Algorithmes pour le traitement interactif des langues naturelles

Jacques Courtin

THESE

présentée à

Université Scientifique et Médicale de Grenoble
Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR es SCIENCES
"Mathématiques"

par

Jacques COURTIN

ALGORITHMES POUR LE TRAITEMENT INTERACTIF DES LANGUES NATURELLES

Thèse soutenue le 28 octobre 1977 devant la Commission d'Examen :

Président	:	B. VAUQUOIS
Examinateurs	:	L. BOLLIET
		J.C. BOUSSARD
		J. COHEN
		Ph. JORRAND
Rapporteur	:	G. VEILLON

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

Monsieur Gabriel CAU : Président
Monsieur Pierre JULLIEN : Vice Président

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

PROFESSEURS TITULAIRES

MM.	AMBLARD Pierre	Clinique de dermatologie
	ARNAUD Paul	Chimie
	ARVIEU Robert	I.S.N.
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme.	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOU Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Clinique de pédiatrie et puériculture
	BELORIZKY Elie	Physique
	BERNARD Alain	Mathématiques pures
Mme.	BERTRANDIAS Françoise	Mathématiques pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques pures
	BEZEZ Henri	Pathologie chirurgicale
	BLAMBERT Maurice	Mathématiques pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Clinique gastro-entérologique
Mme.	BONNIER Marie-Jeanne	Chimie générale
MM.	BOUCHERLE André	Chimie et toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BOUTET DE MONVEL Louis	Mathématiques pures
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologique
	CALAS François	Anatomie
	CARLIER Georges	Biologie végétale
	CARRAZ Gilbert	Biologie animale et pharmocodynamie
	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques pures
	CHARACHON Robert	Clinique oto-rhino-laryngologique
	CHATEAU Robert	Clinique de neurologie
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie pathologique

Mme.	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELORMAS Pierre	Pneumophtisiologie
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée (IUT I)
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	GAGNAIRE Didier	Chimie physique
	GALVANI Octave	Mathématiques pures
	GASTINEL Noël	Analyse numérique
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	KAHANE André	Physique générale
	KOSZUL Jean-Louis	Mathématiques pures
	KLEIN Joseph	Mathématiques pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
Mme.	LAJZEROWICZ Janine	Physique
MM.	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LE ROY Philippe	Mécanique (IUT I)
	LLIBOUTRY Louis	Géophysique
	LOISEAUX Pierre	Sciences nucléaires
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Melle	LUTZ Elisabeth	Mathématiques pures
MM.	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Clinique cardiologique
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NOZIERES Philippe	Spectrométrie physique
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET Jean	Semeiologie médicale (Neurologie)
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REVOL Michel	Urologie
	RINALDI Renaud	Physique
	DE ROUGEMONT Jacques	Neuro-chirurgie
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SIBILLE Robert	Construction mécanique (IUT I)

MM.	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme.	VERAIN Alice	Pharmacie galénique
MM.	VERAIN André	Physique
	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale

PROFESSEURS ASSOCIES

MM.	CRABBE Pierre	CERMO
	DEMBICKI Eugéniuz	Mécanique
	JOHNSON Thomas	Mathématiques appliquées
	PENNEY Thomas	Physique

PROFESSEURS SANS CHAIRE

Melle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Gilbert	Géographie
	BENZAKEN Claude	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
	BILLET Jean	Géographie
	BOUCHET Yves	Anatomie
	BRUGEL Lucien	Energétique (IUT I)
	BUISSON René	Physique (IUT I)
	BUTEL Jean	Orthopédie
	COHEN ADDAD Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie
	CONTE René	Physique (IUT I)
	DELOBEL Claude	M.I.A.G.
	DEPASSEL Roger	Mécanique des fluides
	FONTAINE Jean-Marc	Mathématiques pures
	GAUTRON René	Chimie
	GIDON Paul	Géologie et minéralogie
	GLENAT René	Chimie organique
	GROULADE Joseph	Biologie médicale
	HACQUES Gérard	Calcul numérique
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et médecine préventive
	IDELMAN Simon	Physiologie animale
	JOLY Jean-René	Mathématiques pures
	JULLIEN Pierre	Mathématiques appliquées
Mme.	KAHANE Josette	Physique
MM.	KRAKOWIACK Sacha	Mathématiques appliquées
	KUHN Gérard	Physique (IUT I)
	LUU DUC Cuong	Chimie organique
	MAYNARD Roger	Physique du solide
Mme.	MINIER Colette	Physique (IUT I)
MM.	PELMONT Jean	Biochimie
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
Melle	PIERY Yvette	Physiologie animale

MM.	RAYNAUD Hervé	M.I.A.G.
	REBECQ Jacques	Biologie (CUS)
	REYMOND Jean-Charles	Chirurgie générale
	RICHARD Lucien	Biologie végétale
Mme.	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIROT Louis	Chirurgie générale
Mme.	SOUTIF Jeanne	Physique générale
MM.	STIEGLITZ Paul	Anesthésiologie
	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	ARMAND Yves	Chimie (IUT I)
	BACHELOT Yvan	Endocrinologie
	BARGE Michel	Neuro-chirurgie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamie
MM.	BOST Michel	Pédiatrie
	BOUCHARLAT Jacques	Psychiatrie adultes
Mme.	BOUCHE Liane	Mathématiques (CUS)
MM.	BRODEAU François	Mathématiques (IUT B) (Personne étrangère habilitée à être directeur de thèse)
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARDON Michel	Géographie
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	CORDONNIER Daniel	Néphrologie
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DENIS Bernard	Cardiologie
	DOUCE Roland	Physiologie végétale
	DUSSAUD René	Mathématiques (CUS)
Mme.	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FAURE Gilbert	Urologie
	GAUTIER Robert	Chirurgie générale
	GIDON Maurice	Géologie
	GROS Yves	Physique (IUT I)
	GUIGNIER Michel	Thérapeutique
	GUITTON Jacques	Chimie
	HICTER Pierre	Chimie
	JALBERT Pierre	Histologie
	JULIEN-LAVILLAVROY Claude	O.R.L.
	KOLODIE Lucien	Hématologie
	LE NOC Pierre	Bactériologie-virologie
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et médecine préventive
	MALLION Jean-Michel	Médecine du travail
	MARECHAL Jean	Mécanique (IUT I)
	MARTIN-BOUYER Michel	Chimie (CUS)
	MICHOULIER Jean	Physique (IUT I)

MM.	NEGRE Robert	Mécanique (IUT I)
	NEMOZ Alain	Thermodynamique
	NOUGARET Marcel	Automatique (IUT I)
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B) (Personnalité étrangère habilité à être directeur de thèse)
	PEFFEN René	Métallurgie (IUT I)
	PERRIER Guy	Géophysique-Glaciologie
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAMBAUD André	Hygiène et hydrologie (Pharmacie)
	RAMBAUD Pierre	Pédiatrie
	RAPHAEL Bernard	Stomatologie
Mme.	RENAUDET Jacqueline	Bactériologie (Pharmacie)
MM.	ROBERT Jean-Bernard	Chimie physique
	Romier Guy	Mathématiques (IUT B) (Personnalité étrangère habilité à être directeur de thèse)
	SCHAERER René	Cancérologie
	SHOM Jean-Claude	Chimie générale
	STOEBNER Pierre	Anatomie pathologie
	VROUSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	DEVINE Roderick	Spectro physique
	HODGES Christopher	Transition de phases

Fait à SAINT MARTIN D'HERES, NOVEMBRE 1976.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Monsieur Philippe TRAYNARD : Président

Monsieur Pierre-Jean LAURENT : Vice Président

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BLOCH Daniel	Physique du solide
	BONNETAIN Lucien	Chimie minérale
	BONNIER Etienne	Electrochimie et électrométallurgie
	BOUDOURIS Georges	Radioélectricité
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	COUMES André	Radioélectricité
	DURAND Francis	Métallurgie
	FELICI Noël	Electrostatique
	FOULARD Claude	Automatique
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean-Charles	Chimie-Physique
	PAUTHENET René	Physique du solide
	PERRET René	Servomécanismes
	POLOUJADOFF Michel	Electrotechnique
	SILBER Robert	Mécanique des fluides

PROFESSEUR ASSOCIE

M.	ROUXEL Roland	Automatique

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuel	Electronique
	BOUVARD Maurice	Génie mécanique
	COHEN Joseph	Electrotechnique
	LACOUME Jean-Louis	Géophysique
	LANCIA Roland	Electronique
	ROBERT François	Analyse Numérique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM.	ANCEAU François	Mathématiques appliquées
	CHARTIER Germain	Electronique
	GUYOT Pierre	Chimie minérale
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du solide
	MORET Roger	Electrotechnique nucléaire
	PIERRARD Jean-Marie	Mécanique
	SABONNADIERE Jean-Claude	Informatique fondamentale et appliquée
Mme.	SAUCIER Gabrièle	Informatique fondamentale et appliquée

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan	Automatique

CHERCHEURS DU C.N.R.S. (Directeur et Maîtres de Recherche)

MM.	FRUCHART Robert	Directeur de Recherche
	ANSARA Ibrahim	Maître de Recherche
	CARRE René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	MATHIEU Jean-Claude	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Je tiens à remercier :

Monsieur B. VAUQUOIS, Professeur à l'Université Scientifique et Médicale de Grenoble, qui m'a fait l'honneur de présider le jury de cette thèse.

Monsieur L. BOLLIET, Professeur à l'Université des Sciences Sociales de Grenoble, qui s'est intéressé à mon travail et a bien voulu participer à ce jury.

Monsieur J.C. BOUSSARD, Professeur à l'Université de Nice, qui m'a toujours encouragé dans mes travaux.

Monsieur J. COHEN, Associate Professor of Computer Science, Brandeis University, pour l'intérêt qu'il a porté à ce travail et pour avoir accepté de lire le manuscrit.

Monsieur P. JORRAND, Maître de recherche au C.N.R.S., qui a accepté de juger cette thèse et dont les critiques bienveillantes m'ont été précieuses.

Monsieur G. VEILLON, Professeur à l'Institut National Polytechnique de Grenoble, qui m'a donné les moyens de mener à bien cette recherche. Je veux lui exprimer ma profonde gratitude pour la confiance qu'il m'a toujours témoignée et pour ses nombreux conseils et encouragements.

Je tiens également à remercier D. DUJARDIN et E. GRANDJEAN pour leur compétence et leur collaboration apportées au niveau de la construction d'une version opérationnelle du système P.I.A.F.

Toute ma reconnaissance va également à G. BICAIS qui a assuré la dactylographie de ce texte avec soin et rapidité, et aux membres du service de reprographie pour le tirage de cette thèse.

TABLE DES MATIERES
-:-:-:-:-:-:-:-:-

AVANT-PROPOS

Le Traitement automatique des langues a fait l'objet de développements importants depuis une dizaine d'année.

Initialement orienté vers des applications déjà ambitieuses telles que la traduction automatique [VAUQ], les progrès de l'intelligence artificielle ont fait apparaître un intérêt croissant pour les applications orientées vers la "compréhension" de textes en langue naturelle. Des projets importants ont été consacrés à des applications telles que la consultation de bases de données [PLAT, THOM], les systèmes questions réponses utilisant des réseaux sémantiques [SCHA, SIMM, WIL], la compréhension de la parole [WALK, WOODS]. Des projets de recherche plus sophistiqués sont orientés vers la construction de systèmes de compréhension et de représentation plus ambitieuses [BWKRL].

Parallèlement, les études menées sur le traitement de textes importants (traduction automatique [BOIT, VAUQ], documentation automatique [SALT, SALBER]) ont été poursuivies.

Dans ce domaine, on a assisté à une progression et à une multiplicité des modèles. Après l'approche pragmatique des modèles lexicographiques, l'apparition des grammaires génératives, puis transformationnelles [CHI] ont donné naissance à des systèmes analogues aux compilateurs dirigés par la syntaxe [KNU3].

Les difficultés combinatoires (ambiguités) et les problèmes sémantiques ont conduit à proposer des modèles équivalents fondés sur des techniques procédurales [WIN, WOOD]. Malgré les différences de procédés [PETR, WOODS], on constate, cependant, une convergence pour l'obtention d'une représentation de dépendance conceptuelle des phrases analysées, équivalente aux structures de dépendances. Cette représentation permet notamment une interprétation fonctionnelle de la structure de phrase, facilement traitable par les langages informatiques (par exemple LISP [PLAT]).

Une autre caractéristique de l'évolution actuelle est la nécessité d'intégrer les divers composants du système de traitement automatique des langues. Un des objectifs est de limiter la combinatoire et de lever les ambiguités [BOIT].

Parallèlement, l'utilisation effective du traitement automatique des langues dans des applications industrielles commence à être envisagée. Plus accessible dans le domaine de la communication Homme-Machine, elle pose des problèmes de réalisation lorsque l'on veut aborder des applications sans restriction de la langue naturelle, comme, par exemple, en traduction automatique et en documentation automatique.

C'est dans ce contexte que l'on voit intervenir des problèmes "d'engineering" [HEND], utilisation d'ensemble de programmes de manipulation de données linguistiques. Il faut alors réaliser des systèmes efficaces, transportables et suffisamment puissants utilisables sur des ordinateurs "raisonnables".

C'est dans ce cadre que s'inscrit ce travail :

- obtention de structure de dépendances
- efficacité des algorithmes
- extensibilité et adaptabilité du système (incrémentiel, interactif, modulaire)
- communication avec d'autres langages (LISP)
- transportabilité.

INTRODUCTION

Avant d'aborder la philosophie et le but du système que nous proposons, nous examinons les principes généraux des systèmes proposés à ce jour.

Analyse des systèmes existants

Parmi les principes généraux qui régissent la conception de ces systèmes, on peut noter qu'ils possèdent les caractéristiques suivantes :

a) Ils sont conçus pour une application déterminée.

La plupart des réalisations existantes sont orientées sur une application bien déterminée. Les auteurs définissent des outils informatiques propres à cette application et en général difficilement adaptables à d'autres réalisations.

b) Ils utilisent soit un module unique soit plusieurs modules.

Pour concevoir ces outils, deux stratégies sont envisagées : soit on utilise un seul module capable de traiter tous les problèmes rencontrés , soit on utilise plusieurs modules dont chacun d'eux est spécialisé pour une certaine classe de problèmes. L'emploi du module unique [COLM2] est séduisant car il a l'avantage de ne nécessiter qu'un seul formalisme ; par contre, il a l'inconvénient majeur d'être sur-puissant pour le traitement des problèmes simples.
De plus, il peut être difficilement proposé pour des applications dont la complexité n'atteint pas la possibilité de l'outil.
La deuxième stratégie, qui est la plus utilisée, consiste à définir des outils spécialisés pour chaque classe de problèmes.
Par exemple on définit un module pour l'analyse morphologique, un deuxième pour l'analyse syntaxique et enfin un module pour résoudre les problèmes sémantiques. Le travail de conception est plus important, mais l'efficacité algorithmique en est améliorée. De plus,

si les outils sont très généraux, un ou plusieurs modules peuvent être utilisés pour d'autres applications n'ayant pas la complexité du système complet.

c) L'exécution des différents modules est séquentielle.

Lorsque le système est composé de plusieurs modules, l'exécution de ces différents composants est en général séquentielle, ce qui a l'inconvénient de fournir au module suivant toutes les ambiguités du module précédent. Par contre, une exécution en parallèle ou en coroutine de différents composants telle qu'elle est proposée par WINOGRAD [WIN] ou FENNEL [FEN] permet de résoudre des ambiguités plus facilement et plus rapidement.

d) Les modules proposés sont en général des accepteurs.

Les accepteurs donnent une réponse binaire (le texte d'entrée appartient ou non au langage prévu par les modèles linguistiques). La non appartenance pouvant être détectée à quelque niveau que ce soit. C'est-à-dire que l'exécution d'un module est subordonnée à l'acceptation du module précédent.

e) La localisation des erreurs est difficile.

En cas d'erreurs, ces accepteurs sont capables de déterminer le module défaillant. Quant à la localisation de ces erreurs, elle dépend de la complexité du modèle. Par exemple, pour la morphologie, elle est relativement simple ; par contre, pour la syntaxe, les outils proposés à ce jour sont incapables de donner une information précise sur le type de l'erreur et le lieu où elle se produit. Soit l'erreur provient du texte d'entrée, et une étude attentive, par l'utilisateur, de la phrase incriminée, permet de la détecter ; soit l'erreur provient du modèle (cas non ou mal prévu par les paramètres linguistiques nécessaires à son fonctionnement) et les remises en cause nécessitent souvent de nombreuses retouches des paramètres linguistiques du modèle défaillant voire des modèles en aval et en amont.

Nous abordons ici le point fondamental du traitement automatique des langues. Contrairement au langage de programmation, la langue naturelle est difficilement modélisable. En effet, en programmation, on connait parfaitement le langage à compiler. On peut le décrire à l'aide d'une grammaire formelle, figée une fois pour toute, et produire, par exemple des compilateurs dirigés par la syntaxe [WIRTH]. Par contre, dans le domaine des langues naturelles, on ne peut pas définir à priori une grammaire. Les modèles informatiques proposés ne sont toujours qu'une approximation des phénomènes linguistiques : il s'ensuit qu'ils sont toujours incomplets et qu'ils contiennent toujours des erreurs ou des omissions. On est donc amené à y apporter sans cesse de nombreuses retouches qui sont difficilement maîtrisables si on veut garder toute la langue. De plus, le problème de la convergence de la méthode utilisée reste entier : en effet, rien ne prouve qu'un modèle linguistique apte à traiter un texte, modifié pour en traiter un second, soit encore valide pour le traitement du premier texte. D'énormes progrès informatiques ont été réalisés pour aider l'utilisateur à résoudre ce problème. En particulier, dans quelques systèmes proposés actuellement, les informations linguistiques ne sont plus intégrées dans les programmes. Par contre, il n'existe pas encore, à notre connaissance, de véritables systèmes intéractifs, ce qui amélioreraient considérablement la tâche du constructeur de modèles linguistiques.

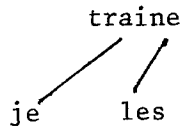
Les insuffisances des systèmes actuels proviennent en grande partie de ces difficultés. Pour y remédier, on peut avoir deux attitudes :

a) Limiter artificiellement la langue. Cette solution, qui est la plus facile, a donné naissance à de nombreuses réalisations donnant de bons résultats (par exemple, les systèmes questions réponses).

b) Ne pas limiter la langue, mais définir des outils informatiques puissants et simples pour aider au maximum l'utilisateur à maîtriser la complexité des paramètres linguistiques.

- Le système P.I.A.F.

En fonction des remarques précédentes, nous avons défini un système de traitement automatique des langues appelé P.I.A.F. "Programme Intéractif d'Analyse du Français" dont le but est de construire des structures de dépendances. Par exemple, pour la phrase : "je les traine", on obtient la structure de dépendances suivante :



où "traine" est le gouverneur et "je" et "les" les deux dépendants. Nous avons choisi ce type d'arborescences, car elles sont facilement interprétables par d'autres modèles tels qu'un modèle sémantique.

Les caractéristiques essentielles de ce système sont les suivantes :

a) Il est composé de plusieurs modules informatiques indépendants dont l'enchaînement peut être soit séquentiel soit coroutine (c'est-à-dire que le module suivant est activé dès qu'un sous-résultat est déterminé par le module précédent).

b) La conception des modules étant très générale, ils peuvent être utilisés pour de nombreuses applications.

c) Le système est intéractif.

Des éditeurs associés à des compilateurs incrémentiels permettent à l'utilisateur, même en cours d'exécution,

- la correction du texte d'entrée
- la modification des paramètres linguistiques.

d) Le texte d'entrée n'étant plus supposé correct, le système permet la localisation des erreurs, voire leurs corrections dans certains cas.

Pour lever les ambiguités et détecter les erreurs, voire les corriger, on utilise les redondances de la langue. En effet, une erreur de codage ne suffit pas toujours à rendre une phrase incompréhensible. Dans le cas des phrases : "Je les traines" et "chien mange soupe" les erreurs n'empêchent pas leurs compréhensions. Pour pouvoir les détecter, il faut disposer de modèles incomplets pouvant intervenir soit en parallèle, ce qui par leur complémentarité permet d'obtenir une solution le plus rapidement possible, soit en séquentiel, ce qui autorise la détection et la localisation de l'erreur. Pour proposer des éléments de correction, il faut disposer de modèles construisant un certain invariant. Dans le cas de la phrase "je les traines", on peut soit utiliser un modèle engendrant la forme du verbe "trainer" à la première personne du singulier du présent de l'indicatif et on obtient une solution, soit utiliser un modèle engendrant toutes les formes possédant le même invariant phonétique que "je les traines" et on obtient deux interprétations possibles : "je les traine" et "je l'étrenne".

Le système P.I.A.F. est actuellement composé de deux modèles principaux : "Un transducteur général d'états finis" et un "système d'analyse syntaxique". Chacun d'eux étant composé de sous-modules dont l'enchaînement peut être séquentiel ou parallèle. Cet exposé est divisé en trois parties :

Dans une première partie, nous exposons le transducteur général d'états finis, dans la seconde, nous abordons le problème de l'analyse syntaxique, pour terminer, dans la troisième partie, par les éditeurs et les applications du système P.I.A.F.

A

LE TRANSDUCTEUR GENERAL D'ETATS FINIS

I - INTRODUCTION

A l'origine, le transducteur général d'états finis avait été conçu pour réaliser l'analyse morphologique d'une langue. Etant très général, il a permis, par la suite, d'autres applications telles que :

- la génération morphologique (système réversible),
- la transduction phonétique,
- la correction automatique des erreurs lexicales par invariance phonétique et invariance morphonémique,
- la traduction d'un texte en braille
- l'aide à la traduction d'un langage de programmation dans un autre langage de la même famille (par exemple : PL 360 vers LP/80).

Dans cette introduction, nous donnons tout d'abord quelques définitions sur les chaînes de caractères, ensuite nous indiquons quels étaient les objectifs de l'analyse morphologique. Dans les chapitres suivants, nous abordons l'étude des principaux composants de ce transducteur, à savoir le dictionnaire et la grammaire.

I.1 - DEFINITIONS

- Vocabulaire :

Soit un ensemble fini V appelé vocabulaire contenant tous les caractères :

$V = \{\sqcup, A, B, \ldots, Z, ., ; , \ldots, 0, 1, \ldots, 9\}$

- Chaîne :

On appelle chaîne sur V une suite finie de symboles de V :

$X_1 X_2 \ldots X_n$ avec $X_i \in V$, $i \in [1,n]$

- V^*, V^+

On désigne par

V^* l'ensemble de toutes les chaînes que l'on peut construire sur V, y compris la chaîne vide notée Λ et par

V^+ l'ensemble $V^* - \{\Lambda\}$.

- On définit une relation d'ordre sur les symboles de V notée "<".

On prend la convention :

... < , < ... < ␣ < A < B < ... < Z < ... < 0 < 1 < ... < 9

- LONGUEUR (α)

C'est une application de V^* dans N (ensemble des entiers naturels)

- si $\alpha = \Lambda$ alors LONGUEUR (α) = 0
  sinon si $\alpha = a_1 \varphi$ alors LONGUEUR(α) = 1+LONGUEUR(φ)
  $\forall a_1 \in V$ et $\forall \varphi \in V^*$

- PREFIXE(α,)

, $\beta \in V^*$

C'est une application de $V^* \times V^*$ dans V^*.

Soient a,b $\in V$, φ, λ , $\mu \in V^*$

- si $\alpha = a\lambda$ et $\beta = b\lambda$
  alors PREFIXE(α,β) = Λ

- si $\alpha = \varphi\lambda$ et $\beta = \varphi\mu$
  alors PREFIXE(α,β) = φ PREFIXE(λ,μ)

- <u>relation inférieure ("<") entre deux chaînes non vides (ordre alphabétique)</u>

Soient α et β deux éléments de V^+, avec $\alpha \neq \beta$,

soient a_1, $a_2 \in V$, λ , φ , $\mu \in V^*$

- si $\alpha = \varphi\, a_1\, \lambda$ et $\beta = \varphi\, a_2\, \mu$ avec $a_1 \neq a_2$
  $\alpha < \beta$ si et seulement si $a_1 < a_2$ sinon $\beta < \alpha$

- si $\beta = \alpha\, a_2\, \lambda$ alors $\alpha < \beta$

- si $\alpha = \beta\, a_1\, \lambda$ alors $\beta < \alpha$

c'est une relation d'ordre strict.

