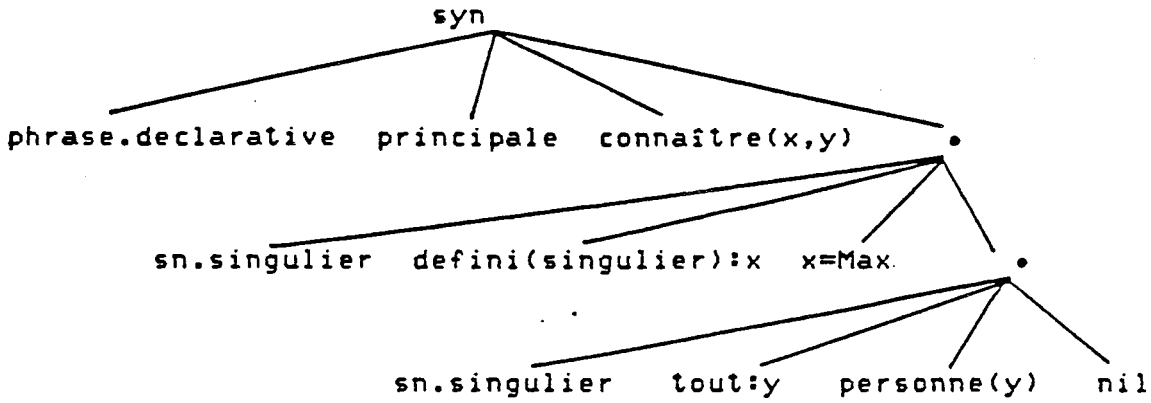
on aura par exemple l'item suivant:



Pour la chaîne:

Max connaît chaque personne

on aura par exemple l'item:



Par la suite, un ensemble de règles d'interprétation sémantique opèrera sur ces items pour déterminer en particulier la portée des modificateurs.

A partir de l'item syntaxique produit pour la chaîne:

Max connaît chaque personne

les règles sémantiques donneront comme interprétation quelque chose comme:

declarative (personne(y) => (x=Max) et (connaître(x,y))

Le formalisme des MSG vise à fournir à l'utilisateur une normalisation dans la représentation des informations linguistiques. Et cela dans un but bien précis: faciliter la mise en oeuvre des mécanismes d'interprétation sémantique des phrases d'un langage naturel.

"DEFINITE CLAUSE TRANSLATION GRAMMARS"

Les "Definite Clause Translation Grammars" (DCTG) [Abramson 1984] sont un formalisme général orienté vers l'expression modulaire des traductions dans le cadre des Definite Clause Grammars. Nous en rappelons ici les principales caractéristiques.

Les règles de DCTG sont de la forme:

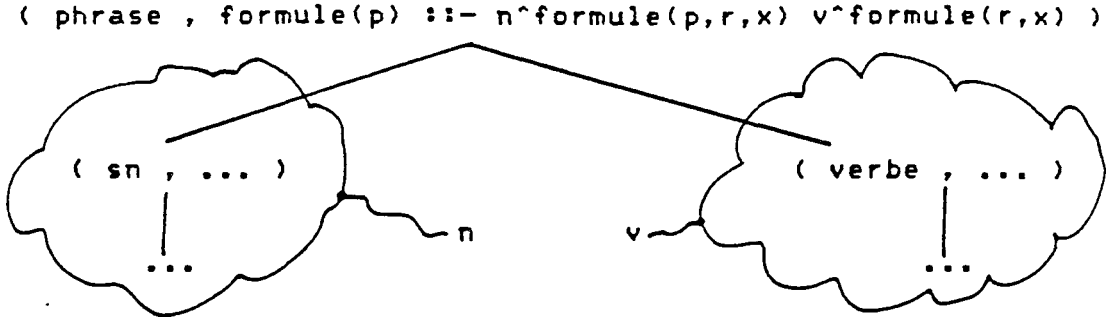
Partie-Réécriture <:> Partie-Attributs

Voici des exemples:

```
(R1)   phrase  ::=  sn^n   verbe^v   {accord-nombre(n,v)}
       <:>
       formule(p) ::=  n^formule(p,r,x)  v^formule(r,x).

(R2)   verbe  ::=  'intéressant'
       <:>
       nombre(singulier).
       formule(intéressant(x),x).

(R3)   sn  ::=  np^n
       <:>
       nombre(singulier).
       formule(p,r,x) ::-  n^formule(x).

(R4)   np  ::=  'Max'
       <:>
       formule(Max).
```

"Partie-Réécriture" est constituée par une règle de type GM. Les deux parties de la règle de réécriture sont séparées par le symbole ":::=". A gauche de ":::=", on a un symbole non-terminal suivi d'une suite (éventuellement vide) de symboles terminaux. A droite de ":::=", on a une suite (éventuellement vide) de symboles non-terminaux et terminaux, et de conditions (notées entre accolades). Dans cette partie droite, une seule variable (précédée de l'opérateur "^" est associée à chaque non-terminal. Que désigne cette variable, et avec quoi s'unifie-t-elle?

Dans les DCTG, le noeud de l'arbre de dérivation produit par la réécriture d'un symbole S est constitué de ce symbole S et des éléments situés à droite de "<:>" (c.-à-d. le contenu de "Partie-Attributs") dans la règle utilisée pour réécrire S. Par exemple, après la réécriture de "phrase" en utilisant R1, l'arbre de dérivation ressemblera à quelque chose comme:



Ainsi, la variable précédée de l'opérateur "^" qui est associée à un symbole non-terminal S désigne et s'unifie avec l'arbre de dérivation dont le noeud est constitué par S et sa "Partie-Attributs".

"Partie-Attributs" contient l'ensemble des attributs associés au symbole concerné. Les attributs peuvent contenir:

(1) des données nécessaires à l'évaluation des conditions. Comme par exemple l'attribut "nombre(singulier)" dans R1 et R3,

(2) des données sur la nature des traductions. Comme par exemple les différentes expressions "formule(...)" dans R1, R2, R3 et R4.

Les attributs constituent une banque de données locale au symbole concerné. Cette banque de données est définie par un ensemble de Clauses de Horn.

Dans:

```
(R1)   phrase  ::=  sn^n   verbe^v   {accord-nombre(n,v)}
       <:>
       formule(p)  ::-  n^formule(p,r,x)   v^formule(r,x).
```

"formule(p)" est l'attribut associé au symbole "phrase". Il est défini par la conjonction des deux expressions:

```
n^formule(p,r,x)
v^formule(r,x)
```

L'ensemble se lit:

Pour calculer "formule(p)",

(1) traverser l'arbre de dérivation unifié à "n", et calculer les attributs qui s'unifient avec "formule(p,r,x)", et

(2) traverser l'arbre de dérivation unifié à "v" et calculer les attributs qui s'unifient avec "formule(r,x)".

Voici une grammaire DCTG complète:

```
phrase  ::=  sn^n   verbe^v
<:>
formule(t)  :-   n^formule(t,x,r)   v^formule(r,x).

sn  ::=   np^n
<:>
formule(t,x,r)  :-   n^formule(x).

sn  ::=   det^d   nc^n
<:>
formule(t,x,r)  :-   d^formule(t,x,v,r)   n^formule(v,x).

np  ::=   'Max'
<:>
formule(Max).

det  ::=   'un'
<:>
formule(existe(x,v,r),x,v,r).

det  ::=  'chaque'
<:>
formule(quelque-soit(x,v,r),x,v,r).

nc  ::=  'exemple'
<:>
formule(exemple(x),x).

nc  ::=  'exercice'
<:>
formule(exercice(x),x).

verbe  ::=  'est'  'stupide'
<:>
formule(stupide(x),x).

verbe  ::=  'est'  'intéressant'
<:>
formule(intéressant(x),x).
```

<Grammaire DCTG1>

A titre de comparaison, nous reproduisons ici la grammaire de métamorphoses GM1. GM1 est équivalente à DCTG1 en ce qui concerne les chaînes définies et les traductions produites :

```
phrase(t)  ->  sn(t,x,r)  verbe(r,x)

sn(t,x,r)  ->  np(x)

sn(t,x,r)  ->  det(t,x,v,r)  nc(v,x)
```

```
np(Max)  ->  'Max'

det(existe(x,v,r),x,v,r)  ->  'un'

det(quelque-soit(x,v,r),x,v,r)  ->  'chaque'

nc(exemple(x),x)  ->  'exemple'

nc(exercice(x),x)  ->  'exercice'

verbe(stupide(x),x)  ->  'est'  'stupide'

verbe(intéressant(x),x)    ->  'est'  'intéressant'
```

<Grammaire GM1>

En introduisant les variables associées aux arbres de dérivation implicites , DCTG1 contient davantage de variables que GM1. Mais les traductions associées aux chaînes semblent plus facilement modifiables dans DCTG1 que dans GM1.

STRATEGIES D'ANALYSE ET DE SYNTHESE : PROGRAMMATION EN PROLOG

Une stratégie de réécriture définit la démarche à suivre pour reconnaître ou produire les chaînes d'un langage à partir d'une grammaire donnée.

Les stratégies de réécriture ont été largement étudiées dans le cadre des théories et des méthodes de compilation des langages de programmation [Aho et Ullman 1972, 1973].

La question que l'on peut poser alors est de savoir dans quelle mesure ces stratégies se prêtent à l'analyse et à la synthèse des langages naturels.

Si on compare les langues naturelles avec des langages formels (comme les langages de programmation, par exemple), on constate, par exemple, que les constructions anaphoriques mettant en jeu des structures coordonnées, elliptiques et pronominales sont caractéristisques des langages naturels.

La méthode qui consiste à ne pas définir au moyen de règles spécifiques certaines chaînes d'un langage, mais à rendre compte de ces chaînes au moyen de procédures simplifiant l'opération de réécriture en la complétant, a donné lieu à plusieurs applications dans le cadre du principe des GM. Par exemple, l'analyse des constructions contenant des conjonction de coordination est confiée à un interpréteur dans [Dahl et McCord 1983], [Sedogbo 1983].

Prolog et le formalisme des Grammaires de Métamorphoses sont indépendants. Cependant, le succès des GM (et de leurs dérivés) est largement dû au fait que leur programmation en Prolog est des plus appropriées. En particulier, pour les raisons suivantes:

(1) Prolog manipule des termes,

(2) le principe d'unification sous-jacent au formalisme des GM est prédéfini en Prolog,

(3) les conditions apparaissant dans la partie droite des règles peuvent être définies simplement et clairement par un ensemble de clauses Prolog.

La programmation en Prolog est quasi immédiate si on a choisi pour la grammaire une stratégie de réécriture descendante, gauche-droite, séquentielle, en-profondeur, non-déterministe: c'est à dire la stratégie de Prolog lui-même. Chaque règle de grammaire peut être alors directement traduite dans une règle Prolog. On peut l'écrire soi-même, ou s'en remettre à un traduc-

teur.

On trouvera dans l'annexe A, un tel traducteur. Il s'agit du programme traduisant les règles linguistiques (lexique et grammaire présentés dans le chapitre VI) du système INTERFACILE dans un programme Prolog d'analyse et de synthèse.

La programmation est un peu plus complexe si on a choisi des stratégies d'analyse et de synthèse autres que celles inhérentes à Prolog. Mais l'exercice est toujours possible: Prolog est d'abord un langage de programmation. Pour la programmation en Prolog de stratégies ascendantes, voir en particulier [Uehara et Toyoda 1981], [Stabler 1983], [Matsumoto et al. 1983], [Uehara et al. 1984]. Pour des stratégies parallèles: [Hirakawa 1983].

APPLICATIONS

Programmées en Prolog, les GM, DCG, GX, MSG et DCTG ont donné lieu à de nombreuses applications, en particulier dans le domaine d'interfaces en langage naturel pour banques de données, systèmes experts et systèmes d'histoire. Nous en citons quelques unes ici. Entre parenthèses est noté le type de grammaire utilisé.

- Système d'histoire, interface en français (GM), [Colmerauer et al. 1973].

- Banque de données sur du matériel informatique, interface en français (GM), [Dahl et Sambuç 1976].

- Banque de données administratives, interface en espagnol (GM), [Dahl 1977].

- TUGA, banque de données sur des références bibliographiques, interface en portugais (DCG), [Coelho 1979].

- DIALOGUES, système d'histoire, interface en français (GM), [Sabatier 1980].

- MICROSIAL, banque de données sur une administration militaire, interface en français (GM), [Giraud et al. 1980], [Pique et Sabatier 1982].

- CHAT-80, banque de données sur la géographie mondiale, interface en anglais (GX), [Pereira et Warren 1981].

- UNI, banque de données sur l'organisation d'enseignements universitaires, interface en anglais (MSG), [McCord 1982].

- ORBI, système expert sur l'aménagement du territoire portugais, interface en portugais (DCG), [Oliveira et al. 1982].

- INTERFRANCE, banque de données sur la France géographique et administrative, interface en français (DCG), [Duchier et Sabatier 1982].

- ORBIS, banque de données sur les planètes, interface en français et en anglais (DCG), [Colmerauer et Kittredge 1982].

- KBO1, système-expert sur l'utilisation de produits de jardinage, interface en anglais (DCG), [Walker et Porto 1983].

- INTERIX, système d'aide à l'utilisation d'Unix, interface en français (DCG), [Guez et Sabbagh 1984].

- INTERFACILE, système didactique pour l'apprentissage d'un langage de commande d'un système d'exploitation, interface en français (DCG), [Mathieu et Sabatier 1985].

Nous donnons dans le chapitre VI les règles de grammaire du

système INTERFACILE et son lexique. Les traducteurs de ces formalismes en règles Prolog sont décrits dans [Sabatier 1985].

Parmi les grammaires à large couverture syntaxique, il faut citer la programmation en Prolog par Célestin Sedogbo [1984] d'une version étendue de la grammaire en chaîne du français [Salkoff 1973, 1979, 1982].

Dans le cadre de la réalisation d'un système de reconnaissance automatique de la parole continue, il faut citer l'utilisation du formalisme des Grammaires de Métamorphoses [Meloni 1982].

CONCLUSION

Les Grammaires de Métamorphose sont comparables à des grammaires de type 0. Les formalismes qui en sont dérivés ne peuvent donc pas étendre leur capacité générative. Les "extensions" envisagées sont justifiées pour des raisons de clarté, de lisibilité et de facilité de modification.

La puissance des GM réside dans une généralisation du principe d'unification et du rôle joué par les variables. A un certain niveau, les formalismes dérivés peuvent conduire à une réduction du nombre des variables grâce à un usage particulier de l'unification.

La mise en oeuvre de mécanismes comme la construction implicite des arbres de dérivation complexes sous-jacents à ces formalismes est le prix technique à payer pour les facilités d'expression offertes.

Les alternatives aux Grammaires de Métamorphose sont-elles alors réellement nécessaires?

Si l'abondance des variables ne gêne pas l'utilisateur d'un tel formalisme, la réponse à cette question sera négative. Si la réduction des variables conduit à une expression plus claire de la grammaire à écrire, la réponse sera positive. Et l'on voudra pouvoir accéder à des formalismes "au-dessus" des GM. La question que l'on ne manquera pas de se poser sera alors :

> Les formalismes dérivés des GM développés jusqu'ici sont-ils adaptés à la formulation et à la résolution des phénomènes dont on veut rendre compte ?

CHAPITRE III

SEMANTIQUE FORMELLE ET INTERPRETATION SEMANTIQUE

Introduction

"Sémantique computationnelle" et "sémantique linguistique"

Sémantique formelle : évaluation et déduction

Représentation sémantique d'un sous-ensemble du français

De la représentation sémantique au système logique

Application aux interfaces pour bases de données

Conclusion

INTRODUCTION

On peut distinguer trois buts essentiels poursuivis par les systèmes reliant des expressions d'un langage naturel et des bases de connaissances (bases de données, systèmes-experts, systèmes documentaires, didactiques, etc.). Ces trois buts sont :

- L'interrogation en langage naturel de bases de connaissances.

- La création de bases de connaissances à partir de formulations en langage naturel.

- La production de textes en langage naturel synthétisés à partir de bases de connaissances.

La réalisation d'un système illustrant l'un de ces trois buts conduit, à un moment donné, à concevoir un composant ayant pour tâche d'exprimer le sens (ou l'"interprétation sémantique") des phrases analysées ou produites, et à définir ainsi un langage formel de "représentation sémantique" pour le langage naturel considéré.

Le langage de représentation choisi pour exprimer le sens des phrases peut être directement le langage dans lequel sont formulées les connaissances de la base qui est interrogée (ou créée, ou à partir de laquelle des textes sont synthétisés). Par exemple, dans le cadre de l'interrogation (ou de la mise à jour) d'une base de données relationnelle, le langage de représentation sémantique peut être le langage formel de requête SQL.

Si elle peut présenter certains avantages pratiques, cette approche présente deux inconvénients majeurs : 1) elle conduit à concevoir des systèmes difficilement portables vers des domaines d'application autres que ceux pour lequel le système a été conçu; et, 2) cette approche est de peu d'intérêt scientifique pour la compréhension de la sémantique du langage naturel.

Dans ce chapitre, nous situons les différentes approches "sémantiques" du traitement du langage naturel : "sémantique computationnelle", "sémantique linguistique" et "sémantique formelle". Nous présentons et justifions l'approche que nous avons retenue et suivie au cours de nos travaux : celle de la sémantique formelle. Nous rappelons les principes essentiels mis en oeuvre dans les systèmes réalisés, en particulier pour la compréhension des phénomènes de quantification et de présupposition, cela à partir des travaux sur le français d'Alain Colmerauer, de Robert Pasero et de Jean-François Pique .

Certains passages de ce chapitre sont repris de l'article "Formal Semantics and Knowledge Representation" écrit en collaboration

avec Franz Guenthner [Guenthner et Sabatier 1985].

"SEMANTIQUE COMPUTATIONNELLE" ET "SEMANTIQUE LINGUISTIQUE"

Sous le terme de "sémantique computationnelle" [Charniak et Wilks 1976], on peut regrouper un certain nombre de formalismes comme les "frames" [Minsky 1975], les dépendances conceptuelles, les "scripts" [Schank et Abelson 1977] ou les réseaux sémantiques [Simmons 1973] qui ont été proposés pour rendre compte de certains aspects sémantiques du langage naturel. On constate cependant que la plupart des approches en "sémantique computationnelle" sont essentiellement représentationnelles. D'une façon générale, il est difficile de savoir ce qu'est une expression bien formée dans ces langages. Sans définition rigoureuse, il est difficile d'évaluer les intuitions sémantiques sous-jacentes aux notations, et de savoir quelles représentations sont systématiques et lesquelles sont purement "ad hoc".

Dans le cadre de théories linguistiques, l'interprétation sémantique des phrases a fait l'objet de plusieurs développements, et a donné lieu à plusieurs propositions (en particulier, [Katz et Postal 1964], [Greimas 1966], [Fillmore 1968]), et que l'on pourrait regrouper sous le terme de "sémantique linguistique". David Lewis [Lewis 1972] formule à sa manière la philosophie sous-jacente à cette approche :

> Mes propositions en ce qui concerne la nature des significations ne sont pas conformes à ce que pensent les linguistes qui conçoivent l'interprétation sémantique comme consistant à assigner, aux phrases et à leurs composants, des traits ("markers") sémantiques ou quelque chose de similaire (Katz et Postal, 1964, en l'occurrence). Les traits ("markers") sémantiques sont des symboles, des éléments du vocabulaire d'un langage artificiel que nous pouvons appeler le "marquerais sémantique". Une interprétation sémantique réalisée en utilisant ces symboles n'est rien d'autre qu'un algorithme de traduction d'un langage objet dans le langage intermédiaire marquerais. Mais nous pouvons connaître la traduction en marquerais d'une phrase anglaise sans savoir la première chose concernant le sens de la phrase en anglais : à savoir, les conditions sous lesquelles la phrase serait vraie. Un sémantique sans traitement des conditions de vérité n'est pas une sémantique.

Nous partageons le point de vue de Lewis. L'approche qu'il met en avant est celle qui s'inscrit dans la tradition logicienne de la théorie des modèles (ou de la "sémantique formelle") héritée des travaux de Frege et Tarski.

SEMANTIQUE FORMELLE : EVALUATION ET DEDUCTION

Avant de poursuivre dans la direction indiquée par Lewis, il faut souligner que deux aspects restent fondamentaux dans la spécification du langage de représentation sémantique recherché.

On s'intéresse d'une part à savoir comment des expressions d'un langage de représentation particulier expriment des connaissances sur un monde (ou "un domaine", dans la terminologie des bases de données); c'est-à-dire à quelles conditions ces expressions caractérisent des états du "monde" définis par la base de connaissances. On peut appeler cet aspect "évaluation".

On s'intéresse aussi aux façons dont les expressions du langage de représentation sont reliées entre elles, et en particulier aux inférences réalisables au sein de l'ensemble des expressions. En accord avec la pratique courante, on appellera cet aspect "déduction".

Lorsque nous considérons l'évaluation, nous sommes intéressé d'abord par les informations que les expressions véhiculent sur le monde, et principalement dans le cas où les expressions sont considérées comme vraies. Lorsque nous considérons la déduction, nous nous intéressons d'abord aux relations entre les expressions, et qui peuvent être déterminées par un ensemble fini de manipulations indépendamment de toute référence sur la nature du monde décrite par la base de connaissances.

Les deux aspects essentiels que sont l'évaluation et la déduction font de la sémantique formelle un langage intéressant pour représenter 1) les informations contenues dans les bases de connaissances, et 2) la sémantique des expressions du langage naturel utilisées pour interroger (ou créer) ces bases, (ou synthétiser des phrases à partir de ces bases).

Dans toute représentation de type logique, ces deux aspects essentiels sont supportés par:

i) une syntaxe rigoureusement définie,
ii) une sémantique rigoureusement définie,
iii) une théorie déductive rigoureusement définie.

Et pour certains sous-ensembles d'entre-elles,

iv) il existe des applications informatiques.

Nous décrivons maintenant chacun de ces points en axant notre description sur un langage logique particulier : la logique des prédicats.

A la différence des formalismes développés en "sémantique computationnelle", les langages de type logique sont rigoureusement définis. Même si la logique des prédicats connaît différentes variantes syntaxiques, chacune d'elles est rigoureusement définie, et il est facile de passer de l'une à l'autre. Tout élément notationnel est sémantiquement pertinent.

La syntaxe de la logique des prédicats est une indication relativement claire des intuitions sémantiques sous-jacentes aux conditions de vérité (Théorie des modèles). Le parti pris sémantique de toute représentation logique est celui d'une vue relationnelle du monde. Les prédicats sont sémantiquement interprétés comme dénotant des relations entre des individus. La quantification, la prédication et la référence constituent les ingrédients essentiels de toute sémantique logique. Ils permettent de puissantes variations: interprétations extensionnelles vs. interprétations intensionnelles, divers types d'interprétations partielles ("logique libre", "sémantique situationnelle", etc.) ou des approches constructivistes (comme la logique intuitionniste). Quoiqu'il en soit des aspects particuliers intégrés dans le cadre logique, la relation entre les expressions du langage, - en particulier les formules -, et les modèles "prévus" est ce qui doit rester clair. Cela est d'autant plus vrai dans le contexte des théories logiques élaborées pour doter les langues naturelles d'une sémantique.

Les concepts de base de tout système de représentation concernent d'une part la nature des représentations elles-mêmes (leurs syntaxes et leurs possibles dénotations), et d'autre part les relations de conséquences entre ces représentations. La façon dont ces deux aspects sont reliés dans la logique des prédicats est des plus intéressantes pour la représentation de connaissances. Etant donné un ensemble R de représentations (en d'autres termes, un ensemble de formules) et une représentation r (une formule), on peut exprimer le fait que r "suit" de R d'une façon sémantique et déductive. Dans la notation usuelle R|=r est lu "r est une conséquence sémantique de R", avec la définition suivante : dans tout modèle où les représentations dans R sont vraies, r est aussi vrai. C'est-à-dire que la vérité de R garantit la vérité de r. D'une façon déductive, nous avons R|-r qui se lit "r peut être déduit (ou inféré) de R" (pour un système de déduction donné). Dans la logique des prédicats, plusieurs types de systèmes déductifs ont été proposés. Tous ont les deux propriétés suivantes:

(1) Si R|-r alors R|=r

C'est-à-dire que le système d'inférence est correct, il déduit les conséquences sémantiques qui suivent des prémisses.

(2) Si R|=r alors R|-r

C'est-à-dire que le système d'inférence est complet, il déduit toutes les conséquences sémantiques qui suivent des prémisses.

D'un point de vue théorique ces deux propriétés sont très importantes. D'un point de vue pratique, elles ne peuvent pas toujours être exploitées car il n'y a pas de démonstrateur de théorème général efficace pour la logique des prédicats complète.

Le fait que la logique des prédicats ne soit pas décidable (c.-à-d., que pour R et r arbitraires, il n'y a pas d'algorithme décidant si R⊢r ou non) ne signifie pas qu'il faille abandonner toute application. L'expérience montre qu'il est possible de "programmer" en logique, en particulier dans le sous-ensemble formé des clauses (universelles) de Horn. Un langage de programmation comme Prolog [Roussel 1975] permet des applications directes pour la représentation et la manipulation des connaissances. Dans la plupart des cas, la puissance d'expression des clauses de Horn suffit pour représenter les bases de connaissances en question. Les recherches actuellement menées pour étendre la puissance d'expression et de déduction des langages de programmation en logique laissent espérer dans un prochain avenir d'efficaces implémentations de diverses extensions de Prolog. Pour de nombreuses applications dans le domaine du traitement des langues naturelles, toute la puissance d'un système déductif ne semble pas nécessaire. Par exemple pour l'interrogation de bases de données, il suffit de pouvoir évaluer des formules dans une base relationnelle. Une telle évaluation est non seulement décidable, mais elle peut être aussi exécutée directement dans des langages comme Prolog.