En conséquence : $\neg(\alpha < \beta)\ \&\ \alpha \neq \beta \Rightarrow \beta < \alpha$

- <u>La relation "<" entre deux chaînes est transitive</u>

si $\alpha < \beta$ et $\beta < \delta$ alors $\alpha < \delta\ \forall\ \alpha, \beta, \delta \in V^+$

Soient $a_1, a_2, a_1', a_2', a_1'', a_2'' \in V, \varphi, \lambda, \mu, \varphi', \lambda', \mu', \varphi'' \in V^*$

1°) $\alpha < \beta$: si $\alpha = \varphi\, a_1\, \lambda$ et $\beta = \varphi\, a_2\, \mu$ avec $a_1 < a_2$

-i) $\beta < \delta$: si $\beta = \varphi' a_1'\, \lambda'$ et $\delta = \varphi' a_2'\, \mu'$ avec $a_1' < a_2'$

i_1) $\varphi = \varphi' \Rightarrow a_2 = a_1'$ donc
$\alpha = \varphi' a_1\, \lambda$, $\delta = \varphi' a_2'\, \mu'$ avec $a_1 < a_2' \Rightarrow \alpha < \delta$

i_2) $\varphi' = \varphi\, a_1''\, \varphi''$
$\alpha = \varphi\, a_1\, \lambda$, $\delta = \varphi\, a_1'' \varphi''\, a_2'\, \mu'$
$\beta = \varphi\, a_2\, \mu = \varphi\, a_1''\, \varphi''\, a_1'\, \lambda'$
$\Rightarrow$ $a_2 = a_1''$; comme $a_1 < a_2$
$\Rightarrow$ $a_1 < a_1''$ donc $\alpha < \delta$

i_3) $\varphi = \varphi'\, a_1''\, \varphi''$
$\alpha = \varphi'\, a_1''\, \varphi''\, a_1\, \lambda$
$\beta = \varphi'\, a_1'\, \lambda' = \varphi' a_1''\, \varphi''\, a_2\, \mu$
$\delta = \varphi'\, a_2'\, \lambda$
comme $a_1' = a_1''$, $a_1' < a_2' \Rightarrow a_1'' < a_2'$ donc $\alpha < \beta$

-j) $\beta < \delta \Rightarrow \delta = \beta\ a'_1\ \lambda'$

d'après 1°) $\alpha = \varphi\ a_1\ \lambda$, $\beta = \varphi\ a_2\ \mu$

donc $\delta = \varphi\ a_2\ \mu\ a'_1\ \lambda'$

$a_1 < a_2 \Rightarrow \alpha < \delta$

2°) $\alpha < \beta$: $\beta = \alpha\ a'_1\ \lambda'$

i) $\beta < \delta$: $\beta = \varphi\ a_1\ \lambda$ et $\delta = \varphi\ a_2\ \mu$ avec $a_1 < a_2$

i_1) $\alpha = \varphi$

$\delta = \varphi\ a_2\ \mu = \alpha\ a_2\ \mu \Rightarrow \alpha < \delta$

i_2) $\alpha = \varphi\ a'_2\ \mu'$

$\beta = \alpha\ a'_1\ \lambda' = \varphi\ a'_2\ \mu' a'_1\ \lambda' = \varphi\ a_1\ \lambda$ donc $a'_2 = a_1$.

Or $\delta = \varphi\ a_2\ \mu\ .\ a_1 < a_2 \Rightarrow a'_2 < a_2$ donc $\alpha < \delta$

i_3) $\varphi = \alpha\ a'_2\ \mu'$

$\delta = \varphi\ a_2\ \mu = \alpha\ a'_2\ \mu'\ a_2\ \mu \Rightarrow \alpha < \delta$

j) $\beta < \delta$: $\delta = \beta\ a_1\ \lambda$

or $\beta = \alpha\ a'_1\ \lambda'$ donc $\delta = \alpha\ a'_1\ \lambda'\ a_1\ \lambda \Rightarrow \alpha < \delta$

I.2 - OBJECTIFS DE L'ANALYSE MORPHOLOGIQUE

Le but de ce modèle est de :

1°) segmenter une phrase pour obtenir des chaînes

2°) déterminer un certain nombre de renseignements linguistiques sur les chaînes déterminées par la segmentation : c'est-à-dire effectuer une transduction.

I.2.1. - La segmentation

Etant donné une phrase $\alpha_1\alpha_2 \ldots\alpha_n$ ($\forall i \in [1,n]$, $\alpha_i \in V$), on doit déterminer des mots ou groupe de mots m_i ($m_i \in V^+$) tels que :

$$m_1\ m_2 \ldots m_k = \alpha_1\ \alpha_2 \ldots \alpha_n \quad \text{avec } k \leq n$$

Pour ce premier objectif, on suppose que le blanc n'est pas un séparateur de mots afin de laisser la possibilité de reconnaître des locutions telles que : "au fur et à mesure, à gauche de, à ce propos, etc..." et ainsi, considérer, à ce niveau, qu'elles ne constituent qu'un seul segment.

D'autre part, on suppose que tout segment est insegmentable, ce qui a pour conséquence de segmenter en un nombre unique de segments.

Exemple :

Pour la phrase :

"Il y a à ce propos une controverse."

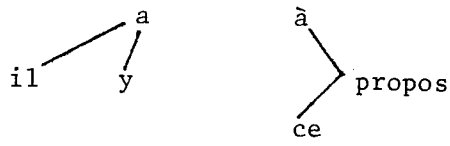
On pourrait déterminer quatre segmentations différentes :

Il y a	à ce propos	une	controverse	.	k = 5
Il y a	à ce propos	une	controverse	.	k = 7
Il y a	à ce propos	une	controverse	.	k = 7
Il y a	à ce propos	une	controverse	.	k = 9

Cette restriction nous est imposée par le modèle suivant, l'analyse syntaxique, dont l'algorithme utilisé doit connaître le nombre d'unités lexicales de la phrase (c'est-à-dire k). On aurait pu trouver une solution à ce problème (par exemple, effectuer l'analyse pour toutes les segmentations différentes) au détriment, bien entendu, de l'efficacité de l'algorithme d'analyse. Dans la réalité, ce cas ne se présente que très rarement et quand il se présente il suffit de choisir une stratégie de segmentation qui préserve la cohérence de l'analyse syntaxique. Pour l'exemple précédent, on choisit la stratégie qui permet d'obtenir la segmentation :

il y a à ce propos une controverse .

car la locution "à ce propos" n'est pas ambigüe comme la locution "il y a". On pourrait également comme dans CHAUCHE [CHA] effectuer une partie de l'analyse syntaxique au niveau de l'analyse morphologique et, pour l'exemple précédent, délivrer :



une controverse .

Détermination d'un m_i

On constate que les segments m_i sont composés de plusieurs éléments que l'on appelle, dans la langue française, suivant leur place dans m_i : préfixe, base ou racine, suffixe, désinence.

Exemple :

"retrouvera " est composé du préfixe "re", de la base "trouv", du suffixe "er" et de la désinence "a␣".

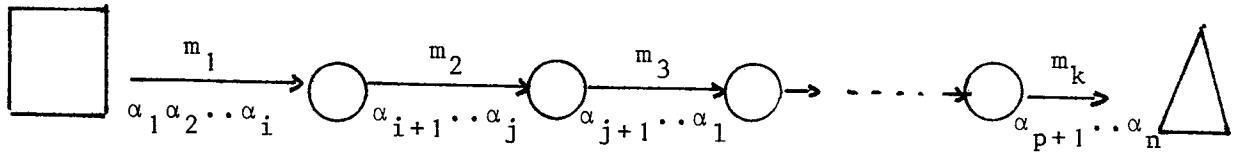
La détermination d'un m_i consiste donc en sa décomposition en préfixe, base, suffixe, et désinence.

Pour ce faire, on dispose :

1°) d'un dictionnaire de préfixes, de bases, de suffixes et de désinences qui permet, par identification, la décomposition d'une partie contigüe de la chaîne d'entrée,

2°) d'une grammaire qui doit infirmer ou confirmer la concaténation de ces éléments pour constituer le segment cherché.

Le principe de la segmentation est donc le suivant :
détermination de m_1 par identification dans le dictionnaire et contrôle par application de règles de grammaire, puis itération du processus jusqu'au point de fin de phrase.



NB : La décomposition d'un m_i n'est pas unique : par exemple "COUVENT␣" admet deux décompositions : "COUV" "ENT␣" et "COUVENT" "␣".

I.2.2. - La transduction

Pour chaque m_i déterminé, il faut donner des renseignements linguistiques qui seront utilisés par le modèle suivant. Il s'agit de donner toutes les interprétations de toutes les décompositions d'un m_i.

Exemple : La chaîne "COUVENT␣" donne lieu à deux décompositions "COUVENT" "␣" et "COUV" "ENT␣" ayant pour interprétation : substantif commun masculin singulier et VERBE à la 3ème personne du pluriel du présent de l'indicatif ou VERBE à la 3ème personne du pluriel du présent du subjonctif.

Cette transduction est réalisée par la grammaire qui vérifie la segmentation d'un m_i étant donné que la plupart des éléments constituant un m_i sont porteurs d'informations.

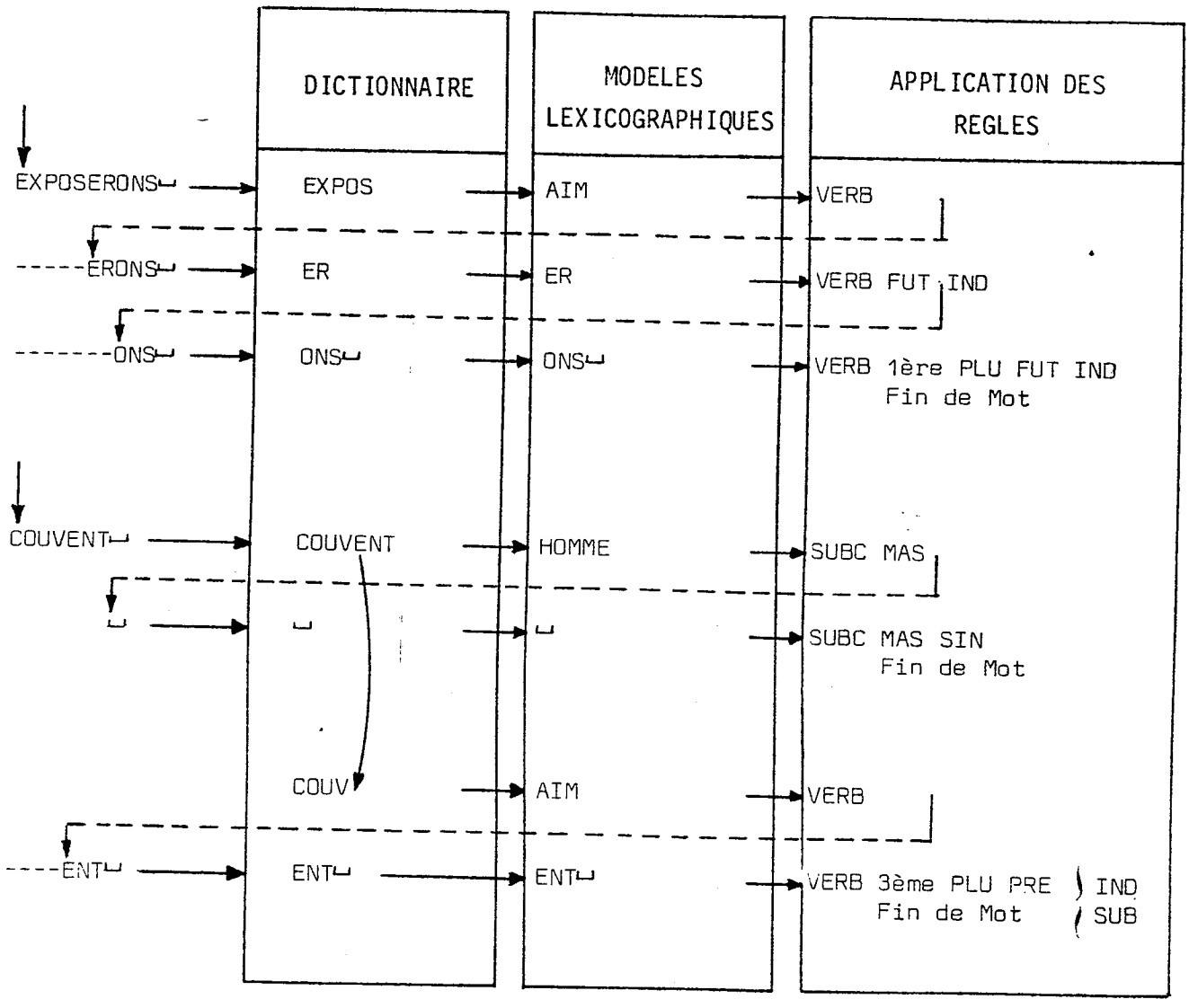
Exemple :
La base "COUVENT" est porteuse de l'information substantif commun masculin et la désinence "␣" est porteuse de l'information singulier.

De même la base "COUV" est une racine de VERBE du premier groupe et la désinence "ENT␣" est une désinence ayant pour information :
3ème personne du pluriel du présent de l'indicatif ou du subjonctif.

Le principe de l'analyse peut être représenté par le schéma suivant où les modèles lexicographiques constituent l'interface entre le dictionnaire et la grammaire.

Dans la suite de cette partie, le chapitre A1 sera consacré au dictionnaire et le chapitre A2 à la grammaire.

SCHEMA DE L'ANALYSE MORPHOLOGIQUE



A1 LE DICTIONNAIRE

I - INTRODUCTION

Comme nous l'avons vu précédemment, on ne connaît pas, comme en Compilation, l'élément que l'on cherche à identifier étant donné que l'on n'a pas de séparateurs de mots. En conséquence, on ne connaît que la première lettre de cet élément :

$$\underset{\uparrow}{\alpha_i}\ \alpha_{i+1} \cdots \alpha_n \qquad \alpha_j \in V \quad \forall j \in [i,n]$$

On ne peut donc pas employer les techniques classiques d'adressage dispersé "HASCH CODING". D'autre part, désirant une gestion intéractive et en temps réel du dictionnaire, il est également exclu d'utiliser les méthodes "batch". Nous allons utiliser une table chaînée par ordre alphabétique dont l'organisation devra tenir compte de la spécificité de l'analyse morphologique.

Nous examinons tout d'abord quelques aspects théoriques.

II - ASPECTS THEORIQUES

Nous ne nous intéressons ici qu'à l'ensemble des clés permettant l'accès aux informations (attributs).

II.1 - DEFINITIONS

- Dictionnaire

  On appelle dictionnaire (D) tout sous-ensemble de V^+

- relation "C" (contenue)

  Soient α et β deux éléments de V^+ : on dit que α C β si et seulement si

1°) PREFIXE(α, β) = α

2°) LONGUEUR(α) < LONGUEUR(β)

- relation "< & ¬ C" (inférieure et non contenue)

Soient α et β deux éléments de V^+ :

On dit que $\alpha < \& \neg C\ \beta$ si et seulement si

1°) $\alpha \prec \beta$

2°) PREFIXE(α, β) $\neq \alpha$

- Ensemble des Fils d'un élément β

Soit $\beta \in D$,

FILS(β) = $\{\alpha \in D \ / \ \alpha\ C\ \beta\}$

- Ensemble des Frères d'un élément α

Soit $\alpha \in D$ tel que $\alpha \notin Fils(\beta) \quad \forall \beta \in D$

Frere(α) = $\{\beta \in D \ / \ \alpha < \& \neg C\ \beta\}$

- Fils direct

Soient α et $\beta \in D$

α est fils direct de β si et seulement si

1°) $\alpha \in Fils(\beta)$

2°) Il n'existe pas d'élément $\gamma \in Fils(\beta)$ tel que $\alpha \in Fils(\gamma)$

- Frère direct

Soient α et $\beta \in D$ tel que $\alpha \notin Fils(\gamma) \quad \forall \gamma \in D$

β est frère direct de α si et seulement si

1°) $\beta \in Frère(\alpha)$

2°) Il n'existe pas d'élément $\delta \in Frere(\alpha)$ tel que $\beta \in Frere(\delta)$

Exemple :

Soit D = {␣␣␣, PARTIEL, PARTIES, PARTIE, PART, PARTOUT, ZZZZ}

FILS(PARTIEL) = FILS(PARTIES) = {PARTIE,PART}
FILS(PARTOUT) = FILS(PARTIE) = {PART}
PART est fils direct de PARTIE et de PARTOUT
PARTIE est fils direct de PARTIES et de PARTIEL
PARTIES est frere direct de PARTIEL
PARTOUT est frère direct de PARTIES

III - ORGANISATION

Le dictionnaire est organisé en liste chaînée par ordre alphabétique (relation <) en tenant compte des relations fils direct et frere direct de telle sorte que si l'élément cherché est α , on obtienne immédiatement l'ensemble de tous ses fils : Fils(α).

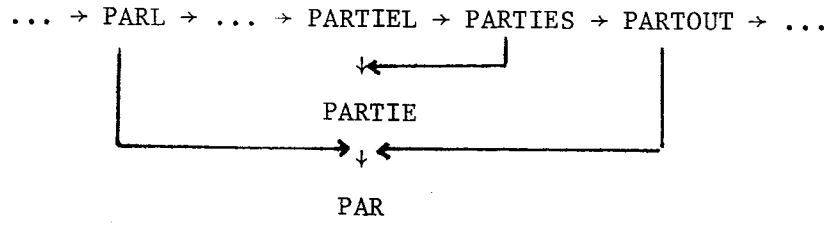
Ceci signifie qu'à tout $\alpha \in D$, est associé un pointeur sur son fils direct

Exemple :

PARTIEL
↓
PARTIE
↓
PART

De même, pour tout $\alpha \in D$ tel qu'il n'existe pas d'éléments $\gamma \in D$ tel que $\alpha \in$ Fils(γ), est associé un pointeur sur son frère direct. On appelle élément dominant un tel élément.

Exemple :



III.1 - ALGORITHME DE RECHERCHE D'UN ELEMENT

Soit un élément $\delta = X_1 X_2 \cdots X_n$, on cherche le plus grand préfixe de δ qui appartient à D.
C'est-à-dire un élément α de D tel que PREFIXE(α,δ) = α et tel que
$\forall \beta \neq \alpha \in D$ tel que PREFIXE(β,δ) = β , on ait :

$$\beta = \text{PREFIXE}(\beta,\alpha)$$

III.1.1. - Aspects théoriques

Théorème : Soient $A_1, A_2, \ldots, A_n$ des éléments de V^+ tels que
$\forall i \in [1,n-1]\ A_i < A_{i+1}$,
soit ELEMENT un élément de V^+ tel que

$$A_n < \text{ELEMENT}$$

alors

$$\text{LONGUEUR}(\text{PREFIXE}(A_j,\text{ELEMENT})) \leq \text{LONGUEUR}(\text{PREFIXE}(A_{j+1},\text{ELEMENT}))$$
$$\forall j \in [1,n-1]$$

Soient les chaînes :

$\alpha = \alpha_1 \alpha_2 \cdots \alpha_n$ $\quad$ $n > 0,\ m > 0,\ 1 > 0$

$\beta = \beta_1 \beta_2 \cdots \beta_m$ $\quad$ $\forall i \in [1,n]\ ,\ \alpha_i \in V$
$\forall i \in [1,m], \beta_i \in V$

$\delta = \delta_1 \delta_2 \cdots \delta_1$ $\quad$ $\forall i \in [1,1]\ ,\ \delta_i \in V$

avec

$$\alpha < \beta < \delta$$

alors
LONGUEUR(PREFIXE(α,δ)) $\leq$ LONGUEUR(PREFIXE(β,δ))

appelons :

φ_k = PREFIXE(α,β) $\quad 0 < k < \min(n,m)$

φ_i = PREFIXE(α,δ) $\quad 0 < i < \min(n,1)$

φ_j = PREFIXE(β,δ) $\quad 0 < j < \min(m,1)$

et

k = LONGUEUR(φ_k), i = LONGUEUR(φ_i), j = LONGUEUR(φ_j)

a) $k > i \Rightarrow i = j$

En effet : $\alpha < \beta < \delta$ s'écrit :

1. $\varphi_i\alpha_{i+1} \ldots\alpha_{k+1}\ldots\alpha_n < \varphi_i\alpha_{i+1} \ldots\alpha_k\beta_{k+1} \ldots\beta_m < \varphi_j\delta_{j+1} \ldots \delta_1$

$\alpha < \delta$ s'écrit

2. $\varphi_i\alpha_{i+1} \ldots \alpha_n < \varphi_i\delta_{i+1} \ldots \delta_1$

d'après 2. $\alpha_1\alpha_2 \ldots\alpha_i = \delta_1\delta_2 \ldots\delta_i$ et $\alpha_{i+1} < \delta_{i+1}$

comme $k > i$ on a $\alpha_1\alpha_2 \ldots\alpha_i = \beta_1\ \beta_2 \ldots \beta_i$ et $\alpha_{i+1} = \beta_{i+1}$

donc :

$\beta_{i+1} < \delta_{i+1}$ et $\beta_1\ \beta_2 \ldots \beta_i = \delta_1\delta_2 \ldots\delta_i$

d'où

$i = j$

b) $k = i \Rightarrow i \leq j$

$\alpha < \beta < \delta$ s'écrit

1. $\varphi_i\alpha_{i+1} \ldots\alpha_n < \varphi_i\beta_{i+1} \ldots\beta_m < \varphi_j\delta_{j+1} \ldots \delta_1$

$\alpha < \delta$ s'écrit

2. $\varphi_i\alpha_{i+1} \ldots\alpha_n < \varphi_i\delta_{i+1} \ldots \delta_1$

d'après 2. $\varphi_i = \alpha_1\alpha_2 \ldots\alpha_i = \delta_1\delta_2 \ldots\delta_i$

comme k = i $\varphi_i = \beta_1\beta_2 \ldots\beta_i = \alpha_1\alpha_2 \ldots\alpha_i$

donc $\beta_1\beta_2 \ldots\beta_i = \delta_1\delta_2 \ldots\delta_i$

et d'après 1

$$\boxed{i \leq j}$$

c) k < i est impossible

$\alpha < \beta < \delta$ s'écrit

1. $\varphi_k\alpha_{k+1} \ldots\alpha_{i+1} \ldots\alpha_n < \varphi_k\beta_{k+1} \ldots\beta_{i+1} \ldots\beta_m < \varphi_j\delta_{j+1} \ldots\delta_1$

$\alpha < \delta$ s'écrit

2. $\varphi_k\alpha_{k+1} \ldots\alpha_i\alpha_{i+1} \ldots\alpha_n < \varphi_k\delta_{k+1} \ldots\delta_i\delta_{i+1} \ldots\delta_1$

or $\varphi_i = \varphi_k\alpha_{k+1} \ldots\alpha_i = \varphi_k\delta_{k+1} \ldots\delta_i$

d'après 1 $\alpha_{k+1} < \beta_{k+1}$

donc $\delta_{k+1} < \beta_{k+1}$ ce qui entraîne :

$\varphi_k\delta_{k+1} \ldots\delta_1 < \varphi_k\beta_{k+1} \ldots\beta_m$ ce qui est en contradiction avec 1.

Conclusion

si $\alpha < \beta < \delta$, α , β , $\delta \in V^+$

alors

LONGUEUR(PREFIXE(α,δ)) $\leq$ LONGUEUR(PREFIXE(β,δ))

LONGUEUR(PREFIXE(α,β)) $\geq$ LONGUEUR(PREFIXE(α,δ))

d'où les théorèmes :

Théorème 1

Soient $A_1, A_2, \ldots, A_n$ des éléments de V^+ tels que
$\forall i \in [1,n-1]$, $A_i < A_{i+1}$,

Soit ELEMENT un élément de V^+ tel que ELEMENT $< A_1$

alors

LONGUEUR(PREFIXE(A_j,ELEMENT)) $\geq$ LONGUEUR(PREFIXE(A_{j+1},ELEMENT)) $\forall j \in [1,n-1]$

Théorème 2

Soient $A_1, A_2, \ldots, A_n$ des éléments de V^+ tels que
$\forall i \in [1,n-1]$, $A_i < A_{i+1}$,

Soit ELEMENT un élément de V^+ et $j \in [1,n-1]$ tel que
$A_j <$ ELEMENT $< A_{j+1}$
alors

LONGUEUR(PREFIXE(A_i,ELEMENT)) $\leq$ LONGUEUR(PREFIXE(A_j,ELEMENT)) $\forall i \in [1,j-1]$
LONGUEUR(PREFIXE(A_{j+1},ELEMENT)) $\geq$ LONGUEUR(PREFIXE(A_i,ELEMENT)) $\forall i \in [j+1,n]$

Ce théorème est également vrai pour la relation " $< \& \neg C$" (par définition de cette relation) d'où le théorème :

Théorème 3

Soient ELEMENT ϵ V^+ et deux éléments A_i et A_{i+1} de D tels que A_{i+1} soit frere direct de A_i vérifiant la relation

$$A_i < \text{ELEMENT} < A_{i+1}$$

alors A_i et A_{i+1} contiennent les plus grands préfixes possibles avec ELEMENT ou A_i est le plus grand préfixe de ELEMENT.

L'algorithme de recherche est donc le suivant :

1°) Parcours des éléments dominants de D jusqu'à ce que l'on ait trouvé un j tel que
A_j < ELEMENT. Si PREFIXE(A_j, ELEMENT) = A_j alors l'algorithme est terminé, sinon

2°) Parcours des éléments FILS(A_j) et des éléments FILS(A_{j+1}) et sélection du plus grand préfixe.

III.1.2. - Aspects pratiques

Tous les éléments dominants sont implantés de la façon suivante :

$$l_i b_{i_1} b_{i_2} \ldots b_{il_i} \quad frd_i \; fsd_i \; attribut_i$$

où l_i : est la longueur de l'élément $B_i = b_{i_1} b_{i_2} \ldots b_{il_i}$

frd_i : est le pointeur sur le frère direct de B_i

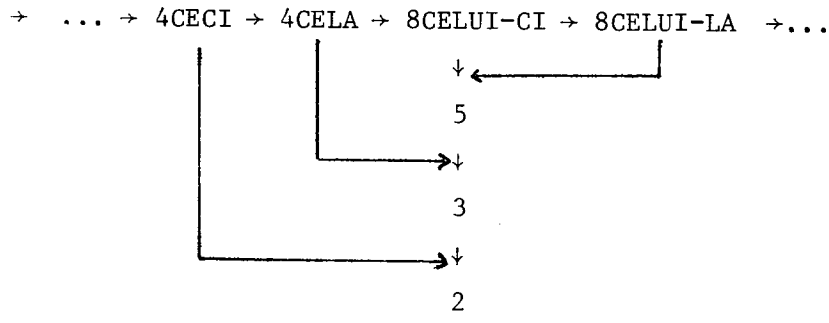
fsd_i : est le pointeur sur le fils direct de B_i

$attribut_i$: est une liste de pointeurs sur les renseignements linguistiques de l'élément Bi.

Quand aux éléments appartenant à l'ensemble FILS, il suffit d'indiquer pour chacun d'eux, sa longueur, le pointeur sur son fils direct, la liste de pointeurs sur les renseignements linguistiques. Ils sont donc de la forme :

l_i fsd_i $attribut_i$

Exemple :



D'autre part, on dispose des primitives suivantes :

Soit NOEUD un élément dominant

NOEUD : l_i b_{i_1} b_{i_2} ... b_{il_i} frd_i fsd_i $attribut_i$

LONGUEUR(NOEUD) = l_i

FREREDIRECT(NOEUD) = frd_i

FILSDIRECT(NOEUD) = fsd_i

De plus on dispose des fonctions sur les chaînes de V^+ :

EXTRCHAINE($X_1X_2...X_n$,i) = $X_1X_2...X_i$ $\quad 1 < i \leq n$

INFER(α,β), SUPER(α,β), EGAL(α,β) qui permettent la comparaison de deux chaînes de caractères :

INFER(α,β) = $\alpha < \beta$

SUPER(α,β) = $\alpha > \beta$

EGAL(α,β) = $\alpha = \beta$

Le prédicat RECHERCHECLASSE délivre la valeur

- VRAI - si un élément dominant de D est un préfixe de ELEMENT, RESULTAT délivre l'adresse de cet élément.

  - si un élément de FILS(α) U FILS(β) avec α < ELEMENT < β , β frere direct de α est le plus grand préfixe possible de ELEMENT.
    RESULTAT délivre l'adresse de cet élément.

- FAUX dans les autres cas.

ENTREE est le point d'entrée dans le dictionnaire, c'est-à-dire le pointeur sur le plus petit élément du dictionnaire qui ne soit préfixe d'un autre.

```
PREDICAT  RECHERCHECLASSE (ENTREES:ELEMENT, ENTREE, SORTIE : RESULTAT)
    |     RESULTAT ← PRECEDENT ← NOEUD ← ENTREE
    |     ELEM ← EXTRCHAINE(ELEMENT, LONGUEUR(NOEUD))
    |     TANTQUE SUPER (ELEM, CHAINE(NOEUD))
    |         FAIRE
    |           |  NOEUD ← FREREDIRECT(PRECEDENT ← NOEUD)
    |           |  ELEM ← EXTRCHAINE(ELEMENT, LONGUEUR(NOEUD))
    |         FINFAIRE
    |     SI INFER(ELEM, CHAINE(NOEUD))
    |         ALORS
    |           |  RECHERCHECLASSE ← RECHERCHEFILS(ELEMENT, PRECEDENT,
    |           |                                  NOEUD,RESULTAT).
    |         SINON
    |           |  RECHERCHECLASSE ← VRAI
    |           |  RESULTAT ← NOEUD
    |         FINSI
FINPREDICAT
```

Le predicat RECHERCHEFILS délivre la valeur

- VRAI si on a trouvé un élément ϵ FILS(α) $\cup$ FILS(β) qui soit la plus longue superposition avec ELEMENT.
  RESULTAT délivre l'adresse de cet élément.

- FAUX dans les autres cas.

PRECEDENT et NOEUD sont les adresses respectives de α et de β .

Le predicat RECHFILS délivre la valeur

- VRAI si l'élément cherché ϵ FILS(γ)
  RESULT délivre l'adresse de cet élément

- FAUX dans les autres cas.

ENTREE est l'adresse de l'élément γ .

```
PREDICAT   RECHERCHEFILS(ENTREES : ELEMENT, PRECEDENT, NOEUD, SORTIE RESULTAT)
|
|          TROUVE1  ← RECHFILS(ELEMENT, PRECEDENT, NOEUD1)
|          TROUVE2 ← RECHFILS(ELEMENT, NOEUD, NOEUD2)
|          SI TROUVE1 ET TROUVE2
|             ALORS
|             |    SI LONGUEUR(NOEUD1) > LONGUEUR(NOEUD2)
|             |       ALORS
|             |       |    RESULTAT ← NOEUD1
|             |       SINON
|             |       |    RESULTAT ← NOEUD2
|             |       FINSI
|             SINON
|             |    SI TROUVE1
|             |       ALORS
|             |       |    RESULTAT ← NOEUD1
|             |       SINON
|             |       |    SI TROUVE2
|             |       |       ALORS
|             |       |       |    RESULTAT ← NOEUD2
|             |       |       FINSI
|             |       FINSI
|             FINSI
|          RECHERCHEFILS ← TROUVE1 OU TROUVE2
FINPREDICAT
```

```
PREDICAT   RECHFILS (ENTREES : ELEMENT, ENTREE, SORTIE : RESULT)
           RECHFILS ← FAUX
           NOEUD ← FILSDIRECT(ENTREE)
           PARCOURS ← VRAI
           TANTQUE NOEUD ≠ NIL ET PARCOURS
             FAIRE
               ELEM ← EXTRCHAINE(ELEMENT, LONGUEUR(NOEUD))
               SI SUPER(CHAINE(NOEUD), ELEM)
                  ALORS
                     NOEUD ← FILSDIRECT(NOEUD)
                  SINON
                     PARCOURS ← FAUX
                     RECHFILS ← EGAL(CHAINE(NOEUD),ELEM)
                     RESULT ← NOEUD
               FINSI
             FINFAIRE
FINPREDICAT
```

III.1.3. - Coût de l'algorithme

Le coût d'un algorithme de recherche séquentielle dans une liste ordonnée est en moyenne égal à $\frac{N}{2}$ où N est le nombre d'éléments de cette liste dans le cas où tous les éléments sont équiprobables.

Pour notre problème, on a en général 3 parcours :
- un premier parcours des éléments dominants
- deux parcours sur des éléments préfixes d'autres éléments.

Le nombre des éléments dominants étant très supérieur aux nombres d'éléments des deux autres listes, on peut dire que le coût moyen est de l'ordre de $\frac{N}{2}$ où N est le nombre d'éléments dominants. N étant dans la pratique de l'ordre de 10000, il est nécessaire de trouver une organisation qui minimise ce coût. Pour ce faire, on va découper la liste en p sous-listes au moyen d'un arbre binaire ordonné par ordre alphabétique.

III.2 - Organisation du dictionnaire

Le dictionnaire est donc composé :

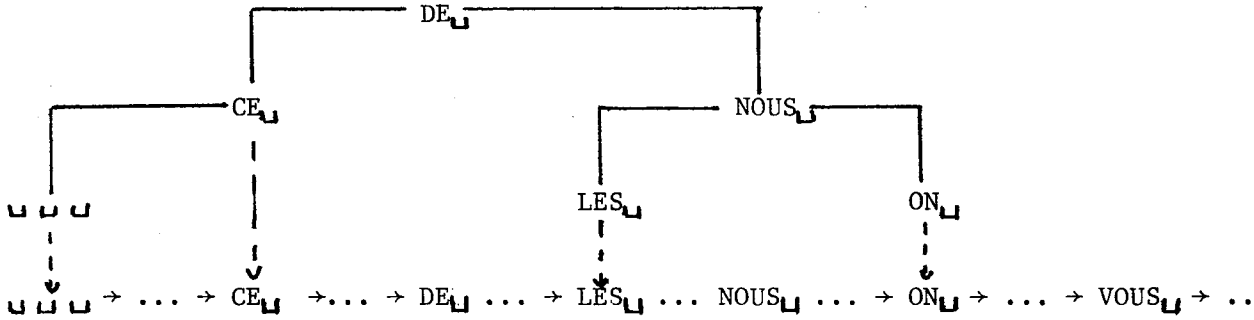
- d'un arbre binaire ordonné par ordre alphabétique (D1) qui permettra l'accès à une sous-liste.

- d'une liste chaînée par ordre alphabétique (D2) tenant compte des relations frere-direct et fils direct composée de p sous-listes.

Le problème qui reste posé est de savoir comment relier les deux structures D1 et D2.

D1 est un arbre binaire ordonné qui ne contient que des éléments dominants.

Exemple :



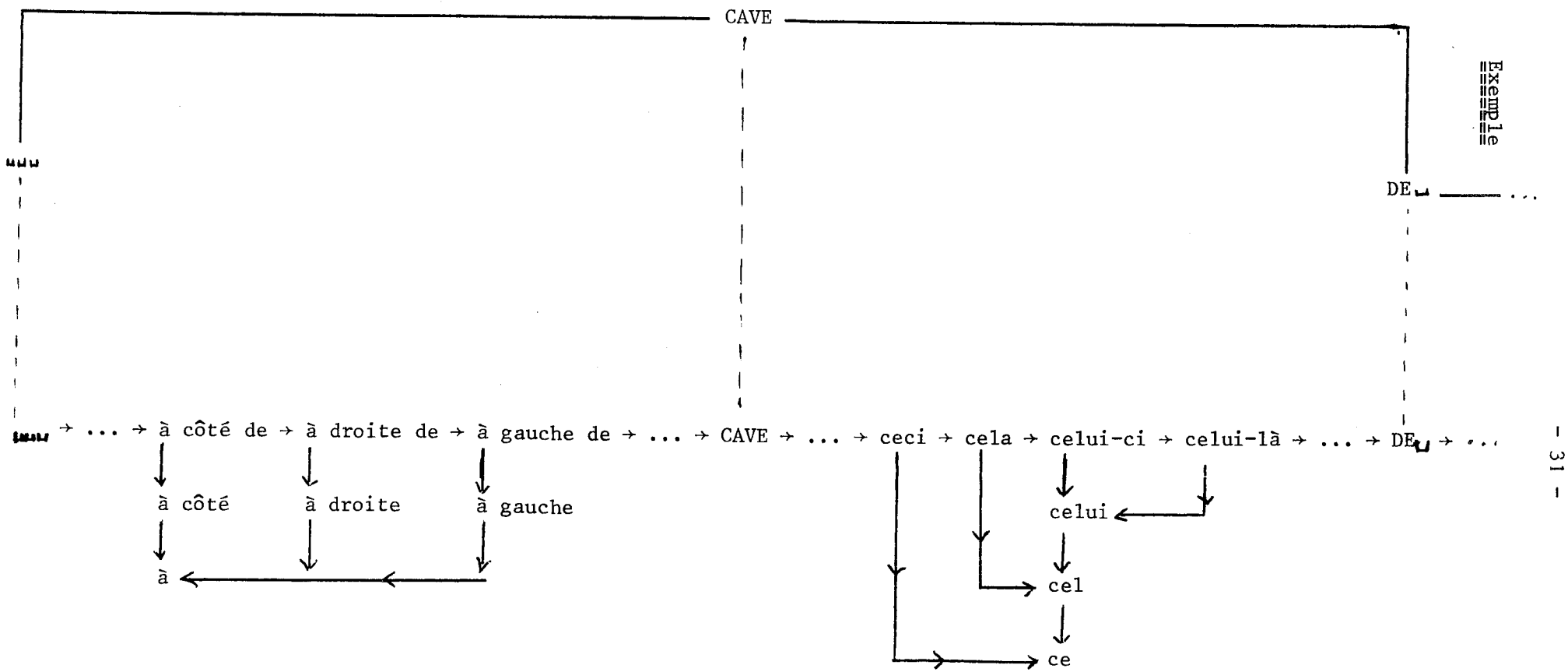
Soient $A_1, A_2, \ldots, A_n$, les n éléments dominants de l'arbre binaire ordonné ($A_i < A_{i+1}$, $\forall i \in [1,n-1]$). Ces n éléments déterminent des classes dans la liste chaînée où chaque classe est définie par :

$$C(A_i) = E(A_i) \cup \{FILS(\alpha) / \alpha \in E(A_i)\}$$

où

$$E(A_i) = \{\alpha \text{ élément dominant } \in D / A_i \leq \alpha < A_{i+1}\}$$
$$\forall i \in [1,n-1]$$

$$E(A_n) = \{\alpha \text{ élément dominant } \in D / A_n \leq \alpha\}$$



classe(␣␣␣) = { ␣␣␣ ,... , à côté de, à droite de,
à gauche de, ..., à côté, à droite
à gauche, à}

classe(cave) = {cave ,..., ceci, cela, celui-ci,
celui-là ,..., celui, cel, ce}

III.2.1. - Construction de l'arbre (D1)

Il est aisé de constater que dans un texte écrit en langue naturelle (de même que dans un programme écrit en langage évolué) certains mots apparaissent plus souvent que d'autres. D'où l'idée de construire un dictionnaire, donc un arbre, qui tienne compte de la fréquence d'utilisation de ses éléments. En consultant l'importante littérature consacrée à ce sujet [KNU1, KNU2, COUS, MAHL, VEILL2, PAIR1], nous avons dans une première étape choisi, pour construire cet arbre, l'algorithme de KNUTH [KNU1, KNU2] qui tient compte non seulement de la fréquence d'utilisation des éléments appartenant à l'arbre mais également de la fréquence des éléments n'y appartenant pas.

Nous donnons, ci-dessous, le principe de cet algorithme.

III.2.1.1. - Algorithme de Knuth [KNU1, KNU2]

Le but de cet algorithme est de construire un arbre binaire ordonné qui minimise le temps de recherche global compte tenu :

a) de l'ordre alphabétique des n éléments A_i de l'arbre :

$$A_i < A_{i+1} \quad \forall i \in [1,n-1]$$

b) de la fréquence $\alpha_i \geq 0$ des éléments A_i

c) de la fréquence $\beta_i \geq 0$ des éléments B_i n'appartenant pas à l'arbre et tel que :

$$B_i \in C(A_i) \qquad i \in [1:n-1]$$

$$B_o < A_1 \quad \text{et} \quad B_n > A_n$$

Pour ce faire, on définit un coût global C :

$$C = \sum_{i=1}^{n} l(A_i)\alpha_i + \sum_{i=0}^{n} l(B_i)\beta_i$$

où $l(A_i)$ est le niveau du noeud A_i

et $l(B_i)$ est le niveau de l'élément B_i

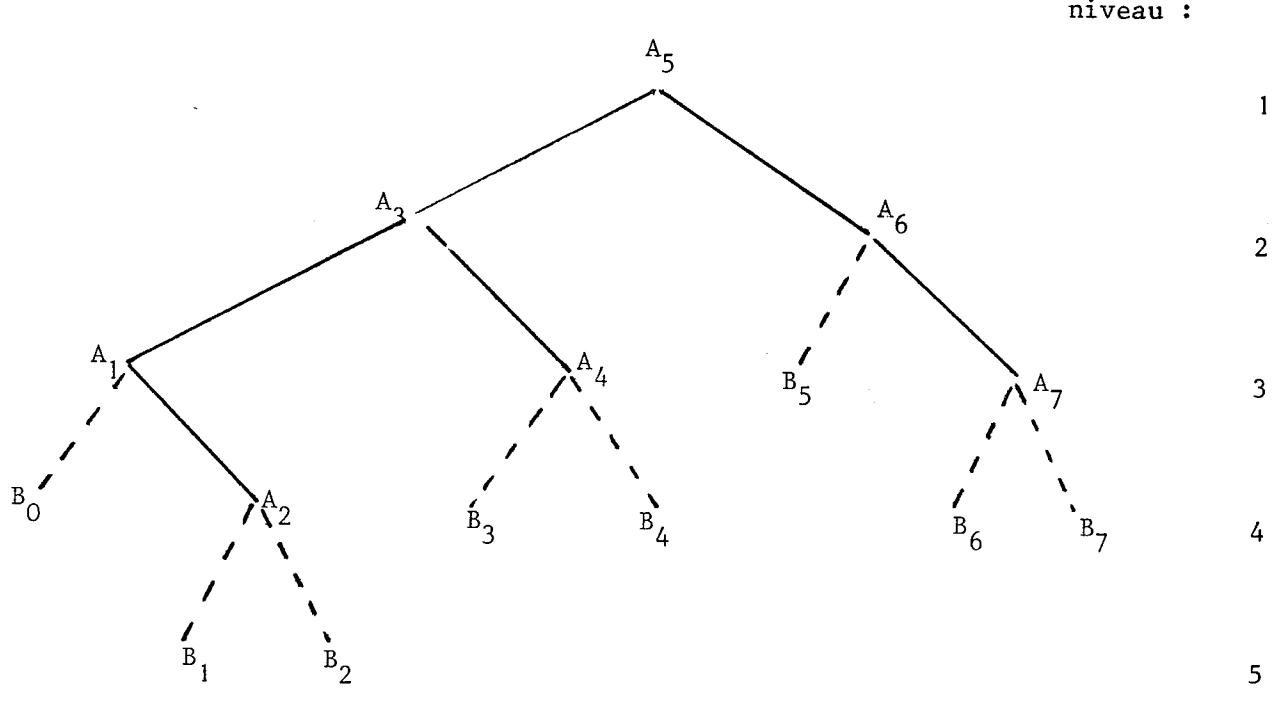
où le niveau d'un noeud est défini par :

l(racine) = 1

l(noeud) = l(père(noeud))+1 où noeud ≠ racine.

Exemple :

Soient 7 éléments $A_1, A_2, \ldots, A_7$ avec $A_1 < A_2 < A_3 \ldots < A_7$ et un arbre binaire ordonné :



Le coût C est égal à :

$$C = 3\alpha_1 + 4\alpha_2 + 2\alpha_3 + 3\alpha_4 + \alpha_5 + 2\alpha_6 + 3\alpha_7$$

$$+ 4\beta_0 + 5\beta_1 + 5\beta_2 + 4\beta_3 + 4\beta_4 + 3\beta_5 + 4\beta_6 + 4\beta_7$$

Si A_r est la racine de l'arbre, le coût C peut s'écrire :

$$C = \sum_{i=1}^{n} l(A_i)\alpha_i + \sum_{i=0}^{n} l(B_i)\beta_i$$

$$= \sum_{i=1}^{r-1} (l(A_i)-1)\alpha_i + \sum_{i=r+1}^{n} (l(A_i)-1)\alpha_i + \sum_{i=1}^{r-1} \alpha_i + \sum_{i=r+1}^{n} \alpha_i + \alpha_r$$

$$+ \sum_{i=0}^{r-1} (l(B_i)-1)\beta_i + \sum_{i=r}^{n} (l(B_i)-1)\beta_i + \sum_{i=0}^{n} \beta_i$$

$$= \sum_{i=1}^{r-1} (l(A_i)-1)\alpha_i + \sum_{i=0}^{r-1} (l(B_i)-1)\beta_i + \sum_{i=r+1}^{n} (l(A_i)-1)\alpha_i + \sum_{i=r}^{n} (l(B_i)-1)\beta_i$$

$$+ \sum_{i=1}^{n} \alpha_i + \sum_{i=0}^{n} \beta_i$$

$$= C_G + C_D + S$$

où C_G est le coût du sous-arbre gauche, C_D le coût du sous-arbre droit et S la somme des fréquences α_i et β_i.

Or ce que l'on cherche, c'est à construire un arbre tel que le coût C soit minimum.

$$\min(C) = \min(C_G + C_D + S) = \min(C_G) + \min(C_D) + S$$

Conclusion

Les sous-arbres gauche et droit d'un arbre ayant un coût minimum ont un coût minimum.

Soit r(i,j) la racine de l'arbre t(i,j) ayant un coût minimum C(i,j) étant donné $\beta_i \ldots \beta_j$ et $\alpha_{i+1} \ldots \alpha_j$ $\forall i,j$ avec $0 \leq i < j \leq n$.

L'algorithme est déterminé par le calcul des formules :

$$
\text{**} \quad \begin{cases} C(i,i) = S(i,i) = \beta_i & 0 \leq i \leq n \\ S(i,j) = S(i,j-1)+\alpha_j + \beta_j & 0 \leq i < j \leq n \\ C(i,j) = S(i,j) + \min(C(i,k-1)+C(k,j)) & i < k \leq j \\ \qquad = S(i,j)+C(i,r(i,j)-1)+C(r(i,j),j) \quad \text{pour} & 0 \leq i < j \leq n \end{cases}
$$

Le principe est donc de construire en partant de gauche à droite et de bas en haut les sous-arbres ayant un coût minimum jusqu'à obtenir l'arbre complet.

C'est-à-dire que l'on va calculer :

r(0,1), r(1,2) ..... r(n-1,n) puis
r(0,2), r(1,3) ..... r(n-2,n)
.....
.....
r(0,n-1), r(1,n) et
r(0,n)

Démontrons que cet algorithme demande un temps de calcul proportionnel à n^3.

Pour chaque calcul de r(i,j) il faut examiner j-i valeurs donc pour :

r(0,1), r(1,2) ..... r(n-1,n) il faut calculer 1 × n valeurs
r(0,2), r(1,3) ..... r(n-2,n) il faut calculer 2 × (n-1) valeurs
.....
.....
r(0,n-1), r(1,n) il faut calculer (n-1) * (n-(n-2)) valeurs
r(0,n) il faut calculer n * (n-(n-1)) valeurs

On arrive donc à :

$$\sum_{i=1}^{n} i(n-(i-1)) = n \sum_{i=1}^{n} i - \sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i = (n+1) \sum_{i=1}^{n} i - \sum_{i=1}^{n} i^2$$

$$= (n+1)\frac{n(n+1)}{2} - \frac{n(n+1)(2n+1)}{6} = \boxed{\frac{n(n+1)(n+2)}{6}}$$

Dans son article [KNU2] Knuth démontre le théorème suivant :

Théorème :

Si on ajoute un nouveau noeud A_{n+1} tel que $A_{n+1} > A_n$, la racine de l'arbre ne se déplace jamais à gauche.
C'est-à-dire que $r(0,n) \leq r(0,n+1)$ pour $n \geq 1$.