Tels sont les principaux avantages de la logique comme formalisme pour la représentation des connaissances et la sémantique du langage naturel, qui bien que puissant et transparent n'est pas encore l'outil idéal.

Dans la mesure où la logique a vu le jour, et s'est développée, pour formaliser ce sous-langage naturel qu'est le langage mathématique, elle reste l'outil idéal pour représenter les connaissances mathématiques et en formaliser le discours.

Rappelons que dans ce sous-langage, la valeur de vérité des énoncés est indépendante des circonstances (locuteur, interlocuteur, lieu, etc.) dans lesquelles ils sont formulés. Dans ce cadre, certains phénomènes naturels comme la prédication, la quantification et certaines formes de conjonction ("les connecteurs") sont parfaitement décrits par la logique des propositions et des prédicats.

Vouloir ramener tous les phénomènes du langage naturel à la logique des prédicats est alors une attitude naïve. Comme le soulignent Guenthner et Nef [1984], "(...) il s'agit plûtot d'une part de faire évoluer les systèmes logiques vers une souplesse qui les rendent susceptibles de rendre compte des subtilités des langues naturelles, et d'autre part de respecter la méthodologie

qui est de règle dans le maniement des systèmes formels".

C'est dans ce cadre que nous situons les travaux d'Alain Colmerauer, de Robert Pasero et de Jean-François Pique, travaux dont nous avons intégré les résultats dans notre recherche sur les interfaces en langage naturel.

REPRESENTATION SEMANTIQUE D'UN SOUS-ENSEMBLE DU FRANCAIS

La sémantique du sous-ensemble du français que nous avons mis en oeuvre a été étudiée et formalisée dans [Pasero 1973, 1982], [Colmerauer 1979], [Colmerauer et Pique 1980] et [Pique 1981]. Ce sous-ensemble est formé par :

- un ensemble de déterminants comprenant : "le", "la" "les", "un", "une", "des", "tout(e)", "tou(te)s les", "chaque" , "aucun(e)", "des", "plusieurs", "beaucoup des", "quelques", les entiers non négatifs.

- les nom communs et les noms propres,

- un ensemble d'adjectifs simples comme : "beau", "grand", etc.,

- la négation verbale : "ne ... pas",

- la conjonction de coordination "et",

- les pronoms relatifs : "qui", "que", "dont", etc.,

- les formes interrogatives : "qui", "quels", "combien", etc.,

- les prépositions : "de", "à", etc.

- un ensemble de verbes à 0, 1 ou plusieurs compléments nominaux, et conjugués au présent de l'indicatif.

Dans la construction de la représentation sémantique d'une phrase, on peut distinguer conceptuellement deux étapes (qui dans la pratique et l'implémentation peuvent être réduites à une seule).

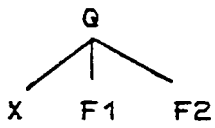
La première étape consiste à traduire la phrase dans une représentation, une formule, où sont spécifiés la quantification et un ensemble de propriétés sur les éléments quantifiés. La seconde étape consiste à traduire cette représentation dans une formule propre au système de la logique considérée (logique à 2

valeurs ou à 3 valeurs, par exemple).

Certains principes régissent la nature de la première représentation, et la hiérarchie des quantifications obéït à un ensemble d'hypothèses. Nous rappelons d'abord les principes essentiels sur la nature de cette représentation.

A chaque nom commun, à chaque adjectif et à chaque verbe est associée une propriété à n arguments explicitant la relation sous-jacente au nom, au verbe ou à l'adjectif. Dans la représentation, ces arguments sont des noms propres ou des variables.

A chaque déterminant du sous-ensemble du français on associe un arbre de la forme :



où :

- Q est un quantificateur correspondant au déterminant.
- X est la variable introduite par Q.
- F1 et F2 sont deux formules.

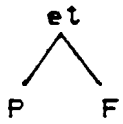
A la différence des quantificateurs classiques, Q lie la variable X aux deux formules F1 et F2, et non à une seule.

L'arbre se lit :

Pour Q X tel que F1, il est vrai que F2.

Dans le cas d'un groupe nominal simple (article + nom), F1 est la propriété introduite par le nom commun quantifié.

Dans le cas d'un groupe nominal comportant des relatives, F1 est de la forme :



où :

- P est la propriété introduite par le nom commun.

- Si la relative contient un déterminant, F est l'arbre correspondant au groupe nominal quantifié; sinon F est directement la propriété associée au verbe.

Si le verbe de la phrase comporte un complément formé d'un groupe nominal avec un déterminant, F2 est l'arbre associé au quantificateur de ce groupe nominal. Sinon F2 est directement la propriété associée au verbe.

Voici en BNF (Bakus Naur Form), la syntaxe de la représentation sémantique :

```
<formule>  ::= <relation1> ( <argument> )
           ::= <relation2> ( <argument> , <argument> )
           ::= <relation3> ( <argument> , <argument> , <argument> )
           ::= non( <formule> )
           ::= et( <formule> , <formule> )
           ::= un( <variable> , <formule> , <formule> )
           ::= le( <variable> , <formule> , <formule> )
           ::= les( <variable> , <formule> , <formule> )
           ::= chaque( <variable> , <formule> , <formule> )
           ::= tousles( <variable> , <formule> , <formule> )
           ::= aucun( <variable> , <formule> , <formule> )
           ::= <entier> ( <variable> , <formule> , <formule> )

<relation1> ::= 'tout nom ou tout verbe avec 1 argument'
<relation2> ::= 'tout nom ou tout verbe avec 2 arguments'
<relation3> ::= 'tout nom ou tout verbe avec 3 arguments'

<argument> ::= 'variable'
           ::= 'nom propre'
```

Un certain nombre d'hypothèses [Colmerauer 1979], [Pique 1981] ont été formulées sur la hiérarchie des éléments quantifiés. Nous les rappelons.

Hypothèse I (Sujet et compléments du verbe)

Dans les phrases de la forme :

```
[ Dét1  ... ]     Verbe   Comp1   [ Det2 ... ]     Compn
    Sujet                                Compi
```

(où Compi est un complément du verbe, avec n=<i=1)

Si :

- Dét1 = "un", et si
- Dét2 = "chaque", "chacun des", et si

- il existe une contrainte d'unicité sur l'argument du verbe quantifié par le sujet;

alors :

les quantifications introduites par les compléments du verbe (Comp1, Compi, Compn) dominent celles introduites par le sujet,

sinon :

la quantification introduite par le déterminant du sujet domine celles introduites par les compléments du verbe.

Hypothèse II (Nom commun suivi d'un complément)

Dans un groupe nominal de la forme :

[Dét1 Nom1 "de" Dét2 Nom2]

la quantification introduite par Dét2 domine celle introduite par Dét1, sauf si :

- Dét1 = "tout", "chaque", "aucun" ("tous les" pour le sujet) et si Dét2 est indéfini;

ou bien si :

- Dét1 est pluriel, et si la propriété attachée à Nom2 est exclusive sur l'argument de Nom1;

ou bien si :

- Dét2 est au pluriel;

Dans ces trois cas, la quantification introduite par Dét2 domine celle introduite par Dét1.

Hypothèse III (Suite de compléments de nom)

Dans un groupe nominal de la forme :

[Dét1 Nom1 "de" ... "de" Détn Nomn]

avec n>2, la quantification s'effectue sur la base de l'hypothèse II et dans l'ordre inverse d'apparition des déterminants.

Hypothèse IV (Verbe et adjectif à plusieurs compléments)

Dans une construction de la forme :

Verbe Comp1 ... Compn

ou de la forme :

Adjectif Comp1 ... Compn

avec n>1, la quantification s'effectue dans l'ordre inverse d'apparition des compléments.

Hypothèse V (Négation)

La negation verbale en "ne ... pas" introduit l'opérateur "non". Dans une phrase de la forme :

[Dét ...] "ne" verbe "pas" ...
Sujet

Si :

Dét = "chaque", "chacun des", "tout", "tous les"

Alors :

l'opérateur "non" porte sur toute la représentation.

Sinon :

l'opérateur est placé en dessous des quantifications introduites par le sujet.

Hypothèse VI (Proposition relative)

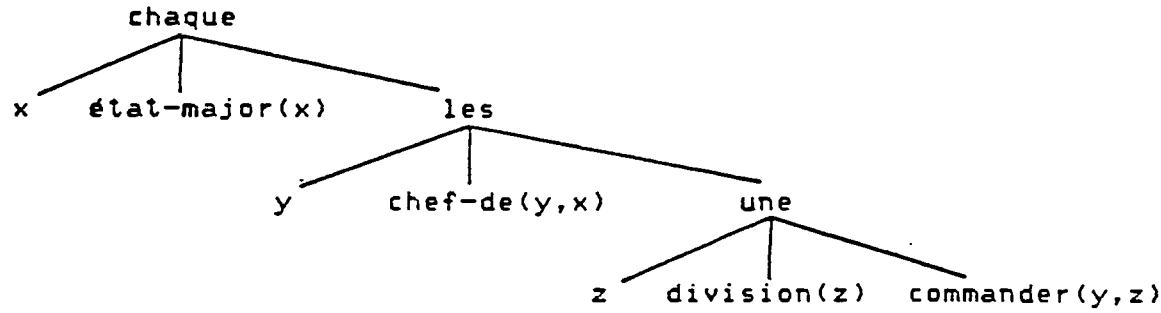
Toute proposition relative restrictive est traitée comme une phrase. Le pronom relatif est remplacé par la variable introduite par son antécédent. L'opérateur "et" relie la formule de l'antécédent et celle de la proposition relative.

De par leurs statuts d'hypothèses, ces règles qui régissent la hiérarchie des quantifications restent falsifiables, et constituent les énoncés de base d'une théorie générale sur la sémantique de notre langue. Cette approche est séduisante sur deux aspects méthodologiques fondamentaux. D'une part, il faut souligner le rôle déterminant des éléments et fonctions syntaxiques dans la construction de la représentation sémantique. Les hypothèses sont formulées à partir de notions syntaxiques maîtrisées, comme celles de "sujet", de "verbe", de "complément de nom" ou de "complément de verbe", qui limitent le recours à des "justifications" psychologiques. D'autre part, ces données syntaxiques sont complétées et pondérées par des principes formalisables comme les contraintes d'unicité et d'exclusivité (dans les hypothèses I et II) qui expriment des données de la réalité extra-linguistique propre au domaine d'application.

A partir de ces hypothèses sur la quantification, voici des exemples de questions et leurs représentations sémantiques produites par les systèmes MICROSIAL et INTERFRANCE :

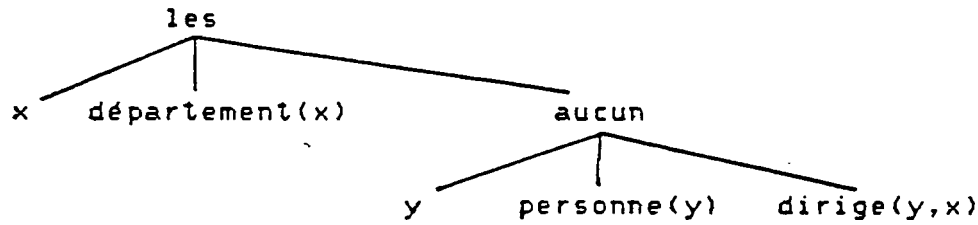
Les chefs de chaque état-major commandent-ils une division ?

Est-ce-que :



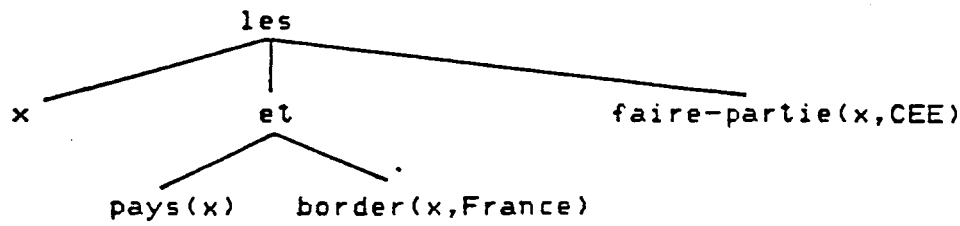
Quels sont les départements qu'aucune personne ne dirige ?

? x :



Les pays qui bordent la France font-ils partie de la CEE ?

? Est-ce-que :



Aux hypothèses régissant l'ordre des quantifications, il faut en ajouter deux autres, l'une concerne les individus vérifiant les

propriétés, l'autre concerne la nature des valeurs de vérité que peut prendre une phrase.

Hypothèse VII (Propriétés et Individus)

Les propriétés n-aires introduites par les verbes, les noms communs et les adjectifs portent sur des ensembles d'individus.

Cette hypothèse permet de rendre compte de propriétés comme "être parallèles", propriétés qui n'ont de sens que si leur argument est un ensemble. Il en va de même pour représenter sémantiquement des phrases ayant la structure syntaxique :

[Dét ...] Verbe
Sujet

où - Dét est un déterminant pluriel

- Verbe est un verbe dit symétrique [Boons et al. 1976] comme "alterner", "jointer", "cohabiter", "coïncider", etc. qui admettent les structures :

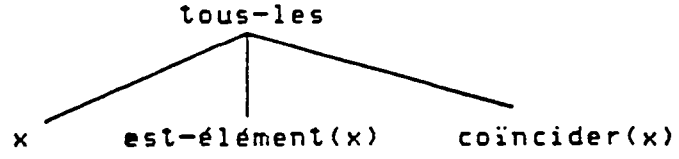
Gn1 Verbe "avec" Gn2
Gn1 "et" Gn2 Verbe
GnPluriel Verbe

Ainsi pour la phrase :

Tous les éléments coïncident

on aura la représentation sémantique :

? x :



où x désigne l'ensemble des éléments qui coïncident entre eux.

Hypothèse VIII (Valeurs de vérité)

Dans une situation donnée, une phrase est soit vraie, soit

fausse, soit absurde.

Le recours à la valeur "absurde" est nécessaire pour traiter les présuppositions associées à une phrase. Une phrase recevra la valeur "absurde" si les présuppositions qui lui sont associées sont fausses. La représentation sémantique d'une phrase S où les présuppositions sont traitées est de la forme :

si(P, Q)

où - P est une formule exprimant les présuppositions associées à S,

- Q est une formule exprimant les assertions associées à S.

Dans une phrase S, les présuppositions sont les énoncés tenus pour vrais et qui restent vrais si S est mis à la forme négative. La vérité des présuppositions de S est le prérequis pour que S puisse recevoir la valeur "vrai" ou "faux". Si P est faux, alors S est absurde.

Dans les systèmes réalisés, on s'est intéressé à rendre compte de deux types de présuppositions :

- les présuppositions propres à la sémantique du domaine d'application.

- les présuppositions exprimant des contraintes sur la cardinalité des ensembles manipulés par les quantificateurs, et qui sont indépendantes du domaine d'application.

Par exemple, dans la question :

Est-ce que les régions bordant deux pays frontaliers bordent la Méditerranée ?

une des présuppositions liée à la sémantique du domaine est que :

Une région peut border un pays frontalier

La présupposition sur la cardinalité porte sur le nombre de régions et des pays frontaliers dans la relation "border". Elle peut être ainsi formulée :

Il existe au moins une région bordant au moins deux pays frontaliers

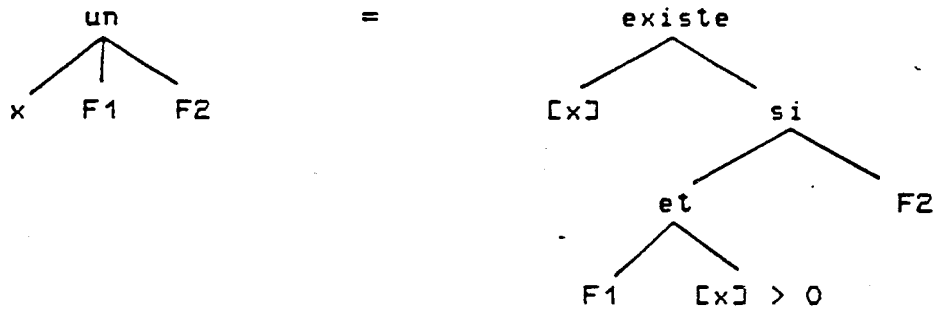
Les présuppositions sur la sémantique du domaine d'application jouent un rôle important dans l'évaluation de phrases interrogatives à la forme négative, comme par exemple, dans la question :

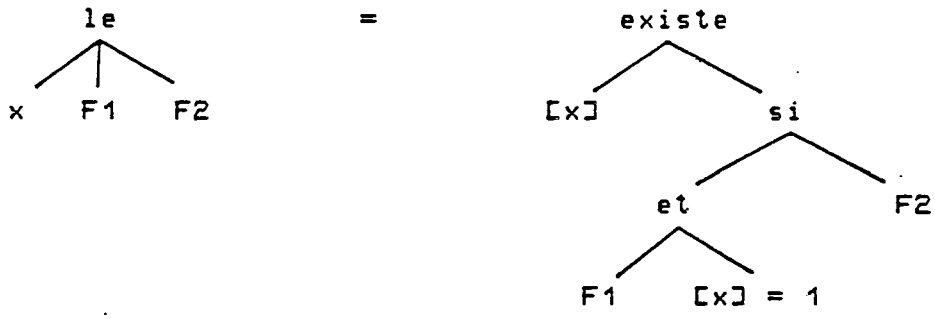
Qui ne dirige pas le département Stratégie ?

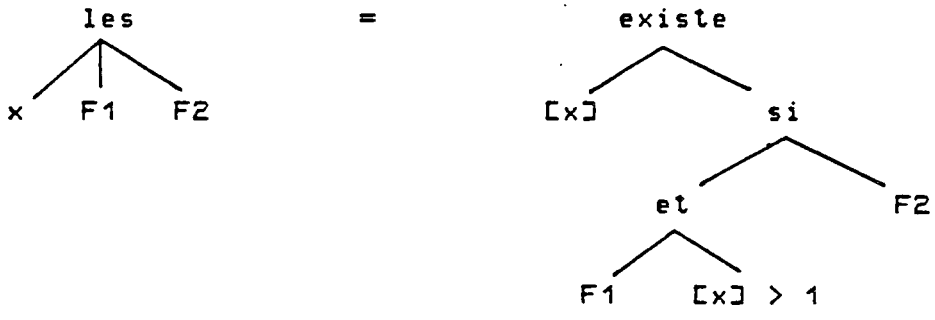
La présupposition associée à l'interrogatif "qui", à savoir une personne, permet de définir le domaine dans lequel les valeurs doivent être recherchées et prises.

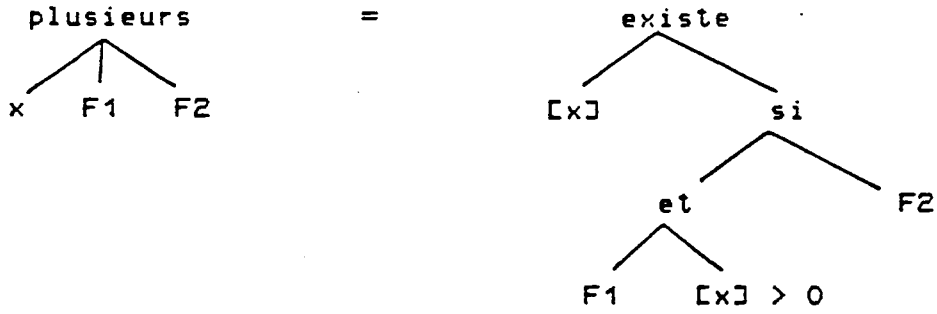
DE LA REPRESENTATION SEMANTIQUE AU SYSTEME LOGIQUE

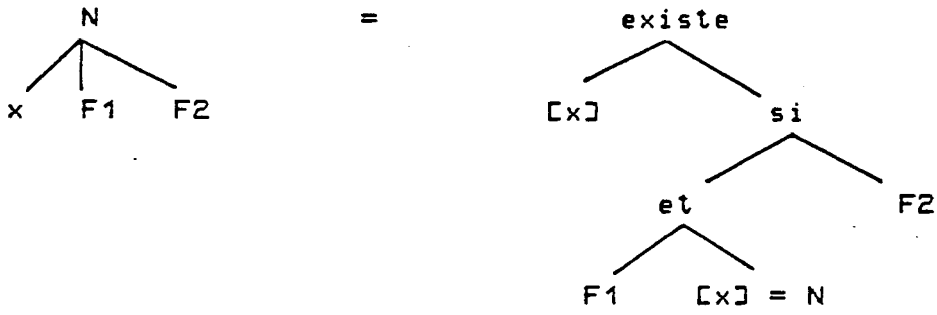
La représentation sémantique d'une phrase établie, le passage au système logique en vue de l'évaluation est réalisé à partir de l'ensemble d'égalités suivant :

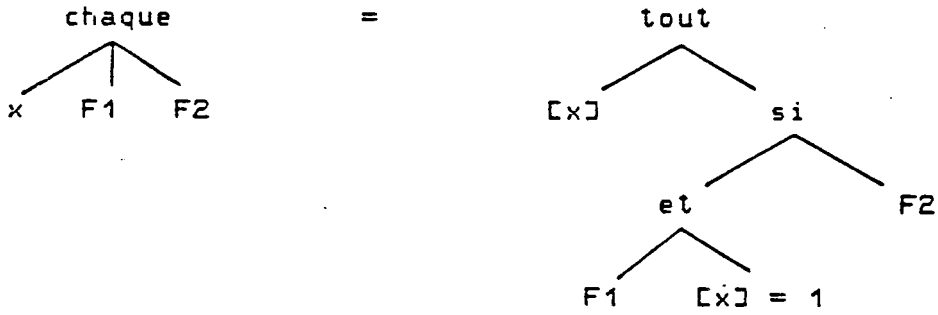


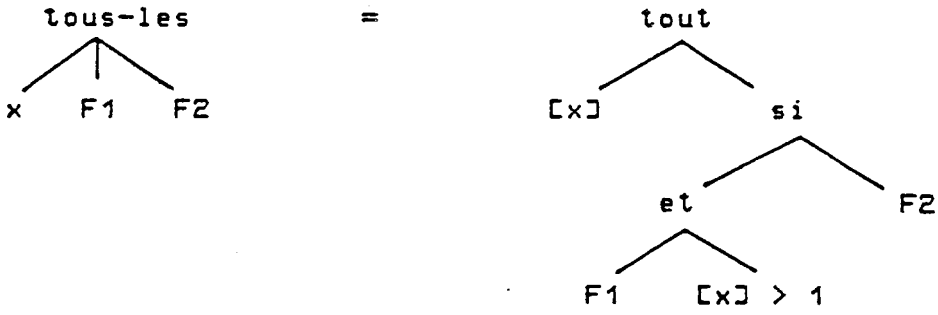


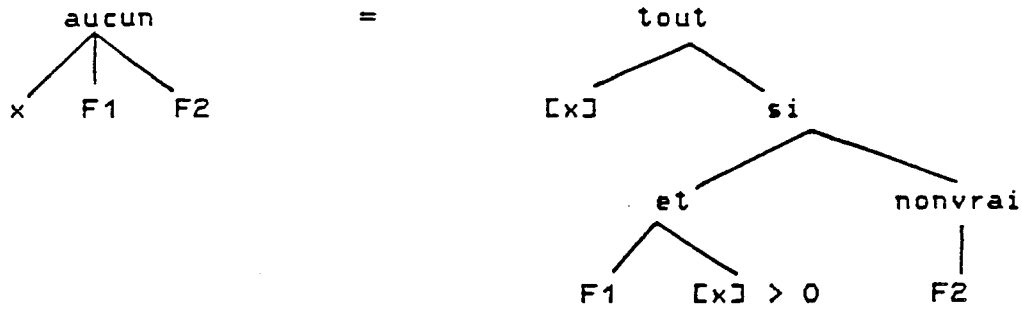



plusieurs
=
x F1 F2
existe
[x] si
et F2
F1 [x] > 0


N
=
x F1 F2
existe
[x] si
et F2
F1 [x] = N


chaque
=
x F1 F2
tout
[x] si
et F2
F1 [x] = 1


tous-les
=
x F1 F2
tout
[x] si
et F2
F1 [x] > 1



Les égalités que nous donnons ci-dessus constituent une version simplifiée de celles proposées par A. Colmerauer. Elles sont suffisantes dans le simple cadre de l'interrogation d'une base de données.

Dans ces égalités, [x] dénote l'ensemble des individus de type x. Les contraintes sur la cardinalité des ensembles d'individus sont formulées au moyen des opérateurs >, < et = .