On peut en déduire un théorème similaire en ajoutant un nouveau noeud plus petit que tous les autres.

Par application de ces résultats à tous les sous-arbres, on peut établir le corollaire suivant :

Corollaire

Il y a toujours une solution aux équations ** telle que :

$$\boxed{r(i,j-1) \leq r(i,j) \leq r(i+1,j) \quad \text{pour } 0 \leq i < j-1 \leq n}$$

Ce corollaire permet d'améliorer l'efficacité de l'algorithme. En effet on peut démontrer qu'il est en $O(n^2)$.

Démonstration

Pour calculer r(i,j), on n'a plus besoin d'examiner j-i valeurs mais uniquement r(i+1,j)-r(i,j-1)+1 valeurs.
En conséquence :

r(0,1), r(1,2) ..... r(n-1,n) demande le calcul de n valeurs
r(0,2), r(1,3) ..... r(n-2,n) demande le calcul de r(n-1,n)-r(0,1)+n-1 valeurs
.....
.....
r(0,n-1), r(1,n) demande le calcul de r(2,n)-r(0,n-2)+2 valeurs
r(0,n) demande le calcul de r(1,n)-r(0,n-1)+1 valeurs

On arrive donc à

$$\sum_{i=1}^{n-1} r(n-i,n)-r(0,i) + \sum_{i=1}^{n} i < \sum_{i=1}^{n-1} n + \sum_{i=1}^{n} i = n(n-1)+\frac{n(n+1)}{2}$$

$$< \frac{3n^2-n}{2}$$

Quant à la place utilisée, il est aisé de voir qu'elle est proportionnelle à n^2.

III.2.1.2. - Algorithme retenu pour la construction de l'arbre

L'algorithme de Knuth est très intéressant lorsque l'on connait les mots qui doivent se trouver dans l'arbre. Dans une application aux langues naturelles, ce choix reste entier et l'utilisateur qui est en général non informaticien se trouve confronté à un problème délicat qu'il résoud en pratique assez mal : en début d'application, il choisit tant bien que mal quelques mots puis ensuite, au fur et à mesure que le dictionnaire augmente, il oublie de mettre à jour cet arbre afin de minimiser le coût de recherche. En conséquence, nous avons essayé de trouver

d'autres algorithmes qui permettent une construction automatique en fonction des mots du dictionnaire afin de simplifier le travail de l'utilisateur qui n'a plus à savoir que le dictionnaire est organisé en deux parties. Ce travail a constitué le sujet de thèse de E. GRANDJEAN [GRAN2].

Coût théorique

Soit n le nombre d'éléments du dictionnaire. Supposons que tous les éléments soient équiprobables, que l'arbre binaire qui contient p éléments soit équilibré et qu'il détermine des classes de tailles sensiblement égales ($= \frac{n}{p}$), alors le coût C est égal à

$$C = \alpha \log_2 p + \beta \frac{n}{2p} + \gamma$$

Cette fonction passe par un minimum lorsque l'on fait varier le nombre p d'éléments de l'arbre pour un nombre n d'éléments total fixé. Par exemple, si le dictionnaire contient 1600 éléments, elle passe par un minimum lorsque p est compris entre 256 et 512. D'autre part, on constate qu'à partir de 128, les gains obtenus deviennent très faibles [GRAN 2].

Pour la détermination des noeuds qui doivent constituer l'arbre, on peut envisager plusieurs stratégies :

1°) Obtenir des classes de tailles identiques.

La taille d'une classe est égale au nombre d'éléments dominants appartenant à cette classe. L'algorithme est simple et donne de bons résultats. [GRAN 2]

2°) Obtenir des classes possédant la même fréquence.

On appelle fréquence d'une classe, la somme des fréquences de ses éléments. L'algorithme est également très simple mais les résultats obtenus sont moins bons. [GRAN 2]

3°) Choisir les éléments de plus grandes fréquences.

On détermine les p éléments dominants qui possèdent la plus grande fréquence. Les résultats sont encore plus mauvais que les deux précédents.

4°) Obtenir des classes de même coût.

Le coût C d'une classe de taille k est égal à :

$$C = \sum_{i=1}^{k} i\ f(i)$$

où i est le rang de l'élément dominant dans la classe et f(i) sa fréquence. Les résultats obtenus sont meilleurs que les précédents.

5°) Obtenir des classes telles que le coût total soit minimal.

Cet algorithme donne naturellement les meilleurs résultats. Nous allons, dans ce qui suit, exposer la méthode que nous avons utilisée.

En fonction des éléments choisis, on peut ensuite soit construire un arbre binaire équilibré, soit faire appel à l'algorithme de Knuth si on désire obtenir le meilleur arbre possible.

III.2.1.2.1. - Algorithmes

Le coût d'une recherche séquentielle dans une liste ordonnée de n éléments de fréquences f(i) est égal à :

$$C = \sum_{i=1}^{n} i\ f(i)$$

Il faut décomposer cette liste de n éléments en p sous-listes de sorte que la somme des coûts des p sous-listes soit minimum.

On doit trouver K_j $\quad 1 \leq K_j < n \quad j \in [1,p-1]$ tel que :

$$\sum_{j=1}^{p-1} \sum_{i=K_j+1}^{k_{j+1}-1} (i-K_j)\, f(i) + \sum_{i=K_p+1}^{n} (i-K_p) \quad \text{soit minimum.}$$

Appelons *

$V_q(i)$ le coût minimum quand on a les i premiers mots du dictionnaire à explorer en q sous-listes (q ≥ 2).

$$V_q(i) = \min_{q \le K_q \le i} \left[\sum_{j=K_q+1}^{i} (j-K_q)\, f(j) + V_{q-1}(K_q-1) \right]$$

On cherche $V_p(n)$.

Pour q = 1, on connaît K_q qui est égal à 1, d'où l'algorithme :

---

* Nous tenons à remercier ici l'équipe de recherche opérationnelle pour l'aide précieuse qu'elle nous a fournie pour la construction de cet algorithme.

```
ACTION   SAKA(ENTREES : F, N, P, SORTIE : K)
|
|   S ← 0
|   I ← 2
|   TANTQUE I ≤ N
|       FAIRE
|       |   S ← S + (I-1)*F(I)
|       |   V(1,I) ← S
|       |   K(1,I) ← 1
|       |   I ← I+1
|       FINFAIRE
|   Q ← 2
|   TANTQUE Q ≤ P
|       FAIRE
|       |   I ← Q+1
|       |   TANTQUE I ≤ N
|       |       FAIRE
|       |       |   KQ ← Q
|       |       |   MIN ← PLUSGRANDEVALEURPOSSIBLE
|       |       |   TANTQUE KQ ≤ I
|       |       |       FAIRE
|       |       |       |   J ← KQ+1
|       |       |       |   S ← 0
|       |       |       |   TANTQUE J ≤ I
|       |       |       |       FAIRE
|       |       |       |       |   S ← S+(J-KQ)*F(J)
|       |       |       |       |   J ← J+1
|       |       |       |       FINFAIRE
|       |       |       |   S ← S+V(Q-1,KQ-1)
|       |       |       |   SI S < MIN
|       |       |       |       ALORS
|       |       |       |       |   MIN ← S
|       |       |       |       |   KB ← KQ
|       |       |       |       FINSI
|       |       |       |   KQ ← KQ+1
|       |       |       FINFAIRE
|       |       |   V(Q,I) ← MIN
|       |       |   K(Q,I) ← KB
|       |       |   I ← I+1
|       |       FINFAIRE
|       |   Q ← Q+1
|       FINFAIRE
|
FINACTION
```

Evaluation du coût de l'algorithme

- Evaluation du nombre d'itérations

Ce nombre d'itérations est égal à :

$$\sum_{q=2}^{p} \sum_{i=q+1}^{n} \sum_{k_q=q}^{i} \sum_{j=k_q+1}^{i} C^{te}$$

a) $$\sum_{j=k_q+1}^{i} C = (i-k_q)c$$

b) $$\sum_{k_q=q}^{i} (i-k_q) = i(i-q+1) - \frac{i(i+1)}{2} + \frac{(q-1)(q)}{2}$$

$$= \frac{i^2}{2} - \frac{i}{2}(2q-1) + \frac{q(q-1)}{2}$$

c) $$\sum_{i=q+1}^{n} \frac{i^2}{2} - \frac{i}{2}(2q-1) + \frac{q(q-1)}{2}$$ est égal à :

$$\frac{n(n+1)(2n+1)}{12} - \frac{q(q+1)(2q+1)}{12} - \frac{n(n+1)}{4}(2q-1)$$

$$+ \frac{q(q+1)}{4}(2q-1) + \frac{q(q-1)}{2}(n-q)$$

en développant et en ordonnant par rapport à q, on trouve :

$$-\frac{1}{6}q^3 + \frac{n+1}{2}q^2 - \frac{3n^2+6n+2}{6}q + \frac{n(n+2)(n+1)}{6}$$

d) $$\sum_{q=2}^{p} -\frac{1}{6}q^3 + \frac{n+1}{2}q^2 - \frac{3n^2+6n+2}{6}q + \frac{n(n+2)(n+1)}{6}$$ est égal à :

$$-\frac{1}{6}\left[\frac{p^4+2p^3+p^2-4}{4}\right] + \frac{n+1}{2}\left[\frac{2p^3+3p^2+p-6}{6}\right]$$

$$-\frac{3n^2+6n+2}{6}\left[\frac{p^2+p-2}{2}\right] + (p-1)\frac{n(n+2)(n+1)}{6}$$

En développant et en ordonnant par rapport à n, on trouve :

$$\frac{p-1}{6}\,n^3 - \frac{p^2-p}{4}\,n^2 + \frac{2p^3-3p^2-p+2}{12}\,n + \frac{-p^4+2p^3+p^2-2p}{24}$$

On peut mettre $\frac{p-1}{24}$ en facteur, d'où le coût :

$$\boxed{\frac{p-1}{24}\,[4n^3-6pn^2+(4p^2-2p-4)n-p^3+p^2+2p]}$$

A ce coût, il faudrait ajouter le coût de la première itération (n-1) qui permet l'initialisation de V(1,I) et de K(1,I).

Evaluation de l'encombrement mémoire :

Il résulte principalement de l'encombrement des deux matrices V et K et du tableau F. Il est donc de l'ordre de n(2p+1).

Optimisation de l'algorithme

Cet algorithme est coûteux en temps et en place. On peut donner un algorithme qui optimise le coût en temps et le coût en place :

a) Optimisation sur le nombre d'itérations

On peut optimiser l'itération sur les KQ. En effet, au lieu de partir de KQ = Q, il suffit de partir de KQ = K(Q-1,I). C'est à dire que :

$$V_q(I) = \min_{K(Q-1,I)\le K_q\le i} \left[\sum_{j=K_q+1}^{i} (j-K_q)\, f(j) + V_{q-1}\,(K_q-1) \right]$$

Pour ce faire, il faut montrer que, pour chaque pas, on a :

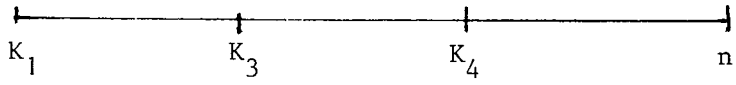
$$K(Q,I) \geq K(Q-1,I).$$

Montrons le pour Q = 3 et I = N. (Ce résultat est généralisable

$\forall Q \in [4,P]$, $I \in [Q+1,N]$)

au premier pas (Q=2),on a obtenu K_2

Montrons qu'au deuxième pas (Q = 3) on obtient K_3, K_4 avec $K_4 \geq K_2$



Pour Q = 2 le coût A est égal à :

$$A = \sum_{i=2}^{K_2-1} (i-1)f(i) + \sum_{i=K_2+1}^{n} (i-K_2)f(i)$$

Pour Q = 3 le coût B est égal à :

$$B = \sum_{i=2}^{K_3-1} (i-1)f(i) + \sum_{i=K_3+1}^{K_4-1} (i-K_3)f(i) + \sum_{i=K_4+1}^{n} (i-K_4)f(i)$$

Posons $K_4 = K_2 - \alpha$, $i-K_3 = i-1-K_3+1$ et $i-K_4 = i-1-K_2+\alpha+1$

d'où

$$B = \sum_{i=2}^{K_3-1} (i-1)f(i) + \sum_{i=K_3+1}^{K_2-\alpha-1} (i-1)f(i) - \sum_{i=K_3+1}^{K_2-\alpha-1} (K_3-1)f(i) + \sum_{i=K_2-\alpha+1}^{K_2-1} (i-1)f(i)$$

$$- \sum_{i=K_2-\alpha+1}^{K_2-1} (K_2-\alpha-1)f(i) + \sum_{i=K_2}^{n} (i-K_2)f(i) + \alpha \sum_{i=K_2}^{n} f(i)$$

$$B = \sum_{i=2}^{K_2-1} (i-1)f(i) + \sum_{i=K_2}^{n} (i-K_2)f(i)$$

$$- \left[\sum_{i=K_3}^{K_2-\alpha-1} (K_3-1)f(i) + \sum_{i=K_2-\alpha}^{K_2-1} (K_2-\alpha-1)f(i) - \sum_{i=K_2}^{n} f(i) \right]$$

d'où

$$B = A - D(\alpha)$$

où

$$D(\alpha) = \sum_{i=K_3}^{K_2-\alpha-1} (K_3-1)f(i) + \sum_{i=K_2-\alpha}^{K_2-1} (K_2-\alpha-1)f(i) + \alpha \sum_{i=K_2}^{n} f(i)$$

Or A est minimum, donc pour que B soit minimum, il faut que $D(\alpha)$ soit maximum.

Etudions $D(\alpha)$.

$$D(\alpha) = \sum_{i=K_3}^{K_2-\alpha-1} (K_3-1)f(i) + \sum_{i=K_2-\alpha}^{n} (K_2-\alpha)f(i) - \sum_{i=k_2-\alpha}^{k_2-1} f(i) - \sum_{i=K_2}^{n} k_2 f(i)$$

or

$\sum_{i=K_2}^{n} k_2 f(i)$ est une constante.

a) $\alpha > 0$.

$$\sum_{i=K_3}^{K_2-\alpha-1} (K_3-1)f(i) \leq \sum_{i=K_3}^{K_2-1} (K_3-1)f(i)$$

$$\sum_{i=K_2-\alpha}^{n} (K_2-\alpha)f(i) \leq \sum_{i=K_2}^{n} K_2 f(i) \quad \text{puisque A est minimum}$$

$$\sum_{i=k_2-\alpha}^{k_2-1} f(i) \geq \sum_{i=K_2}^{k_2-1} f(i) = 0.$$

En conclusion $D(\alpha) \leq D(0)$ pour $\alpha > 0$.

b) $\alpha < 0$

$$\sum_{i=K_3}^{K_2-\alpha-1} (K_3-1)f(i) \geq \sum_{i=K_3}^{K_2-1} (K_3-1)f(i)$$

$$\sum_{i=K_2-\alpha}^{n} (K_2-\alpha)f(i) \leq \sum_{i=K_2}^{n} K_2\, f(i) \quad \text{puisque A est minimum}$$

$$\sum_{i=K_2-\alpha}^{k_2-1} f(i) = 0.$$

Ici, on ne peut rien dire entre $D(\alpha)$ et $D(0)$.

Conclusion α n'est jamais positif. Comme $K_4 = K_2 - \alpha$, c'est que $K_4 \geq K_2$.

Le calcul de l'évaluation du nombre d'itérations est impossible car il dépend de la répartition. Expérimentalement, on constate que le gain est très important (cf. p.51)

b) Optimisation sur la place

Au pas Q, on calcule les V(Q,I) et les K(Q,I).
Pour calculer un V(Q,I), on a besoin de V(Q-1,I). On peut donc optimiser en ne gardant qu'une matrice V de dimension 2n. Pour ce faire, on peut utiliser dans le schéma d'algorithme une variable booléenne B pour savoir si l'on utilise V(1,I) ou V(2,I). Le changement de valeur se faisant par l'instruction d'affectation B ← non B. Un procédé plus simple consiste à trouver une relation telle que si J est égal à 1, sa prochaine valeur sera 2 et vice-versa. Il suffit d'utiliser l'instruction d'affectation J ← 3-J.

Quant à K(Q, I) , on est obligé de les garder tous puisque

K(p,n), K(p-1, K(p,n)-1)

K(p-2, K(p-1, K(p,n))-1),... etc sont les solutions de l'algorithme.

Cet algorithme, bien qu'optimisé, étant très coûteux, nous avons cherché un algorithme simple et moins coûteux qui permette d'approcher le résultat. Le principe de l'algorithme est le suivant : ayant déterminé q sous-listes, on détermine la q+1ème sous-liste en supposant que les éléments K_1, K_2,...,K_q du pas précédent sont valables au pas q+1.

On procède de la façon suivante : ayant déterminé $K_1, K_2, \ldots, K_q$ et les coûts $C_1, C_2, \ldots, C_q$ de chaque sous-liste, on partage la sous-liste ayant le plus grand coût en deux sous-listes telles que la somme du coût gauche et du coût droit soit minimum.

Principe:

Au premier pas, il faut trouver K_2 tel que

$$\sum_{i=2}^{K_2-1} i\, f(i) + \sum_{i=K_2+1}^{n} (i-K_2)\, f(i) \quad \text{soit minimum}$$

donc que

$$\sum_{i=2}^{n} i\, f(i) - K_2 \sum_{i=K_2}^{n} f(i) \text{ soit minimum.}$$

Cette expression est minimum lorsque $K_2 \sum_{i=K_2}^{n} f(i)$ est maximum.

Ayant déterminé K_2, on calcule C_1 et C_2 les coûts respectifs de la 1ère et de la 2ème sous-liste. On réitère le processus avec la sous-liste ayant le plus grand coût et ainsi de suite jusqu'à obtenir p sous-listes. D'où les schémas d'algorithme :

```
ACTION   GRAND(ENTREES : F, N, P SORTIE : K)
|
|        K(1) ← 1
|        K(2) ← N+1
|        COUT(1) ← PLUSGRANDEVALEURPOSSIBLE
|        COUT(2) ← 0
|        Q ← 2
|        TANTQUE Q ≤ P
|           FAIRE
|           |     COUTMAX(COUT,Q,K,INDG)
|           |     MILIEU(F,N,K,INDG,MIL,MCOUT)
|           |     INSERER(K,COUT,Q+1,MIL,MCOUT)
|           |     Q ← Q+1
|           FINFAIRE
FINACTION
```

```
ACTION   COUTMAX(ENTREES : COUT, NK, K, SORTIE : INDG)
|
|        MAX ← COUT(1)
|        I ← 2
|        INDG ← 1
|        TANTQUE I ≤ NK
|           FAIRE
|           |     SI MAX < COUT(I)
|           |        ALORS
|           |        |     MAX ← COUT(I)
|           |        |     INDG ← I
|           |        FINSI
|           |     I ← I+1
|           FINFAIRE
FINACTION
```

```
ACTION   MILIEU(ENTREES : F, K, INDG SORTIES : MIL, MCOUT)
    |    J ← K(INDG+1)-1
    |    MAX ← 0
    |    S ← 0
    |    MIL ← J
    |    L ← K(INDG)
    |    TANTQUE J > L
    |       FAIRE
    |          |    S ← S+F(J)
    |          |    C = (J-L)*S
    |          |    SI MAX < C
    |          |      ALORS
    |          |         |    MAX ← C
    |          |         |    MIL ← J
    |          |      FINSI
    |          |    J ← J-1
    |       FINFAIRE
    |    CO CALCUL DES COUTS GAUCHE ET DROIT CO
    |    CALCULCOUT (K(INDG), MIL, F, COUT(INDG))
    |    CALCULCOUT(MIL, K(INDG+1), F, MCOUT)
FINACTION
```

```
ACTION   CALCULCOUT(ENTREES : BORNEG, BORNED, F SORTIE COUT)
    |    S ← 0
    |    J ← BORNEG+1
    |    TANTQUE J < BORNED
    |       FAIRE
    |          |    S ← S+(J-BORNEG)*F(J)
    |          |    J ← J+1
    |       FINFAIRE
    |    COUT ← S
FINACTION
```

Quant à l'action INSERER, elle trie par insertion de MIL et de MCOUT, les $Q+1$ premiers éléments de K et de COUT.

Evaluation du nombre d'itérations

Le coût dépend principalement du nombre d'itérations de l'action MILIEU.

Pour Q = 2 on effectue n-1 itérations

pour Q = 3 on effectue soit K_2-1, soit $n-K_2+1$ itérations

etc...

On peut donc dire que C_{milieu} est très inférieur à (p-1)(n-1).

Le coût total de l'algorithme est inférieur à : (p-1)(n-1+p)

Evaluation de la place :

2 tableaux de p+1 éléments pour COUT et K et un tableau de n éléments pour F. L'encombrement mémoire est donc de l'ordre de 2(p+1)+n .

Expérimentalement, on constate que les résultats fournis par cet algorithme sont très proches de ceux obtenus par l'action SAKA. Naturellement, si on appelle C_q^S et C_q^G les coûts totaux de recherche lorsque le dictionnaire est divisé en q sous-listes pour l'algorithme "SAKA" et l'algorithme "GRAND", on a :

$$C_q^S \leq C_q^G \qquad \forall q \in [1,p\ ]$$

On trouve ci-dessous un tableau donnant les temps d'exécution pour l'algorithme SAKA, SAKA optimisé et GRAND.

N	P	SAKA	SAKAOPT	GRAND
30	2	0,26 s	0,26 s	0,07 s
30	5	0,70 s	0,36 s	0,09 s
30	10	1,18 s	0,37 s	0,10 s
30	20	1,52 s	0,48 s	0,11 s
60	2	1,48 s	1,58 s	0,13 s
60	5	5,15 s	2,07 s	0,14 s
60	10	10,30 s	2,27 s	0,13 s
60	20	16,89 s	2,40 s	0,16 s
60	30	20,69 s	2,50 s	0,18 s
117	2	10,12 s	10,42 s	0,21 s
117	5	39,72 s	15,75 s	0,22 s
117	10	83,85 s	17,96 s	0,24 s
117	20	155,80 s	18,88 s	0,26 s
117	30	208 s	18,96 s	0,29 s

Si ensuite on applique l'algorithme de Knuth sur les p noeuds ainsi déterminés, on s'aperçoit que le gain est extrêmement faible par rapport à la construction d'un arbre binaire équilibré. De plus, cette méthode qui est la meilleure des quatre que nous avons testées permet une construction automatique de l'arbre ce qui libère l'utilisateur de toute contrainte pour obtenir un dictionnaire efficace.

IV - ALGORITHME DE RECHERCHE DANS LE DICTIONNAIRE

L'algorithme consiste à effectuer d'abord un parcours de l'arbre et, si l'élément n'a pas été trouvé, un parcours de la classe qui a été sélectionnée.

Le schéma du prédicat RECHERCHEDICTIONNAIRE décrit l'algorithme de parcours du dictionnaire.

```
PREDICAT   RECHERCHEDICTIONNAIRE (ENTREES : ELEMENT, RACINE,
                                  SORTIE : ADRESSE)
    SI RECHERCHEARBRE (ELEMENT, RACINE, ENTREE)
       ALORS
          RECHERCHEDICTIONNAIRE ← VRAI
          ADRESSE ← ENTREE
       SINON
          SI RECHERCHECLASSE (ELEMENT, ENTREE, RESULTAT)
             ALORS
                RECHERCHEDICTIONNAIRE ← VRAI
                ADRESSE ← RESULTAT
             SINON
                RECHERCHEDICTIONNAIRE ← FAUX
                ADRESSE ← NIL
             FINSI
       FINSI
FINPREDICAT
```

IV-1 - Parcours de l'arbre

Chaque élément de l'arbre a été implanté de la manière suivante :

$$l_i \; a_{i_1} \; a_{i_2} \; \cdots \; a_{i l_i} \; g_i \; d_i \; e_i$$

où - l_i est la longueur (nombre de caractères) de l'élément $A_i = a_{i_1} \; a_{i_2} \; \cdots \; a_{i l_i}$

- g_i, d_i, e_i sont trois pointeurs
  - g_i sur le sous-arbre gauche
  - d_i sur le sous-arbre droit
  - e_i sur le point d'entrée de la classe.