Pour évaluation, la valeur de vérité val(F) d'une formule F de type :

si(F1,F2)

est ainsi définie :

val(si(F1,F2)) = val(F2) si val(F1) = vrai
val(si(F1,F2)) = absurde si val(F1) = faux

La valeur de vérite d'une formule du type :

nonvrai(F)

est ainsi définie :

val(nonvrai(F)) = faux si val(F) = vrai
val(nonvrai(F)) = vrai si val(F) = faux
val(nonvrai(F)) = vrai si val(F) = absurde

Les valeurs de vérité des autres types de formules sont celles définies à partir des tables de vérité de la logique classique auxquelles on ajoute les lignes suivantes :

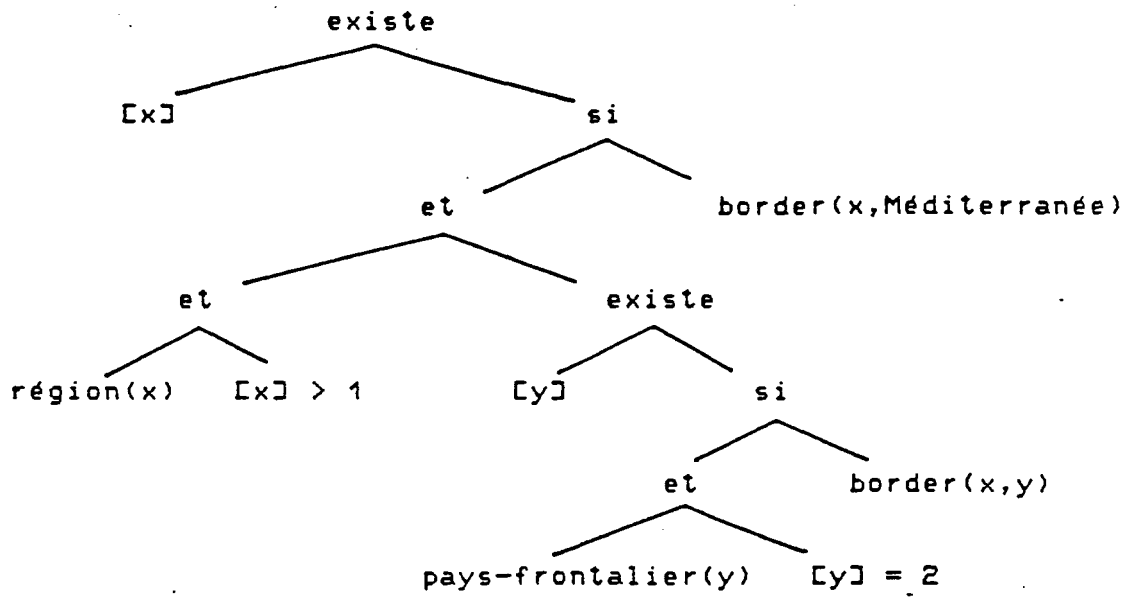
non(absurde) = absurde
et(vrai,absurde) = absurde
et(faux,absurde) = absurde
et(absurde,absurde) = absurde

Par exemple, pour la question précédente :

Est-ce que les régions bordant deux pays frontaliers bordent la Méditerranée ?

on aura à évaluer la formule :

Est-ce-que :



Une phrase sera considérée comme ambigüe si plusieurs formules sémantiques lui sont associées. C'est à partir de ces formules que des paraphrases non ambigües seront produites. Dans le chapitre VI, on trouvera des exemples de paraphrases produites par le système INTERFRANCE.

APPLICATION AUX INTERFACES POUR BASES DE DONNEES

Lorsqu'on dispose d'un sous-ensemble du français à la sémantique rigoureusement définie, deux tâches complémentaires restent à faire pour concevoir une interface en langage naturel pour une base de données. Il s'agit d'une part de définir la sémantique propre au domaine d'application de la base. D'autre part, il faut assurer le passage entre le modèle sémantique (représentation et système logique) et le modèle de la base données (structure et système de requête).

Dans la première tâche, on sera amené à définir le caractère bien formé des relations entre les différents individus et type d'individus manipulables dans la base. Par exemple, pour un

domaine d'application comme celui d'INTERFRANCE, il sera dit que:

un département a un nom,
une préfecture,
une superficie,
une population,
une densité,
etc.

un département fait partie d'une région

un département frontalier est un département
qui borde un pays étranger

une région se compose de plusieurs départements

une préfecture est une ville

une ville a un nom
une population
un code postal
etc.

etc.

D'une manière générale, on distingue trois types de relation :

- des relations du type :

x ETRE y

qui définissent des classes et une hiérarchie dans l'ensemble des individus.

- des relations du type :

x AVOIR y

qui définissent l'ensemble des caractéristiques (ou "attributs") Y d'une classe d'individus.

- des relations d'autres types spécifiant une sémantique particulière, comme les relations "border" et "traverser" dans les exemples suivants :

un département peut border un département

un fleuve peut traverser un département

Ces informations peuvent être directement codées au niveau du lexique et de la grammaire, au moyen d'un ensemble de traits et de contraintes plus ou moins complexes, comme c'est le cas dans les systèmes MICROSIAL et INTERFRANCE. Ces informations peuvent

être aussi organisées dans une base indépendante qui est consultée au fur et à mesure de l'analyse, comme c'est le cas dans les systèmes ORBI et INTERFACILE.

Comme nous le verrons dans le chapitre V, la qualité du traitement des erreurs sémantiques dépend du soin apporté à la description du domaine d'application et du modèle choisi pour le décrire.

Le passage du modèle sémantique (représentation et système logique) à celui de la base de données (structure et système de requête) est quasi-immédiat si la base est (ou se comporte comme) un ensemble de formules logiques. C'est le cas des bases associées aux systèmes que nous avons réalisés. Les bases sont des ensembles de clauses de Horn écrite en Prolog. L'évaluation des formules est confiée à un court programme écrit en Prolog, langage qui permet aussi des applications directes pour l'évaluation et la déduction de formules en logique du 1er ordre.

Dans le cas d'un autre type de base de données, la traduction des formules logiques devra se faire dans le langage de requête du système concerné, SQL par exemple dans le cas d'un système relationnel. Nous n'avons pas mis en oeuvre cet aspect au cours de nos travaux. Sur ce sujet, nous avons consulté en particulier les travaux de [Li 1984] et [Wallace 1984].

CONCLUSION

Beaucoup de phénomènes sémantiques propres au langage naturel restent encore à élucider et à maîtriser. S'il fallait s'en convraincre, on pourrait reprendre le "puzzle" de Peter Strawson [1957, p.194] :

> "Comparons ce que nous pourrions appeler "un syllogisme naturel" avec les exemples obstinément irréalistes que l'on trouve dans les livres. Un morceau de dialogue pourrait être:
>
> - Un homme vient d'avaler toute une bouteille d'alcool à brûler.
> - Quiconque avale une telle dose ne survit pas.
> - Donc il mourra.
>
> Remarquez comment le premier terme indéfini conduit au pronom défini "il" dans la conclusion, et comment le temps se modifie à chaque pas. A la différence des autres déviations par rapport au schéma strict (tel que "mourir"

pour "ne pas survivre" et "avaler une telle dose" pour "boire toute une bouteille", etc.) ces deux là ne peuvent être éliminées sans détruire le sens de l'original."

Il y a une différence importante entre les systèmes de compréhension du langage naturel conçus pour l'interrogation (et la mise à jour) de bases de données existantes et formatées, et des systèmes qui nécessitent la construction de bases de données étendues, c'est-à-dire, de représentations de connaissances à partir de formulations en langage naturel, qu'il s'agisse de phrases isolées ou de textes.

Nous avons essayé de montrer dans ce chapitre que pour les premiers, les techniques de la sémantique formelle permettent la résolution des principaux problèmes sémantiques et logiques, même si une somme importante de travail reste encore à faire. Les autres systèmes, pensons nous, constituent une entreprise beaucoup plus compliquée encore, même si l'on s'en tient à la seule manipulation logique des représentations.

CHAPITRE IV

ANAPHORES : SYNTAXE ET RESOLUTION

Introduction

Proformes

Ellipses

Analyse et synthèse d'anaphores

Résolution d'anaphores dans les systèmes
Dialogues, Orbi et Interfacile

Conclusion

INTRODUCTION

Une des caractéristiques d'un langage naturel, comparé à un langage formel, réside dans l'usage d'anaphores. Qu'est-ce qu'une anaphore? Voici une possible définition:

> Une ANAPHORE est une unité linguistique qui fait référence soit à d'autres unités linguistiques qui la précédent ou la suivent dans le discours, soit à des entités extra linguistiques propres à la situation où se déroule le discours. On appellera REFERENT LINGUISTIQUE l'unité linguistique à laquelle une anaphore peut faire référence.

Les pronoms et les ellipses sont des exemples d'anaphores. Toute interface en langage naturel devrait permettre l'emploi de ces éléments qui assurent un dialogue concis, vif et spontané.

Voici l'extrait d'une session avec le système DIALOGUES [Sabatier 1980] comportant des phrases elliptiques. Ici l'utilisateur définit un ensemble d'individus et décrit une suite d'actions mettant en jeu ces individus:

- Jacques est une personne.
- Luc aussi.
- Mais Okapi, non.
- Milou, non plus.
- Luc si.
- Et Bob aussi.
- Mais Médor, non.
- Jacques rend Milou à Bob.
- Max, Médor à Luc.
- Luc échange Médor contre Milou.
- Bob donne Médor à Max.
- Et Luc, Milou contre Médor.

L'extrait suivant est tiré d'une suite de questions posées au système INTERFACILE [Mathieu et Sabatier 1985]. Ces questions mettent en jeu un ensemble de formes anaphoriques:

- Comment éditer un fichier ?
- Comment l'imprimer ?
- Comment en faire une copie ?
- Comment ajouter son contenu à un fichier ?
- Comment effacer ce dernier ?
- Comment changer le nom de celui-ci ?

Une anaphore et son référent linguistique peuvent dénoter des

personnes, des choses ou des situations identiques. Par exemple dans:

Un homme entra. Il s'appelait Max.

Si "un homme" est le référent linguistique du pronom "il", alors ce référent et le pronom sont ici COREFERENTS: ils désignent le même individu.

Une anaphore et son référent linguistique peuvent dénoter des personnes, des choses ou des situations distinctes. Comme dans l'exemple donné par [Gross 1975]:

Max achète des lits, Luc en vend de grands à baldaquin.

Le pronom "en" et son référent "des lits" n'entretiennent pas forcément une relation de coréférence: les lits qu'achètent Max peuvent être distincts de ceux que Luc vend.

Le référent de certaines anaphores peut être extra linguistique. L'anaphore renvoie alors à un (ou des) élément(s) de l'univers où se déroule le discours. L'anaphore a une interprétation DEICTIQUE. Par exemple, associé à la scène où Max pointe du doigt un objet, Max dit à Luc:

Donne-moi ça !

Ici l'anaphore "ça" a une valeur déictique.

Nous nous intéresserons ici aux cas où les référents linguistiques d'une anaphore existent.

La résolution d'une anaphore est l'opération qui consiste à trouver son référent linguistique et dans certains cas, à déterminer l'existence d'une relation de coréférence.

Si on devine l'intérêt de l'usage des anaphores dans une interface en langage naturel, on imagine aussi la complexité des problèmes à résoudre qu'entraîne leur introduction.

Dans ce chapitre, nous essayerons alors de répondre aux questions suivantes:

Quels sont les différents types d'anaphores ?

Quelle est la nature des informations nécessaires pour les résoudre ?

Nous verrons comment intégrer ces éléments dans le cadre général d'une interface. Nous étudierons la façon dont les systèmes DIALOGUES, ORBI et INTERFACILE en particulier, opèrent pour rendre compte de certaines anaphores.

Une première distinction peut être faite entre 1) les anaphores lexicalement réalisées, comme les pronoms par exemple, et 2) les anaphores qui ne sont pas lexicalement réalisées. Dans ce dernier cas, on parlera alors d'anaphores nulles (que l'on notera []), comme dans l'exemple suivant:

Max mange une pomme. Luc [] une poire.

Nous appellerons PROFORMES les anaphores lexicalement réalisées, et ELLIPSES, celles qui ne le sont pas.

(Comme précédemment, dans les règles de grammaire, toute expression notée entre simples apostrophes est un symbole terminal. Toute expression commençant par au moins deux lettres est un symbole non-terminal. Une expression formée d'une seule lettre suivie (ou non) d'un entier est une variable).

PROFORMES

Sous la catégorie "proforme", nous rangeons les formes linguistiques suivantes:

- les articles définis,
- les adjectifs possessifs,
- les pronoms possessifs,
- les adjectifs démonstratifs,
- les pronoms démonstratifs,
- les pronoms relatifs,
- les proformes interrogatives,
- les pronoms propremenTs dits,
- les formes en "le faire",
- l'adverbe "ainsi",
- les formes en "tel",

que nous allons maintenant étudier.

Les articles définis

Les articles définis (ARTDEF) que l'on peut décrire par les règles suivantes:

ARTDEF(mas.sin) -> 'le'

ARTDEF(fem.sin) -> 'la'

ARTDEF(x.sin) -> 'l''

ARTDEF(x.plu) -> 'les'

sont-ils deş proformes ?

(la variable x est mise pour une valeur inconnue du genre)

Lorsqu'il figure dans l'intitulé d'un nom propre, comme par exemple:

La France ou Les Antilles

l'article défini n'a pas de valeur anaphorique.

Par contre l'interprétation non générique d'un article défini passe par la référence avec des unités linguistiques qui peuvent le précéder dans le discours, comme dans:

Un garçon et une fille entrèrent.

Le gars s'appelait Max.

ou qui peuvent le suivre, comme dans l'exemple suivant:

Max m'a raconté l'histoire que Luc lui a racontée.

Dans ce dernier cas, le groupe nominal est de la forme:

ARTDEF NOM MODIFIEUR

où MODIFIEUR peut être un complément de nom, une relative, une superlative, etc. Le référent linguistique est immédiatement précisé par le modifieur. La valeur anaphorique de l'article défini est évidente si on compare l'exemple précédent au suivant:

Max m'a raconté une histoire que Luc lui a racontée.

L'interprétation générique d'un article défini tient à ce qu'aucun référent n'est à sélectionner dans le discours, comme dans l'exemple suivant:

L'article et le nom s'accordent en genre et en nombre.

En terme de quantification, l'article défini reçoit ici une lecture universelle. Une paraphrase possible de l'exemple précédent serait:

Tout article et tout nom s'accordent en genre et en nombre.

Intéressons-nous au cas d'un groupe nominal simple formé d'un article défini et d'un nom commun, c'est-à-dire:

GN -> ARTDEF(a) NOM-COMMUN(a,s)

(la variable "a" désigne la liste des traits de genre et de nombre, et "s" celle des traits sémantiques)

et où le référent le précède. Comme dans le premier exemple, que nous rappelons:

Un garçon et une fille entrèrent.
Le gars s'appelait Max.

Accompagné du nom "gars", l'article "le" se conduit comme une proforme et introduit ici une relation de coréférence entre la personne qu'il dénote et une autre personne dénotée par son référent.

En règle générale, la recherche des référents linguistiques possibles s'effectue sur la base des traits "sémantiques" du nom

associé à l'article défini. Les noms "garçon", "fille" et "gars" pourraient être ainsi décrits:

NOM(mas.sin,humain.homme.garçon) -> 'garçon'

NOM(fem.sin,humain.femme.fille) -> 'fille'

NOM(mas.x,humain.homme.gars) -> 'gars'

Sur la base des traits sémantiques (s) qu'il partage avec "garçon", "gars" (au contraire de "fille") est un référent possible, et les groupes nominaux "un garçon" et "un gars" peuvent être coréférents.

Dans le discours suivant:

Un gars sortit.
Un garçon et une fille entrèrent.
Le gars s'appelait Max.

"le gars" aurait pour référent linguistique "un gars" et non "un garçon": la personne qui est sortie et celle qui s'appelait Max seraient alors le même individu.

Les exemples que nous venons de donner sont relativement simples. Le problème se complique avec la multiplication des traits sémantiques auxquels il faut avoir recours pour rendre compte de relations implicites. Comme dans le petit discours suivant:

Luc a un fils.
Max a trois fils.
L'aîné s'appelle Léo.

"L'aîné" et un des "trois fils" de Max sont coréférents.

Une façon de rendre compte de la relation anaphorique serait:

1) d'avoir recours au trait "lien de parenté" dans la description des noms "frère" et "aîné", et

2) d'analyser le groupe nominal "l'aîné" comme une possible réduction du groupe nominal complexe "l'aîné de trois fils".

L'exemple ci-dessus illustre le cas où le référent est un groupe nominal quantifié. La relation de coréférence introduit une opération de sélection parmi l'ensemble des individus.

Le discours suivant:

Max n'a résolu aucun problème.
Les problèmes étaient trop difficiles.

montre que le nombre (singulier vs pluriel) n'est pas toujours un élément pertinent dans la recherche du référent d'un groupe nominal défini.

Ces exemples illustrent quelques uns des problèmes à résoudre pour sélectionner le référent d'un groupe nominal simple contenant un article défini.

Les adjectifs possessifs

Les adjectifs possessifs (APOSS) peuvent être décrits par les règles suivantes:

```
APOSS(1.mas.sin) -> 'mon'
APOSS(1.fem.sin) -> 'ma'
APOSS(1.x.plu) -> 'mes'

APOSS(2.mas.sin) -> 'ton'
APOSS(2.fem.sin) -> 'ta'
APOSS(2.x.plu) -> 'tes'

APOSS(3.mas.sin) -> 'son'
APOSS(3.fem.sin) -> 'sa'
APOSS(3.x.plu) -> 'ses'

APOSS(4.x.sin) -> 'notre'
APOSS(4.x.plu) -> 'nos'

APOSS(5.x.sin) -> 'votre'
APOSS(5.x.plu) -> 'vos'

APOSS(6.x.sin) -> 'leur'
APOSS(6.x.plu) -> 'leurs'
```

Rappelons que les formes masculines "mon", "ton" et "son" précèdent une expression de genre féminin lorsqu'elle sont immédiatement suivies d'une voyelle. Exemples:

son aventure

son éternelle jeunesse

Dans un discours, une fois le groupe nominal "Max" introduit, une paraphrase possible de:

l'exemple de Max

peut être:

son exemple

L'interprétation d'un groupe nominal de la forme:

APOSS(i.g.n) NOM

s'effectue à partir de la structure équivalente:

ARTDEF(g.n) NOM de GN(i)

Le problème se ramène alors à trouver quelle est la valeur de GN(i).

Pour GN(i), les règles sont les suivantes:

GN(1) -> 'moi'
GN(2) -> 'toi'
GN(3) -> GN
GN(4) -> 'nous'
GN(5) -> 'vous'
GN(6) -> GN

Dans le cas où i prend les valeurs 3 ou 6, un groupe nominal doit être sélectionné. Le référent peut être dans la même phrase, comme dans:

Max place toujours son exemple.

ou dans des phrases antérieures, comme dans le discours:

Max a donné un exemple.
Son exemple était percutant.

D'une façon générale, la recherche d'un GN(i) possible s'effectue sur la base d'une compatibilité sémantique de l'association de NOM et GN(i) dans la construction (et la relation):

NOM de GN(i)

Certains cas peuvent être cependant résolus par le simple jeu des propriétés syntaxiques. Dans les constructions de la forme:

GN1 VERBE GN2 pour GN3

où GN3 est de la forme:

APOSS NOM

on aura:

GN(i) = GN2

Exemple:

Max apprécie Luc pour son courage

Les pronoms possessifs

Les pronoms possessifs (PPOSS) peuvent être décrits comme constitués d'un article défini suivi d'une forme possessive s'accordant en genre et en nombre avec l'article. On a les règles:

PPOSS(i.g.n) -> ARTDEF(g.n) FPOSS(i.g.n)

avec:

FPOSS(1.mas.sin) -> 'mien
FPOSS(1.fem.sin) -> 'mienne
FPOSS(1.mas.plu) -> 'miens
FPOSS(1.fem.plu) -> 'miennes

FPOSS(2.mas.sin) -> 'tien'
FPOSS(2.fem.sin) -> 'tienne'
FPOSS(2.mas.plu) -> 'tiens'
FPOSS(2.fem.plu) -> 'tiennes'

FPOSS(3.mas.sin) -> 'sien'
FPOSS(3.fem.sin) -> 'sienne'
FPOSS(3.mas.plu) -> 'siens'
FPOSS(3.fem.plu) -> 'siennes'

FPOSS(4.g.sin) -> 'notre'
FPOSS(4.g.plu) -> 'notres'

FPOSS(5.g.sin) -> 'votre'
FPOSS(5.g.plu) -> 'votres'

FPOSS(6.g.sin) -> 'leur'
FPOSS(6.g.plu) -> 'leurs'

La fonction sémantique caractéristique d'un pronom démonstratif est de différencier des entités qui partagent des propriétés communes quant aux entités qui les possèdent. Comme dans le discours suivant:

Max a réparé une montre.
La mienne avance.

Et comme le suggère le discours suivant:

Max a réparé sa montre.
La sienne est cassée.

dans le cas où la montre que Max a réparée et la montre qui est cassée appartiennent à Max.

Ce dernier discours est à comparer au discours cohérent suivant:

Max a réparé sa montre.
La mienne avance.
La sienne est cassée.

où la montre que Max a réparée et la montre qui est cassée appartiennent à Max (et dénotent le même objet).

La résolution d'un pronom possessif:

PPOSS(i.g.n)

s'effectue, comme pour un groupe à adjectif possessif, à partir de la structure équivalente:

ARTEDF(g.n) NOM de GN(i)

avec une condition spécifique à la résolution du pronom démonstratif, à savoir:

- dans le discours une entité de la forme (ou dérivable de la forme):

NOM de GN(j)

précède PPOSS(i.g.n).

- et GN(i) et GN(j) ne sont pas coréférents.

Les adjectifs démonstratifs

Les adjectifs démonstratifs:

ce cet cette ces

sont des proformes. Le référent d'un goupe nominal contenant un tel adjectif comme déterminant peut précéder dans le discours cet adjectif. Comme dans:

Un gars est entré.
Ce gars s'appelait Max.

Le référent peut suivre l'adjectif démonstratif comme dans le cas où le référent et l'adjectif font partie d'un même groupe nominal complexe. Comme dans l'exemple suivant:

Ce malin de Max a trouvé la solution.

Lorsqu'il précède l'adjectif, le référent peut être un groupe nominal explicite, comme dans le premier exemple. Dans ce cas, la recherche du référent ne pose pas de problèmes particuliers. Par contre, le référent peut être implicite. Comme dans l'exemple suivant:

Max n'a pas respecté un stop.
Cette imprudence lui a coûté cher.

Le référent de "cette imprudence" peut être résolu en ayant recours à une phrase du type classificatoire [Gross 1977], comme:

Ne pas respecter un stop est une imprudence.

On aurait alors à reconstruire le discours suivant:

Max n'a pas respecté un stop.
(Ne pas respecter un stop est une imprudence)
Cette imprudence lui a coûté cher.

pour résoudre le référent de "cette imprudence".

Les pronoms démonstratifs

On trouvera dans [Gross 1977] une étude détaillée des groupes nominaux avec pronom démonstratif. Nous rappelons ici la structure de ces groupes.

- "ce" en position sujet du verbe "être", comme dans:

  Ce sera fait.

- "ce" suivi d'une relative en position de sujet ou de complément, comme dans:

  Max approuve ce que Luc a dit.

- "ça" , "ceci" et "cela" en position de sujet ou de complément, comme dans:

Max mange cela tous les jours.

- les groupes nominaux formés par les combinaisons:

celui	-ci
ceux	-là
celle	RELATIVE
celles	de GN
	Part-PASSE
	Part-PRESENT
	ADJECTIF(-able)
	ADJECTIF(-ible)
	COMPLETIVE
	INFINITIVE

Dans ces dernières constructions, le genre (masculin vs féminin) des démonstratifs "celui", "ceux", "celle" et "celles" est une une information pertinente pour la recherche du référent linguistique. Cette recherche peut se faire à partir des structures de base que Gross propose pour dériver ces groupes anaphoriques, à savoir:

ce NOM MODIFIEUR

ou

ce NOM de { GN / NOM } MODIFIEUR

Un groupe anaphorique constitué du seul démonstratif "ce", "ça", "ceci" ou "cela" ne contient pas de traits sémantiques pertinents qui pourraient faciliter la selection d'un référent linguistique lorsque celui-ci existe.

Lorsque le démonstratif est sujet du verbe être, le référent peut le suivre, comme dans:

C'est inutile de continuer.

Le référent peut précéder le démonstratif, comme dans:

Max n'a pas respecté un stop.

Cela lui a coûté cher.

ou dans:

Max n'a pas respecté un stop.
C'est dingue.

Le référent est, dans ces cas, de nature phrastique. Il pourrait être reconstruit à partir de la structure:

le fait que PHRASE

où PHRASE renvoie à l'énoncé précédent dans le discours. On aurait alors:

PHRASE1 = Max n'a pas respecté un stop.

PHRASE2 = C'est dingue.

avec:

ce = le fait que PHRASE1

Le référent linguistique peut être un ensemble de phrases. En particulier lorsque le démonstratif est précédé du déterminant "tout". Comme dans le discours suivant:

PHRASE1 = Max n'a pas respecté un stop.

PHRASE2 = Luc a trouvé la solution du problème.

PHRASE3 = Tout cela est incroyable

On a:

Tout cela = le fait que PHRASE1 et PHRASE2

Les pronoms relatifs

Les pronoms suivants:

qui que dont où lequel laquelle lesquels lesquelles

entrent dans la construction de phrases relatives. Les pronoms "qui", "où" et les formes en "lequel" peuvent être précédés d'une préposition.

Précédés de la préposition "à", les pronoms "lequel", "lesquels" et "lesquelles" deviennent respectivement "auquel", "auxquels" et

"auxquelles". Précédés de la préposition "de", ces pronoms deviennent respectivement "duquel", "desquels" et "desquelles".

Le référent linguistique d'un pronom relatif précède immédiatement ce dernier (d'où le nom d'"antécédent"). La résolution d'un pronom relatif ne pose pas de problèmes particuliers si ce n'est le cas où l'antécédent est un groupe nominal complexe de la forme:

NOM de NOM

comme dans la phrase:

le frère de la personne que Max connaît est Luc.

L'antécédent de "que" peut être ici "la personne" ou "le frère de la personne".