Soit ELEMENT = $\delta_1 \; \delta_2 \; \cdots \; \delta_n$ une chaîne à analyser. On recherche la plus longue superposition possible commençant à δ_1 entre cette chaîne et les éléments A_i de l'arbre.

On dispose des fonctions suivantes :

Soit NOEUD : $l \; a_1 \; a_2 \; \cdots \; a_l \; g \; d \; e$ un élément de l'arbre

ADRESSEDROITE(NOEUD) = d
ADRESSEGAUCHE(NOEUD) = g
ADRESSECLASSE(NOEUD) = e
LONGUEUR(NOEUD) = l
CHAINE(NOEUD) = $a_1 a_2 \cdots a_l$

De plus, on dispose de

EXTRCHAINE(φ,n) = n premiers caractères de φ et des prédicats INFER(φ,Ψ), SUPER(φ,Ψ), EGAL(φ,Ψ) qui permettent de comparer deux chaînes de caractères φ et Ψ

- INFER(φ,Ψ) - VRAI si la chaîne φ est inférieure à Ψ
  - FAUX dans les autres cas.

- SUPER(φ,Ψ) - VRAI si la chaîne φ est supérieure à Ψ
  - FAUX dans les autres cas.

- EGAL(φ,Ψ) = NON INFER(φ,Ψ) ET NON SUPER(φ,Ψ).

Le prédicat RECHERCHEARBRE permet d'effectuer le parcours de l'arbre et si

-RECHERARBRE a pris la valeur VRAI, ENTREE est le pointeur sur l'élément recherché.

-RECHERCHEARBRE a pris la valeur FAUX, ENTREE est le point d'entrée sur la classe qui permettra d'accéder à l'élément cherché s'il existe dans le dictionnaire.

```
PREDICAT RECHERCHEARBRE(ENTREES : ELEMENT, RACINE, SORTIE : ENTREE)
    PRECEDENT ← NOEUD ← RACINE
    TROUVE ← FAUX
    TANTQUE NOEUD ≠ NIL ET NON TROUVE
       FAIRE
          ELEM ← EXTRCHAINE(ELEMENT, LONGUEUR(NOEUD))
          SI SUPER(ELEM, CHAINE(NOEUD))
              ALORS
                  PRECEDENT ← NOEUD
                  NOEUD ← ADRESSEDROITE(NOEUD)
              SINON
                  SI INFER(ELEM, CHAINE(NOEUD))
                    ALORS
                        NOEUD ← ADRESSEGAUCHE(NOEUD)
                    SINON
                        TROUVE ← VRAI
                    FINSI
              FINSI
       FINFAIRE
    SI TROUVE
          ALORS
              ENTREE ← ADDRESSECLASSE(NOEUD)
          SINON
              ENTREE ← ADRESSECLASSE(PRECEDENT)
          FINSI
       RECHERCHEARBRE ← TROUVE
FINPREDICAT
```

Démonstration du prédicat RECHERCHEARBRE

Ce prédicat effectue un parcours d'arbre : c'est-à-dire qu'il énumère n_1, n_2,...,n_1 noeuds de cet arbre. Le but de cette démonstration est de prouver que cette énumération est finie et qu'elle donne le résultat cherché.

Premier pas

n_1 est la racine de l'arbre qui est $\neq$ nil (arbre non vide)
Après avoir exécuté le "TANTQUE", on a :
- Si le prédicat SUPER a délivré la valeur vrai :

$$\text{chaîne}(n_1) < \text{elem} \;\&\; \text{chaine}(n_1) < \text{chaîne}(n_i) \quad \forall i \in [2,1]$$

Sinon si le prédicat INFER a délivré la valeur vrai :

$$\text{elem} < \text{chaîne}(n_1) \;\&\; \text{chaîne}(n_1) > \text{chaîne}(n_i) \quad \forall i \in [2,1]$$

- Sinon le parcours est terminé: n_1 est la solution cherchée.

Deuxième pas

Après avoir exécuté le "TANTQUE", on a :
- Si le prédicat SUPER a délivré la valeur vrai pour la deuxième fois :

$$\text{chaîne}(n_1) < \text{chaîne}(n_2) \;\&\; \text{chaîne}(n_2) < \text{elem} \;\&\; \text{chaîne}(n_2) < \text{chaîne}(n_i) \quad \forall i \in [3,1]$$

- Si le prédicat SUPER a délivré la valeur VRAI pour la première fois :

$$\text{chaîne}(n_2) < \text{elem} \;\&\; \text{chaîne}(n_1) > \text{elem} \;\&\; \text{chaîne}(n_2) < \text{chaîne}(n_i) \;\&\; \text{chaîne}(n_1) > \text{chaîne}(n_i) \quad \forall i \in [3,1]$$

Sinon : si le prédicat INFER a délivré la valeur vrai pour la deuxième fois :

elem < chaîne(n_2) & chaîne(n_2) < chaîne(n_1) & chaîne(n_i) < chaîne(n_2) $\forall i \in [3,1]$

Si le prédicat INFER a délivré la valeur vrai pour la première fois :

chaîne(n_1) < elem & elem < chaîne(n_2) & chaîne(n_1) < chaîne(n_i) & chaîne(n_i) < chaîne(n_2) $\forall i \in [3,1]$

Sinon le parcours est terminé : n_2 est la solution cherchée.

au pas i+j = 1

On a donc une suite $p_1, p_2, \ldots, p_j$

et une suite $s_1, s_2, \ldots, s_i$

telles que :

chaîne(p_1) < chaîne(p_2) < ...< chaîne(p_j) < elem < chaîne(s_i) < ...< chaîne(s_1)

On a donc deux suites : l'une croissante et bornée supérieurement, l'autre décroissante et bornée inférieurement : elles sont donc convergentes ce qui démontre la terminaison de l'algorithme.

A la sortie du tantque

- TROUVE est vrai (donc noeud ≠ nil)

On a donc : ... chaîne(p_j) < elem ≤ chaîne(s_i) ...

s_i qui est différent de nil est la solution cherchée.

- TROUVE est faux (donc noeud = nil)

On a (adressegauche(s_i) = nil) ou (adressedroite(p_j) = nil).

p_j qui est différent de nil est la solution cherchée.

Il reste à démontrer qu'il existe au moins p_1.
Ceci résulte du fait que dans l'arbre, on inclut une chaîne plus petite que toutes les autres.

IV.2 Parcours d'une classe

L'algorithme de parcours a été donné précédemment (cf. III.1) lorsqu'on considérait que le dictionnaire n'était formé que d'une seule classe.

IV.3 - Gestion du dictionnaire

Pour la gestion des classes, il est aisé de voir qu'on est amené à utiliser les techniques classiques de gestion de listes[COUS, KNU1] et à utiliser l'algorithme de recherche précédent. Après un certain nombre de modifications, une réorganisation automatique est déclenchée en fonction d'un certain pourcentage de désordre afin de limiter la pagination. Quant à l'arbre sa reconstruction est automatique.

A 2
GRAMMAIRES

La grammaire a pour rôle d'infirmer ou de confirmer la décomposition d'un segment effectuée par identification dans le dictionnaire et de produire toutes les interprétations possibles de la (ou des) décomposition(s) d'un segment. Pour ce faire, il suffit de se placer dans le cas des modèles d'états finis. Etant donné que le formalisme habituel des grammaires d'états finis est mal adapté au problème, nous avons défini des grammaires à validations et saturations qui permettent une écriture plus aisée et plus condensée que ces grammaires. Nous donnons tout d'abord quelques aspects théoriques qui permettent de montrer l'équivalence de ces grammaires et comment on peut obtenir l'automate associé. Ensuite nous examinons le problème de la transduction des informations linguistiques pour terminer par la définition des langages qui permettent à l'utilisateur de définir les paramètres linguistiques.

I - ASPECTS THEORIQUES

I.1 - GRAMMAIRE A VALIDATIONS

I.1.1. - Définitions

On définit une grammaire à validations par

- la donnée d'un vocabulaire terminal $V_{TV} = \{a,b,\ldots\}$
- la donnée d'un sous-ensemble E des entiers naturels $E = \{1,2,\ldots,n\}$
- un élément particulier $SV \in \mathcal{P}(E)$ appelé axiome.
- un ensemble fini $\mathcal{P}V$ de productions.

Par la suite, on note $GV = (V_{TV}, SV, \mathcal{P}V, E)$

Production (ou règle)

Une production est un élément d'une application finie notée $\rightarrow$ entre les éléments de E d'une part et les éléments de $V_{TV} \times \mathcal{P}(E)$ d'autre part.

Les productions sont de la forme :

$$i \rightarrow a\,[j_1,j_2,\ldots,j_q]$$

ou

$$i \rightarrow a[\emptyset]$$

avec $i \in E$, $[j_1,j_2,\ldots,j_q] \in \mathcal{P}(E)$ et $a \in V_{TV}$

NB - Par abus de langage, on note les parties de E entre crochets.

Relation $\overset{=}{\underset{GV}{>}}$

Soient α et $\beta \in V_{TV}^{*} \times \mathcal{S}(E)$.
On dit que $\alpha \overset{=}{\underset{GV}{>}} \beta$ si et seulement si il existe une chaîne $\varphi \in V_{TV}^{*}$ et $[k_1,k_2,\ldots,k_r,i,k_{r+2},\ldots,k_q] \in \mathcal{S}(E)$ tels que :

$$\alpha = \varphi[k_1,k_2,\ldots,k_r,i,k_{r+2},\ldots,k_q]$$

$$\beta = \varphi\ a[j_1,j_2,\ldots,j_q] \quad (\text{ou } \beta = \varphi\ a[\emptyset])$$

$i \rightarrow a[j_1,j_2,\ldots,j_q]$ (où $i \rightarrow a[\emptyset]$ étant une production de GV.

On appelle dérivation directe une telle relation.

Relation transitive $\overset{*}{\underset{GV}{>}}$

Soient α et $\beta \in V_{TV}^{*} \times \mathcal{S}(E)$.
On dit que $\alpha \overset{*}{\underset{GV}{>}} \beta$ s'il existe une suite finie $\alpha_o,\alpha_1,\ldots,\alpha_n$ $(n > 0)$ $(\alpha_i \in V_{TV}^{*} \times \mathcal{S}(E))$ telle que

$$\alpha_o \overset{=}{\underset{GV}{>}} \alpha_1 \overset{=}{\underset{GV}{>}} \alpha_2 \cdots \overset{=}{\underset{GV}{>}} \alpha_n \quad \text{avec } \alpha = \alpha_o \text{ et } \beta = \alpha_n$$

On appelle dérivation une telle relation.

Langage défini par une grammaire GV

On appelle langage défini par GV, que l'on note L(GV), le sous-ensemble de V_{TV}^{*} tel que :

$$L(GV) = \{\ \alpha\ /\ SV \overset{*}{\underset{GV}{>}} \alpha[\emptyset],\ \alpha \in V_{TV}^{*}\}$$

Exemple 1

$$GV = (V_{TV}, SV, \mathcal{P}V, E)$$

avec :

$V_{TV} = \{a,b,c,d\}$

$E = \{1,2,3,4\}$

$SV = [1,2]$

$\mathcal{P}V$:

$1 \rightarrow a[2,3]$

$2 \rightarrow b[2,4]$

$3 \rightarrow c[3,4]$

$4 \rightarrow d[\emptyset]$

Dérivation :

$[1,2] \Rightarrow a[2,3] \Rightarrow ab[2,4] \Rightarrow ab^2[2,4] \Rightarrow \ldots$

$\Rightarrow ab^n[2,4] \Rightarrow ab^n d[\emptyset]$

$[1,2] \Rightarrow a[2,3] \Rightarrow ac[3,4] \Rightarrow ac^2[3,4] \Rightarrow \ldots$

$\Rightarrow ac^m[3,4] \Rightarrow ac^m d[\emptyset]$

$[1,2] \Rightarrow b[2,4] \Rightarrow \ldots \Rightarrow b^p[2,4] \Rightarrow b^p d[\emptyset]$

$$L(GV) = \{a\ b^n\ d,\ a\ c^m\ d,\ b^p\ d \ /\ n > 0,\quad m > 0,\quad p > 0\}.$$

Nous allons montrer maintenant que l'on peut passer d'une grammaire d'états finis à une grammaire à validations et réciproquement. Ces grammaires engendrant le même langage.

I.1.2. - GRAMMAIRE D'ETATS FINIS → GRAMMAIRE A VALIDATIONS

Nous rappelons tout d'abord quelques définitions concernant les grammaires d'états finis. [CHO, HOUL]

I.1.2.1. - Definitions

- Une grammaire d'états finis $G = (V_T, V_N, S, \mathcal{P})$ avec

  V_T : vocabulaire terminal

  V_N : vocabulaire non terminal tel que :

  $$V_T \cap V_N = \emptyset, \quad V_T \cup V_N = V$$

  S : axiome avec $S \in V_N$

  $\mathcal{P}$: ensemble fini de productions de la forme :

  $$A \to a \text{ ou } A \to aB \text{ avec } A,B \in V_N,\ a \in V_T$$

- relation $\overset{=}{G}>$ entre deux chaînes

  On dit que $\alpha \overset{=}{G}> \beta$ (α , $\beta \in V^+$) si et seulement s'il existe une chaîne $\varphi \in V_T^*$ et un non terminal A tel que :

  $\alpha = \varphi A$

  $\beta = \varphi a B$ (ou $\beta = \varphi a$)

  $A \to a B$ (ou $A \to a$) étant une production de G

  On appelle dérivation directe une telle relation.

- relation $\overset{*}{\overset{=}{G}}>$ entre deux chaînes

  On dit que $\alpha \overset{*}{\overset{=}{G}}> \beta$ (α, $\beta \in V^+$) s'il existe une suite finie $\alpha_o, \alpha_1, \ldots, \alpha_n$ ($n > 0$) telle que

  $\alpha_o \overset{=}{G}> \alpha_1 \overset{=}{G}> \alpha_2 \overset{=}{G}> \alpha_n$ avec $\alpha = \alpha_o$ et $\beta = \alpha_n$

  On appelle dérivation une telle relation.

- Langage défini par une grammaire G

  On appelle langage défini par G, que l'on note L(G), le sous-ensemble de V_T^* tel que :

  $$L(G) = \{\alpha \ / \ S \overset{*}{\overset{=}{G}}> \alpha,\ \alpha \in V_T^* \}$$

I.1.2.2. - Transformation

I.1.2.2.1 - Définitions

On numérote toutes les productions de $\mathcal{P}$ de 1 à n et on définit les applications suivantes :

- GAUCHE(i) application de E dans V_N

  $i \in [1,n]$ avec n = cardinal($\mathcal{P}$)

la ième production de $\mathcal{P}$ étant de la forme

$A \rightarrow aB$ ou $A \rightarrow a$

| GAUCHE(i) = A |

- VAL(A) application de V_N dans $\mathcal{P}(E)$

  $A \in V_N$

Pour tout non terminal A on définit l'ensemble des numéros de production de $\mathcal{P}$ ayant A en partie gauche.

| VAL(A) = {i / GAUCHE(i) = A} |

I.1.2.2.2 - Transformation

On définit GV = (V_{TV}, SV, $\mathcal{P}$V, E) à partir de G = (V_T, V_N, S, $\mathcal{P}$)
avec :

- E = {1,2,...,n} avec n = cardinal($\mathcal{P}$)
- $V_{TV} = V_T$
- SV = VAL(S)
- $\mathcal{P}$V :

Pour toutes les productions de $\mathcal{P}$, on définit les productions de $\mathcal{P}$V de la manière suivante :

- si la ième production de $\mathcal{P}$ est de la forme A → aB alors
  i → a VAL(B) est une production de $\mathcal{P}$V

- si la ième production de $\mathcal{P}$ est de la forme A → a alors
  i → a[∅] est une production de $\mathcal{P}$V

Exemple 2

Soit G = (V_T, V_N, S, $\mathcal{P}$)

avec :

V_T = {a,b,c,d,e,f}

V_N = {S,A,B,C}

$\mathcal{P}$:

S → aA

S → bB

A → bA

A → d

B → cB

B → eC

C → f

alors

GV = (V_{TV}, SV, $\mathcal{P}$V, E)

avec :

V_{TV} = {a,b,c,d,e,f}

E = {1,2,3,4,5,6,7}

SV = [1,2]

$\mathcal{P}_V$:

$1 \rightarrow a[3,4]$

$2 \rightarrow b[5,6]$

$3 \rightarrow b[3,4]$

$4 \rightarrow d[\emptyset]$

$5 \rightarrow c[5,6]$

$6 \rightarrow e[7]$

$7 \rightarrow f[\emptyset]$

Dérivation

- Definition de VALC(α)

Soient $a_1, a_2, \ldots, a_i \in V_T$, $A \in V_N$

- si $\alpha = a_1 a_2 \ldots a_i A$ alors $VALC(\alpha) = a_1 a_2 \ldots a_i \; VAL(A)$
- si $\alpha = a_1, a_2 \ldots a_i$ alors $VALC(\alpha) = a_1 a_2 \ldots a_i[\emptyset]$

A toute dérivation $\alpha_o \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} \alpha_2 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \alpha_m$ de G on fait correspondre la dérivation de GV

$$VALC(\alpha_o) \underset{GV}{\Rightarrow} VALC(\alpha_1) \underset{GV}{\Rightarrow} VALC(\alpha_2) \underset{GV}{\Rightarrow} \cdots \underset{GV}{\Rightarrow} VALC(\alpha_m)$$

Théorème

Soit G une grammaire d'états finis, il existe une grammaire GV telle que $L(G) \subseteq L(GV)$

A partir de G, on définit une grammaire GV comme précédemment

$L(G) = \{\alpha \; / \; S \overset{*}{\underset{G}{\Rightarrow}} \alpha, \; \alpha \in V_T^+\}$

A toute dérivation

$S \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} \alpha_2 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \alpha_m \quad \alpha_m \in V_T^+$, on associe la dérivation

$SV = VAL(S) \underset{GV}{\Rightarrow} VALC(\alpha_1) \underset{GV}{\Rightarrow} \cdots \underset{GV}{\Rightarrow} VALC(\alpha_m) = \alpha_m[\emptyset], \quad \alpha_m \in V_{TV}^+$

Par contruction $L(G) \subseteq L(GV)$ et de plus à toute dérivation de G correspond une dérivation de GV.

I.1.3 - GRAMMAIRE A VALIDATIONS → GRAMMAIRE D'ETATS FINIS

I.1.3.1. - Definitions

- DROITE(i) application de E dans $\mathcal{P}(E)$.

  Pour toute production de $\mathcal{P}$ V de la forme

  $i \rightarrow a\ [i_1, i_2, \ldots, i_m]$ on définit

$$\boxed{\text{DROITE}(i) = [i_1, i_2, \ldots, i_m]}$$

- SYMBOLE(X)

  Soit un ensemble VSYMB composé de 2^n-1 symboles (où n est le nombre de productions). On définit une application notée SYMBOLE qui a tout élément $X \in \mathcal{P}(E)-\emptyset$ associe un élément de VSYMB.

$$\text{SYMBOLE}([i_1, i_2, \ldots, i_m]) = A$$

avec

$$[i_1, i_1, \ldots, i_m] \in \mathcal{P}(E)-\emptyset \quad \text{et} \quad A \in \text{VSYMB}$$

I.1.3.2 - Propriétés :

Si GV a été construite à partir de G, alors

a) $\forall i, \forall j,\ i \neq j \in E = \{1,2,\ldots,n\}$

$$\boxed{\begin{array}{l} \text{DROITE}(i) = \text{DROITE}(j) \\ \text{ou} \\ \text{DROITE}(i) \cap \text{DROITE}(j) = \emptyset \end{array}}$$

Remarque : cette propriété n'est plus vraie dans le cas où l'on construit directement une grammaire à validations GV (cf. exemple 1).

b) $SYMBOLE(DROITE(i)) = SYMBOLE([i_1, i_2, \ldots, i_m])$

$= SYMBOLE(i_1) = SYMBOLE(i_2) = \ldots = SYMBOLE(i_m)$

c) $DROITE(i) = DROITE(j) \Rightarrow SYMBOLE(DROITE(i)) = SYMBOLE(DROITE(j))$

d) $DROITE(i) \cap DROITE(j) = \emptyset \Rightarrow SYMBOLE(DROITE(i)) \neq SYMBOLE(DROITE(j))$

I.1.3.3. - Transformation dans le cas où la propriété ci-dessus est vérifiée

a) Définition de V_N

A partir de SV, $\mathcal{P}$ V, nous donnons ci-dessous un schéma d'algorithme de construction de l'ensemble V_N ($V_N \subseteq VSYMB$)

```
ACTION   CONSTRUIREVN (ENTREES : SV, P V, SORTIES : S, VN)
|        S ← SYMBOLE(SV)
|        V_N ← S
|        L ← CARDINAL(P V)
|        I ← 1
|        TANTQUE I ≤ L
|           FAIRE
|           |   SI DROITE(i) ≠ ∅
|           |      ALORS
|           |      |   V_N ← V_N ∪ SYMBOLE(DROITE(i))
|           |      FINSI
|           |   I ← I+1
|           FINFAIRE
FINACTION
```

b) Définition de $G = (V_T, V_N, S, \mathcal{P})$

On définit $G = (V_T, V_N, S, \mathcal{P})$ à partir de $G_V = (V_{TV}, SV, \mathcal{P} V, E)$ avec :

- $V_{TV} = V_T$
- V_N = (cf. a))

S = SYMBOLE(SV)

$\mathcal{P}$:

- si $i \rightarrow a[i_1, i_2, \ldots, i_m]$ est une production de $\mathcal{P}V$, alors SYMBOLE([i]) $\rightarrow$ a SYMBOLE($[i_1]$) est une production de $\mathcal{P}$.

- si $i \rightarrow a[\emptyset]$ est une production de $\mathcal{P}V$, alors SYMBOLE([i]) $\rightarrow$ a est une production de $\mathcal{P}$.

c) Théorème L(GV) = L(G)

- Définitions

SYMBOLEG(α_i) est une application de $V_{TV}^{*} \times \mathcal{P}(E)$ dans $V_T^{*} \times (V_N \cup \emptyset)$

. SYMBOLEG(α_i) = SYMBOLEG($a_1\ a_2 \ldots a_i\ [i_1, i_2, \ldots, i_m]$) = $a_1\ a_2 \ldots a_i$ SYMBOLE($[i_1, i_2, \ldots, i_m]$)

. SYMBOLEG($\alpha_m[\emptyset]$) = α_m avec $\alpha_m \in V_{TV}^{+}$

- Dérivation directe

A toute dérivation directe $\alpha \underset{GV}{\Rightarrow} \beta$ ($\alpha, \beta \in V_{TV}^{*} \times \mathcal{P}(E)$), on fait correspondre

SYMBOLEG(α) $\underset{G}{\Rightarrow}$ SYMBOLEG(β)

- Dérivation

A toute dérivation $SV \underset{GV}{\Rightarrow} \alpha_1 \underset{GV}{\Rightarrow} \alpha_2 \Rightarrow \ldots \underset{GV}{\Rightarrow} \alpha_m[\emptyset]$, on fait correspondre

SYMBOLEG(SV) $\underset{G}{\Rightarrow}$ SYMBOLEG(α_1) $\underset{G}{\Rightarrow}$ SYMBOLEG(α_2) $\underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow}$ SYMBOLEG($\alpha_m[\emptyset]$) = α_m

Par construction :

$$L(GV) = \{\alpha / SV \overset{*}{\underset{GV}{\Longrightarrow}} \alpha[\emptyset],\ \alpha \in V_{TV}^{+}\} \subseteq L(G) = \{\alpha /\ S \overset{*}{\underset{G}{\Longrightarrow}} \alpha,\ \alpha \in V_{T}^{+}\}\ .$$

Et à toute dérivation de GV correspond une dérivation de G.