Dans la phrase suivante:

L'adresse de la personne que Max estime est illisible.

on s'accordera pour dire que le pronom relatif "que" a pour antécédent "la personne" et non "l'adresse de la personne". On dira le contraire pour la phrase:

L'adresse de la personne que Max a écrite est illisible.

Si le pronom est "qui" et si "qui" est sujet, l'antécédent est celui qui s'accorde en nombre avec le verbe de la relative. Comme dans:

Le nom des personnes qui gagneront ne sera pas diffusé.

S'il y a ambiguïté, comme dans le premier exemple, on appliquera la procédure générale suivante:

- déterminer la fonction (sujet, objet direct, objet indirect, ...) du pronom dans la proposition relative,

- prendre le verbe de la relative et extraire du lexique la sous-liste des traits du verbe propre à la fonction qu'occupe le pronom,

- comparer cette sous-liste avec la liste des traits de chaque NOM du groupe nominal antécédent,

- retenir comme antécédent le NOM dont la liste de traits est la plus largement partagée avec la sous-liste du verbe.

Les proformes interrogatives

Les pronoms "qui", "que" et "quoi" et les formes en "lequel" entrent dans la construction de phrases interrogatives.

"qui" et les formes en "lequels" en fonction sujet sont toujours placés au début de l'interrogative. Exemples:

Qui viendra ?

Lequel viendra ?

Pour les autres fonctions, "qui" et les formes en "lequel" peuvent être placés au début ou à la fin de l'interrogative. Exemples:

Avec qui Max parle-t-il ?
Max parle avec qui ?

"que" est toujours placé au début de l'interrogative et ne peut pas être précédé d'une préposition.

"quoi" ne peut être placé au début de l'interrogative que s'il est précédé d'une préposition. Il peut être placé en fin, précédé ou non d'une préposition. Exemples:

Avec quoi Max a-t-il fait cela ?
Max a fait cela avec quoi ?

Les pronoms pouvant être placés au début d'une interrogative peuvent être suivis de l'expression "est-ce que". Exemples:

Qui est-ce que Max a vu ?

Lequel est-ce que Max préfère ?

Avec quoi est-ce que Max a fait cela ?

Les formes en "lequel" peuvent être suivies de la structure:

de GN

où GN est au pluriel et dénote un ensemble d'entités. Exemples:

Lequel de ces trois livres Max préfère-t-il ?

Lequel d'entre nous partira le premier ?

Les formes:

quel quelle quels quelles

sont des pronoms interrogatifs. Ils entrent en particulier dans la structure:

quel ETRE GN

où GN est un groupe nominal qui ne peut être un nom propre. La forme de base de ce GN est:

ARTDEF NOM MODIFIEUR

où MODIFIEUR peut être en particulier un complément de nom ou une relative. Exemples:

Quelles sont les chances de Max de réussir ?

Quels sont les étudiants qui ont réussi ?

Dans les autres constructions interrogatives, les formes en "quel" sont suivies d'un groupe nominal sans article, et peuvent être précédées d'une préposition. Elles peuvent être placées en tête ou en fin de l'interrogative. Exemples:

Quelle solution Max a-t-il donné ?
Max a donné quelle solution ?

Avec quelle voiture Max roule-t-il ?
Max roule avec quelle voiture ?

Comme les pronoms précédents, les formes:

où quand comment pourquoi

sont des proformes à fonction interrogative.

La résolution des proformes interrogatives est l'objet même du recours à une proposition les contenant.

Même si un possible référent linguistique peut se trouver dans l'interrogative, comme dans:

Qui a fait le coup, Max ou Luc ?

ou dans le discours antérieur, comme dans:

Max connaît Luc.
Qui connaît Max?

la résolution d'une proforme interrogative dépasse les simples données linguistiques.

Si ces données s'avèrent insuffisantes, elles contiennent cependant un ensemble d'informations pertinentes pour la recherche d'un référent linguistique ou non linguistique.

Ainsi l'interrogatif "qui" laisse entendre que l'antécédent est de type animé, à la différence des formes "que" et "quoi".

Les interrogatifs "où", "quand", "comment", "pourquoi" présupposent que l'antécédent dénote respectivement un lieu, une entité temporelle, une manière, une raison, etc.

Il est intéressant de noter que certaines proformes interrogatives peuvent être paraphrasées par une construction du type:

(PREPOSITION) quel ETRE GN

Exemples:

où: à quel l'endroit ...

quand: à quel moment ...

pourquoi: pour quelles raisons ...

comment: de quelle façon ...

qui: quelle est l'"entité animée" ...

Les pronoms proprement dits

Les pronoms proprement dits sont des substituts de groupes nominaux. Ils peuvent être sujets ou compléments d'un verbe. Dans la table ci-dessous nous listons ces pronoms, accompagnés de certaines propriétés.

	sujet	complément		genre	nombre	humain
		PV	VP			
je	+	−	−		s	h
tu	+	−	−		s	h
il	+	−	−	m	s	
ils	+	−	−	m	p	
elle	+	−	+	f	s	
elles	+	−	+	f	p	
on	+	−	−		s	
nous	+	+	+		p	h
vous	+	+	+		p	h
lui	+	+	+	m	s	
eux	+	−	+	m	p	
me	−	+	−		s	h
m'	−	+	−		s	h
moi	−	−	+		s	h
te	−	+	−		s	h
t'	−	+	−			h
toi	−	−	+		s	h
se	−	+	−			
s'	−	+	−			
soi	−	−	+		s	h
leur	−	+	−		p	
en	−	+	−			
y	−	+	−			
le	−	+	−	m	s	
la	−	+	−	f	s	
les	−	+	−		p	
l'	−	+	−		s	

La première colonne des propriétés rend compte de la fonction sujet. Si le pronom peut occuper à lui seul une telle fonction, le signe "+" est marqué, sinon le signe "−" est noté. Les deux colonnes suivantes rendent compte des fonctions compléments. La première (PV) précise si le pronom peut occuper une position préverbale en fonction complément. Comme "le" dans:

Max le croit.

La seconde colonne (VP) précise si le pronom peut occuper en fonction complément une position postverbale (et toujours précédé d'une préposition). Comme "eux" dans:

Max est avec eux.

Rappelons que des verbes comme "penser" n'admettent pas "lui" en position préverbale alors que cela est possible avec des verbes comme "donner". Nous donnons ci-dessous les différentes structures combinant les ordres possibles des pronoms compléments en position préverbale. (Le séparateur "," se lit "ou").

SUJET [nous,vous,lui,me,m',te,t',se,s',lui,en,y,le,la,les,l'] VERBE

SUJET [nous,vous,me,m',te,t',se,s'] [le,la,les,l',en,y] VERBE

SUJET [lui,leur] en VERBE

SUJET [le,la,les,l'] [lui,leur] VERBE

SUJET [les,l'] [en,y] VERBE

La quatrième colonne de la table précise le genre associé au pronom. "m" pour masculin, "f" pour féminin, et rien dans le cas où le trait n'est pas distinctif. La colonne suivante précise le nombre. "s" pour singulier, "p" pour pluriel, et rien lorsque le trait n'est pas distinctif.

La dernière colonne précise si le trait humain (h) est obligatoirement associé au pronom. Si le trait n'est pas distinctif, rien n'est noté.

Les pronoms "se", "s" et "soi" ont pour référent linguistique le sujet du verbe dont ils sont compléments. Les pronoms "me", "m'" et "moi" ont pour référent linguistique le locuteur de l'énoncé, et les pronoms "te" , "t'" et "toi" l'interlocuteur.

Gross [1975] a montré que dans certaines constructions, le référent linguistique d'un pronom est localisable par le simple jeu des propriétés syntaxiques. Il s'agit des constructions à complétives du type:

GN1 dire de GN2 que PHRASE

Comme dans l'exemple suivant:

Max a dit de Luc qu'il était sympathique

où le référent linguistique de "il" ne peut être que "Luc" .

Dans un discours où la phrase suivante est énoncée:

Il est en retard parce que Max ne s'est pas réveillé à temps.

"Max" ne peut pas être le référent linguistique du pronom "il". Ce phénomène est généralisable aux phrases formées d'une proposition principale avec un pronom sujet et d'une proposition subordonnée dont le sujet n'est pas un pronom. Par contre, dans la phrase:

Il est en retard parce qu'il ne s'est pas réveillé à temps.

Les deux pronoms "il" peuvent désigner le même individu.

Dans les autres cas, la procédure générale de résolution d'un pronom consiste à appliquer les opérations suivantes:

- déterminer la fonction (sujet, objet direct, indirect, complément de phrase etc.) du pronom,
- prendre le verbe associé au pronom et extraire du lexique la sous-liste des traits du verbe propre à la fonction qu'occupe le pronom,
- comparer cette sous-liste avec la liste des traits de chaque groupe nominal précédent,
- retenir comme référent linguistique possible du pronom le groupe nominal dont la liste de traits est la plus largement partagée avec la sous-liste du verbe.

"le faire"

Dans certaines structures, l'expression formée du verbe "faire" (à l'infinitif ou conjugué) précédé du pronom "le" a une valeur anaphorique, comme dans les deux exemples suivants:

(Max a refusé de démissionner)
Luc n'a pas hésité à le faire.

(Max a donné sa démission en 1984)
Luc l'a fait en 1985.

Le référent linguistique d'une anaphore en "le faire" peut être une infinitive complément d'un verbe, comme dans le premier exemple ou un groupe verbal, comme dans le second exemple. On a:

le faire = démissionner
l'a fait = a donné sa démission

"aussi", "si", "oui", "non" et "non plus"

Les adverbes "aussi", "si", "oui", "non" et "non plus" ont une valeur anaphorique comme le montre le petit discours suivant:

Max, connaît une solution.
Luc, aussi.
Léo, non.
Bob, oui.
Ida, non.
Guy, non plus.
Léa, si.

La résolution de ce type d'anaphore consiste à reprendre le groupe verbal (explicite ou implicite) de la phrase précédente et de le modifier (ou non) compte tenu de la valeur affirmative ou négative contenu dans la forme de l'anaphore. Ainsi, pour l'exemple précédent, on aura les équivalences:

aussi = connaît une solution
non = ne connaît pas une solution
oui = connaît une solution
non plus = ne connaît pas une solution
si = connaît une solution

Ces proformes sont d'un type particulier dans la mesure où elles conduisent à une modification de la nature du référent linguistique.

On trouvera une description plus complète de ces proformes dans [Sabatier 1979, 1984].

"Ainsi"

L'adverbe "ainsi" dans des exemples comme:

Ainsi vécut Max.

C'est ainsi que l'histoire se termine.

a une valeur anaphorique. Les expressions "de cette manière", "de cette façon", etc. sont des paraphrases possibles. Le référent linguistique de "ainsi" peut être la phrase précédente ou l'ensemble du discours précédent. Le référent linguistique peut suivre directement "ainsi" lorsqu'il s'agit par exemple d'un discours rapporté, comme dans:

Max conclut ainsi: "On ne m'y reprendra plus".

"Tel" (pour "de ce type")

Précédées d'un article défini et suivies d'un nom, les formes "tel", "tels", "telle" et "telles" ont une valeur anaphorique, comme dans:

(Max a acheté une voiture de course)
Une telle voiture dépasse le 200 km heure.

Le référent linguistique précède l'anaphore. L'anaphore peut avoir pour référent linguistique le modifieur (adjectif, complément de nom, relative, etc.) d'un groupe nominal précédent, comme ci-dessus. Le nom qui suit la proforme "tel" et celui qui précède le modifieur du référent sont identiques.

Le référent linguistique peut être aussi un groupe nominal complet, comme dans:

(Max a acheté une Ferrari)
Un tel véhicule dépasse le 200 km heure.

Le nom qui suit la proforme en "tel" est alors un classifieur du groupe nominal auquel la proforme renvoie.

"Tel" peut renvoyer à l'action décrite par la phrase précédente, comme dans:

(Max a acheté une Ferrari)
Un tel achat doit être fêté.

Le nom qui suit la forme en "tel" peut être une nominalisation du verbe de la phrase à laquelle renvoie la proforme, comme ci-dessus. Dans ce cas, la résolution de l'anaphore ne pose pas de problème particulier.

Le nom qui suit l'anaphore peut être un nom classifieur de l'action décrite par la phrase précédente, comme dans:

(Max a coulé le bateau de Luc)

Un tel sabotage
  Un tel acte est condamnable
Une telle action
  Un tel geste

Dans ces cas, la résolution de l'anaphore nécessite une analyse plus complexe.

ELLIPSES

Nous allons maintenant donner les principales constructions dans lesquelles se manifestent des ellipses, principalement dans le cadre:

- de phrases coordonnées,
- de propositions comparatives,
- et de subordonnées à valeur interrogative.

Dans la mesure où leur reconnaissance et leur résolution ne posent pas de problèmes particuliers, nous laissons de côté les cas d'ellipses apparaissant dans les réponses à des questions. Comme par exemple:

Qui a vu Luc ?

Max [].

([] = a vu Luc)

Ellipses nominales

Par ellipses nominales, nous désignons les ellipses affectant les groupes nominaux.

Le groupe nominal sujet d'une phrase coordonnée peut faire l'objet d'une ellipse. Exemple:

Max estime Luc et [] l'encourage.

([] = Max)

Deux structures de la forme:

DETERMINANT NOM MODIFIEUR

peuvent être coordonnées, et une ellipse du nom est possible dans la seconde structure, comme le montrent les exemples suivants:

Une voiture rouge et une [] verte

([] = voiture)

Trois litres de vin et deux [] de bière.

([] = litres)

Un enfant qui rit et un [] qui pleure

([] = enfant)

Certaines contraintes existent sur la nature du déterminant et du modifieur du nom en cas d'ellipse. Si l'exemple suivant est correct:

La voiture rouge et la [] verte

les deux suivants sont incorrects:

* Le litre de vin et le [] de bière.

* Le bambin qui rit et le [] qui pleure.

Si l'article est défini, une ellipse du nom est possible à condition que le modifieur soit un adjectif.

Dans une phrase coordonnée, le nom (et son mofificateur) d'un groupe nominal peut faire l'objet d'une ellipse lorsque ce nom est précédé d'un déterminant à valeur nominale et exprimant une partition avec un groupe nominal précédent. Exemple:

Max a donné la moyenne à trois étudiants et il a recalé les autres [].

([] = étudiants)

Ellipses verbales

Par ellipse verbale, nous désignons les ellipses d'entités linguistiques comprenant au moins une entité de type verbal.

Dans le cadre de structures coordonnées, c'est à dire de la forme:

PHRASE1 CONJ PHRASE2 ... CONJ PHRASEn

le verbe de PHRASEi (i>1) peut faire l'objet d'une ellipse si sa structure autorise et présente un complément de ce verbe. Exemples:

Max mange une pomme, Luc [] une poire et Guy [] une orange.

([] = mange)

Max dépend de Luc et Bob [] de Guy.

([] = dépend)

Le complément du verbe sous ellipse peut être une complétive, comme dans:

Max croit que Bob dit la vérité, et Bob [] que Guy ment.

([] = croit)

Le complément du verbe peut être une infinitive, comme dans:

Max essaye de rire, et Luc [] de pleurer

([] = essaye)

L'ellipse semble inappropriéee lorsque l'infinitive n'est pas précédée d'une préposition, comme le suggère l'exemple incorrect suivant:

* Max pense partir, et Luc [] rester

Toujours dans le cadre de phrases coordonnées, une ellipse du verbe de la proposition principale et de celui de la proposition infinitive est possible. Exemple:

Max veut manger une pomme et Luc [] une poire.

([] = veut manger)

Une ellipse de l'infinitive objet de la phrase coordonnée est possible, comme dans:

Max a demandé à Luc de démissionner mais Luc a refusé [].

Luc a refusé [] quand Max lui a demandé de démissionner.

([] = de démissionner)

L'ellipse peut recouvrir le verbe et un complément de phrase, comme dans:

Max mange souvent une pomme et Luc [] une poire.

([] = mange souvent)

L'ellipse d'un verbe intransitif est possible dans le cas de la présence d'un complément de phrase. Exemple:

Max ment souvent mais Luc [] jamais.

([] = ment)

Le verbe et les particules négatives qui lui sont attachées peuvent être sous ellipse, comme dans:

Max n'aime pas les pommes et Luc [] les poires.

([] = n'aime pas)

Si l'exemple précédent est correct, le suivant paraît peu acceptable:

Max ne mange pas de pomme et Luc [] de poire.

On dira plutot:

Max ne mange pas de pomme, ni Luc [] de poire.

Il semblerait que la conjonction "ni" exprimant une négation est préférable pour coordonner deux propositions négatives quand la seconde contient une ellipse du verbe dont le complément est quantifié existentiellement.

L'exemple incorrect suivant:

* Max donne une pomme à Luc, et Bob [] à Guy

([] = donne une pomme)

laisse entendre qu'une ellipse du verbe et d'un de ses compléments est impossible même si ce verbe admet et présente dans la phrase un autre complément. Ce type d'ellipse est pourtant possible dans le cas où le verbe autorise un complément, et où le complément présent dans la proposition coordonnée est un complément de phrase. Exemple:

Max mange une pomme au petit déjeuner, et Luc [] au diner.

([] = mange une pomme)

L'analyse de l'exemple suivant:

Max a demandé Léa en mariage, et Luc [] Evy.

([] = a demandé en mariage)

ne doit pas être confondue avec celle des précédents. Cet exemple met ici en jeu une expression figée, à savoir:

demander quelqu'un en mariage

Dans cette expression le verbe est "demander en mariage" plutôt que "demander".

Le verbe et son sujet peuvent faire l'objet d'une ellipse, en particulier dans les structures coordonnées. Exemples:

Max mange une pomme et [] une poire.

([] = Max mange)

Max a mangé une pomme et [] une poire.

([] = Max a mangé)

Une ellipse de l'auxiliaire du verbe et de son sujet est possible, comme dans:

Max a pelé une pomme et [] mangé une poire.

([] = Max a)

Le nom d'un groupe nominal sujet et son verbe peuvent faire l'objet d'une ellipse. Exemples:

La voiture rouge appartient à Max, et la [1] bleue [2] à Luc.

([1] = voiture, [2] = appartient)

Certaines personnes préfèrent le café, d'autres [] le thé.

([] = personnes préfèrent)

40 % des électeurs ont voté pour Max et 30 % [] pour Luc.

([] = des électeurs ont voté)

L'ellipse peut affecter le nom du groupe nominal sujet, le verbe, mais aussi les compléments de ce verbe, comme dans:

30 étudiants ont réussi leurs examens en juin, et
20 [] en septembre.

([] = étudiants ont réussi leurs examens)

L'exemple suivant:

Max a mangé trois pommes rouges, et Luc [1] deux [2].

([1] = a mangé, [2] = pommes rouges)

montre que dans certaines conditions une ellipse du verbe et du nom (et de son modifieur) d'un groupe nominal complément est possible. En particulier lorsque le déterminant des groupes nominaux compléments exprime une cardinalité.

Les constructions exprimant une comparaison autorisent certaines ellipses. Par exemple, avec la conjonction "comme" une ellipse du verbe et de ses compléments est possible. Exemple:

Max chante comme un oiseau [].

([] = chante)

L'ellipse peut précéder son référent linguistique, comme dans:

Comme Max [], Luc connaît la vérité.

([] = connaît la vérité)

Les comparatives de la forme:

[plus,moins,autant,davantage, etc.] ... que ...

autorisent une ellipse du groupe verbal, comme dans:

Max apprécie davantage le thé que Luc [].

([] = apprécie le thé)

ou du sujet et du verbe, comme dans:

Max apprécie davantage le thé que [] le café.

([] = Max apprécie)

Ellipses de phrases

Certaines propositions subordonnées peuvent être décrites en termes de structures elliptiques. Comme par exemple:

Max viendra mais Luc ne sait pas quand []

([] = Max viendra)

Il s'agit des subordonnées se terminant par une proforme interrogative comme:

qui quoi pourquoi quand comment où

L'ellipse renvoie alors la proposition principale.

Reconstruction de l'ellipse

Si la syntaxe des constructions elliptiques n'a pas fait l'objet d'études à vocation exhaustive, de nombreuses théories ont vu le jour, en particulier dans le cadre de la grammaire générative, où il faut citer en particulier les travaux de [Ross 1967, 1970], [Jackendoff 1971], [Kuno 1976], [Hankamer et Sag 1976], [Williams 1977], et de [Sag 1977].

A la différence d'une proforme, la distance qui sépare une ellipse de son référent linguistique est limitée. Le référent se trouve en général dans la même phrase (ou proposition) ou dans la phrase précédente ou qui suit. L'exemple suivant illustre ce principe:

Max pèle une pomme, Luc mange une poire, et Léo [] une orange.

Le référent linguistique de l'ellipse est "mange" et non "pèle".

Dans des structures coordonnées, la prise en compte des contraintes générales sur la nature des éléments coordonnables peut faciliter la localisation d'une ellipse et la recherche de son référent linguistique.

Le référent linguistique localisé, il reste à en extraire les informations pertinentes pour reconstruire l'ellipse en vue de l'interprétation de la proposition la contenant. En effet le référent linguistique d'une ellipse ne peut pas toujours être substitué tel quel à cette dernière. Par exemple, la version non elliptique de la phrase:

Les enfants ont mangé les pommes, et Luc [] les poires.

est la phrase:

Les enfants ont mangé les pommes, et Luc a mangé les poires.

et non la phrase:

* Les enfants ont mangé les pommes, et Luc ont mangé les poires.

La reconstruction de l'ellipse s'effectue sur la base des traits lexicaux, syntaxiques et sémantiques associés au référent linguistique. Ces traits sont repris et ajustés selon le contexte où figure l'ellipse.

ANALYSE ET SYNTHESE D'ANAPHORES

Les pronoms relatifs et les proformes interrogatives sont couramment traitées par les systèmes d'analyse ou de synthèse du langage naturel. Ces types d'anaphore mis à part, il faut souligner que c'est la résolution des pronoms qui a fait l'objet du plus grand nombre d'études et de réalisations. Cet état de fait est sans doute dû 1) à la relative simplicité de la localisation morphologique et syntaxique des pronoms (comparée à celle beaucoup plus complexe des ellipses, par exemple), et 2) à l'homogénéité du référent linguistique d'un pronom: une entité du type groupe nominal.

Parmi les systèmes analysant des pronoms, on peut citer en particulier les suivants:

- STUDENT [Bobrow 1964]
- SHDRLU [Winograd 1972]
- LUNAR [Woods et al. 1972]
- Le système décrit dans [Colmerauer et al. 1973]
- ROBOT [L.Harris 1977]

Diverses approches pour la résolution des anaphores ont été proposées à travers les systèmes réalisés jusqu'ici. Ces approches varient quant aux méthodes suppléant les informations morphologiques et syntaxiques nécessaires mais insuffisantes pour résoudre la plupart des formes anaphoriques.

La philosophie générale de ces méthodes consiste à avoir recours à des connaissances extra-linguistiques (ou "pragmatiques") propres au domaine de l'application du système. Ces connaissances sont représentées et activées sous diverses formes: représentation logique et inférence [Colmerauer et al. 1973], représentation de type "Frames" [Minsky 1975], de type "Scripts" [Schank 1975] ou de type "sémantique préférentielle" [Wilks 1973].

L'exemple suivant, adapté de [Hirst 1981]:

Max a invité Luc à déjeuner. Il lui a servi un cassoulet.

illustre la problématique. Les pronoms "il" et "lui" ne peuvent être résolus au moyen des seules données lexicales et syntaxiques. La seule information pertinente est que "il" et "lui" ne peuvent pas avoir le même référent linguistique. Une connaissance sur les coutumes et les usages en matière de reception est nécessaire pour associer au pronom "il" le référent linguistique

"Max" et "Luc" à "lui".

Ici cette connaissance pourrait être formulée au moyen de la règle:

Si X invite Y à déjeuner,
X sert (offre, propose, donne, etc.) quelque chose à manger à Y

Dans le cadre de la génération automatique de textes, la synthèse d'anaphores pose plusieurs problèmes intéressants. En particulier les problèmes connexes concernant le caractère obligatoire ou facultatif de la production d'une anaphore et la synthèse de textes non ambigus. Si cette problématique semble mineure dans le cas de la synthèse des ellipses (Il semble que l'ellipse n'ait jamais un caractère obligatoire, et qu'elle engendre peu d'ambiguïté), elle reste majeure dans le cas de la synthèse de proformes.

Un pronom ne peut être produit s'il engendre une ambiguïté. Comme dans:

Max a vu Luc. Il était heureux.

Il en va de même, si son référent linguistique est trop distant dans le texte.

Sa production est obligatoire dans certains cas. Comme dans les cas de coréférence, que nous avons vus plus haut. Exemples:

Max se regarde parler

(et non: Max regarde Max parler)

Max apprécie Luc pour son courage

(et non: Max apprécie Luc pour le courage de Luc)

Max a dit de Luc qu'il était sympathique

(meilleur que: Max a dit de Luc que Luc était sympathique)

Comme le montrent les résultats présentés dans [Danlos 1985], la synthèse de textes constitue un champ d'expériences et d'applications de choix pour le traitement des anaphores en général.

RESOLUTION D'ANAPHORES DANS LES SYSTEMES DIALOGUES, ORBI ET INTERFACILE

Le traitement des anaphores a fait l'objet d'une attention particulière dans les systèmes DIALOGUES [Sabatier 1980] , ORBI [Oliveira, Pereira et Sabatier 1982] et INTERFACILE [Mathieu et Sabatier 1985].

DIALOGUES rend compte de phrase à phrase des ellipses du verbe (1), du sujet et du verbe (2), du verbe et d'un de ses compléments (3) et des proformes du type "aussi" mises pour un groupe verbal (4) :

(1) Milou appartient à Max.
Et Médor à Luc.

(2) Milou appartient à Max.
Et à Luc.

(3) Max donne Milou à Luc.
Et Bob Okapi.

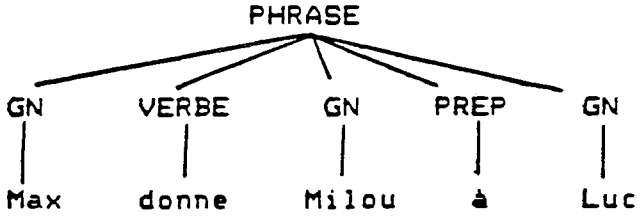
(4) Milou appartient à Max.
Okapi aussi.

Les structures de phrases acceptées par DIALOGUES sont très simples. Cette simplicité autorise une généralisation dans le traitement des anaphores.

Pour la phrase:

Max donne Milou à Luc.

le système produira la structure de surface:



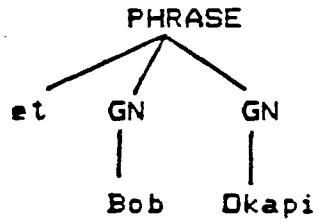
puis la représentation interne normalisée suivante:

< declarer(oui) , <donner,Max,Milou,Luc> >

Pour la phrase:

Et Bob Okapi.

la structure de surface sera:



Associée à cette analyse, deux représentations internes normalisées seront produites:

< x , <v,Bob,Okapi> > (1ère représentation)

< x , <v,Bob,Okapi,c> > (2ème représentation)

où x, v et c sont des variables.

La première représentation rend compte d'une ellipse du verbe (v), la seconde d'une ellipse du verbe (v) et d'un de ses compléments (c).

La valeur du type (x) de la phrase (déclarative négative ou positive, question, ordre) est alors inconnue.

L'interprétation de l'ellipse est résolue par un algorithme d'unification particulier opérant sur la représentation interne de la phrase précédente dans le dialogue, et sur celle(s) de la phrase elliptique.

En reprenant notre exemple, si on a le dialogue suivant:

Max donne Milou à Luc.
Et Bob Okapi.

l'opération d'unification retiendra pour la phrase elliptique la seconde représentation interne. Le résultat sera la représentation interne sans variable:

< declarer(oui) , <donner,Bob,Okapi,Luc> >

Cette nouvelle représentation interne sera conservée au cas où la prochaine phrase du dialogue serait elliptique, comme dans:

Max donne Milou à Luc.
Et Bob Okapi.

Et Léo Médor.

Le principe est le même pour ce qui est du traitement des proformes "aussi", "non", "non plus", "si", "oui", "pas". L'algorithme d'unification diffère dans la mesure où la proforme peut exprimer une quantification (positive ou négative) différente, comme dans:

Max est une personne.
Okapi non.

L'algorithme permet aussi de signaler les discours incohérents, comme le suivant:

Max est une personne.
Okapi non plus.

Le système ORBI a une couverture linguistique sans commune mesure avec celle de DIALOGUES. De l'ordre de 75 constructions syntaxiques, et de 150 mots environ. ORBI rend compte d'ellipses au sein d'une même question, comme par exemple:

Est-ce que chaque point dont le facteur résistance à l'incendie est inférieur à 2 ou dont le descripteur risque d'érosion est supérieur à 3 ne possède aucune aptitude supérieure à 4 ?

ou de question à question, comme par exemple:

Quels descripteurs de l'aptitude Agriculture Intensive du point 56,78 sont inférieurs à 3 ?

(...)

Et du point 67,78 ?

La méthode retenue ici pour résoudre les ellipses diffère de celle que nous avions mis au point dans le système précédent. Les généralisations sont beaucoup plus difficiles à capter lorsque la couverture linguistique de l'application est importante, comme c'est le cas dans ORBI.

Le principe est le suivant. Lorsqu'une question non elliptique a été analysée correctement, ORBI mémorise certains constituants (et les suites de mots associés) qui pourraient faire l'objet d'une ellipse dans la question suivante du dialogue.

Par exemple, après l'analyse de la question:

Quels descripteurs de l'aptitude Agriculture Intensive du

point 56,78 sont inférieurs à 3 ?

ORBI mémorisera les constituants et les suites de mots associés:

GUEL NOM = quels descripteurs
de GN = de l'aptitude Agriculture Intensive
de GN = de l'aptitude
NP = Agriculture Intensive
de GN = du point 56,78
GVERBAL = sont inférieurs à 3

Si l'utilisateur pose maintenant la question suivante:

Et du point 12,95 ?

cette question est prise en compte par une grammaire des constructions elliptiques. Cette grammaire opère à partir de symboles et de constitutants généraux. Elle a pour fonction de synthétiser une question non elliptique à partir de la question elliptique et des constituants mémorisés lors de l'analyse de la question précédente. La question synthétisée, celle-ci sera alors analysée comme une question non elliptique.

Par exemple, à partir de la question:

Et du point 12,95 ?

la question suivante sera synthétisée:

Quels descripteurs de l'aptitude Agriculture Intensive du point 12,95 sont inférieurs à 3 ?

puis analysée.

La synthèse ne rend pas compte des accords en genre et en nombre. Lorsqu'une question synthétisée est analysée comme une question non elliptique, les tests d'accord sont suspendus.

L'une des caractéristiques du système INTERFACILE est de rendre compte d'un sous-ensemble des proformes du français. Les pronoms personnels compléments:

le la l' les lui elle leur en y

les possessifs:

son sa ses

les démonstratifs:

celui-ci celle-ci ceux-ci celles-ci

et les groupes nominaux à valeur démonstrative:

ce dernier cette dernière ces derniers ces dernières

Voici par exemple une suite de questions posées à INTERFACILE:

Comment créer un fichier ?

Comment en faire la copie ?

Comment l'imprimer ?

Comment ajouter son contenu à un fichier ?

Comment renommer ce dernier ?

Comment modifier le contenu de celui-ci ?

Comment envoyer un message à des utilisateurs ?

Comment dialoguer avec eux ?

Comment le système procède-t-il pour résoudre ces proformes ?

Les référents linguistiques de ces proformes sont des groupes nominaux. Après avoir répondu à une question, INTERFACILE mémorise les analyses des groupes nominaux des arguments du verbe de la question. Par exemple, pour la question:

Comment recopier la version d'un fichier dans un catalogue ?

les analyses des groupes nominaux suivants:

la versipn d'un fichier

un catalogue

seront conservées.

Si la prochaine question est:

Comment la renommer ?

le référent linguistique du pronom "la" sera recherché à partir des analyses précédemment enregistrés. Le pronom "la" ayant le trait féminin, c'est le groupe nominal de même genre, "la version d'un fichier", qui sera retenu comme référent linguistique.

S'il y a une ambiguïté, c'est à dire si plusieurs référents sont

linguistiquement possibles, INTERFACILE en fait part à l'utilisateur et lui demande son avis. Voici un exemple d'un tel échange:

comment recopier un fichier dans un catalogue ?

- tapez:

COPY nomfichier.type;version [nomcatalogue]

comment l'effacer ?

- le pronom "l'" renvoie à:

(1) fichier ?

(2) catalogue ?

- votre choix ? tapez le numero correspondant !

2

- tapez:

DELETE [nomcatalogue]

Si le système ne trouve pas de référent possible pour une proforme donnée, il le signale à l'utilisateur, comme dans l'échange suivant:

comment créer un catalogue ?

- tapez:

CREATE/DIRECTORY [nomcatalogue]

comment la renommer ?

- le pronom "la" renvoie à quoi ?

Les règles de résolution des proformes dans le système INTERFACILE reposent sur les seules données morphologiques, lexicales et syntaxiques. Cette façon de procéder assure un bon degrès de portabilité du système vers d'autres domaines d'application.

La résolution des ambiguïtés relève toujours de la sémantique du domaine de l'application. INTERFACILE a dans ce cas recours à l'avis de l'utilisateur, à travers un court échange. Cette façon de faire peut géner un utilisateur qui maîtrise parfaitement le

domaine d'application et pour lequel il n'y a pas d'ambiguïté possible.

Cette façon de faire est recommandée dans le cas d'utilisateurs non spécialistes du domaine (auxquels s'adresse INTERFACILE) afin de contrer tout présupposé erroné. Comme le montre l'échange suivant:

comment recopier un fichier dans un catalogue ?

- tapez:

COPY nomfichier.type [nomcatalogue]

comment renommer sa version ?

- version de ...

(1) fichier ?
(2) catalogue ?

- votre choix, tapez le numero correspondant !

2

- INTERFACILE parle plutot de:

version d'un fichier

A la différence des autres proformes, la proforme "ce dernier" (et ses variantes morphologiques) limite le choix du référent linguistique au dernier groupe nominal (et qui peut être le seul) énoncé dans la question précédente. Il n'y a pas d'ambiguïté possible.

La résolution des autres proformes obéit aux règles données plus haut.

Une fois qu'une anaphore a été résolue, son analyse est remplacée par celle de son référent linguistique. Par exemple, si on a l'échange suivant:

Comment imprimer un fichier ?

(...)

Comment le renommer ?

une fois le pronom "le" résolu, l'analyse de "le" est remplacée par celle de "un fichier". Cela afin de faciliter la résolution de tout pronom pouvant apparaître dans les question suivantes.

Comme par exemple:

Comment l'imprimer ?

CONCLUSION

La compréhension des constructions anaphoriques est un atout fondamental pour une interface en langage naturel, et décisif par rapport au choix d'autres types d'interface (formel, graphique, etc.)

Les travaux que nous avons menés dans le domaine des anaphores nous ont montré la complexité du phénomène linguistique. Les résultats obtenus sont élémentaires compte tenu de la réalité linguistique qui reste encore à décrire pour une compréhension automatique plus large.

CHAPITRE V

DETECTION ET TRAITEMENT DES ERREURS

Introduction

Types d'erreur

Messages d'erreurs

Traitement des erreurs : deux approches

Erreurs lexicales

Erreurs syntaxiques

Erreurs sémantiques

Traitement des erreurs et conception de l'interface

Composition de phrases assistée

Programmation en Prolog

Conclusion

INTRODUCTION

Comparée à la richesse des structures syntaxiques et des usages de toute langue naturelle, la compétence linguistique des interfaces reste encore limitée. Si on a recours à de tels systèmes pour obtenir une information que l'on ignore, on en sait généralement plus que l'interface elle-même sur les nombreuses façons dont nos requêtes pourraient être formulées.

Nous pensons que la qualité d'une interface en langage naturel ne doit pas seulement être appréciée d'après l'étendue de sa couverture linguistique, mais aussi d'après la qualité des procédures qu'elle met en oeuvre face aux phrases qui sont soit incorrectes, soit extragrammaticales, c'est à dire correctes mais non attendues par l'interface.

Dans le cadre d'une interface en langage naturel, on entend par détection et traitement d'erreurs l'ensemble des procédures et des techniques mises en oeuvre pour rendre compte des incompréhensions de l'interface devant certains propos de l'utilisateur : mots mal orthographiés (ou mal prononcés) ou inconnus de l'interface, fautes d'accord, emploi par l'utilisateur de structures correctes mais inconnues de l'interface, ou encore, de phrases faisant apparaître des malentendus sur le domaine d'application de l'interface.

Sachant que la détection et le traitement des erreurs est une tâche compliquée dans le cadre pourtant "restreint" des langages de programmation (des syntaxes limitées et des sémantiques rigoureusement définies), on peut déjà apprécier les difficultés pour doter les interfaces en langage naturel de telles capacités.

Dans ce chapitre, nous rappelons les différents types d'erreur qui peuvent se produire dans le cadre de la communication homme-machine. Nous présentons les différentes techniques de traitement, et leurs incidences sur la conception de l'interface elle-même. Nous montrons l'intérêt d'une approche où l'aide à la (re)formulation est privilégiée. Enfin nous précisons en quoi la programmation en Prolog se prête à la détection, à la récupération et aux traitements des erreurs.

TYPES D'ERREURS

Vue de l'extérieur, une interface peut ne pas comprendre la phrase d'un utilisateur pour deux raisons majeures, soit parce que sa phrase contient des éléments incorrects, soit parce que certains éléments, bien que corrects, sont inconnus de l'interface, et dépassent donc sa compétence.

La distinction entre un élément incorrect ou inconnu n'est pas directement perceptible par une interface (et par tout automate, d'une façon générale). Le terme technique d'"erreurs" recouvre ici ces deux cas de figure.

On peut distinguer trois grands types d'erreurs. Ces erreurs peuvent concerner le vocabulaire, la syntaxe ou la sémantique de l'interface. On parlera alors d'erreurs lexicales, d'erreurs syntaxiques et d'erreurs sémantiques.

MESSAGES D'ERREURS

D'une façon générale, les messages d'erreurs doivent être dépourvus de termes techniques étrangers au domaine de l'application. Et cela en particulier dans les cas de messages portant sur l'aspect linguistique du dialogue. On peut signaler à l'utilisateur que le complément qu'il a employé avec tel verbe n'est pas celui attendu par l'interface, ou qu'il a fait une faute d'accord entre tel mot et tel mot. L'emploi de ces notions relève encore du sens commun. Mais il ne faut pas oublier que l'on a recours à de tels systèmes pour obtenir un renseignement ou un service précis, et non pour recevoir une leçon de grammaire. N'oublions pas que, verbalement, l'utilisateur est généralement plus compétent que l'interface.

Les messages d'erreurs doivent être formulés autant que possible en langage naturel et ne doivent pas tromper l'utilisateur sur les réelles capacités linguistiques de l'interface. Il faut distinguer ici deux types de messages :

- ceux qui sont pré-enregistrés et qui contiennent généralement des mots que l'interface ne pourrait pas comprendre si l'utilisateur venait à les employer dans le dialogue,

- ceux qui sont totalement synthétisés par l'interface à partir des données linguistiques (lexique, syntaxe et sémantique) utilisées pour l'analyse. Dans ce cas, les mots et les constructions de ces messages peuvent être repris par l'utilisateur dans les dialogues avec l'interface.

Ce dernier type de message est sans aucun doute le plus intéressant mais aussi le plus coûteux à mettre en oeuvre. Dans la pratique, il serait pertinent que l'utilisateur puisse reconnaître dans les messages les mots et les constructions qu'il pourrait réutiliser. Dans le cadre d'un dialogue écrit, l'usage d'une typographie spéciale (mots en majuscule vs. mots en minuscule) pourrait faciliter la distinction.

TRAITEMENT DES ERREURS : DEUX APPROCHES

Une fois qu'une erreur (lexicale, syntaxique ou sémantique) est détectée, deux types de traitement sont possibles, et correspondent en fait à deux approches du problème. Ces deux approches ne sont pas incompatibles, et une interface conviviale devrait mêler astucieusement les deux.

La première approche consiste à essayer d'accepter les questions de l'utilisateur qui dévient par rapport aux attentes de l'interface en essayant de les transformer dans des expressions que l'interface pourra comprendre.

Cette première approche présente deux inconvénients, que l'on peut formuler ainsi :

(1) Il n'y a aucune garantie que l'interprétation de la question déviante fournie par l'interface corresponde à l'intention de l'utilisateur lorsqu'il a posé sa question.

(2) Il y aura toujours une limite au degré de déviation qu'un système pourra tolérer avant d'analyser la question d'un utilisateur. Si la transformation de questions déviantes dans des expressions analysables est réalisée automatiquement, l'utilisateur ne saura jamais où se trouvent les limites, et pourra croire que le système peut comprendre tout ce que l'utilisateur demande. La distinction entre les phrases que l'interface interprètera correctement et celles qu'elle rejettera complètement lui semblera purement arbitraire.

La seconde approche consiste à conduire l'utilisateur à ajuster ses compétences linguistiques à celle de l'interface, et cela de la façon la moins contraignante possible. Nous allons discuter maintenant des différents types d'erreurs sous l'angle des deux approches de leur traitement.

ERREURS LEXICALES

On peut distinguer divers types d'erreurs lexicales. Ces erreurs peuvent être dues à :

(1) des fautes de frappe
(2) des fautes d'orthographe
(3) des fautes de prononciation
(4) l'emploi de mots nouveaux

Les erreurs de type (1) et (2) sont liées à l'utilisation d'un clavier. On rencontre des erreurs du type (3) dans le cadre d'un dialogue avec entrée vocale. Les erreurs du type (4) sont indépendantes du mode choisi pour communiquer (frappe au clavier ou entrée vocale). Nous nous intéresserons ici aux erreurs de type (1), (2) et (4).

Concernant les fautes de frappe, Damerau [1964] a noté que 80 % des erreurs qui concernent une seule lettre peuvent être réparties dans les quatre cas suivants :

- une lettre est erronée,
  exemple : "fivhier" au lieu de "fichier"

- une lettre manque,
  exemple : "fihier" au lieu de "fichier"

- une lettre bizarre a été insérée,
  exemple : "ficchier" ou "fiuchier" au lieu de "fichier"

- l'ordre de deux lettres adjacentes a été inversé,
  exemple : "fihcier" au lieu de "fichier"

Le traitement de ce type d'erreurs a fait l'ojet d'une attention particulière dans le cadre de la compilation de programmes, et a donné lieu à plusieurs algorithmes plus ou moins efficaces (pour une présentation générale, voir [Peterson 1980], [Durham, Lamb, Saxe 1983]).

En particulier, l'algorithme de Damerau permettant de traiter ces erreurs consiste à sélectionner dans le dictionnaire un sous-ensemble de chaînes et à appliquer une comparaison pour déterminer si la chaîne d'entrée (considérée comme erronée) est ou n'est pas une forme altérée de la chaîne sélectionnée dans le dictionnaire.

Les chaînes du dictionnaire sont sélectionnées selon le nombre de caractères contenus dans la chaîne incorrecte. Si la chaîne incorrecte contient n caractères, la chaîne sélectionnée dans le dictionnaire contiendra :

- n caractères lorsque l'erreur est du type lettre erronée, ou lettres inversées.

- n-1 caractères dans le cas où la chaîne d'entrée contient un caractère bizarre.

- n+1 caractères dans le cas où un caractère manque dans la chaîne d'entrée.

C'est cette technique que nous avons retenue dans le système INTERFACILE, illustrée par le dialogue suivant :

> Comment envoyre unn mesage à un utikisateur ?

("envoyre"	devient	"envoyer")
("unn"	devient	"un")
("mesage"	devient	"message")
("utikisateur"	devient	"utilisateur")

- en tapant :
  MAIL

D'autres algorithmes, comme celui présenté dans [Pollock et Zamora 1984], opèrent à partir d'un codage particulier des mots du dictionnaire et du mot considéré comme erroné, et d'un ensemble d'opérations de recherche de similarité sur les codages obtenus.

En ce qui concerne les fautes de frappe, une erreur lexicale peut être aussi due à une segmentation incorrecte. Deux cas de figure sont possibles . A la suite d'une mauvaise frappe, des mots peuvent être accolés, comme "un" et "fichier" dans :

Comment créer unfichier ?

Ou bien, un mot peut être scindé dans des formes séparées par des espaces, comme "fichier" dans :

Comment créer un fic hier ?

Si le traitement de ce type d'erreur est potentiellement réalisable, dans bien des cas, il conduit rapidement à une explosion combinatoire, comme le suggère tout traitement de l'exemple suivant :

Comment créerunfic hier ?

En ce qui concerne les fautes d'orthographe, certaines peuvent être assimilées à des fautes de frappe, et traitées comme telles. Il va ainsi des fautes portant sur le redoublement (ou non) de

certaines lettres. Comme par exemple :

"renomer" au lieu de "renommer"
"courrir" au lieu de "courir"

De même pour certaines fautes concernant un caractère erroné, comme par exemple :

"essenciel" au lieu de "essentiel"
"différance" au lieu de "différence"
"enporter" au lieu de "emporter"

Ou dans le cas d'erreurs d'accentuation, comme par exemple :

"creer" au lieu de "créer"
"érreur" au lieu de "erreur"
"évènement" au lieu de "événement"
"fête" au lieu de "fête"
"ambigue" au lieu de "ambiguë"

Les fautes d'orthographe concernant l'omission d'un caractère peuvent être aussi assimilées à des fautes de frappe, et traitées comme telles, comme par exemple :

"notament" au lieu de "notamment"
"picine" au lieu de "piscine"
"cataloge" au lieu de "catalogue"

Les fautes d'orthographe portant sur plusieurs lettres d'un même mot nécessitent un traitement particulier. Comme par exemple les fautes suivantes :

"connection" au lieu de "connexion"
"apparament" au lieu de "apparemment"
"agraphe" au lieu de "agrafe"

Ce type de faute est fréquent dans l'orthographe des noms propres. Exemples :

"Méditérané" au lieu de "Méditérranée"
"Rocancour" au lieu de "Rocquencourt"
"Châtenais Mallabrie" au lieu de "Châtenay Malabry"

Dans certains cas, ces erreurs peuvent être traitées sur la base d'un codage phonétique des mots du dictionnaire et du mot erroné. Ainsi, dans les exemples précédents les formes attendues et les formes erronées sont identiques sur la base de leur description phonétique. Préliminaire à ce type de traitement, la transcription graphie-phonétique a donné lieu à l'élaboration de nombreux programmes et dictionnaires phonétiques automatisés, en par-

ticulier pour le français [Bulot 1983] et [Laporte 1987].

D'une façon générale, pour le traitement des erreurs du type (1) et (2), la sélection des chaînes dans le dictionnaire peut être rendue plus fine en tenant compte des contraintes syntaxiques propres au contexte dans lequel apparaît la chaîne incorrecte. Par exemple en reprenant l'exemple précédent, dans la grammaire d'INTERFACILE, l'interrogatif "comment" ne peut être suivi que d'un pronom ou d'un verbe. Cette information syntaxique limiterait l'ensemble des chaînes sélectionnables.

Cette optimisation dans la sélection n'est possible que si l'analyse lexicale est retardée au niveau de l'analyse syntaxique. Ce qui n'est pas le cas dans INTERFACILE, où l'analyse syntaxique s'effectue une fois l'analyse lexicale d'une question terminée.

On peut aussi prendre en compte des contraintes sémantiques pour optimiser la sélection des chaînes dans le dictionnaire. Là aussi les analyses lexicale, syntaxique et sémantique doivent être menées conjointement.

L'efficacité peut être aussi améliorée en optimisant l'organisation des chaînes dans le dictionnaire, comme cela se pratique, en compilation, pour les tables de symboles [Morgan 1970].

D'une façon générale, la demande de validation par l'utilisateur des solutions proposées par l'interprétation est facilement formulable. Si un mot a été mal orthographié (ou mal prononcé), l'interface propose un mot (ou un ensemble de mots) connu se rapprochant du mot erroné. Dans le cas où il s'agit en fait d'un mot nouveau pour le système, le problème reste entier.

En ce qui concerne le traitement des mots nouveaux, une interface peut être facilement dotée d'une procédure permettant, au cours du dialogue, la définition de synonymes. Voici l'extrait d'un échange avec le système LADDER où une telle procédure est mise en oeuvre :

> Define "med" like "mediterranean".

> How many carriers are in the med ?

– 15.

La définition de mots nouveaux qui ne sont pas des synonymes de mots existants nécessite la mise en oeuvre de procédures complexes liées à l'extension du domaine d'application et de sa

sémantique. Toute extension de ce type est d'une façon générale difficilement réalisable dans le cadre d'un simple dialogue avec l'utilisateur.

ERREURS SYNTAXIQUES

L'ensemble des mots d'une phrase formulée par un utilisateur peut être connu de l'interface mais cette dernière peut ne proposer aucune analyse correcte à cause d'erreurs de nature syntaxique.

Par erreurs syntaxiques, nous entendons celles dont la détection nécessite au moins une analyse syntaxique de la phrase où elles se manifestent. On peut distinguer trois types d'erreurs, selon qu'elles sont dues à :

(1) des fautes d'accord
(2) l'emploi de structures incorrectes
(3) l'emploi de structures inconnues

Parmi les erreurs du type (1), celles concernant les accords en genre (masculin vs féminin) et en nombre (singulier vs pluriel) sont les plus fréquentes. Comme dans le dialogue suivant :

> Quel sont les département qui bordent la Principauté de Monaco ?

— Fautes d'accord entre :
"quel" et "sont"
"les" et "département"

La stratégie générale concernant le traitement de ce type d'erreur consiste à signaler l'erreur à l'utilisateur, et à essayer de la corriger automatiquement. La correction peut être effectuée sur la base de règles de priorité que l'on aura définie au préalable, comme les règles suivantes, par exemple :

Si le déterminant et le nom d'un groupe nominal divergent sur leur genre (ou leur nombre), on retiendra comme genre (ou comme nombre) celui du déterminant.

Si un verbe et son sujet divergent sur leur nombre on retiendra comme nombre celui du sujet.

Lorsqu'une seule solution est possible, comme dans l'exemple précédent, la correction automatique ne pose pas de problème particulier. Le cas peut se présenter où plusieurs solutions sont possibles, chacune d'elles conduisant alors à une interprétation différente. L'avis de l'utilisateur est alors nécessaire.

Il est rare qu'un utilisateur maîtrisant une langue donnée emploie des structures incorrectes. L'emploi de structures incorrectes se manifeste généralement dans le cas où l'utilisateur formule ses propos dans une langue qui lui est étrangère, comme l'illustre la question suivante tirée d'un échange avec le système ORBIS :

> What is the planet which diameter is 5000 km ?

Dans ce contexte, le pronom relatif attendu est "whose" et non "which".