Exemple 3

Reprenons l'exemple 2 :

$GV = (V_{TV},\ SV,\ \mathcal{P}V,\ E)$

avec :

$E = \{1,2,3,4,5,6,7\}$

$V_{TV} = \{a,b,c,d,e,f\}$

$SV = [1,2]$

$\mathcal{P}V$:

$1 \rightarrow a[3,4]$

$2 \rightarrow b[5,6]$

$3 \rightarrow b[3,4]$

$4 \rightarrow d[\emptyset]$

$5 \rightarrow c[5,6]$

$6 \rightarrow e[7]$

$7 \rightarrow f[\emptyset]$

$S = SYMBOLE([1,2]) = SYMBOLE([1]) = SYMBOLE([2])$

$A = SYMBOLE([3,4]) = SYMBOLE([3]) = SYMBOLE([4])$

$B = SYMBOLE([5,6]) = SYMBOLE([5]) = SYMBOLE([6])$

$C = SYMBOLE([7])$

d'où $G = (V_T,\ V_N,\ S, \mathcal{P})$

avec $V_T = \{a,b,c,d,e,f\}$

$V_N = \{S,A,B,C\}$

S = SYMBOLE([1,2])

$\mathcal{P}$:

$S \rightarrow aA$

$S \rightarrow bB$

$A \rightarrow bA$

$A \rightarrow d$

$B \rightarrow cB$

$B \rightarrow eC$

$C \rightarrow f$

I.1.3.4. - Transformation dans le cas où la propriété n'est plus vérifiée

(DROITE(i) $\cap$ DROITE(j) $\neq \emptyset$)

$V_T = V_{TV}$

V_N = Cf. I.1.3.3. a)

S = SYMBOLE(SV)

$\mathcal{P}$: pour tout $i_j \in$ SV

- si $i_j \rightarrow$ a DROITE(i_j) $\in \mathcal{P}$V
  alors S $\rightarrow$ a SYMBOLE(DROITE(i_j)) $\in \mathcal{P}$
- si $i_j \rightarrow a[\emptyset] \in \mathcal{P}$V
  alors S $\rightarrow$ a $\in \mathcal{P}$

De même soient DROITE(i_1), DROITE(i_2),..., DROITE(i_n) les parties droites des productions $i_1, i_2, \ldots, i_n$ telles que

DROITE(i_1) $\neq$ DROITE(i_j) $\quad 1 \neq j \quad 1 \leq 1,j \leq n$

pour tout $i_j \in$ DROITE(i_1)

- si $i_j \rightarrow$ a DROITE(i_j) $\in \mathcal{P}_V$
  alors SYMBOLE(DROITE(i_1)) $\rightarrow$ a SYMBOLE(DROITE(i_j)) $\in \mathcal{P}$

- si $i_j \rightarrow$ a[$\emptyset$] $\in \mathcal{P}_V$
  alors SYMBOLE(DROITE(i_1)) $\rightarrow$ a $\in \mathcal{P}$

et ceci pour tous les éléments de DROITE(i_1), DROITE(i_2), ..., DROITE(i_n).

Dérivation

On établit les mêmes correspondances avec la fonction SYMBOLEG que dans le cas précédent.

En conséquence $L(GV) \subseteq L(G)$.

Théorème : Les grammaires G et GV sont donc équivalentes

Exemple 4 :

Reprenons l'exemple 1

GV = ({a,b,c,d}, [1,2], $\mathcal{P}_V$, {1,2,3,4})

avec :

$\mathcal{P}_V$:

1 $\rightarrow$ a[2,3]

2 $\rightarrow$ b[2,4]

3 $\rightarrow$ c[3,4]

4 $\rightarrow$ d[$\emptyset$]

On définit

$S = SYMBOLE([1,2])$

$A = SYMBOLE([2,3])$

$B = SYMBOLE([2,4])$

$C = SYMBOLE([3,4])$

d'où $G = (V_T, V_N, S, \mathcal{P})$

avec $V_T = \{a,b,c,d\}$, $V_N = \{S,A,B,C\}$

$\mathcal{P}$:

$S \rightarrow aA$

$S \rightarrow bB$

$A \rightarrow bB$

$A \rightarrow cC$

$B \rightarrow bB$

$B \rightarrow d$

$C \rightarrow cC$

$C \rightarrow d$

I.1.3.5 - Nombre de règles de G et de GV

Dans le cas de la transformation G → GV, on obtient, par construction, le même nombre de règles.

Dans le cas de la transformation GV → G, le nombre de règles est égal à :

$$\text{Card(SV)} + \sum_{\substack{\text{sur les} \\ \text{DROITE(i) différents}}} \text{Card(DROITE(i))}$$

Théorème

Le nombre de règles de GV est toujours inférieur ou égal au nombre de règles de G.

Dans le cas ou DROITE(i) = DROITE(j) ou

DROITE(i) ∩ DROITE(j) = ∅

le nombre de règle est égal.

Par contre, dans le cas où cette propriété n'est plus vérifiée il est supérieur.

C'est là que réside tout l'intérêt de construire des grammaires à validations plutôt que des grammaires d'états finis.

I.2 - GRAMMAIRE A VALIDATIONS ET SATURATIONS [COURS]

I.2.1 - Motivations

Nous allons étendre les grammaires à validations en introduisant des "saturations" qui permettent d'interdire l'application de certaines règles.

Supposons que l'on ait le problème suivant : on veut écrire une grammaire engendrant les chaînes

$$W_1 \boxed{W_2} W_3 \quad \text{et} \quad W_4 \boxed{W_2} W_5$$

avec $W_1 = a_1 a_2 \ldots a_p$

$W_2 = b_1 b_2 \ldots b_1$

$W_3 = c_1 c_2 \ldots c_q$

$W_4 = d_1 d_2 \ldots d_m$

$W_5 = e_1 e_2 \ldots e_r$

Pour ce faire, on peut écrire

$S_1 = [1,1']$	
$[1] \rightarrow a_1\ [2]$	$[1'] \rightarrow d_1[2']$
$[2] \rightarrow a_2\ [3]$	$[2'] \rightarrow d_2[3']$
---	---
$[p-1] \rightarrow a_{p-1}[p]$	$[(m-1)'] \rightarrow d_{m-1}[m']$
$[p] \rightarrow a_p[p+1]$	$[m'] \rightarrow d_m\ [(m+1)']$
$[p+1] \rightarrow b_1[p+2]$	$[(m+1)'] \rightarrow b_1[(m+2)']$
---	---
$[p+1] \rightarrow b_1[p+1+1]$	$[(m+1)'] \rightarrow b_1[(m+1+1)']$
$[p+1+1] \rightarrow c_1[p+1+2]$	$[(m+1+1)'] \rightarrow e_1[(m+1+2)']$
---	---
$[p+1+q-1] \rightarrow c_{q-1}[p+1+q]$	$[(m+1+r-1)'] \rightarrow e_{r-1}[(m+1+r)']$
$[p+1+q] \rightarrow c_q[\emptyset]$	$[(m+1+r)'] \rightarrow e_r[\emptyset]$

On constate que l'on écrit deux fois des règles contenant l'élément terminal b_i

$$[s+i] \rightarrow b_i[s+i+1]$$

Afin d'éviter cette réécriture, on introduit, parallèlement aux validations, des saturations. Si dans une dérivation, une règle est saturée, celà signifie qu'on ne peut plus appliquer cette règle. Pour l'exemple précédent, il suffit que les règles permettant la reconnaissance des chaînes W_1 et W_4 autorisent celle de la chaîne W_2 et interdisent respectivement celles des chaînes W_5 et W_3. Par contre les règles permettant la reconnaissance de la chaîne W_2 autorisent celles des chaînes W_3 et W_5.

I.2.2. - Definitions

Une grammaire à validations et saturations est définie par :

- la donnée d'un vocabulaire terminal $V_{TVS} = \{a,b,...\}$
- la donnée d'un sous-ensemble E des entiers naturels $E = \{1,2,..,n\}$
- un élément particulier $SVS \in \mathcal{P}(E) \times \mathcal{P}(E)$ appelé axiome
- un ensemble fini $\mathcal{P}VS$ de productions.

Par la suite, on note une telle grammaire par

$$GVS = (V_{TVS}, SVS, \mathcal{P}VS, E)$$

Production

Une production est un élément d'une application finie notée entre les éléments de E d'une part et les éléments de $V_{TVS} \times \mathcal{P}(E) \times \mathcal{P}(E)$.

Les productions sont de la forme :

$$i \to a[i_1,i_2,\ldots,i_m][j_1,j_2,\ldots,j_n]$$

ou

$$i \to a[\emptyset][\emptyset]$$

avec

$$i \in E, \quad a \in V_{TVS}, \ [i_1,i_2,\ldots,i_m] \in \mathcal{P}(E),$$

$$[j_1,j_2,\ldots,j_n] \in \mathcal{P}(E) \text{ et } [i_1,i_2,\ldots,i_m] \cap [j_1,j_2,\ldots,j_n] = \emptyset$$

On appelle $[i_1, i_2,\ldots,i_m]$ les validations de la règle numéro i et $[j_1, j_2,\ldots,j_n]$ les saturations de la règle numéro i.

Relation $\overset{}{\underset{GVS}{\Longrightarrow}}$

On dit que $\alpha \underset{GVS}{\Longrightarrow} \beta$ ($\alpha,\beta \in V_{TVS}^{*} \times \mathcal{P}(E) \times \mathcal{P}(E)$) si et seulement s'il existe une chaîne $\varphi \in V_{TVS}^{*}$ et

$[k_1,\ldots,k_r,i,k_{r+2},\ldots,k_q][s_1,s_2,\ldots,s_q] \in \mathcal{P}(E) \times \mathcal{P}(E)$ tels que

$$\alpha = \varphi\,[k_1,\ldots,k_r,i,k_{r+2},\ldots,k_q][s_1,s_2,\ldots,s_q]$$

$$\beta = \varphi\, a[j_1,j_2,\ldots,j_q][s_1,s_2,\ldots,s_q,\ldots,s_r]$$

$$i \rightarrow a[j_1,j_2,\ldots,j_1][s_{q+1},\ldots,s_r]$$

étant une production de GVS

avec

$$[s_1,s_2,\ldots,s_r] = [s_1,s_2,\ldots,s_q] \cup [s_{q+1},\ldots,s_r]$$

et

$$[j_1,j_2,\ldots,j_q] = [j_1,j_2,\ldots,j_1] \cap [\overline{s_1,s_2,\ldots,s_r}]$$

On appelle <u>dérivation directe</u> une telle relation

<u>Dérivation</u> : <u>relation transitive $\alpha \overset{*}{\underset{GVS}{\Longrightarrow}} \beta$</u>

On dit que $\alpha \overset{*}{\underset{GVS}{\Longrightarrow}} \beta$ $(\alpha,\beta \in V^*_{TVS} \times \mathcal{P}(E) \times \mathcal{P}(E))$ si et seulement s'il existe une suite finie $\alpha_o, \alpha_1,\ldots, \alpha_n$ $(n > 0)$ telle que

$\alpha_o \overset{}{\underset{GVS}{\Longrightarrow}} \alpha_1 \overset{}{\underset{GVS}{\Longrightarrow}} \alpha_2 \overset{}{\underset{GVS}{\Longrightarrow}} \cdots \overset{}{\underset{GVS}{\Longrightarrow}} \alpha_n$

avec $\alpha = \alpha_o$ et $\beta = \alpha_n$

<u>Langage défini par une grammaire GVS</u>

On appelle langage défini par GVS, que l'on note L(GVS) le sous-ensemble de V^*_{TVS} tel que

$$L(GVS) = \{\ \alpha\ /\ SVS \overset{*}{\underset{GVS}{\Longrightarrow}} \alpha\ [\emptyset][s_1,s_2,\ldots,s_q],$$

$$\alpha \in V^*_{TVS}\ ,\ [s_1,s_2,\ldots,s_q] \in \mathcal{P}(E)\}$$

Exemple 5 :

$GVS = (V_{TVS}, SVS, \mathcal{P}_{VS}, E)$

$V_{TVS} = \{a,b,c,d,f,y,z,t,u\}$

$E = \{1,2,3,4,5,6,7,8,9,10\}$

$SVS = [1,2][\emptyset]$

$\mathcal{P}_{VS}$:

$1 \rightarrow f[2][6,10]$

$2 \rightarrow a[3][\emptyset]$

$3 \rightarrow b[3,4][\emptyset]$

$4 \rightarrow c[5,6][\emptyset]$

$5 \rightarrow d[7][\emptyset]$

$6 \rightarrow e[5][9]$

$7 \rightarrow y[7,8][\emptyset]$

$8 \rightarrow z[9,10][\emptyset]$

$9 \rightarrow t[\emptyset][\emptyset]$

$10 \rightarrow u[\emptyset][\emptyset]$

Dérivations :

$[1,2][\emptyset] \Rightarrow f[2][6,10] \Rightarrow fa[3][6,10] \Rightarrow \ldots \Rightarrow fab^n[3,4][6,10]$

$\Rightarrow fab^n c[5][6,10] \Rightarrow fab^n cd[7][6,10] \Rightarrow \ldots$

$\Rightarrow fab^n cdy^m[7,8][6,10] \Rightarrow fab^n cdy^m z[9][6,10]$

$\Rightarrow fab^n cdy^m zt[\emptyset][6,10]$

$$[1,2][\emptyset] \Rightarrow a[3][\emptyset] \Rightarrow \ldots \Rightarrow ab^n[3,4][\emptyset] \Rightarrow ab^nc[5,6][\emptyset]$$
$$\Rightarrow ab^ncd[7][\emptyset] \Rightarrow \ldots \Rightarrow ab^ncdy^m[7,8][\emptyset]$$
$$\Rightarrow ab^ncdy^mz[9,10][\emptyset] \Rightarrow ab^ncdy^mzt[\emptyset][\emptyset]$$
$$\Rightarrow ab^ncdy^mzu[\emptyset][\emptyset]$$

$$[1,2][\emptyset] \Rightarrow \ldots \Rightarrow ab^nc[5,6][\emptyset] \Rightarrow ab^nce[5][9]$$
$$\Rightarrow ab^nced[7][9] \Rightarrow \ldots \Rightarrow ab^ncedy^m[7,8][9]$$
$$\Rightarrow ab^ncedy^mz[10][9] \Rightarrow ab^ncedy^mzu[\emptyset][9]$$

$$L(GVS) = \{fab^ncdy^mzt,\ ab^ncdy^mzt,\ ab^ncdy^mzu,$$
$$ab^ncedy^mzu\} \quad \text{avec } n,m > 0.$$

Nous allons montrer maintenant l'équivalence entre les grammaires GVS et les grammaires G d'états finis.

I.2.3. - Equivalence entre les grammaires à validation saturations et les grammaires d'états finis.

I.2.3.1. - A partir d'une grammaire G d'états finis, on peut toujours trouver une grammaire GVS qui engendre le même langage.

Dans le cas où les saturations de toutes les règles de GVS sont vides, une grammaire GVS est une grammaire à validations (GV). Or nous avons déjà démontré l'équivalence entre les grammaires à validations et les grammaires d'états finis.

I.2.3.2. - A partir d'une grammaire GVS, on peut trouver une grammaire G d'états finis qui engendre le même langage.

I.2.3.2.1. - Définitions

- VAL(i), SAT(i)

si la ième production de GVS est de la forme :

$i \rightarrow a[i_1,i_2,\ldots,i_m][j_1,j_2,\ldots,j_n]$ alors

$VAL(i) = [i_1,i_2,\ldots,i_m]$

$SAT(i) = [j_1,j_2,\ldots,j_n]$

- VALID, SATUR

Ce sont deux applications de $\mathcal{P}(E) \times \mathcal{P}(E)$ dans $\mathcal{P}(E)$.

Soit VALSAT $\in \mathcal{P}(E) \times \mathcal{P}(E)$:

$VALSAT = [i_1,i_2,\ldots,i_m][j_1\ j_2\ ,\ldots,\ j_n]$, alors

$VALID(VALSAT) = [i_1,i_2,\ldots,i_m]$

$SATUR(VALSAT) = [j_1,j_2,\ldots,j_m]$

- TERMINAL(i)

Si la ième production de GVS est de la forme :

$i \rightarrow a[i_1,i_2,\ldots,i_m][j_1,j_2,\ldots,j_n]$, alors

TERMINAL(i) = a

- SYMBOLE(VALSAT)

Soit un ensemble VSYMB de 2^{2n} symboles où n est le nombre de productions. On définit une application SYMBOLE de $\mathcal{P}(E) \times \mathcal{P}(E)$ dans VSYMB.

$SYMBOLE([i_1,i_2,\ldots,i_m][j_1,j_2,\ldots,j_n]) = A, \qquad A \in VSYMB$

I.2.3.2.2. - Transformation de GVS en G

A partir de SVS, $\mathcal{P}VS$, nous donnons un schéma d'algorithme de construction de S, $\mathcal{P}$ et de V_N.

Par fermeture transitive de $\mathcal{P}VS$ sur l'ensemble des VALSAT à partir de SVS, on construit V_N qui est un sous-ensemble de VSYMB et l'ensemble des productions :

```
ACTION  CONSTRUIREG(ENTREES : SVS, 𝒫VS SORTIES : S, 𝒫 , V_N)
    LISTEDEVALSAT ← SVS
    S ← SYMBOLE(SVS)
    𝒫 ← VIDE
    V_N ← S
    PRENDRE(LISTEDEVALSAT, VALSAT)
    TANTQUE NON DERNIER(LISTEDEVALSAT)
       FAIRE
          REGLESAPPLICABLES ← VALID(VALSAT)
          PRENDRE(REGLESAPPLICABLES, i)
          TANTQUE NON DERNIER(REGLESAPPLICABLES)
             FAIRE
                SAT ← SATUR(VALSAT) U SAT(i)
                VALSAT1 ← [VAL(i)-SAT,SAT]
                SI VALID(VALSAT1) ≠ [Ø]
                  ALORS
                     𝒫 ← 𝒫U (SYMBOLE(VALSAT) → TERMINAL(i)
                                SYMBOLE(VALSAT1))
                     V_N ← V_N U SYMBOLE(VALSAT1)
                     SI NON APPARTIENT(LISTEDEVALSAT, VALSAT1)
                       ALORS
                          LISTEDEVALSAT ← LISTEDEVALSAT,VALSAT1
                       FINSI
                  SINON
                     𝒫 ← 𝒫U (SYMBOLE(VALSAT) → TERMINAL(i))
                  FINSI
                  PRENDRE(REGLESAPPLICABLES, i)
             FINFAIRE
          PRENDRE(LISTEDEVALSAT, VALSAT)
       FINFAIRE
FINACTION
```

Terminaison de l'algorithme

Il faut montrer la terminaison des deux schémas itératifs (TANTQUE).

a) TANTQUE NON DERNIER(REGLESAPPLICABLES)

Le nombre de règles applicables est toujours fini.
Il s' ensuit que ce schéma est exécuté un nombre fini de fois.
(< n si n = cardinal(E))

b) TANTQUE NON DERNIER(LISTEDEVALSAT)

Il faut montrer que cette liste est finie. Elle l'est car :

1°) l'opération
VALSAT1 ← [VAL(i)-SAT, SAT ← SATUR(VALSAT) ∪ SAT(i)]
est une opération interne de
$\mathcal{P}(E) \times \mathcal{P}(E)$ dans $\mathcal{P}(E) \times \mathcal{P}(E)$ où le nombre d'éléments est fini puisqu'égal à 2^{2n}.

2°) Tous les éléments de LISTEDEVALSAT sont distincts par construction.
Il s'ensuit que ce schéma est exécuté un nombre fini de fois.

d'où $G = (V_T, V_N, S, \mathcal{P})$
avec

$V_T = V_{TS}$

Quant à V_N, S, $\mathcal{P}$, ils sont donnés par le schéma de construction précédent.

Lemme $L(GVS) \subseteq L(G)$

- Définition de SYMBOLE(α)
c'est une application de $V_{TV}^* \times \mathcal{P}(E) \times \mathcal{P}(E)$ dans $V_T^* \times (V_N \cup \emptyset)$

. $SYMBOLEG(\alpha) = SYMBOLEG(a_1\ a_2 \ldots a_i[i_1,i_2,\ldots,i_m][j_1,j_2,\ldots,j_n])$

$= a_1\ a_2 \ldots a_i\ SYMBOLE([i_1,i_2,\ldots,i_m][j_1,j_2,\ldots,j_n])$

. $SYMBOLEG(\alpha) = SYMBOLEG(a_1\ a_2 \ldots a_m[\emptyset][j_1,j_2,\ldots,j_n])$

$= a_1\ a_2 \ldots a_m$

- Dérivation directe

A toute dérivation directe $\alpha \underset{GVS}{\Longrightarrow} \beta$, on fait correspondre la dérivation

$$SYMBOLEG(\alpha) \underset{G}{\Rightarrow} SYMBOLEG(\beta)$$

- Dérivation

A toute dérivation : $SVS \underset{GVS}{\Longrightarrow} \alpha_1 \underset{GVS}{\Longrightarrow} \alpha_2 \Rightarrow \ldots \underset{GVS}{\Longrightarrow} \alpha_m$, on fait correspondre

$$SYMBOLE(SVS) \underset{G}{\Rightarrow} SYMBOLEG(\alpha_1) \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} SYMBOLEG(\alpha_m)$$

par construction : $L(GVS) \subseteq L(G)$.

Lemme $L(G) \subseteq L(GVS)$

Supposons qu'il n'en soit pas ainsi. Cela voudrait donc dire qu'il existe une chaîne $\alpha \in L(G)$ et $\notin L(GVS)$.
Pour ce faire, il faut donc qu'il existe au moins une dérivation directe $SYMBOLEG(\alpha) \underset{G}{\Rightarrow} SYMBOLEG(\beta)$ et que $\alpha \underset{GVS}{\Longrightarrow} \beta$ n'existe pas. Ce qui est impossible de par la construction de G.

Théorème $L(GVS) = L(G)$

(par application des 2 lemmes ci-dessus).

Exemple 6

Soit la grammaire GVS de l'exemple 5

alors

$$G = (V_T, V_N, S, \mathcal{P})$$

avec

$$V_T = V_{TVS} = \{a,b,c,d,e,f,y,z,t,u\}$$

$$V_N = \{A,B,C,D,E,F,G,H,I,J,K,L,M,N,P,Q,R,S\}$$

Les éléments de V_N étant définis par la fonction SYMBOLE

SYMBOLE([1,2][Ø]) = S

SYMBOLE([2][6,10]) = A

SYMBOLE([3][6,10]) = B

SYMBOLE([3,4][6,10]) = C

SYMBOLE([5][6,10]) = D

SYMBOLE([7][6,10]) = E

SYMBOLE([7,8][6,10]) = F

SYMBOLE([9][6,10]) = G

SYMBOLE([5][9]) = N

SYMBOLE([7,8][9]) = Q

SYMBOLE([3][Ø]) = H

SYMBOLE([3,4][Ø]) = I

SYMBOLE([5,6][Ø]) = J

SYMBOLE([7][Ø]) = K

SYMBOLE([7,8][Ø]) = L

SYMBOLE([9,10][Ø]) = M

SYMBOLE([7][9]) = P

SYMBOLE([10][9]) = R

$\mathcal{P}$: S → fA

A → aB

B → bC

C → bC

C → cD

D → dE

E → yF

F → yF

F → zG

G → t

S → aH

H → bI

I → bI

I → cJ

J → dK

K → yL

L → yL

L → zM

M → t

M → u

J → eN

N → dP

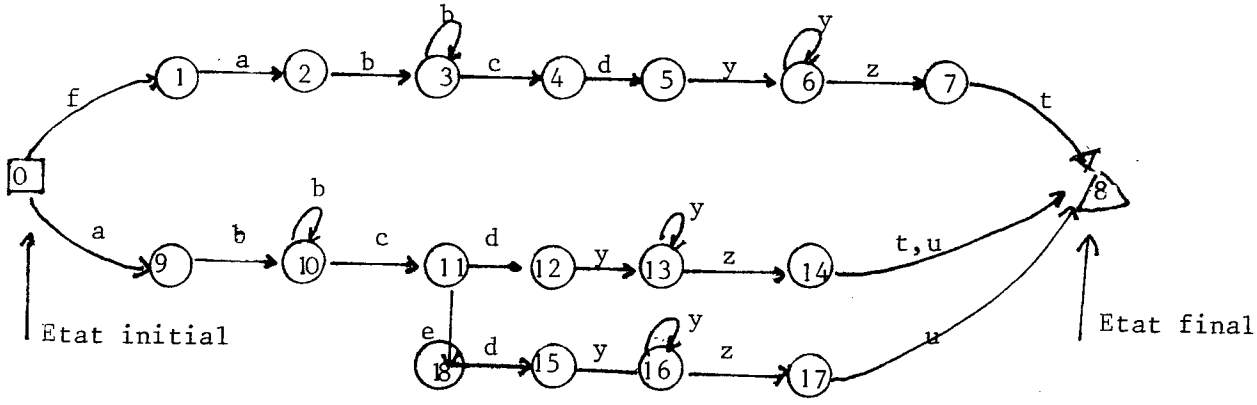
P → yQ

Q → yQ

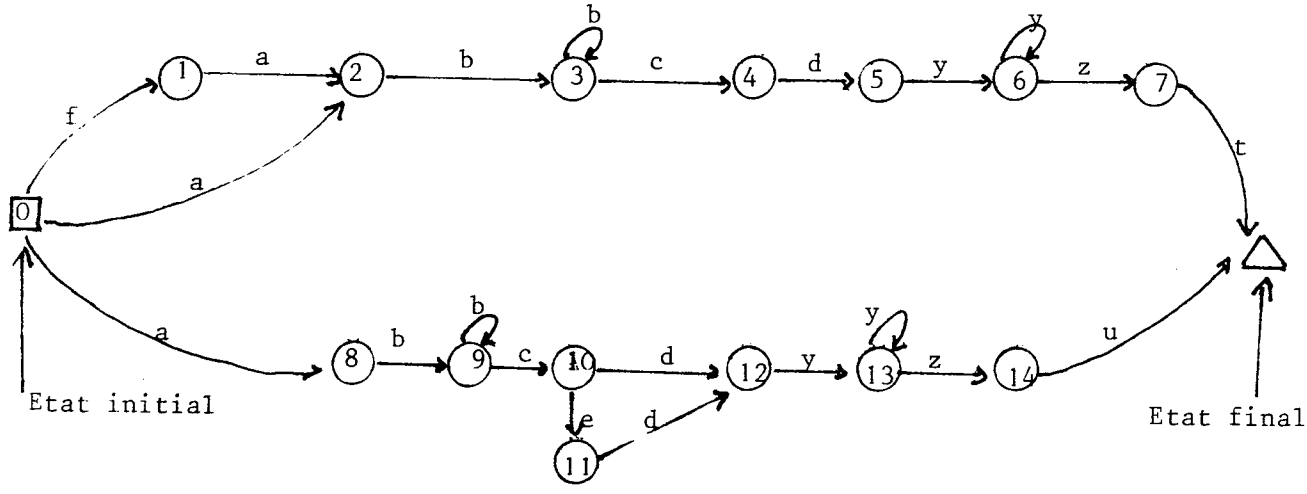
Q → yR

R → u

On obtient une grammaire G d'états finis comprenant 26 règles correspondant au diagramme suivant :



alors que directement, on peut trouver une grammaire G possédant 22 règles et correspondant au diagramme suivant :



1.2.3.2.3. - Nombre de règles de G et de GVS

- Dans le cas de la transformation G → GVS, c'est-à-dire GV, on a vu qu'on obtenait le même nombre de règles.