Dans la question suivante :

> Which planets Galileo discovered ?

l'utilisateur a vraisemblablement traduit mot à mot sa question du français vers l'anglais. Il a ainsi oublié que la construction interrogative nécessite ici l'emploi de l'auxiliaire "do". La construction attendue est :

> Which planets did Galileo discover ?

Les erreurs syntaxiques les plus fréquentes sont celles du type (3). Elles concernent l'emploi de structures syntaxiques correctes mais inconnues de l'interface. Elles témoignent de la richesse des constructions d'une langue et des problèmes liés à l'énumération de leur combinaison. Voici par exemple, les variantes syntaxiques d'une même requête :

Comment dans un catalogue copier un fichier ?
Comment copier dans un catalogue un fichier ?
Comment copier un fichier dans un catalogue ?

La mobilité de certains constituants comme les compléments de verbe ou les compléments de phrase de type circonstanciel pose quelques problèmes quant à leur énumération et leur analyse.

Dans le cadre d'une grammaire opérant selon le principe classique de réécriture l'énumération des combinaisons ne peut se faire qu'en multipliant le nombre des règles. Le concepteur de la grammaire n'est pas à l'abri de l'oubli de certaines combinaisons et donc du traitement des erreurs syntaxiques de ce type. Dans ce cas, l'idée est de prévoir le plus grand nombre de combinaisons, et d'amener l'utilisateur à s'y conformer en cas d'erreurs. C'est la philosophie retenue dans le système ORBIS.

En cas d'erreur syntaxique (mais aussi lexicale ou sémantique) le système ORBIS fournit à l'utilisateur la liste des mots (ou des classes de mots) attendus après le dernier mot correctement analysé de la phrase (l'analyseur parcourant la phrase de la gauche vers la droite). Comme dans l'échange suivant :

> Quels sont les astres découverts par Galilée ?

- Quels sont les astres ...
  doit se poursuivre par l'un des mots :
  qui ¦ qu' ¦ que ¦ auxquels ¦ desquels ¦
  à ¦ d' ¦ de ¦ au ¦ aux ¦ dont ¦

  Reformulez la fin de votre question :
  Quels sont les astres (à vous) : que Galilée a découverts ?

Io
Europe
Ganymède
Callisto

Le point intéressant de ce système est que les suites de mots fournies sont produites en faisant fonctionner en synthèse la grammaire utilisée aussi pour l'analyse.

Nous avons retenu une stratégie similaire dans le système INTERFACILE pour le traitement des erreurs syntaxiques. Comme l'illustre l'échange suivant:

> Comment envoyer à un utilisateur un message ?

- Pour "envoyer", employez la construction :

  envoyer [quelque chose] à [quelqu'un]

> Comment envoyer un message à un utilisateur ?

- Utilisez la commande :
  MAIL

L'information fournie ici est obtenue à partir des données syntaxiques contenues dans le lexique-grammaire associé à l'interface. Il est consulté comme une banque de données pour fournir une aide à reformulation.

Dans le cadre d'un domaine d'application limité, et d'une couverture linguistique simple, une autre approche pour traiter les erreurs syntaxiques consiste à organiser l'analyse de toute phrase autour de certains mots considérés comme clés : les verbes, les prépositions, les noms, par exemple. L'avantage de cette approche est de permettre une certaine flexibilité au niveau de l'analyse [Hayes et Maroudian 1981], [Carbonell et Hayes 1983]. Cette approche présente cependant trois inconvénients majeurs :

- l'analyse d'une phrase correcte prendra autant de temps que celle d'une phrase déviante,

- pour être efficace cette approche doit opérer sur une grammaire sémantique étroitement liée au domaine d'application,

donc difficilement portable vers d'autres applications,

- cette approche n'est pas viable dans le cadre d'une couverture linguistique un peu complexe (relatives, structures coordonnées, subordonnées, etc).

ERREURS SEMANTIQUES

La phrase d'un utilisateur peut ne contenir aucun mot inconnu, recevoir une analyse syntaxique mais être dépourvue de sens. Le terme d'erreur sémantique recouvre ce cas de figure.

D'une façon générale, les erreurs de nature sémantique concernent les relations entre les différents éléments de la phrase comme :

- les relations entre un nom et ses attributs (adjectifs, compléments du nom, relatives, etc.),
- les relations entre un verbe, son sujet et ses compléments.

La validé de ces relations est entièrement dépendante du domaine d'application.

A titre d'illustration, voici l'extrait d'un dialogue avec INTERFRANCE :

> Quelles sont les populations des fleuves ?

- INTERFRANCE ignore le sens de la construction : population ... de ... fleuve

La requête de l'utilisateur est syntaxiquement correcte mais dépourvue de sens dans l'application concernée. On peut imaginer le cas où cette question pourrait avoir un sens, comme par exemple, dans le cadre d'une banque de données sur la faune aquatique.

Les erreurs sémantiques ne sont pas toujours le lot des utilisateurs néophytes ou occasionels. Les deux questions suivantes ont été posées aux systèmes MICROSIAL et INTERFRANCE par un utilisateur ayant une connaissance du domaine d'application :

> Quelle entité dirige l'ingénieur Lagarde ?

- Lagarde n'est pas ingénieur !

> Quelle est la préfecture de la Garonne ?

- La Garonne n'est pas un département !

Dans ces deux derniers exemples, les erreurs concernent les présuppositions associées aux questions.

Si la détection des erreurs sémantiques ne pose pas de problèmes particuliers, leur traitement reste complexe si on souhaite fournir une réponse coopérative.

Les deux exemples qui suivent sont tirés d'un échange avec le système INTERFACILE :

> Comment supprimer la version d'un catalogue ?

- "version" de quoi? INTERFACILE parle plutôt de :

    version de fichier
    catalogue de fichier

> Comment supprimer un utilisateur ?

- INTERFACILE n'en sait rien mais comment :

    dialoguer avec un utilisateur
    lister un utilisateur

    supprimer un fichier
    supprimer un catalogue

Chaque question de l'utilisateur concerne ici une action (effacer, copier, envoyer, etc.) portant sur des objets (fichier, utilisateur, catalogue, message, etc.). INTERFACILE vérifie tout d'abord la cohérence de chacun des objets de l'action décrite, puis vérifie la cohérence de la relation entre l'action et les objets.

Cette vérification est réalisée à partir de la base de connaissances propre à l'application. Lorsqu'une incohérence se manifeste au niveau d'un objet (cas du premier exemple) ou de l'action (deuxième exemple), INTERFACILE fait part à l'utilisateur des objets cohérents et des actions possibles autour des objets attendus. L'aide est dispensée en consultant, comme précédemment, la base de connaissances, puis en synthétisant les informations déduites de la base. Bien que très simple cette stratégie se révèle satisfaisante dans la majorité des cas.

Dans les questions précédentes aux systèmes MICROSIAL et INTERFRANCE, ces derniers auraient pu, par exemple, compléter leur réponse en disant respectivement que :

- Lagarde n'est pas ingénieur, mais le général Lagarde dirige l'Etat-Major de l'Armée de Terre.

- La Garonne n'est pas un département, mais un fleuve. Toulouse est la préfecture de la Haute-Garonne, Montauban

est la préfecture du Tarn-et-Garonne, et Agen est la préfecture du Lot-et-Garonne.

Un comportement coopératif n'est pas seulement attendu en cas d'erreurs. Une interface doit être capable dans certains cas de fournir une réponse apportant davantage d'information que celle strictement demandée. Par exemple, une question dont la réponse "stricte" est "oui" ou "non" peut être complétée pour explicitation. Comme dans le dialogue suivant :

> Est-ce que le diamètre de la terre est inférieur à celui de la lune ?

— non.
(diamètre de la terre : 12 756 km)
(diamètre de la lune : 3 473 km)

La production de ces dernières réponses nécessite la modélisation d'un comportement coopératif dans le cadre des actes de langage, comme Grice a pu les étudier [Grice 1975]. La stratégie générale consiste à deviner les intentions de l'utilisateur dans le but de les devancer. La mise en oeuvre d'un tel comportement relève du domaine de la génération de plans appliquée au dialogue homme-machine en langage naturel [Allen et Perrault 1980], [Hayes et Reddy 1984].

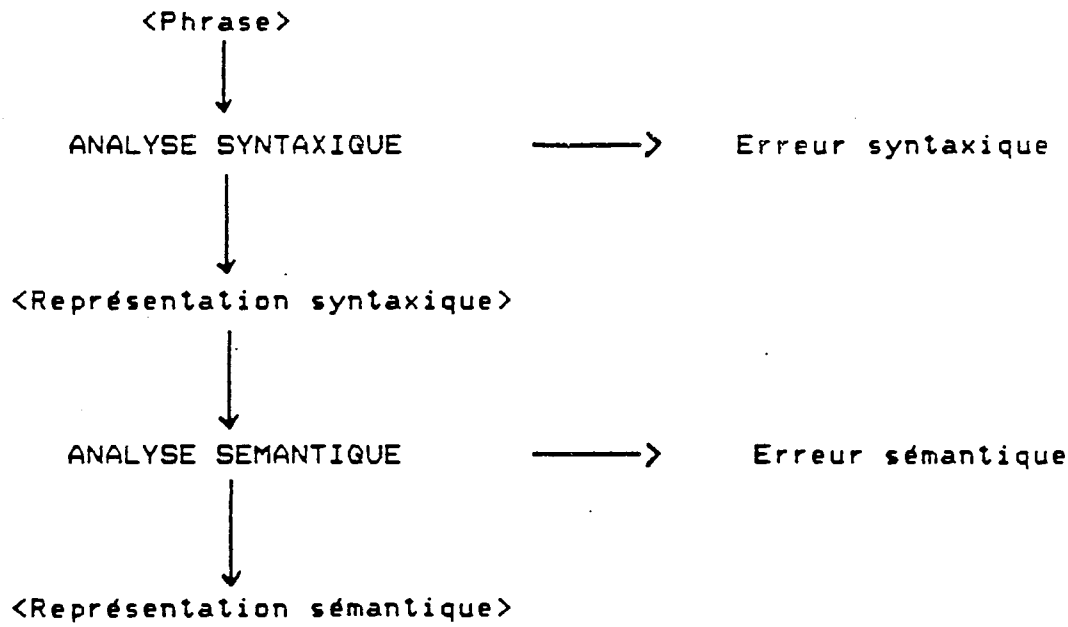
Dans le cadre d'une interface à une banque de données, en cas d'erreurs sémantiques (ou comme complément d'information), des réponses coopératives peuvent être produites à partir des contraintes d'intégrité contenues dans la banque [Janas 1981], [Kaplan 1982], [Gal et Minker 1985].

TRAITEMENT DES ERREURS ET CONCEPTION DE L'INTERFACE

La diversité des erreurs détectées et la qualité de leur traitement dépendent largement de l'architecture de l'interface. Ainsi, le traitement des erreurs doit faire l'objet d'une attention particulière dès la phase de conception d'une interface.

Comme on l'a vu précédemment, pour des raisons de qualité dans les diagnostics à produire et d'efficacité dans la recherche des corrections, les erreurs lexicales gagnent à être détectées et traitées en tenant compte du contexte syntaxique et sémantique où elles se manifestent. L'analyse lexicale d'une phrase doit être donc retardée et intégrée au niveau des analyses syntaxique et sémantique de la phrase, ou menée en parallèle avec ces dernières.

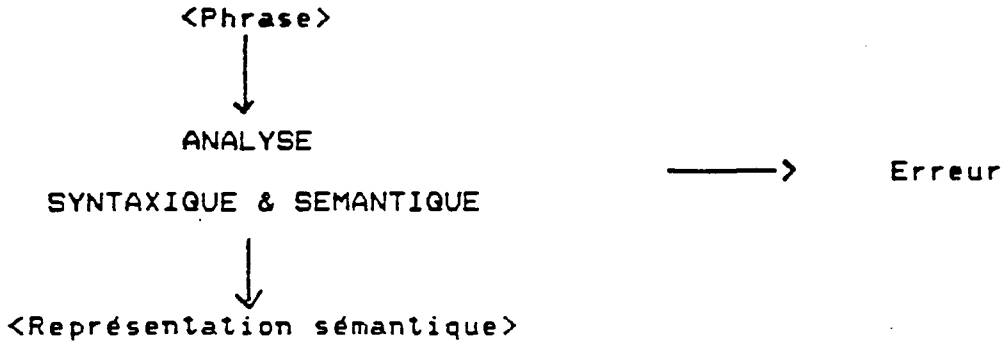
Dans le cas où l'analyse est dirigée par la syntaxe, cette dernière étant suivie par l'analyse sémantique, on a l'architecture suivante :



Avec une telle architecture, les diagnostics portant sur des erreurs syntaxiques ne donneront lieu qu'à des messages proposant des corrections purement syntaxiques. Si l'utilisateur connaît la sémantique du domaine d'application, ces informations lui suffiront. Par contre, si l'utilisateur a une vue partielle ou erronée du domaine d'application, les informations purement syntaxiques fournies seront d'un intérêt limité : il utilisera les constructions syntaxiques attendues sans savoir si elles conduiront à une phrase sémantiquement correcte. Pour le savoir il faudra attendre le résultat de l'analyse sémantique. En cas d'erreur, le système aura produit deux messages l'un sur des

informations syntaxiques, l'autre sur des informations sémantiques.

La conduite en parallèle des analyses syntaxique et sémantique est sans doute la solution à retenir pour fournir des messages d'erreurs les plus informatifs (c'est-à-dire les plus dépendants du domaine de l'application) et des corrections les plus appropriées. On a alors l'architecture suivante :



D'une façon générale, la représentation sémantique d'une phrase consiste en une formule faisant apparaître un ensemble de présuppositions. Par exemple, pour la question suivante :

> Quelle est la préfecture de la Garonne?

on aura pour représentation sémantique une expression comme :

quel(x , si(département(Garonne) , préfecture(x,Garonne)))

où département(Garonne) est la présupposition associée à la question (la nature de la présupposition est ici dépendante du domaine de l'application).

L'évaluation consiste à vérifier la vérité des présuppositions puis à évaluer le reste de la formule pour trouver les éléments de la réponse. Dans le cas où les présuppositions ne sont pas vraies (comme dans l'exemple ci-dessus), la question est absurde, et l'erreur sémantique doit être signalée à l'utilisateur.

Il y a peu de chance qu'un utilisateur ayant une bonne connaissance du domaine d'application fasse des erreurs sémantiques sur la nature des présuppositions, et pose une question comme la précédente. Dans ce cas, l'évaluation d'une phrase peut être rendue plus efficace en supprimant la vérification des présuppositions associées à la requête. Une autre possibilité est de vérifier les présuppositions seulement dans le cas où la réponse à une question est nulle, la réponse pouvant résulter de présuppositions fausses.

S'il est toujours possible de détecter et de traiter des erreurs indépendamment de la stratégie d'analyse choisie, certaines stratégies sont mieux adaptées que d'autres pour réaliser certains types de traitement. Ainsi l'avantage d'une stratégie descendante tient au fait qu'en cas d'erreur, ce qui est déjà analysé constitue une forte hypothèse sur les intentions de l'utilisateur. En guise de correction, le système pourra par exemple proposer (par synthèse) les suites attendues, comme le fait le système ORBIS. A la différence d'une stratégie descendante, une stratégie ascendante est particulièrement adaptée pour réaliser des analyses fragmentaires d'une phrase incorrecte.

Toute analyse sémantique est dans une large mesure dépendante du domaine d'application de l'interface, c'est à dire de la base de connaissances qui lui est associée. Ainsi, la vérification de la cohérence sémantique des requêtes d'un utilisateur ne relève pas entièrement des composants linguistiques de l'interface mais du système de gestion de la base de connaissances. C'est donc à ce niveau que les erreurs sémantiques seront détectées et signalées à l'utilisateur via l'interface.

Dans le cas où le système de gestion de la base de connaissance se comporte comme une boite noire, (comme dans la plupart des systèmes de bases de données existants), la récupération des erreurs et leur interprétation restent limitées. Un traitement efficace des erreurs ne peut se faire qu'en dotant l'interface d'une certaine autonomie. La solution consiste alors à intégrer au niveau de l'interface une partie des connaissances déjà contenues dans la banque de données, comme par exemple les contraintes d'intégrité dans le cas d'une interface pour une banque de données. Outre la redondance d'informations qui est créée en opérant ainsi, l'inconvénient majeur de cette solution réside dans la double mise à jour qu'impose toute modification des connaissances: la mise à jour doit se faire dans la banque de données et dans la base des connaissances propre à l'interface.

COMPOSITION DE PHRASES ASSISTEE

Compte tenu des limites linguistiques et conceptuelles inhérentes à toute interface, nous avons vu en particulier comment les systèmes ORBIS et INTERFACILE pouvaient amener l'utilisateur à reformuler ses requêtes grâce à une synthèse des connaissances.

Dans le cadre de l'interrogation de systèmes de connaissances (banques de données, systèmes-experts, etc.) une approche similaire est généralisable au niveau de la composition même des requêtes. On parlera alors de composition de phrases assistée.

Par exemple, les systèmes NLMenu [Tennant 1984], INGLISH [Phillips et Nicholl 1985] et le système décrit dans [Filgueras et Silva 1986] permettent à l'utilisateur de composer ses requêtes mot après mot à partir des informations synthétisées sur l'écran, et des choix qu'il précise à l'aide d'une souris (ou d'un curseur).

Cette méthode présente de nombreux avantages. En voici quelques uns.

(1) Les erreurs dues aux fautes de frappe ou d'orthographe sont évitées. Si dans l'un des choix possibles, le système demande à l'utilisateur d'entrer un mot au clavier (un nom propre, par exemple, qui n'est pas défini dans le lexique), et si ce dernier tape un mot incorrect ou inconnu, la recherche des corrections par le système est simplifiée car la classe du mot attendu est connue.

(2) Le système comprend toutes les phrases que l'utilisateur compose selon cette méthode. Les erreurs syntaxiques et sémantiques sont évitées.

(3) Les utilisateurs perçoivent facilement et rapidement les limites syntaxique et sémantique de l'interface.

(4) Il n'est pas nécessaire de multiplier dans la grammaire les synonymes et les paraphrases. Il n'est donc pas nécessaire de pouvoir disposer d'importantes ressources en mémoire pour faire tourner de tels systèmes. Des applications sur micro-ordinateurs sont donc envisageables.

Cette technique présente un inconvénient majeur:

Il est difficile de rendre compte des processus anaphoriques (pronoms, ellipses, etc.) dans le cadre de cette méthode. En particulier les références au sein d'une même phrase ou de phrase à phrase font appel à des procédures qui dépassent la simple prédiction et synthèse du mot à mot. D'autre part, pour une simple phrase, une quantité importante d'anaphores risque d'être produite et proposée à l'utilisateur.

La détection d'une erreur (lexicale, syntaxique ou sémantique) doit pouvoir conduire à la fois (1) à la suspension temporaire ou définitive du cours de la boucle générale d'analyse, et (2) à la récupération de l'erreur.

La stratégie de base de Prolog étant le non déterminisme avec retour en arrière, la récupération des points d'erreurs ne pouvait se faire dans les premières versions du langage que par des ajouts dynamiques de clauses. Ainsi, lorsqu'une erreur était détectée, une clause spécifiant la nature de l'erreur était ajoutée dans le programme. Avant de passer à l'étape suivante de l'analyse, on vérifiait si une telle clause existait ou non. Cette méthode présentait l'inconvénient majeur de multiplier les ajouts (et les suppressions) de clauses, opération fort coûteuse en espace mémoire et en temps d'exécution.

En ayant recours aux prédicats évaluables "block" et "block-exit", la suspension d'une boucle de résolution et la récupération des erreurs sont facilement spécifiables et gérables en Prolog II. Nous rappelons ici la définition de ces deux prédicats:

```
block(p,b)
```

Pour résoudre block(p,b), on crée d'abord une paire de parenthèses fictives étiquetée par p, puis on résout le but b.

```
block-exit(p)
```

La rencontre de block-exit(p) provoque une suspension brutale dans la résolution des buts inclus entre les parenthèses étiquetées par p. Le contexte avant la résolution de p est restauré, et la résolution se poursuit normalement.

On trouvera dans l'annexe B, un exemple pour la récupération des erreurs lexicales et syntaxiques. C'est selon cette méthode que l'ensemble des erreurs lexicales, syntaxiques et sémantiques sont détectées et récupérées dans le système INTERFACILE. Cette technique est aussi utilisée dans le sytème OPERA [Sedogbo et Guenthner 1986]. Il faut signaler qu'un tel mécanisme permet aussi de récupérer les erreurs que l'on pourrait faire en programmant l'interface, les "bugs". Ces derniers peuvent être ainsi récupérés et signalés, tout en redonnant la main à l'utilisateur pour qu'il formule une nouvelle requête. Signalons enfin que cette méthode a été retenue pour récupérer et spécifier les erreurs dans l'expressions des règles linguistiques.

En ce qui concerne le traitement des erreurs, on trouvera dans l'annexe B un programme Prolog pour la correction automatique des mots présentant une lettre en trop, une lettre omise, deux lettres inversées ou une lettre impropre.

L'aide à la formulation et à la reformulation repose dans son principe sur une synthèse plus ou moins complexe d'informations. La mise en oeuvre de cette technique est grandement facilitée si un langage de programmation comme Prolog a été choisi : grâce au principe d'unification, les mêmes procédures et données peuvent être utilisées tant en analyse qu'en synthèse.

La faisabilité d'un tel principe a été démontrée par A. Colmerauer dans le système ORBIS. Dans ce dernier système, en cas d'erreur lexicale, syntaxique ou sémantique, une aide à la reformulation est dispensée en faisant fonctionner l'interface en synthèse. Comment procède ORBIS ? A chaque mot d'une question, ORBIS associe au cours de l'analyse un entier spécifiant le numéro d'ordre du mot dans la phrase. Un compteur est incrémenté au fur et à mesure de l'avancement de l'analyse. En cas d'erreur, le point maximal d'analyse est retenu. ORBIS reprend alors le début de la question et produit le mot (ou les mots) suivant. Le même programme est utilisé en analyse et en synthèse. Dans INTERFACILE, le même programme est utilisé pour vérifier la cohérence sémantique d'une question et pour spécifier ce qui est correct, en cas d'erreur.

CONCLUSION

Une interface peu coopérative est condamnée à être rapidement délaissée par les utilisateurs. La détection et le traitement des erreurs constituent un des points clés du développement et de la viabilité des interfaces en langage naturel.

Les expériences que nous avons pu mener dans le domaine nous ont montré que la détection et le traitement des erreurs sont difficilement intégrables a posteriori dans un système existant. Le problème doit être pensé dès la conception de l'interface.

Jusqu'où aller dans le traitement des erreurs? Telle est la question que l'on ne manque pas de se poser lorsqu'on essaie d'étendre les capacités de correction d'une interface. Si la correction des erreurs lexicales reste un objectif réaliste, le traitement des erreurs syntaxiques et sémantiques peut devenir difficilement maîtrisable compte tenu du grand nombre de paramètres requis. L'aide à la reformulation est un excellent compromis entre, d'une part, l'absence de toute information et, d'autre part, la production automatique de corrections qui, elle, restera toujours coûteuse en temps de calcul et sans garantie certaine quant aux résultats fournis.

CHAPITRE VI

REALISATIONS

Microsial

Orbi

Interfrance

Interfacile

Pour illustrer un certain nombre de résultats de nos recherches sur les interfaces en langage naturel, nous avons été amené à développer différents systèmes. Dans ce chapitre nous présentons ces réalisations et décrivons quatre systèmes dont nous avons, en particulier, conçu et programmé les interfaces en langage naturel. Il s'agit des systèmes réalisés entre 1980 et 1986, à savoir :

- MICROSIAL [Giraud, Pique et Sabatier 1980], [Pique et Sabatier 1982]
- ORBI [Oliveira, Pereira et Sabatier 1982]
- INTERFRANCE [Duchier et Sabatier 1982]
- INTERFACILE [Mathieu et Sabatier 1986]

MICROSIAL

Réalisé en 1980 pour le compte de la société CAP Sogeti, MICROSIAL est un système comportant :

- une base de données sur un sous-ensemble d'organismes relevant du Ministère de la Défense,

- une interface en langage naturel permettant une interrogation en français de la base de données.

Le domaine de la base de données comprend différentes directions, divisions (bureaux, services, départements, adresses, etc.) et le personnel (noms, fonctions et grades).

MIROSIAL a été entièrement programmé en Prolog sur un Solar 16-35 et un Mitra 125. La base de données de type relationnel-déductif comporte environ 200 faits et 50 règles.

La partie interface en langage naturel comprend un lexique de 150 mots et une grammaire syntactico-sémantique de 80 règles. Nous donnons ci-dessous des exemples de types de questions analysables par MICROSIAL.

> La division Plan-Budget se trouve-t-elle dans l'Etat major de l'Armée de l'Air?

> Quelles sont les fonctions de tous les capitaines de vaisseau?

> Combien y a-t-il de colonels qui dirigent une division et qui dépendent du général Dupont?

> L'adresse de l'EMAT ?

> Tous les généraux commandent-ils une division?

> Quelles sont les personnes qui ne sont pas amiraux et qui dirigent un organisme faisant partie de l'EMM?

> De quel organisme relève le département Informatique?

ORBI

ORBI est un système expert d'évaluation des ressources du territoire du Portugal réalisé en 1981-1982 pour le compte du Ministère de l'Environnement de ce pays.

Le domaine d'expertise concerne l'évaluation des ressources biologiques et physiques d'un ensemble de régions du Portugal. Chaque région est définie par un ensemble de points (6000 environ). Chaque point correspond à un carré de 200 m de côté. A chaque point sont associés 23 descripteurs spécifiant des valeurs sur :

- l'inclinaison du terrain
- la qualité des micro-climats
- la perméabilité du sous-sol
- l'écoulement pluvial superficiel
- la présence de sites archéologiques
- le risque d'érosion,
- etc.

Au moyen des valeurs des descripteurs un ensemble de valeurs pour des facteurs peuvent être calculées. Parmi ces facteurs on a :

- l'accessibilité du terrain
- la richesse du sous-sol
- la richesse de la faune et de la flore
- la résistance à l'incendie
- la résistance à l'érosion
- la résistance à la pollution des nappes phréatiques
- etc.

Au moyen des valeurs des descripteurs et des facteurs un ensemble d'aptitudes pour le point ou la région concerné peut être déduit. Les principales aptitudes sont :

- l'habitat concentré
- l'agriculture intensive
- l'agriculture non intensive
- l'infrastructure industrielle
- l'infrastructure de loisirs

Chaque valeur d'un descripteur, d'un facteur et d'une aptitude pour un point donné est un entier. A chaque valeur est associé un autre entier (entre 1 et 5) spécifiant l'homogénéité (ou "représentativité") de la valeur donné pour le point concerné.

Les points et les valeurs des descripteurs associées constituent la base de faits du système ORBI. Codés en Prolog, ces faits sont sous la forme de clauses unaires.

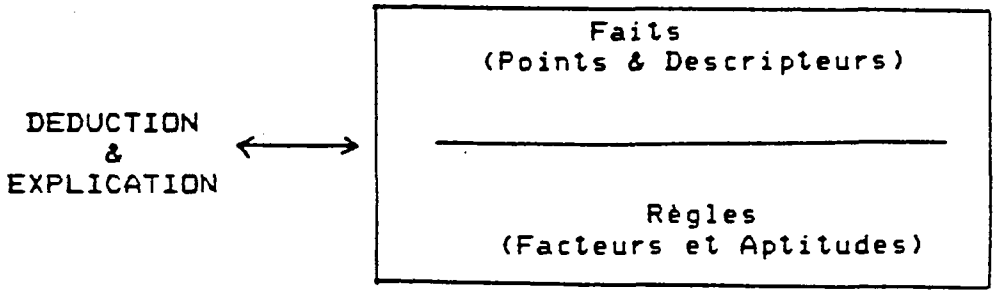
Les facteurs et les aptitudes sont déduits à partir d'un ensemble de règles. Codés en Prolog, ces règles sont sous la formes de

clauses complexes. Faits et règles constituent la base de connaissances du système expert.

Le système de déduction opérant sur les faits et les règles est le moteur d'inférence inhérent à Prolog.

Doté d'une capacité explicative, ORBI peut, à la demande de l'utilisateur, expliciter tous les résultats fournis.

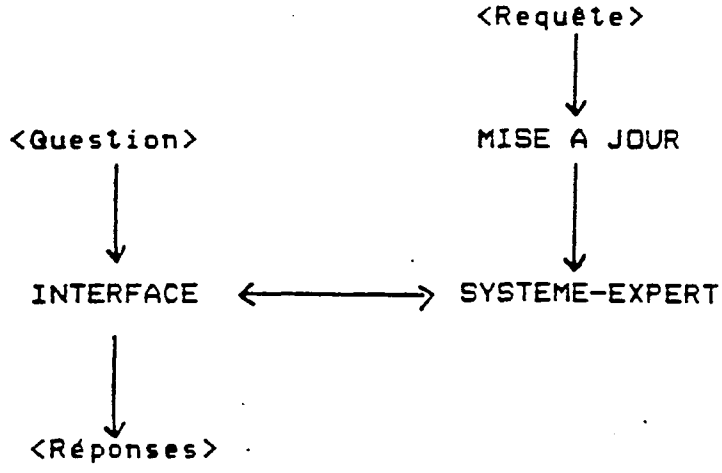
Le système-expert d'ORBI présente l'architecture suivante :



BASE DE CONNAISSANCES

ORBI comporte un module permettant de réaliser en interactif une mise à jour de la base de connaissances (ajouts, suppression et modifications des faits et des règles).

Le système expert d'ORBI est doté d'une interface en langage naturel qui permet à l'utilisateur de poser ses requêtes en portuguais. L'architecture complète d'ORBI est la suivante :



Voici un exemple de dialogue (nous donnons ici la traduction en

français) :

> Quelle est la valeur de l'aptitude à l'agriculture non intensive du point de coordonnées 103,709 ?

- Au point de coordonnées 103,709, l'aptitude à l'agriculture non intensive (A3) a la valeur très apte (5) (avec la représentativité 2).

- Voulez-vous des explications détaillées ?

> Oui.

- Très détaillées ?

> Non.

- Au point de coordonnées 103,709, l'aptitude à l'agriculture non intensive (A3) a la valeur très apte (5) (avec la représentativité 2).
  La valeur 5 a été obtenue à partir de :
  (25*(6-1)+10*4+5+60*5+50)/100
  au moyen de la règle :
  (25*(6-F6d)+10*F3+5+F2b+60*F4+50)/100

  - le facteur résistance à la pollution des nappes phréatiques (F2b) a la valeur 4 (avec la représentativité 5),

  - le facteur résistance du sol à l'érosion (F3) a la valeur 4 (avec la représentativité 4),

  - le facteur possibilité de production agricole (F4) a la valeur 5 (avec la représentativité 1),

  - le facteur paysages déstabilisés (F6d) a la valeur 1 (avec la représentativité 1).

Si on avait demandé des explications très détaillées, ORBI aurait expliqué comment chaque valeur des facteurs F2b, F3, F4 et F6d ont été obtenues.

Les explications fournies en langage naturel sont pour la plupart pré-enregistrées. Il en va tout autrement pour le module d'interface qui analyse les questions de l'utilisateur. Nous donnons ci-dessous quelques exemples de questions illustrant les différents types de phénomènes linguisitiques que l'interface est capable de traiter.

> Est-ce que chaque point dont le facteur résistance à l'incendie est égal à 2 ne possède aucune aptitude supérieure à 3 ?

> Quelle est la région n'ayant aucun point avec la valeur 1 pour l'aptitude à une infrastructure industrielle ?

> Pourquoi le point 103,225 est très apte à un habitat concentré ?

> Est-il vrai que pour quelques points, l'aptitude à l'agriculture non intensive est égale au facteur résistance à l'incendie?

> Quels points de la région 23,78;84,89 sont aptes à un habitat concentré?

> Et inaptes à une infrastructure industrielle ?

> Dans la région 98,123;129,620, combien de points sont très aptes à l'agriculture intensive ?

La représentation sémantique associée à une question comme :

> Quels sont les points ayant la même valeur pour l'aptitude à l'agriculture intensive ?

est l'expression :

```
ensemble( V-S,
          ensemble(point:(X,Y) ,
                   (point(X,Y,D),aptitude(a2,D,V-R)) même V,
                   S) ,
          L )
```

où - a2 est le code pour agriculture intensive

- L est la réposne sous la forme d'un ensemble de couples V-S
  tel que pour chaque valeur V de l'aptitude a2, S est l'ensemble des points de coordonnées X,Y ayant la valeur V

- D est la liste des descripteurs du point concerné

- R est la représentativité associée à la valeur de l'aptitude

ORBI a été entièrement écrit en Prolog sur une petite machine, un PDP 11/02 (LSI/11 CPU, 256 KB).

Pour ce qui est de la partie interface en langage naturel, le lexique se compose de 150 mots environ. La grammaire contient environ 75 règles combinant syntaxe, sémantique et logique dans

le formalisme des grammaires de métamorphoses.

INTERFRANCE

Réalisé en 1982 dans le cadre du Laboratoire de Recherche de la CGE, INTERFRANCE est un prototype d'interface en langage naturel, en français.

INTERFRANCE a pour domaine d'application la France géographique et administrative. Voici quelques exemples de questions :

> Quels sont les pays qui côtoient la France ?

> Donnez-moi l'indicatif téléphonique de la Lozère ?

> Où se trouve Saint-Pierre et Miquelon ?

> De combien de départements se compose chaque région ?

> Quelle est la superficie de tous les départements qui bordent la Manche et qui ne font pas partie de la région Basse-Normandie ?

> Combien y a-t-il de territoires ?

> Plusieurs fleuves traversent-ils plusieurs départements ?

> De quel département Toulouse est la préfecture ?

INTERFRANCE a été conçue de façon modulaire pour faciliter son adaptation à différents domaines d'application. Pour cela, le dictionnaire et la grammaire sont divisés en deux parties autonomes et complémentaires : l'une contient les mots (100 environ) et constructions syntaxiques (70 environ) indépendants du domaine d'application (appelés dictionnaire noyau et grammaire noyau); l'autre contient les mots et constructions propres au domaine d'application.

Une attention particulière a été apportée à la nature de la représentation sémantique associée à une phrase. Cette représentation est une formule logique facilement lisible par un non spécialiste, et peut être proposée telle quelle comme paraphrase de la phrase analysée, comme le montre l'exemple suivant :

> Le Tarn fait-il partie de la région Languedoc-Roussillon ?

(Représentation sémantique et paraphrase :)

Je dois dire si oui ou non
faire-partie-de(X,Y)

à condition que :

```
Il doit y avoir un X si je prends tous les cas où
je trouve
          département(X)
      et  nom-de-département(X,Tarn)

Il doit y avoir un Y si je prends tous les cas où
je trouve
          région(Y)
      et  nom-de-région(Y,Languedoc-Roussillon)
      et  faire-de-partie(X,Y)
```

INTERFRANCE illustre aussi une première approche dans le traitement des erreurs sémantiques. Comme le montrent les exemples suivants :

> Quelle est la population des fleuves qui se jettent dans la Méditerranée ?

- J'ignore le sens que vous associez à la construction : population ... de ... fleuve

> Le département de la Garonne fait-il partie de la région Languedoc-Roussillon ?

- La Garonne est un fleuve, pas un département.

INTERFRANCE a été programmé en Prolog sur un Vax 780.

INTERFACILE

Développé dans le cadre d'un contrat avec l'Agence de l'Informatique (ADI), INTERFACILE est un système associant :

- une base de connaissances sur les commandes d'un système d'exploitation d'ordinateur;

- une interface permettant de poser des questions en français sur l'utilisation de ces commandes.

Le système se compose de :

- une base de connaissances dépendante du domaine d'application et écrite dans un sous-ensemble de la logique (Clauses de Horn);

- une grammaire et un lexique-grammaire indépendants du domaine;

- un lexique-grammaire dépendant de l'application.

Opérant sur ces données linguistiques et les connaissances de la base :

- un analyseur couplant grammaire et lexiques-grammaires produit une représentation intermédiaire des questions, et (2) synthétise des messages en cas d'erreurs.

- un analyseur sémantique transforme la représentation intermédiaire dans une formule sémantique en consultant la base de connaissances;

- à partir d'une formule sémantique associée à une question, un évaluateur consulte la base pour produire les réponses.

- un système général d'aide guide l'utilisateur pour formuler et reformuler ses questions.

INTERFACILE a été programmé en Prolog II et tourne sur Vax, SPS et compatibles PC.

Trois aspects ont été privilégiés dans la réalisation de ce système :

- la résolution des proformes

- le traitement des erreurs

- la portabilité de la base de connaissances

En ce qui concerne la portabilité, le système a été conçu de façon à faciliter l'adaptation des connaissances à différents systèmes d'expoitation. INTERFACILE a été adapté avec succès aux langages de commande des systèmes DCL de DEC, UNIX et MS-DOS.

Afin d'assurer un dialogue concis et spontané, les proformes suivantes ont été introduites et traitées :

- les pronoms prersonnels (le, la, les, y, en, lui, leur, eux, etc.)

- les possessifs (son, sa, ses, etc.)

- les démonstratifs (celui-ci, ceux-ci, ce dernier, etc.)

En ce qui concerne le traitement des erreurs, INTERFACILE explique exactement quelle partie d'une question est incorrecte : un mot inconnu, un verbe muni d'une préposition non attendue, un ordre non prévu des compléments d'un verbe, etc. En cas d'erreurs lexicales, des solutions de corrections sont proposées (Voir l'algorithme et sa programmation dans l'annexe B).

Quand l'utilisateur a une vue erronée ou incomplète du domaine d'application, INTERFACILE localise l'erreur et produit ce qui est attendu. Les informations produites ne sont pas préprogrammées mais sont (1) déduites de la base de connaissances, et (2) synthétisées au moyen des composants linguistiques (grammaire, lexiques-grammaires et analyseurs).

Nous donnons ci-dessous l'extrait d'un dialogue avec INTERFACILE illustrant les aspects présentés ci-dessus. La base de connaissances concerne ici le langage des commandes DCL du système d'exploitation VMS de DEC.

> comment créer un fichier ?

- utilisez la commande :
  CREATE nomfichier

> comment l'imprimer ?

- utilisez la commande :
  PRINT nomfichier

> comment modifier son contenu ?

- utilisez l'éditeur avec la commande :
  EDIT nomfichier

```
> comment copire un fihier das un répertooire ?

     - ("copire"  devient  "copier")
       ("fihier"  devient  "fichier")
       ("das"  devient  "dans")
       ("répertooire"  devient  "répertoire")

     - utilisez la commande :
          COPY  nomfichier  [nomrépertoire]


> comment renommer ce dernier ?

     - utilisez la commande :
          RENAME/DIRECTORY  [ancienrépertoire]
                            [nouveaurépertoire]


> comment envoyer un message à un utilisateur ?

     - utilisez la commande :
          MAIL


> comment le supprimer ?

     - "le" renvoie à :
                         message ?        (1)
                         utilisateur ?    (2)
        tapez le numéro de votre choix.


> 2

     - INTERFACILE n'en sait rien, mais sait comment :

          envoyer un message
          écrire un message

          supprimer un fichier
          supprimer un répertoire
          supprimer un message


> comment ajouter à un fichier le contenu d'un fichier ?

     - pour "ajouter", utilisez la construction :
          ajouter   [quelque chose] à [quelque chose]

       complétez (ou tapez / pour annuler)


> comment ajouter /
```

```
> comment envoyer un message dans un terminal ?

     - pour "envoyer", utilisez les constructions :

          envoyer    [quelque chose] à     [quelqu'un]
          envoyer    [quelque chose] sur   [quelque chose]

       complétez (ou tapez / pour annuler)


> comment envoyer /


> comment imprimer le fichier d'une version ?

     - fichier de quoi ?  INTERFACILE parle plutôt de :

          fichier (tout simplement)
          fichier d'un répertoire
          version d'un fichier


> comment initialiser un fichier ?

     - INTERFACILE n'en sait rien, mais sait comment :

          créer un fichier
          éditer un fichier
          effacer un fichier

          initialiser un disque
          initialiser une disquette
```

Nous donnons ici les règles morphologiques, lexicales (lexique) et syntaxiques (grammaire) du système INTERFACILE. Ces règles linguistiques sont traduites dans des règles Prolog au moyen du programme de traduction présenté dans l'annexe A.

REGLES MORPHOLOGIQUES

```
-  ->  ;            (suppression du tiret)

'  ->  ;            (suppression de l'apostrophe)

au   ->  a  le   ;
aux  ->  a  les  ;

du  ->  de  le  ;
```

LEXIQUE

```
a  :  preposition("a") ;

afficher  :  verbe( afficher , objet.nil ) ;

ajouter  :  verbe ( ajouter , objet."a".nil ) ;

avec  :  preposition(avec) ;

catalogue  :  nom( catalogue.masculin.singulier ) ;
catalogues  :  nom( catalogue.masculin.pluriel ) ;

ce dernier :  pronom( ce-dernier.masculin.singulier ) ;
ces derniers  :  pronom( ce-dernier.masculin.pluriel ) ;
cette derniere  :  pronom( ce-dernier.feminin.singulier ) ;
ces dernieres  :  pronom( ce-dernier.feminin.pluriel );

celui ci  :  pronom( celui-ci.masculin.singulier ) ;
celle ci  :  pronom( celui-ci.feminin.singulier ) ;
ceux ci  :  pronom( celui-ci.masculin.pluriel ) ;
celles ci  :  pronom( celui-ci.feminin.pluriel );

changer  :  verbe( changer , objet.nil )
            verbe( changer , de.nil )    ;

comment  :  comment ;

comparer  :  verbe( comparer , objet.nil )
             verbe( comparer , objet.avec.nil ) ;
```

```
contenu  :  nom( contenu.masculin.singulier ) ;
contenus  :  nom( contenu.masculin.pluriel ) ;

copie  :  nom( copie.feminin.singulier ) ;
copies  :  nom( copie.feminin.pluriel ) ;

copier  :  verbe( copier , objet.nil ) ;

creer  :  verbe( creer , objet.nil ) ;

d  :  preposition( de ) ;
de  :  preposition( de ) ;

des  :  determinant( un.x.pluriel )
        preposition( de )            ;

dialoguer  :  verbe( dialoguer , avec.nil ) ;

editer  :  verbe( editer , objet.nil ) ;

effacer  : verbe( effacer , objet.nil ) ;

elle  :  pronom( lui.feminin.singulier ) ;
elles  :  pronom( lui.feminin.pluriel ) ;

en  : pronom( en.t ) ;

envoyer  :  verbe( envoyer , objet."a".nil ) ;

eux  :  pronom( lui.masculin.pluriel ) ;

faire  :  verbe( faire , objet.nil ) ;

fichier  :  nom( fichier.masculin.singulier ) ;
fichiers  :  nom( fichier.masculin.pluriel ) ;

imprimer  :  verbe( imprimer , objet.nil ) ;

l  :  determinant( le.g.singulier )
      pronom( le.g.singulier ) ;

la  :  determinant( le.feminin.singulier )
       pronom( le.feminin.singulier ) ;

le  :  determinant( le.masculin.singulier )
       pronom( le.masculin.singulier ) ;

les  :  determinant( le.g.pluriel )
        pronom( le.g.pluriel ) ;

leur  :  pronom(lui.g.pluriel ) ;

liste  :  nom( liste.feminin.singulier ) ;
listes  :  nom( liste.feminin.pluriel ) ;

lire  :  verbe( lire , objet.nil ) ;
```

```
lister  :  verbe( lister , objet.nil ) ;

lui  :  pronom( lui.g.singulier ) ;

message  :  nom( message.masculin.singulier ) ;
messages  :  nom( message.masculin.pluriel ) ;

modifier  :  verbe( modifier , objet.nil ) ;

nom  :  nom( nom.masculin.singulier ) ;
noms  :  nom( nom.masculin.pluriel ) ;

protection  :  nom( protection.feminin.singulier ) ;
protections  :  nom( protection.feminin.pluriel ) ;

proteger  :  verbe( proteger , objet.nil ) ;

purger  :  verbe( purger , objet.nil ) ;

recopier  :  verbe( recopier , objet.nil ) ;

renommer  :  verbe( renommer , objet.nil ) ;

sa  :  pronom( son.feminin.singulier ) ;

ses  :  pronom( son.g.pluriel ) ;

son  :  pronom( son.masculin.singulier ) ;

supprimer  :  verbe( supprimer , objet.nil ) ;

un  :  determinant( un.masculin.singulier ) ;
une  :  determinant( un.feminin.singulier ) ;

utilisateur  :  nom( utilisateur.masculin.singulier ) ;
utilisateurs  :  nom( utilisateur.masculin.pluriel ) ;

y  :  pronom("y".t) ;
```

GRAMMAIRE

Les symboles terminaux sont précédés du caractère '#'. Les conditions sont notées entre accolades. Les variables ont la syntaxe des variables Prolog.

```
question =>
 # comment
 ppv(11)
 # verbe(v,12)
```

```
   { complements-possibles(l1,l2,l3) }
   complements(l3);

ppv(nil) => ;

ppv(objet) =>
  # pronom(le.z) ;

ppv(objet) =>
  # pronom(se.z) ;

ppv("a") =>
  # pronom(lui.z) ;

ppv("a") =>
  # pronom("y".z);

ppv(de) =>
  # pronom(en.t) ;

complements(nil) => ;

complements(c.l) =>
  complement(c)
  complements(l);

complement(objet) =>
  groupe-nominal ;

complement(p) =>
  { dif(p,objet) }
  # preposition(p)
  # pronom(lui.z) ;

complement(p) =>
  { dif(p,objet) }
  # preposition(p)
  groupe-nominal ;

groupe-nominal =>
  #pronom(ce-dernier.z) ;

groupe-nominal =>
  #pronom(celui-ci.z) ;

groupe-nominal =>
  #pronom(son.z)
  #nom(a) ;

groupe-nominal =>
  #determinant(a1)
  # nom(a2)
  modificateur(a3) ;

groupe-nominal =>
  # nom(a2)
```

```
   modificateur(a3) ;

modificateur(a) => ;

modificateur(a) =>
   # preposition(de)
   groupe-nominal ;
```

C O N C L U S I O N

Nos travaux permettent de montrer que des solutions peuvent être apportées au problème du dialogue en langage naturel avec des systèmes de bases de connaissances, et que des interfaces peuvent être construites.

La conception d'interfaces reste une tâche réaliste dans le cadre de domaines d'application bien définis. Hors de frontières précises, le problème reste indécidable.

Compte tenu de l'étendue et de la diversité des problèmes à résoudre, le caractère partiel des solutions proposées et proposables doit conduire au développement de systèmes où l'utilisateur puisse maîtriser rapidement, et de façon peu contraignante, les capacités de l'interface. Le traitement des erreurs par l'aide à la reformulation, ainsi que l'assistance à la composition de phrases sont des éléments indispensables qui doivent faire l'objet d'une attention particulière dans le développement de toute interface.

Le portage d'une interface vers des domaines d'application différents doit nous conduire à étudier et à formaliser un sous-ensemble portable de notre langue doté d'une syntaxe et d'une sémantique rigoureusement définies. Dans ce cadre, des phénomènes spécifiques du langage naturel comme les anaphores, ellipses et proformes, restent les acteurs indispensables de toute interface conviviale.

Malgré tout cela, nous sommes bien conscient des limites de certains éléments de notre étude. En particulier, si le modèle de la logique des prédicats standard retenu pour l'interprétation sémantique est suffisant dans le cadre de l'interrogation de bases de connaissances, il se révèle impuissant pour rendre compte, par exemple, des aspects temporels et argumentatifs que l'on ne manquera pas de rencontrer si l'on veut concevoir des interfaces qui, à partir de formulations en langage naturel, permettront de construire des bases de connaissances.

Notre but était d'identifier, d'étudier et d'assembler de façon homogène les différents éléments composant une interface, en donnant à chacun le rôle indispensable qu'il a à jouer dans la construction.

Grâce aux mécanismes prédéfinis, comme l'unification, la réécriture et la déduction, à partir desquels nous avons pu très naturellement exprimer et résoudre nos problèmes, PROLOG s'est révélé être un outil pleinement adapté à notre entreprise.

ANNEXE A : DES REGLES LINGUISTIQUES AUX REGLES PROLOG

Nous présentons ici la façon dont les règles linguistiques sont traduites dans des règles Prolog directement exécutables.