- Dans le cas de la transformation GVS → G, le nombre de règles est égal à :

$$\sum_{VALSAT} CARD(VALID(VALSAT))$$

$$\epsilon \; LISTEDEVALSAT$$

Théorème : Le nombre de règles de GVS est donc inférieur ou égal au nombre de règles de G.

I.3 - AUTOMATES ASSOCIES AUX GRAMMAIRES GV et GVS

I.3.1 - Automate associé à GV

I.3.1.1. - Rappel de définition

Nous utilisons ici le formalisme défini par HOPCROFT et ULLMAN [HOUL].

Un automate d'états finis M sur un alphabet V_T est un quintuplet (Q, V_T, δ, q_o, F) où

- Q est un ensemble fini, non vide d'états,
- V_T est un ensemble fini de symboles
- δ est une application de $Q \times V_T$ dans Q
- q_o état initial qui appartient à Q
- $F \subseteq Q$: l'ensemble des états finaux

On peut étendre l'application δ à une application $\hat{\delta}$ de $Q \times V_T^*$ dans Q :

$$\hat{\delta}(q,\Lambda) = q$$

$$\hat{\delta}(q,\varphi a) = \delta(\hat{\delta}(q,\varphi),a) \text{ avec } \varphi \epsilon V_T^*, \quad a \epsilon V_T \quad q \epsilon Q$$

Etant donné qu'il ne peut y avoir de confusion entre δ et $\hat{\delta}$, on utilise par la suite la notation $\cdot$ pour δ et $\hat{\delta}$.

Une chaîne $\varphi \in V_T^*$ est acceptée par M si $\delta(q_o, \varphi) = p \quad \forall p \in F$.

L'ensemble des chaînes $\varphi \in V_T^*$ acceptées par M, noté T(M) est défini par :

$$T(M) = \{ \varphi / \delta(q_o, \varphi) \in F \}$$

de même on peut définir un automate d'états finis non déterministe $M = (Q, V_T, \delta, q_o, F)$, avec δ une application de $Q \times V_T$ dans $\mathcal{P}(Q)$.

On montre également que si un langage L est accepté par un automate d'états finis non déterministe M, alors il existe un automate d'états finis M' qui accepte le même langage.

I.3.1.2. - Automate d'états finis associé à GV

Soit $GV = (V_{TV}, SV, \mathcal{S}V, E)$, alors il existe un automate d'états finis $MV = (Q, V_{TV}, \delta, SV, F)$ avec T(MV) = L(GV).
En effet, soit MV non déterministe
avec :

- $Q = \{ SV, \bigcup_{i=1}^{n} DROITE(i), T \}$ avec $T \notin \mathcal{P}(E)$

- l'état initial de MV est SV

- F = {T} car on n'a jamais de production vide.

- $T \in \delta(DROITE(i), a)$
  si $j \in DROITE(i)$ et $j \to a[\emptyset] \in \mathcal{S}V$

- $DROITE(j) \in \delta(DROITE(i), a)$
  si $j \in DROITE(i)$ et $j \to a\ DROITE(j) \in \mathcal{P}V$

- $\delta(T, a) = \emptyset \qquad \forall a \in V_{TV}$

Montrons que T(MV) = L(GV)

1°) si $\varphi \in L(GV)$ alors $\varphi \in T(MV) \quad \varphi \in V_{TV}^+$

Soit $\varphi = a_1\ a_2 \ \ldots\ a_n$ avec $n \geq 1$

$\varphi \in L(GV)$ entraîne qu'il existe une dérivation telle que :

$$SV \underset{GV}{\Longrightarrow} a_1\ \text{DROITE}(i_1) \underset{GV}{\Longrightarrow} a_1\ a_2\ \text{DROITE}(i_2) \underset{GV}{\Longrightarrow} \ldots$$

$$\underset{GV}{\Longrightarrow} a_1\ a_2 \ldots a_{n-1}\ \text{DROITE}(i_{n-1}) \underset{GV}{\Longrightarrow} a_1\ a_2 \ldots a_n[\emptyset]$$

d'après la définition de δ

- $\delta(SV, a_1)$ contient $\text{DROITE}(i_1)$
- $\delta(\text{DROITE}(i_1), a_2)$ contient $\text{DROITE}(i_2)$

- $\delta(\text{DROITE}(i_{n-2}), a_{n-1})$ contient $\text{DROITE}(i_{n-1})$
- $\delta(\text{DROITE}(i_{n-1}), a_n)$ contient T

$\delta(SV, \varphi)$ contient T, et $T \in F$, donc $\varphi \in T(MV)$

2°) réciproquement, si $\varphi \in T(MV)$, $\varphi \in V_{TV}^{+}$, alors $\varphi \in L(GV)$

$\varphi \in T(MV)$ entraîne qu'il existe une séquence finie d'états SV, $\text{DROITE}(i_1)$, $\text{DROITE}(i_2)$,... $\text{DROITE}(i_{n-1})$, T telle que :

$\delta(SV, a_1)$ contient $\text{DROITE}(i_1)$

$\delta(\text{DROITE}(i_1), a_2)$ contient $\text{DROITE}(i_2)$

$\delta(\text{DROITE}(i_{n-1}), a_n)$ contient T

Il existe donc des règles de $\mathcal{G}V$ telles que :

$i_1 \rightarrow a_1$ DROITE(i_1) avec $i_1 \in$ SV

$i_2 \rightarrow a_2$ DROITE(i_2) avec $i_2 \in$ DROITE(i_1)

$i_{n-1} \rightarrow a_{n-1}$ DROITE(i_{n-1}) avec $i_{n-1} \in$ DROITE(i_{n-2})

$i_n \rightarrow a_n[\emptyset]$ avec $i_n \in$ DROITE(i_{n-1})

Il s'ensuit que SV $\overset{}{\underset{GV}{\Longrightarrow}}$ a_1 DROITE(i_1) $\overset{}{\underset{GV}{\Longrightarrow}}$ a_1 a_2 DROITE(i_2)

$\overset{}{\underset{GV}{\Longrightarrow}} \cdots \overset{}{\underset{GV}{\Longrightarrow}} a_1\ a_2 \cdots a_{n-1}$ DROITE(i_{n-1}) $\overset{}{\underset{GV}{\Longrightarrow}} a_1\ a_2 .. a_n[\emptyset]$

est une dérivation de GV.

Donc $\varphi \in L(GV)$.

Exemple 7

Reprenons l'exemple n° 1

$GV = (\{a,b,c,d\}, [1,2], \mathcal{S}_V, \{1,2,3,4\})$

avec

$\mathcal{S}_V$:

$1 \rightarrow a[2,3]$

$2 \rightarrow b[2,4]$

$3 \rightarrow c[3,4]$

$4 \rightarrow d[\emptyset]$

$MV = (Q, V_{TV}, \delta, [1,2], F)$

avec $Q = \{[1,2], [2,3], [2,4], [3,4], [5]\}$

$SV = [1,2]$

$F = \{[5]\}$

δ :

$\delta([1,2],a) = [2,3]$ car $1 \in SV$ et $1 \to a[2,3] \in \mathcal{P}V$

$\delta([1,2],b) = [2,4]$ car $2 \in SV$ et $2 \to b[2,4] \in \mathcal{P}V$

$\delta([2,3],b) = [2,4]$

$\delta([2,3],c) = [3,4]$

$\delta([2,4],b) = [2,4]$

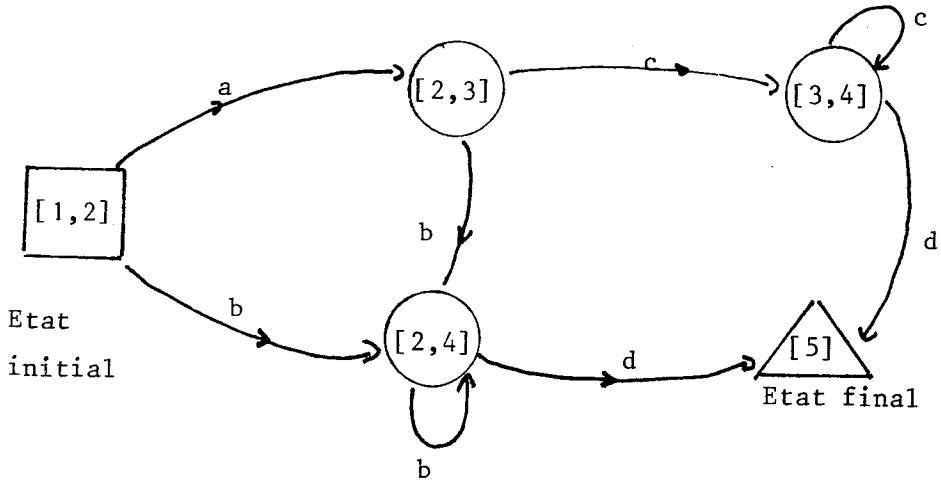
$\delta([2,4],d) = [5]$

$\delta([3,4],c) = [3,4]$

$\delta([3,4],d) = [5]$

$\delta([1,2],c) = \delta([1,2],d) = \ldots = \emptyset$

D'où le diagramme :



I.3.2 - Automate associé à GVS

On définit MVS = $(Q, V_{TVS}, \delta, SVS, F)$ non déterministe avec

- $Q = \{SVS, LISTEDEVALSAT\text{-}SVS, T\}$ $\quad T \subset \mathcal{P}(E) \times \mathcal{P}(E)$
- l'état initial est SVS
- $F = \{T\}$ car on n'a jamais de production vide

Définition de δ :

- $T \in \delta(VALSAT,a)$
  si $j \in VALID(VALSAT)$ et $j \rightarrow a[\emptyset][\emptyset] \in \mathcal{P}_{VS}$

- $VALSAT1 \in \delta(VALSAT,a)$
  si $\exists j \in VALID(VALSAT)$ et $j \rightarrow a\ VAL(j)\ SAT(j) \in \mathcal{P}_{VS}$
  avec

  $$VALSAT1 = [VAL(j)-(SATUR(VALSAT) \cup SAT(j)), SATUR(VALSAT) \cup SAT(j)]$$

- $\delta(T,a) = \emptyset$ pour tout $a \in V_{TVS}$.

Soit $\varphi = a_1\ a_2 \ldots a_n \in L(GVS)$ pour $n \geq 1$
On a donc la dérivation

$$SVS \overset{\overset{=}{=}}{GVS}> a_1\ VALSAT1 \overset{\overset{=}{=}}{GVS}> a_1\ a_2\ VALSAT2 \overset{\overset{=}{=}}{GVS}> \ldots$$

$$\overset{\overset{=}{=}}{GVS}> a_1\ a_2 \ldots a_{n-1}\ VALSATN\text{-}1 \overset{\overset{=}{=}}{GVS}> a_1a_2a_3 \ldots a_n[\emptyset][s_1s_2\ldots s_q]$$

d'après la définition de δ

- $\delta(SVS,a_1)$ contient VALSAT1
- $\delta(VALSAT1, a_2)$ contient VALSAT2
  etc...
- $\delta(VALSATN\text{-}1, a_n)$ contient T

donc $\varphi \in T(MVS)$ car $\delta(SVS,\varphi)$ contient T qui appartient à F.

Réciproquement :

Si $\varphi \in T(MVS)$ pour $|\varphi| \geq 1$ alors il existe une séquence finie d'états, SVS, VALSAT1, VALSAT2,... VALSATN-1, T telle que :

$\delta(SVS,a_1)$ contient VALSAT1

$\delta(VALSAT1,a_2)$ contient VALSAT2

etc...

il existe donc des règles de $\mathcal{S}$VS

$$i_1 \rightarrow a_1\ VAL(i_1)\ SAT(i_1) \text{ avec } i_1 \in VALID(SVS)$$

et

$$SAT = SATUR(SVS) \cup SAT(i_1)$$

$$VALSAT1 = [VAL(i_1)\text{-}SAT, SAT]$$

$$i_2 \rightarrow a_2\ VAL(i_2)\ SAT(i_2) \text{ avec } i_2 \in VALID(VALSAT1)$$

et

$$SAT = SATUR(VALSAT1) \cup SAT(i_2)$$

$$VALSAT2 = [VAL(i_2)\text{-}SAT, SAT]$$

$$i_{n-1} \rightarrow a_{n-1}\ VAL(i_{n-1})SAT(i_{n-1}) \text{ avec } i_{n-1} \in VALID(VALSATN\text{-}2)$$

et

$$SAT = SATUR(VALSATN\text{-}2) \cup SAT(i_{n-1})$$

$$VALSATN\text{-}1 = (VAL(i_{n-1})\text{-}SAT, SAT)$$

$$i_n \rightarrow a_n[\emptyset][\emptyset] \text{ avec } i_n \in VALID(VALSATN\text{-}1)$$

il s'ensuit que :

$$SVS \overset{\overset{=}{GVS}}{\Longrightarrow} a_1\ VALSAT1 \overset{\overset{=}{GVS}}{\Longrightarrow} a_1\ a_2\ VALSAT2 \overset{\overset{=}{GVS}}{\Longrightarrow} \cdots$$

$$\overset{\overset{=}{GVS}}{\Longrightarrow} a_1a_2\cdots a_{n-1}\ VALSATN\text{-}1 \overset{\overset{=}{GVS}}{\Longrightarrow} a_1a_2\cdots a_n[\emptyset][s_1, s_2, \ldots, s_q]$$

est une dérivation de GVS, donc $\varphi \in L(GVS)$.

Théorème

Pour toute grammaire GVS, il existe un automate d'états finis MVS avec T(MVS) = L(GVS).

Exemple 8

Reprenons l'exemple 5

GVS = ({a,b,c,d,f,y,z,t,u}, [1,2][Ø], $\mathcal{P}$VS, {1,2,3,4,5,6,7,8,9,10})

avec $\mathcal{P}$VS :

1 → f[2][6,10]

2 → a[3][Ø]

3 → b[3,4][Ø]

4 → c[5,6][Ø]

5 → d[7][Ø]

6 → e[5][9]

7 → y[7,8][Ø]

8 → z[9,10][Ø]

9 → t[Ø][Ø]

10 → u[Ø][Ø]

MVS = {QVS, V_{TVS}, δ, SVS, F}

avec

Q = {[1,2][Ø], [2][6,10], [3][6,10], [3,4][6,10], [5][6,10], [7][6,10], [7,8][6,10], [9][6,10], [3][Ø], [3,4][Ø], [5,6][Ø], [7][Ø], [7,8][Ø], [9,10][Ø], [5][9], [7][9], [7,8][9], [10][9], T}

T ∈ δ([9][6,10],t), T ∈ δ([9,10][Ø],t)

T ∈ δ([9,10][Ø],u), T ∈ δ([10][9],u)

[2][6,10] ∈ δ([1,2][Ø],f) [3][6,10] ∈ δ([2][6,10],a)

[3,4][6,10] ∈ δ([3][6,10],b) [3,4][6,10] ∈ δ([3,4][6,10],b)

$[5][6,10] \in \delta([3,4][6,10],c)$ $[7][6,10] \in \delta([5][6,10],d)$

$[7,8][6,10] \in \delta([7][6,10],y)$ $[7,8][6,10] \in \delta([7,8][6,10],y)$

$[9][6,10] \in \delta([7,8][6,10],z)$

$[3][\emptyset] \in \delta([1,2][\emptyset],a)$ $[3,4][\emptyset] \in \delta([3][\emptyset],b)$

$[3,4][\emptyset] \in \delta([3,4][\emptyset],b)$ $[5,6][\emptyset] \in \delta([3,4][\emptyset],c)$

$[7][\emptyset] \in \delta([5,6][\emptyset],d)$ $[7,8][\emptyset] \in \delta([7][\emptyset],y)$

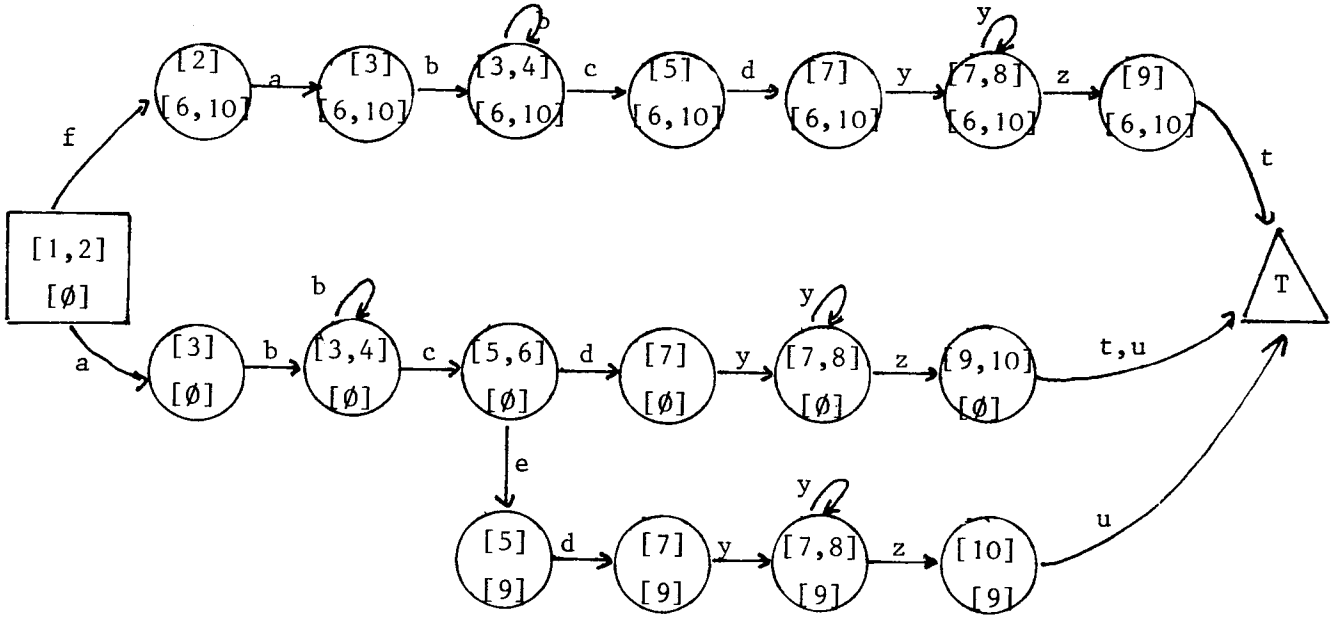
$[7,8][\emptyset] \in \delta([7,8][\emptyset],y)$ $[9,10][\emptyset] \in \delta([7,8][\emptyset],z)$

$[5][9] \in \delta([5,6][\emptyset],e)$ $[7][9] \in \delta([5][9],d)$

$[7,8][9] \in \delta([7][9],y)$ $[7,8][9] \in \delta([7,8][9],y)$

$[10][9] \in \delta([7,8][9],z)$

ce qui correspond au diagramme suivant :



II - REALISATION PRATIQUE [COUGD, COU4]

II.1 - RECONNAISSANCE ET TRANSDUCTION

Ayant reconnu qu'un mot appartient à la langue, il faut également lui associer toutes les informations linguistiques que l'on peut déduire de sa construction. Cette transduction est effectuéeen même temps que l'on vérifie la concaténation des différents éléments constituant l'entité reconnue. On a été amené à définir des opérations pour réaliser cette transduction. Une règle de grammaire est composée de deux parties : l'une utilisée pour la reconnaissance, l'autre pour obtenir les informations linguistiques.

II.1.1 - Informations linguistiques

Comme nous l'avons vu, on distingue deux sortes d'informations : une que l'on appelle la catégorie, l'autre l'ensemble des valeurs de variable grammaticale. Au niveau des opérations, elles se différencient par le fait que la catégorie ne peut prendre qu'une seule valeur, alors que la variable grammaticale est constituée d'un ensemble de valeurs.

On appelle type une information qui ne peut prendre qu'une seule valeur et variable une information composée d'un ensemble de valeurs.

Exemple : HOMMES : SUBSTANTIF MASCULIN PLURIEL

La catégorie SUBSTANTIF est un type alors que {MASCULIN, PLURIEL} est une variable grammaticale.

II.1.2 - Les modèles

Pour réaliser l'interface entre le dictionnaire et la grammaire nous avons défini des modèles qui sont des représentants d'une classe dont tous les éléments ont le même comportement morphologique (par

exemple, les verbes du premier groupe). Les modèles contiendront les informations permettant de faire la liaison avec les règles de grammaires, des renseignements linguistiques et également des validations et des saturations afin de limiter le nombre de règles. De ce fait, pour contrôler le cheminement, nous avons défini un calcul supplémentaire réalisant l'union des validations contenues dans une règle et un modèle ainsi que l'union des saturations.

II.1.3 - Définition des informations linguistiques

II.1.3.1. - Les TYPES

Nous avons défini deux types :

- la classe lexicale notée CL qui peut être modifiée au cours de la dérivation.
- la classe de base notée CB.

Exemple :

DIVISION qui se décompose en DIVIS+ION a pour classe lexicale "substantif" et pour classe de base "verb" si l'on a fait dériver le substantif de la base verbale "DIVIS".

L'instruction d'affectation est utilisée pour réaliser l'une des opérations suivantes :

1°) transmettre la valeur d'un type d'un état à un autre
2°) créer une valeur
3°) accéder à la valeur définie au niveau du modèle.

Par convention, les symboles "G" et "D" signifient respectivement gauche et droite (ou dictionnaire).

En conséquence on écrit les instructions d'affectations suivantes :

1°) CL := CL(G) ; transmission de l'état précédent

2°) CL := VERB ; affectation d'une valeur

3°) CL := CL(D) ; transmission de la valeur définie dans le modèle.