Exemple de traduction d'une règle de prétraitement morphologique :

```
>pre;
   (règle morphologique :)
du  #  de le  ;
   (traduction Prolog :)
PRE("du".x,"de"."le".x) ->;
{}
```

Exemple de traduction d'une règle lexicale :

```
>lex;
   (règle lexicale :)
fichiers : nom(fichier.masculin.pluriel) ;
   (traduction Prolog :)
LEX("fichiers".x,x,"fichiers",8,nom(fichier.masculin.pluriel).nil) ->
{}
```

Exemple de traduction d'une règle de grammaire :

```
>mg;
   (règle de grammaire :)
question =>
  # comment
  ppv(11)
  # verbe(v,12)
  { complements-possibles(11,12,13) }
  complements(13);
```

(traduction Prolog :)

```
<question,x1,x0,<question,comment,a1,verbe(v,l2),a2>> ->
  <terminal(comment),x1,x2,comment>
  <ppv(l1),x2,x3,a1>
  <terminal(verbe(v,l2)),x3,x4,verbe(x130,x131)>
  complements-possibles(l1,l2,l3)
  <complements(l3),x4,x0,a2>;

{}
```

PROGRAMME DE TRADUCTION :

```
to-begin ->
  entete
  aide;

entete ->
  page
  outml("(C) Paul Sabatier, Juin 1985")
  outml(" Traduction : prétraitement, lexique, grammaire ---> Prolog")
  line;

aide ->
   line
   line
   outml(">traduire(n ,""fichier-source"",""fichier-resultat"",e)")
   line
   outml("            n=pre, n=lex, n=mg")
   outml("            e=1 (echo), e=0 (no-echo)")
   line;


"--- traduire ---"

traduire(n,s,r,e) ->
   si-echo(e)
   input-is(i)
   output-is(o)
   input(s)
   output(r)
   block(x,traduit(n))
   line
   outml(";")
   input(i)
   output(o)
   close
   recuperer(x)
   fin-echo(e)
   /;

si-echo(1) ->
   echo;

si-echo(0) ->;

fin-echo(x) ->
   no-echo;

traduit(n) ->
   n
   impasse;

traduit(n) ->
   traduit(n);
```

```
close ->
    close-input
    close-output;

recuperer(104) ->
    line
    outml(". . . TRADUCTION TERMINEE");

recuperer(x) ->
    dif(x,104)
    line
    line
    outml(". . . ERREUR . . . TRADUCTION INTERROMPUE !");

recuperer(x) ->
    block-exit(x);
    .

"--- lire-mots ---"
" l: resultat: liste de strings (avec nil) "
" a: caractere d'arret "

lire-mots(nil,a) ->
    next-char'(a)
    in-char'(a)
    /;

lire-mots(m.l,a) ->
    in-word(m,z)
    lire-mots(l,a);


"--- nil-variable ---"
" e1.e2.---.nil  devient  e1.e2.---.x"

nil-variable(nil,x,x) ->;

nil-variable(e.l1,x,e.l2) ->
    nil-variable(l1,x,l2);

"--- list-of-strings-one-string ---"

list-of-strings-one-string(s.nil,s) ->;

list-of-strings-one-string(s1.s2.l,s) ->
    conc-string(s1," ",s0)
    conc-string(s0,s2,s3)
    list-of-strings-one-string(s3.l,s);

"--- ecrire une regle prolog ---"

ecrire-regle-prolog(g,d) ->
    out(g)
    outm(" ->")
    ecrire-droite-regle-prolog(d)
    line;
```

```
ecrire-droite-regle-prolog(nil) ->
   outml(";");

ecrire-droite-regle-prolog(d.l) ->
   line
   outm("  ")
   out(d)
   ecrire-droite-regle-prolog(l);


"--- lecture et traduction d'une règle lexicale ---"

lex ->
   lire-mots(e1,":")
   lire-definitions(d)
   nil-variable(e1,x,e2)
   list-of-strings-one-string(e1,s)
   arg(0,s,n)
   ecrire-regle-prolog(LEX(e2,x,s,n,d),nil);

lire-definitions(nil) ->
   next-char'(";")
   in-char'(";")
   /;

lire-definitions(d.l) ->
   in(d)
   lire-definitions(l);


"-- lecture et traduction d'une règle de prétraitement morphologique --"

pre ->
   lire-mots(g1,"#")
   lire-mots(d1,";")
   nil-variable(g1,x,g2)
   nil-variable(d1,x,d2)
   ecrire-regle-prolog(PRE(g2,d2),nil);


"--- lecture et traduction d'une règle de mg ---"

mg ->
   lire-commentaire-mg
   lire-regle-mg(g1,d1,v)
   variables-communes(v)
   traduire-regle-mg(g1,d1,g2,d2)
   ecrire-regle-prolog(g2,d2);

lire-commentaire-mg ->
   in-string(s)
   /
   lire-commentaire-mg;

lire-commentaire-mg ->;
```

```
lire-regle-mg(g,d,v1.v2) ->
   in(g,v1)
   in-char'("=")
   in-char'(">")
   droite(d,v2);

droite(nil,nil) ->
   next-char'(";")
   in-char'(";")
   /;

droite(d,v) ->
   next-char'("{")
   in-char'("{")
   /
   conditions(d,v);

droite(l1,l2) ->
   next-char'("#")
   /
   in-char'("#")
   in(u,v)
   cas-terminal(u,v,l1,l2);

droite(nonter(t).l,v1.v2) ->
   in(t,v1)
   /
   droite(l,v2);


"--- cas terminal 1: #aaa (atome)  ou  #""chaine"" "
"                    2: #v (variable), 3: #determinant(x,y)    "

cas-terminal(t,nil,ter(t).l1,l2) ->
   /
   droite(l1,l2);

cas-terminal(t,v,ter(t).l1,v.l2) ->
   tuple(t)
   /
   droite(l1,l2);

cas-terminal(t,<t,m>.nil,ter(t).l1,<t,m>.l2) ->
   droite(l1,l2);


"--- conditions ---"

conditions(cond(c).l,v1.v2) ->
   in(c,v1)
   reste-conditions-et-apres(l,v2);

reste-conditions-et-apres(l,v) ->
   next-char'("}")
   in-char'("}")
   /
```

```
    droite(l,v);

reste-conditions-et-apres(cond(c).l,v1.v2) ->
    in(c,v1)
    reste-conditions-et-apres(l,v2);


"--- variables-communes ---"

variables-communes(l1) ->
    en-liste(l1,l2)
    var-com(l2);

en-liste(nil,nil) ->;

en-liste(nil.l1,l2) ->
    /
    en-liste(l1,l2);

en-liste((x.l1).l2,x.l3) ->
    /
    en-liste(l1.l2,l3);

en-liste(x.l1,x.l2) ->
    en-liste(l1,l2);

var-com(nil) ->;

var-com(<v,n>.l) ->
    voyage-sur(<v,n>,l)
    var-com(l);

voyage-sur(<v,n>,nil) ->;

voyage-sur(<v,n>,<v,n>.l) ->
    /
    voyage-sur(<v,n>,l);

voyage-sur(e1,e2.l) ->
    voyage-sur(e1,l);


"--- traduire-regle-mg ---"

traduire-regle-mg(g,d1,<g,p1,p0,a>,d2) ->
    droite-regle-mg(d1,p1,p0,d2,f)
    pour-construire-arbre(g.f,a);

droite-regle-mg(nil,p,p,nil,nil) ->
    /;

droite-regle-mg(cond(c).l1,p1,p0,c.l2,f) ->
    /
    droite-regle-mg(l1,p1,p0,l2,f);

droite-regle-mg(l1,p1,p0,l2,f) ->
```

```
    droite-non-vide(l1,p1,p0,l2,f);

droite-non-vide(nil,p,p,nil,nil) ->;

droite-non-vide(cond(c).l1,p1,p0,c.l2,f) ->
    /
    droite-non-vide(l1,p1,p0,l2,f);

droite-non-vide(<t,s1>.l1,p1,p0,<s2,p1,p2,f>.l2,f.l3) ->
    type-symbole(t,s1,s2,f)
    droite-non-vide(l1,p2,p0,l2,l3);

type-symbole(nonter,s,s,f) ->;

type-symbole(ter,s,terminal(s),s) ->;


"-- pour-construire-arbre ---"

pour-construire-arbre(r.nil,<r,nil>) ->;

pour-construire-arbre(r.f.l,a) ->
    list-tuple(r.f.l,a);
```

ANNEXE B : DETECTION ET RECUPERATION DES ERREURS EN PROLOG

Nous présentons ici deux programmes :

- l'un illustrant la façon générale de détecter les erreurs

- l'autre permettant de calculer les solutions de correction pour des erreurs lexicales.

Le programme Prolog II que nous donnons ici montre une façon de détecter et de récupérer des erreurs lexicales et syntaxiques au moyen des prédicats "block" et "block-exit".

Le programme est donné à titre d'illustration. Celui mis en oeuvre dans le système INTERFACILE s'inspire de la même philosophie mais reste plus complexe compte tenu de la qualité des diagnostics d'erreurs qui doit être fournie.

PROGRAMME :

```
analysons ->
  in-sentence(p,u)
  block(e,analyse(p))
  suite-a-donner(e);

analyse(p) ->
  analyse-lexicale(p,l)
  analyse-syntaxique(l);

analyse-lexicale(nil,nil) ->
  / ;

analyse-lexicale(x.l1,c.l2) ->
  lex(x,c)
  /
  analyse-lexicale(l1,l2);

analyse-lexicale(x.l1,l2) ->
  block-exit(erreur-lexicale(x));

analyse-syntaxique(l) ->
  phrase(l,nil)
  / ;

analyse-syntaxique(l) ->
  block-exit(erreur-syntaxique);

suite-a-donner(e) ->
  free(e)
  /;

suite-a-donner(e) ->
  line
  message(e);

message(erreur-lexicale(x)) ->
  outm("ERREUR LEXICALE: ") out(x) outm(" EST INCONNU");

message(erreur-syntaxique) ->
  outm("ERREUR SYNTAXIQUE");
```

COMMENTAIRES :

Le prédicat in-sentence(p,u) est prédéfini. Il lit une suite de caractères se terminant par ".", "?" ou "!", et transforme cette suite en deux listes p et u. p est la liste des mots formée par la suite de caractères. Les mots sont codés sont la forme de chaînes. u est la liste des identificateurs de p.

Chaque mot du lexique donne lieu à une règle Prolog lex(x,c) où x est le mot sous la forme d'une chaîne, c est le symbole non terminal (un terme) associé à x. Exemples :

```
lex("fichier",nom(commun,masculin,singulier,fichier)) ->;

lex("un",determinant(masculin,singulier,un)) ->;
```

Le prédicat phrase(x,y) est l'axiome de la grammaire de métamorphose sur laquelle l'analyse syntaxique est lancée.

EXEMPLES :

```
>analysons;
Comment imprimer un ficchier?

ERREUR LEXICALE: "ficchier" EST INCONNU
{}


>analysons;
Comment sur une disquette copier un fichier?

ERREUR SYNTAXIQUE
{}
```

Le programme Prolog II ci-dessous propose des solutions de corrections pour des erreurs lexicales, en particulier pour :

- un mot contenant un caractère en trop, comme

  "fichjier" au lieu de "fichier"

- un mot où un caractère a été omis, comme

  "répertore" au lieu de "répertoire"
  "renomer" au lieu de "renommer"

- un mot contenant deux caractères inversés, comme

  "envoeyr" au lieu de "envoyer"

- un mot contenant un caractère impropre, comme

  "effaccer" au lieu de "effacer"

PROGRAMME :

```
correction-lexicale(s1,s2) ->
   arg(0,s1,n)
   split(s1,l)
   erreurs-lexicales(n,l,s2);

erreurs-lexicales(n1,l1,s) ->
   val(sub(n1,1),n2)
   entree(s,n2)
   split(s,l2)
   caractere-en-trop-ou-omis(l1,l2);

erreurs-lexicales(n1,l1,s) ->
   val(add(n1,1),n2)
   entree(s,n2,d)
   split(s,l2)
   caractere-en-trop-ou-omis(l2,l1);

erreurs-lexicales(n,l1,s) ->
   entree(s,n)
   split(s,l2)
   caractere-impropre-ou-inverse(l1,l2);

caractere-en-trop-ou-omis(c.l,l) ->;

caractere-en-trop-ou-omis(c.l1,c.l2) ->
   caractere-en-trop-ou-omis(l1,l2);

caractere-impropre-ou-inverse(l1,l2) ->
   caractere-impropre(l1,l2);

caractere-impropre-ou-inverse(l1,l2) ->
   caractere-inverse(l1,l2);

caractere-inverse(c.l1,c.l2) ->
   caractere-inverse(l1,l2);

caractere-inverse(c1.c2.l,c2.c1.l) ->;

caractere-impropre(c.l1,c.l2) ->
   caractere-impropre(l1,l2);

caractere-impropre(c1.l,c2.l) ->;
```

COMMENTAIRES :

L'algorithme est basé sur le nombre de caractères du mot considéré comme erroné et du mot correct recherché. Les prédicats suivants sont prédéfinis :

```
arg(0,x,y)
y est la longueur de la chaîne x.
```

split(x,y)
décompose la chaîne (ou le n-uplet) x
dans la liste de ses éléments y

val(x,y)
évalue la fonction arithmétique x et
donne le résultat y.
En particulier, on a pour x :
add(u,v) addition de u et v
sub(u,v) soustraction de u par v.

Les mots du lexique sont donnés par un ensemble de règles Prolog de la forme :

lexique(s,n)

où s est le mot sous la forme d'une chaîne de caractères
n est le nombre de caractères du mot.

Comme par exemple:

```
lexique("effacer",7)  ->;
lexique("envoyer",7)  ->;
lexique("fichier",7)  ->;
lexique("renommer",8)  ->;
lexique("répertoire",10)  ->;
```

EXEMPLES :

```
>correction-lexicale("fichjier",x);

{x="fichier"}


>correction-lexicale("renomer",x);

{x="renommer"}


>correction-lexicale("envoeyr",x);

{x="envoyer"}

>corection-lexicale("répertore",x)

{x="répertoire"}
```

B I B L I O G R A P H I E

Abramson H., "Definite Clause Tanslation Grammars", Proceedings of International Symposium on Logic Programming, IEEE, 1984.

Abramson H., "Definite Clause Translation Grammars and the logical specification of data types as unambiguous context free grammars", Proceedings of the International Conference on Fifth Generation Computer Systems, ICOT, Tokyo, 1984.

Aho A., Ullman J., The Theory of Parsing, Translation, and Compiling, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1972 (Vol.1), 1973 (Vol.2).

Allen J., Perrault R., "Analysing intentions in utterances", Artificial Intelligence, 15, 1980.

Ballard B., User-specification of syntactic case frame in TELI, A transportable, user-customized natural language processor, Technical Report, AT&T Bell Laboratories, Murray Hill, N.J., 1985.

Bates M., Bobrow R., "Information retrieval using a transportable natural language interface", Proceedings of the 6th Annual International ACM SIGIR, ACM, New York, 1983.

Bobrow D., "A question-answering system for hight school algebra word problems", Proceedings of the AFIPS Conference, 1964.

Boons J.P., Guillet A., Leclère C., La structure des phrases simples en français, constructions intransitives, Droz, 1976.

Boons J.P., Guillet A., Leclère C., La structure des phrases simples en français, classes des constructions transitives, Rapport, LADL, CNRS, Université Paris 7, 1976.

Bulot R., Programme de transcription graphémo-phonétique du français conçu et réalisé avec la version de Prolog II, Mémoire de DEA, GIA, Faculté des Sciences de Luminy, 1983.

Burton R., Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems, BBN Report, 1976.

Carbonell J., Hayes P., "Recovery strategies for parsing extragrammatical language", American Journal of Computational Linguistics, 9, 3-4, 1983.

Charniak E., Wilks Y., Computational Semantics : An Introduction to Artificial Intelligence and Natural Language Comprehension, North- Holland, Amsterdam, 1976.

Chomsky N., "On certain formal properties of grammars", Information and Control, 2, 4, 1959.

Coelho H., A program conversing in Portuguese providing a library Service, Ph. D. Thesis, University of Edinburgh, 1979.

Colmerauer A., Les systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur un ordinateur, Rapport 43, Département d'Informatique, Université de Montréal, 1970.

Colmerauer A., Les Grammaires de Métamorphose, Rapport, GIA, Luminy, Université Aix-Marseille 2, 1975; "Metamorphosis Grammars", dans Natural language communication with computers, Bolc L. ed., Springer Verlag 1978.

Colmerauer A., "Un sous-ensemble intéressant du français", RAIRO, Informatique Théorique, 13, 4, Dunod, 1979.

Colmerauer A., Kanoui H., Pasero R., Roussel P., Un système de communication homme-machine en français, Rapport, GIA, Luminy, Université Aix-Marseille 2, 1973.

Colmerauer A., Kittredge R., Présentation du système ORBIS, Conférence COLING, 1982.

Colmerauer A., Pique J.F., "About Natural Logic" dans Advances in Data Base Theory, Vol.1, Gallaire et Minker eds, Plenum Press, 1980.

Dahl V., Sambuc R., Un système de banque de données en logique du premier ordre, en vue de sa consultation en langue naturelle, Rapport, GIA, Luminy, Université Aix-Marseille 2, 1976.

Dahl V., Un système déductif d'interrogation de banques de données en espagnol, Thèse, GIA, Luminy, Université Aix-Marseille 2, 1977.

Dahl V., McCord M., Treating coordination in logic grammars, Report, Computing Sciences Department, Simon Fraser University, 1983.

Dahl V., "More on Gapping Grammars", Proceedings of International Conference on Fifth Generation Computer Systems, ICOT, Tokyo, 1984.

Dahl V., Abramson H., "On Gapping Grammars", Proceedings of Second International Conference on Logic Programming, Uppsala, 1984.

Damerau F., "A technique for computer detection and correction of spelling errors", Communications of the ACM, 7, 3, 1964.

Danlos L., Génération automatique de textes en langues naturelles, Masson, Paris, 1985.

Duchier D., Sabatier P., "INTERFRANCE: un prototype d'interface en langue française", Rapport, CGE, Marcoussis, 1982.

Durham I., Lamb D., Saxe J., "Spelling correction in user interfaces", Communications of the ACM, 26, 10, 1983.

Filgueiras M., Silva F., Natural Menus, Report, Centro de Infor-

matica, Universidade do Porto, 1986.

Fillmore C., "The case for case", dans Universal in Linguistic Theory, (Bach E. et Harms R. éd.), Holt, Rinehart et Winston, New York, 1968.

Gal A., Minker J., "Des réponses coopératives données par une interface SGBD en langage naturel", Actes du Congrès AFCET, 1985.

Giraud Ch., Pique J.F., Sabatier P., Manuel d'utilisation de la banque de données MICROSIAL, Rapport, GIA, Luminy, Université Aix-Marseille 2, 1980.

Greimas A., Sémantique structurale, Larousse, Paris, 1966.

Grice H., "Logic and conversation", dans Syntax and Semantics (Cole P. et Morgan J Eds.) Academic Press, New-York, 1975.

Gross M., Méthodes en syntaxe, Herman, 1975.

Gross M., Grammaire transformationnelle du français, syntaxe du nom, Larousse, 1977.

Guenthner F., Nef F., "Introduction à l'analyse des langues naturelles", dans L'analyse logique des langues naturelles (1968-1978), Editions du CNRS, Paris, 1984.

Guenthner F., Sabatier P., Formal Semantics and Knowledge Representation, FNS-Bericht-85-7, Univeristé de Tübingen, 1985.

Guez S., Sabbagh S., INTERIX : Système d'aide à l'utilisation d'Unix, Rapport de Recherche, Laboratoires de Marcoussis, CGE, 1984.

Hafner C., Godden K., "Portability of syntax and semantics in Datalog", ACM, Transactions on Office Information Systems, 3, 2, 1985.

Hankamer J,, Sag I., "Deep and Surface Anaphora", Linguistic Inquiry 7, 1976.

Harris L., "User-oriented data base query with the Robot natural language query system", International Journal of Man-Machine Studies, 9, 1977.

Hayes P., Maroudian G., "Flexible parsing", American Journal of Computational Linguistics, 7, 4, 1981.

Hayes P., Reddy R., "Steps towards graceful interaction in spoken and written man-machine communication", International Journal of Man-Machine Studies, 19, 3, 1984.

Hendrix G., Sacerdoti E., Sagalowicz J., Slocum J., "Developping a natural language interface to complex data", ACM, Transaction on Database systems, 3, 2, 1978.

Hirschman L., Puder K., "Restriction Grammars in Prolog", Proceedings of the First International Logic Programming Conference, Marseille, 1982.

Hirst G., Anaphora in Natural Language Understanding : A survey, Springer-Verlag, 1981.

Hirakawa H., Chart parsing in Concurrent Prolog, Report 8, ICOT, 1983.

Jackendoff R., "Gapping and Related Rules", Linguistic Inquiry 2, 1971.

Janas J., "On the feasibility of informative answers", dans Advances in database theory, Gallaire H., Minker J. et Nicolas J. (eds), Plenum Press, 1981.

Kaplan J., "Cooperative responses from a portable natural language query system", Artificial Intelligence, 19, 1982.

Katz J., Postal P., An Integrated Theory of Linguistics Descriptions, MIT Press, Cambridge, Mass., 1964.

Kuno S., "Gapping: A Functional Analysis", Linguistic Inquiry, 1976.

Laporte E., Application de la morpho-phonologie à la production automatique de textes phonétiques, Rapport, LADL, CNRS, Université Paris 7, 1987.

Lewis D., "General Semantics", Semantics of Natural Language, (Davidson et Harman éds), Reidel, Dordrecht, Hollande, 1972.

Martin P., Appelt D., Pereira F., "Transportability an generality in a natural-language interface system", Proceedings of COLING, 1983.

Mathieu Y., Sabatier P., INTERFACILE: Linguistic Coverage and Query Reformulation, Rapport, LADL, CNRS, Université Paris 7, 1985; et Proceedings of COLING, 1986.

Matsumoto Y., Tanaka H., Hirakawa H., Miyoshi H., Yasukawa H., "BUP: A bottom-up parser embedded in Prolog", New Generation Journal, Vol. 1, Springer Verlag, 1983.

McCord M., Focalizers, the scoping problem, and semantic interpretation rules in logic grammars, Report, Université du Kentucky, 1981.

McCord M., "Using slots and modifiers in logic grammars for natural language", Artificial Intelligence, Vol. 18, 1982.

Meloni H., Etude et réalisation d'un système de reconnaissance de la parole continue, Thèse d'Etat, Faculté des Sciences de Luminy, Université Aix-Marseille II, 1982.

Minsky M., "A framework for representing knowledge", dans Pyschology of Computer Vision, Winston Ed., Mac Graw-Hill, 1975.

Morgan H., "Spelling correction in systems programs", Communications of the ACM, 13, 2, 1970.

Oliveira E., Pereira L., Sabatier P., "ORBI: an expert system for environmental resources evaluation through natural language", Proceedings of the First International Logic Programming Conference, Marseille, 1982.

Pasero R., Représentation du français en logique du 1er ordre en vue de dialoguer avec un ordinateur, Thèse, GIA, Luminy, Université Aix-Marseille II, 1973.

Pereira F., "Extraposition Grammars", American Journal of Computational Linguistics, Vol. 7, 1981.

Pereira F. Warren D., "Definite Clauses Grammars for language analysis, a survey of the formalism and a comparison with augmented transition networks", Artificial Intelligence, Vol 13, 1980.

Pereira F., Warren D., "An efficient easily adaptable system for interpreting natural language queries", Report, DAI, University of Edinburgh, 1981; American Journal of Computational Linguistics, Vol.8, 1982.

Peterson J., "Computer programs for detecting and correcting errors", Communications of the ACM, 23, 12, 1980.

Phillips B., Nicholl S., Inglish: A natural language interface, Report, Tektronix, Inc., Beaverton, Oregon, 1985.

Pique J.F., Sur un modèle logique du langage naturel et son utilisation pour l'interrogation de bases de données, Thèse, GIA, Luminy, Université Aix-Marseille 2, 1981.

Pique J.F., Sabatier P., "An informative, adaptable and efficient natural language consultable database system", Proceeding of ECAI, 1982.

Pollock J., Zamora A., "Automatic spelling correction in scientific and scholary text", Communications of the ACM, 27, 4, 1984.

Robinson J., "A machine-oriented logic based on the resolution principle", Journal of the ACM, Vol. 12, 1965.

Ross J., Guess Who ?, dans Papers from the Fifth Regional Meeting of the Chicago Linguistic Society, Chicago, Illinois, 1967.

Ross J., Gapping and the Order of Constituents, dans Progress in Linguistics, Bierwish and Heidolph Eds., Mouton and Co., 1970.

Roussel P., PROLOG: manuel de référence et d'utilisation, Rapport, GIA, Luminy, Université Aix-Marseille 2, 1975.

Sabatier P., "Sur quelques placements et interprétation d'AUSSI", Linguisticae Investigationes, II, 2, John Benjamins, Amsterdam, 1979.

Sabatier P., Dialogues en français avec un ordinateur, Thèse, GIA, Luminy, Université Aix-Marseille 2, 1980.

Sabatier P., "Contextual grammars in Prolog", Proceedings of Logic Programming Workshop, University of Lisbon, 1983.

Sabatier P., "Une règle d'effacement de VP en français", dans De la Syntaxe à la Pragmatique, Attal P. et Muller C., Eds., John Benjamins, Amsterdam, 1984.

Sabatier P., "Puzzle grammars", Proceedings of Workshop on Natural Language Understanding and Logic Programming, Rennes, 1984; in Natural Language Understanding and Logic Programming, Dahl and Saint-Dizier Eds., North Holland, 1985.

Sabatier P., Un ensemble d'outils pour l'analyse de phrases en Prolog II, Rapport Interne, LADL, CNRS, Univ. Paris 7, Juillet 1985.

Sag I., Deletion and Logical Form, Ph. D. Thesis, MIT, Cambridge, 1977.

Salkoff M., Une grammaire en chaîne du français, Dunod, 1973.

Salkoff M., Analyse syntaxique du français: grammaire en chaîne, Amsterdam, J. Benjamins, 1979.

Salkoff M., "A context-free grammar of French", Proceedings of COLING, 1982.

Schank R., Conceptual Information Processing, North-Holland, Amsterdam, 1975.

Schank P., Abelson R., Scripts, Plans, Goals, and Understanding, Lawrence Erlbaum, Hillsdale, N.J., 1977.

Sedogbo C., About coordination treatment in Metamorphosis Grammars, Rapport, Bull, 1983.

Sedogbo C., Tree Grammar: A PROLOG implementation of the String Grammar of French, Rapport, BULL, 1984.

Sedogbo C., Guenthner F., Some remarks on the treatment of errors in natural language processing systems, Rapport, Bull, 1986.

Simmons R., "Semantic networks : their computation and use for understanding English sentences", Computer Models of Thought and Language, (Schank et Colby éds), Freeeman, San Francisco, 1973.

Stabler E., "Deterministic and bottom-up parsing in Prolog", Proceedings of the AAAI Conference, 1983.

Tennant H., "Menu-based natural language understanding", Proceedings of the National Computer Conference, 1984.

Uehara K., Ochitani R., Kakusho O., Toyoda J., "A bottom-up parser in predicate logic: a survey of the formalism and its implementation technique", Proceedings of International Symposium on Logic Programming, Altlantic City, 1984.

Uehara K., Toyoda J., "A pattern matching directed parser: PAMPS", Proccedings of ACL/LSA Meeting, 1981.

Walker A., Porto A., "KBO1: a knowledge based garden store assistant", Proceedings of Logic Programming Workshop, University of Lisbon, 1983.

Wilks Y., Preference Semantics, Report, Stanford I.A. Laboratory, Stanford University, 1973.

Williams E., "Discourse and Logical form", Linguistic Inquiry 8, 1977.

Winograd T., Understanding Natural Language, Academic Press, New York, 1972.

Woods W., Kaplan R., Nash-Webber B., The Lunar Sciences Natural Language Information System, Final Report, Bolt Beranek and Newman Inc., 1972.