II.1.3.2. - Les variables

Pour les valeurs de variables, nous avons défini les opérations suivantes :

1°) transmettre des valeurs d'un état à un autre,

2°) imposer un ensemblede valeurs,

3°) accéder aux valeurs définies au niveau du modèle,

4°) ajouter ou retrancher une valeur ou un ensemble de valeurs,

5°) combiner les quatre opérations précédentes

Ces opérations se traduisent par :

1°) VAR := VAR(G) ; transmission

2°) VAR := MAS+SIN ; création des valeurs masculin et singulier

3°) VAR := VAR(D) ; valeurs définies au niveau du modèle

4-5°) VAR:= VAR(G)+VAR(D)-SIN+MAS ;

II.1.3.3 - Type particulier CD (code dérivation)

Il permet, au cours de la dérivation, de produire une chaîne de caractères. Ce n'est pas exactement un type par le fait que le code dérivation n'est pas une valeur mais une concaténation de valeurs. Il est utilisé soit pour indiquer le cheminement, soit pour effectuer une traduction de chaîne à chaîne (par exemple, la traduction en phonétique).

Les opérations sont les suivantes :

1°) transmission d'un état à un autre : CD := CD(G) ;

2°) création d'une valeur : CD := ON ;

3°) accés à la valeur définie par le modèle : CD := CD(D) ;

4°) concaténation : CD := CD(G) || CD(D) ;

ou CD := CD(G) || ON ;

II.1.4 - Règle d'écriture des règles

Une règle est définie par :

nom de la règle : informations linguistiques ; VAL := (liste de noms de règles) ou valeur prédéfinie ; SAT := (liste de noms de règles) ou valeur prédéfinie ; indicateurs.

- Les informations linguistiques sont de la forme indiquée précédemment où toutes les opérations que nous avons définies sont permises.

- VAL et SAT sont deux mots clés qui permettent de définir les validations et les saturations de la règle.

- Indicateurs : on a soit FIM (règle terminale)
soit ND (voir CM paragraphe II.1.6.)

Exemples :

RIB : CL := CL(D) ; CB := CB(D) ; VAR := VAR(D) ;
VAL := ( ) ; SAT := ( ).

PR4 : CL := CL(G) ; CB := CB(G) ; VAR := VAR(G)+VAR(D)
+PRE+IND ; VAL := ( ) ; SAT := ( ) ; FIM.

Syntaxe d'écriture des règles

Nous donnons ci-après la syntaxe d'écriture des règles.

<REGLE>::=<ETIQUETTE DE REGLE>:<CORPS DE REGLE>.

<ETIQUETTE DE REGLE>::=<LETTRE ou CHIFFRE>|<LETTRE ou CHIFFRE><ETIQUETTE DE REGLE>

<LETTRE ou CHIFFRE>::= A|B|...Z|0|1|...|9

<CORPS DE REGLE>::=*COMMENTAIRE*<CORPS DE REGLE S/S COMMENTAIRE>|<CORPS DE REGLE S/S COMMENTAIRE>

<CORPS DE REGLE S/S COMMENTAIRE>::=<AFFECTATION DES TYPES><AFFECTATION DE VARIABLES><AFFECTATION DE VAL-SAT><INDICATEUR>

<AFFECTATION DES TYPES>::=<AFFECTATION CL><AFFECTATION CB><AFFECTATION CD><AFFECTATION CM >

<AFFECTATION CL>::=CL:=<PARTIE DROITE CL>;|vide

<PARTIE DROITE CL>::=CL(<INDICE>)|<VALEUR CL>

<VALEUR CL>::=Une des valeurs de CL déclarées

<INDICE>::=G|D

<AFFECTATION CB>::=CB:=<PARTIE DROITE CB>;|vide

<PARTIE DROITE CB>::=CB(<INDICE>)|<VALEUR CB>

<VALEUR CB>::=Une des valeurs de CB déclarées

<AFFECTATION CM>::=CM:=<PARTIE DROITE CM>;|vide

<PARTIE DROITE CM>::=CM(<INDICE>)|<VALEUR CM>

<VALEUR CM>::=Une des valeurs de CM déclarées

<AFFECTATION CD>::=CD:=<PARTIE DROITE CD>;|vide

<PARTIE DROITE CD>::=CD(<INDICE>)|<VALEUR CD>|CD(G)||CD(D)|CD(G)||<VALEUR CD>

<VALEUR CD>::=Une des valeurs de CD déclarées

<AFFECTATION DE VARIABLES>::=VAR:=<PARTIE DROITE DE VAR>;|vide

<PARTIE DROITE DE VAR>::=<VARIABLE><OPERATEUR><EXPRESSION DE VAR>|<VARIABLE>|<EXPRESSION DE VAR>

<VARIABLE>::=<VAR INDICEE>|<VAR INDICEE>±<VAR INDICEE>

<VAR INDICEE>::=VAR(<INDICE>)

<EXPRESSION DE VAR>::=<EXPRESSION DE VAR><OPERATEUR><VAR SIMPLE>|<VAR SIMPLE>

<VAR SIMPLE>::=Valeur de variable déclarée

<OPERATEUR>::= +|-

<AFFECTATION DE VAL-SAT>::=VAL:=<PARTIE DROITE DE VAL-SAT>;SAT:=<PARTIE DROITE DE VAL-SAT>

<PARTIE DROITE DE VAL-SAT>::=VAL<NOMBRE>|<SAT NOMBRE>|(<LISTE D'ETIQUETTES DE REGLE>)|( )

<INDICATEUR>::=vide|;ND|;FIM|;ND;FIM|

Remarque : VAL<NOMBRE> et SAT<NOMBRE>doivent avoir été préalablement déclarés.

Le nombre de règles est limité à 256.

II.1.5 - Règle d'écriture des modèles

Un modèle est défini par :

/nom du modèle/: REG := (liste de noms de règles) ou valeur prédéfinie ; informations linguistiques ; VAL:= (liste de noms de règles) ou valeur prédéfinie ; SAT := (liste de noms de règles) ou valeur prédéfinie.

- Les règles déterminées par le mot clé REG ont comme symbole terminal le nom du modèle.

- Les informations linguistiques sont de la forme indiquée précédemment où les opérations permises ne sont que celles qui concernent la création d'une valeur (2°) .

- VAL et SAT sont deux mots clés qui permettent de définir les validations et les saturations du modèle.

Exemple

/AIM/: REG := (RIB) ; CL := VERB ; CB := BVRB ; VAL := VAL5 ; SAT := ( ).

où - VAL5 est une liste de noms de règles prédéfinie par l'utilisateur (VAL5 := (liste de noms de règles)). C'est une commodité d'écriture qui évite de définir plusieurs fois la même liste de noms de règles ; quant aux saturations, elles sont vides.

- RIB est un nom de règle (règle initiale de base)

- syntaxe d'écriture des modèles

  On trouve ci-après la syntaxe d'écriture des modèles .

<MODELE>::=/<NOM DE MODELE>/:<CORPS DE MODELE>.

<NOM DE MODELE>::=<CARACTERE>|<CARACTERE><NOM DE MODELE>

<CARACTERE>::=A|B|... Z|0|1 ...|9

<CORPS DE MODELE>::=*COMMENTAIRE*<CORPS DE MODELE S/S COMMENTAIRE>|
<CORPS DE MODELE S/S COMMENTAIRE>

<CORPS DE MODELE S/S COMMENTAIRE>::=<AFFECTATION DE REGLE><AFFECTATION DES TYPES><AFFECTATION DE VARIABLE><AFFECTATION DE VAL-SAT>

<AFFECTATION DE REGLE>::=REG:=<PARTIE DROITE DE VAL-SAT>;

<AFFECTATION DES TYPES>::=<AFFECTATION CL><AFFECTATION CB><AFFECTATION CM><AFFECTATION CD>

<AFFECTATION CL>::=CL:=<VALEUR CL>;|vide

<AFFECTATION CB>::=CB:=<VALEUR CB>;|vide

<AFFECTATION CM>::=CM:=<VALEUR CM>;|vide

<AFFECTATION CD>::=CD:=<VALEUR CD>;|vide

<VALEUR CL>::= Une des valeurs de CL déclarées

<VALEUR CB>::= Une des valeurs de CB déclarées

<VALEUR CM>::= Une des valeurs de CM déclarées

<VALEUR CD>::= Une des valeurs de CD déclarées

<AFFECTATION DE VARIABLE>::=VAR:=<EXPRESSION DE VAR>;|vide

<AFFECTATION DE VAL-SAT>::= voir syntaxe règles

<EXPRESSION DE VAR>::= voir syntaxe règles.

Le nom d'un modèle est limité à 17 caractères.

II.1.6 - Le code morphologique (CM)

Il sert à faire référence à un système de désinences.
Son introduction vient du fait suivant : prenons par exemple la base "centr". Nous pouvons obtenir la conjugaison du verbe centrer ou bien un adjectif (central) ou un substantif (centre ou centralisation) ou la conjugaison du verbe centraliser etc... Afin de simplifier les validations et les saturations, le code morphologique qui peut être modifié au cours de la dérivation permet de définir à chaque pas le système de désinences adéquat. En reprenant l'exemple "CENTR", cette base aura comme valeur de CM le système de désinences correspondant à la conjugaison des verbes du premier groupe et au substantif commun et comme validation la règle permettant de concaténer le suffixe "AL".

Par contre la règle qui permet le suffixe "AL" aura comme valeur de CM le système de désinence, des adjectifs et comme validation une règle permettant de concaténer le suffixe IS. etc...
Le CM ayant été défini comme un type, on peut le transmettre d'un état à un autre, ce qui est très intéressant au niveau des applications.

II.1.7. - Calcul des validations et des saturations

On effectue le calcul défini précédemment (cf. grammaires à validations et saturations) en ayant au préalable réalisé l'union entre les validations de la règle et du modèle et l'union entre les saturations de la règle et du modèle.

Règles applicables à l'étape i+1 de la dérivation

Appelons SAT_i, SAT_r et SAT_m respectivement les saturations à l'étape i de la dérivation, les saturations portées par la règle et les saturations portées par le modèle. De même, appelons VAL_i, VAL_r et VAL_m les validations à l'étape i de la dérivation, les validations portées par la règle et celles portées par le modèle, on obtient à l'étape i+1 :

$$SAT_{i+1} = SAT_i \cup SAT_r \cup SAT_m$$

$$VAL_{i+1} = (VAL_r \cup VAL_m) \cap \overline{SAT_{i+1}}$$

si le modèle sélectionné est m+1, alors la liste des règles applicables à l'étape i+1 de la dérivation avec le modèle "m+1" est égale à :

$$REG_{m+1} \cap VAL_{i+1}$$

où REG_{m+1} est l'ensemble défini par le mot clé REG du modèle m+1.
Les règles initiales sont définies par le mot clé VALOO.

Applications des règles contenues dans le CM

Si à l'étape i+1 de la dérivation, la liste des règles applicables avec le modèle "m+1" est vide et que la règle "r" ne contient pas l'indicateur "ND" (non désinence), alors, la liste des règles applicables est égale à :

$$REG_{m+1} \cap CM_r$$

où CM_r est le code morphologique défini par l'application de la règle r et du modèle m+1.

II.1.8 - Exemples d'utilisations

Soient VALOO := (RIB,...)
CM12 := (PR4, SU1,...)
VAL1 := (E1,E8)

les règles :

RIB : CL:=CL(D) ; CB:=CB(D) ; CM:=CM(D) ;
VAR:=VAR(D) ; VAL:=( ) ; SAT:=( ).

E1 : CL:=CL(G) ; CB:=CB(G) ; VAR:=VAR(G)+SIN ;
VAL:=( ) ; SAT:=( ) ; FIM.

PR4 : CL:=CL(G) ; CB:=CB(G) ; VAR:=VAR(D)+PRE+IND ;
VAL:=( ) ; SAT:=( ) ; FIM.

SU1 : CL:=CL(G) ; CB:=CB(G) ; VAR:=VAR(D)+PRE+SUB ;
VAL:=( ) ; SAT:=( ) ; FIM.

Les modèles :

/FEMME/: REG:=(RIB) ; CL:=SUBC ; CB:=BSBC ;
VAR:=FEM ; VAL:=VAL1 ; SAT:=( ).

/␣/ : REG:=(NF1,PR7,E1) ; VAR:=TRE+SIN ;
VAL:=( ) ; SAT:=( ).

/AIM/ : REG:=(RIB) ; CL:=VERB ; CB:=BVRB ;
CM:=12 ; VAL:=( ) ; SAT:=( ).

/ENT␣/ : REG:=(PR4,IS3,SU1) ; VAR:=TRE+PLU ;
VAL:=( ) ; SAT:=( ).

Le dictionnaire

/MAISON/FEMME/
/CHANT/AIM/
/␣/␣/
/ENT␣/ENT␣/

Analysons les mots "MAISON␣" et "CHANTENT␣"

a) "MAISON␣"

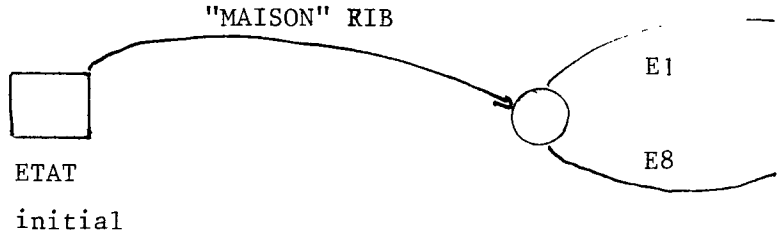
Par identification entre la chaîne d'entrée et les éléments du dictionnaire, on détermine "MAISON" dont le modèle est "FEMME".

D'autre part VALOO := (RIB,...) et dans le modèle "FEMME", on a REG:=(RIB), ce qui permet de sélectionner la règle RIB (VALOO ∩ REG).

Cette règle impose :

CL:=CL(D)	donc	CL:=SUBC (substantif commun)
CB:=CB(D)	donc	CB:=BSBC (base de substantif)
VAR:=VAR(D)	donc	VAR:=FEM (variable féminin)
CM:=CM(D)	donc	CM:=( ) puisque pas de CM dans le modèle.

Quant aux validations, elles sont données par VAL1 et égales à {E1, E8}. On a donc le diagramme suivant :



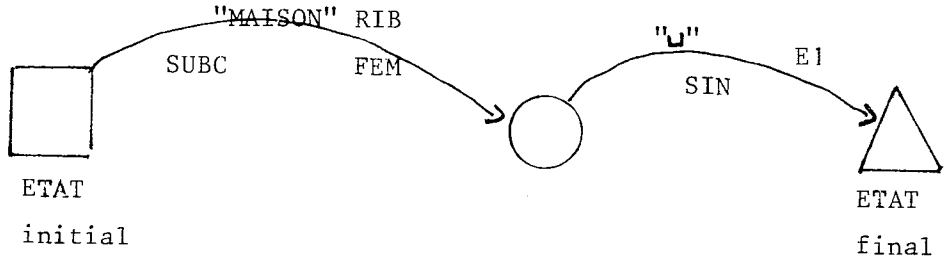
Ayant trouvé "MAISON" on identifie ensuite la désinence "␣" dont le modèle est "␣".

Dans le modèle, on a REG:=(NF1,PR7,E1), d'où sélection de la règle E1 par intersection entre REG et les validations E1 et E8. La règle E1 impose :

CL:=CL(G)	transmission de CL	donc	CL:=SUBC
CB:=CB(G)	transmission de CB	donc	CB:=BSBC
VAR:=VAR(G)+SIN	transmission de la valeur FEM à laquelle on ajoute SIN (singulier).		

D'autre part, la présence de l'indicateur FIM permet l'acceptation du mot "MAISON␣" substantif commun formé à partir de la base de substantif "MAISON", et ayant comme variable grammaticale féminin singulier.

D'où le diagramme :



b) "chantent␣"

On procède de la même façon. Le modèle de /CHANT/ qui est /AIM/, ayant REG:=(RIB), permet la sélection de la règle RIB ceci nous donne :

CL:=VERB, CB:=BVRB (base de verbe) et CM:=12.

Les validations étant vides et l'indicateur ND étant absent on utilise les règles du code morphologique.

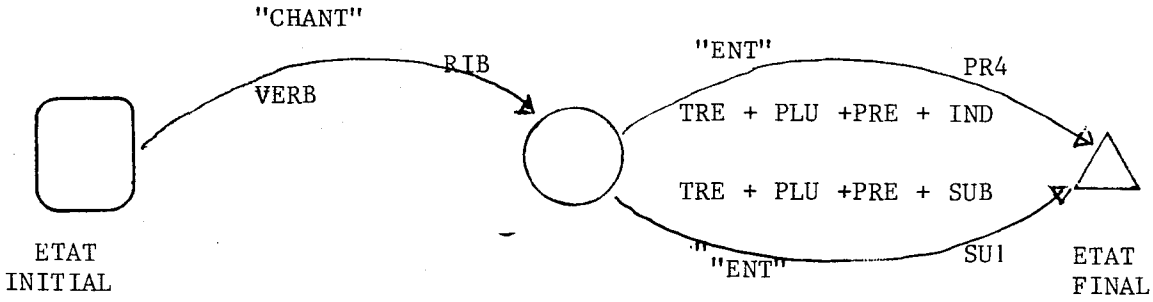
La désinence "ENT␣" dont le modèle a pour REG:=(PR4, IS3, SU1) permet, par intersection avec CM12, la sélection de deux règles : PR4 et SU1.

L'application de PR4 permet d'obtenir le résultat suivant :

CL:=VERB, CB:=BVRB, VAR:=TRE (troisième personne)
+ PLU(du pluriel) + PRE (du présent) + IND (de l'indicatif)

Quant à SU1 il permet d'obtenir le même résultat sauf en ce qui concerne le mode qui est SUB (subjonctif).

d'où le diagramme



Ayant obtenu un résultat, on essaye d'en trouver d'autres avec des bases homographes ou des bases plus courtes. Dans les cas que nous avons traités, il n'y en a pas. Par contre avec "COUVENT" on aurait obtenu deux solutions.

c) Nous donnons maintenant un exemple illustrant l'utilisation des saturations.

Considérons les mots FINIR, FINIRONS, FINIREZ ,...
DIRE, DIRONS, DIREZ ,...

On peut admettre les découpages suivants :

FINI + R + { ONS␣ / ez␣ / e␣ } DI + R + { ONS␣ / ez␣ / e␣ }

Celà signifie qu'après application de la règle permettant la concaténation du suffixe "R" il faut autoriser :

- pour la base FINI les désinences "ONS␣, ez␣" du futur et la désinence "␣" de l'infinitif

- pour la base DI les désinences "ONS␣, ez␣" du futur et la désinence "e␣" de l'infinitif.

On peut résoudre le problème de la manière suivante :

Après application de la règle autorisant le suffixe "R", valider les règles autorisant "ONS␣", "ez␣", "␣" et "e␣" , mais par contre au niveau

de la base FINI interdire la règle du "E␣"
de la base DI interdire la règle du "␣".

d'où

Les modèles

/FINI/ : REG:=(RIB) ; CL:=VERB ; CB:=BVRB ; VAL:=(...FU5...) ;
SAT:=(NF2).

/DI/ : REG:=(RIB) ; CL:=VERB ; CB:=BVRB ; VAL:=(...FU5...) ;
SAT:=(NF1).

/R/ : REG:=(FU5) ; VAL:=( ) ; SAT:=( ).

/␣/ : REG:=(NF1,PR7,E1) ;...; VAL:=( ) ; SAT:=( ).

/E␣/ : REG:=(SU1,PR1,RO3,NF2) ;...; VAL:=( ) ; SAT:=( ).

/ONS␣/ : REG:=(PR2,RO5,FU7,SU4) ; VAR:=UNO+PLU ;
VAL:=( ) ; SAT:=( ).

/ez␣/ : REG:=(PR2,RO5,FU7,SU4) ; VAR:=DUO+PLU ;
VAL:=( ) ; SAT:=( ).

Les règles

FU5 : ..... ; VAL:=(FU7,FU8,NF1,NF2) ; SAT:=( ).
FU7 : ..... ; VAR:=VAR(D)+FUT+IND ; VAL:=( ) ; SAT:=( ) ; FIM.
NF1 : CL:=INFI ; CB:=CB(G) ; VAL:=( ) ; SAT:=( ) ; FIM.
NF2 : CL:=INFI ; CB:=CB(G) ; VAL:=( ) ; SAT:=( ) ; FIM.
RIB : CL:=CL(D) ; CB:=CB(D) ; CM:=CM(D) ; VAR:=VAR(D) ; VAL:=( ) ; SAT:=( ).

Analyse de "FINIR".

Après avoir analysé "FINI", on a

VAL:=(...,FU5,...) et SAT:=(NF2)

L'identification de "R" permet l'analyse de "FINIR" avec

VAL:=(FU7,FU8,NF1,NF2) et SAT:=(NF2)

Les validations de l'état suivant sont donc égales à

(FU7,FU8,NF1), ce qui permet l'analyse de "FINIR␣" mais interdit par contre la reconnaissance de "FINIRE␣".

II.1.9. - Les déclarations

Toutes les valeurs de type ou de variable utilisées ainsi que les valeurs prédéfinies de validations, de saturations et de code morphologique doivent avoir été préalablement déclarées avant leur utilisation. Ces déclarations sont nécessaires au compilateur afin d'effectuer les vérifications nécessaires pour contrôler la validation des opérations effectuées sur ces valeurs. D'autre part, elles assurent à l'utilisateur une cohérence au niveau des noms utilisés. On trouve ci-dessous la syntaxe de ces déclarations.

Syntaxe d'écriture des déclarations des validations, des saturations et des CM

<DECLARATION>::=<DECLARATION DES VALIDATIONS>|<DECLARATION DES SATURATIONS>|<DECLARATION DES CM>

<DECLARATION DES VALIDATIONS>::=VAL<NOMBRE>:=<PARTIE DROITE DE VAL>

<DECLARATION DES SATURATIONS>::=SAT<NOMBRE>:=<PARTIE DROITE DE VAL>

<DECLARATION DES CM>::=CM<NOMBRE CM>:=<PARTIE DROITE DE VAL>

<NOMBRE>::=<CHIFFRE>|<CHIFFRE><CHIFFRE>

<NOMBRECM>::=<NOMBRE>|<NOMBRE><CHIFFRE>

<PARTIE DROITE DE VAL>::=(<LISTE D'ETIQUETTES DE REGLE>)

<LISTE D'ETIQUETTES DE REGLE>::=<ETIQUETTE DE REGLE>|<ETIQUETTE DE REGLE>,<LISTE D'ETIQUETTES DE REGLE>

<ETIQUETTE DE REGLE>::=<LETTRE OU CHIFFRE>|<LETTRE OU CHIFFRE><ETIQUETTE DE REGLE>

<LETTRE OU CHIFFRE>::=<LETTRE>|<CHIFFRE>

<LETTRE>::= A|B|...|Z

<CHIFFRE>::= 0|1|...|9

Par convention VALOO définit l'état initial de l'automate d'états finis.

Une étiquette de règle est limitée à 4 caractères.

Syntaxe d'écriture des déclarations des variables et des types

<DECLARATION>::=<DECLARATION DES NOMS DE VARIABLE><AFFECTATION DES VALEURS DE VARIABLE>|<DELCARATION DES NOMS DE TYPE>
<AFFECTATION DES VALEURS DE TYPE>

<DECLARATION DES NOMS DE VARIABLE>::=VAR:=<LISTE DES NOMS DE VARIABLE>;

<LISTE DES NOMS DE VARIABLE>::=<NOM DE VARIABLE>|
<NOM DE VARIABLE>,<LISTE DE NOMS DE VARIABLE>

<NOM DE VARIABLE>::=<CARACTERE><CARACTERE><CARACTERE>

<CARACTERE>::=<LETTRE>|<CHIFFRE>

<LETTRE>::= A|B|C|...|Z|

<CHIFFRE>::= |0|1|...|9|

<AFFECTATION DES VALEURS DE VARIABLE>::=<NOM DE VARIABLE>:=<LISTE DES VALEURS DE VARIABLE>;

<LISTE DES VALEURS DE VARIABLE>::=<LISTE DES NOMS DE VARIABLE>

<DECLARATION DES NOMS DE TYPE>::=TYP:=CM,CL,CB,CD ;

<AFFECTATION DES VALEURS DE TYPE>::=CM:=<LISTE DES CM>;
CL:=<LISTE DES VALEURS DE CL>;
CB:=<LISTE DES VALEURS DE CB>;
CD:=<LISTE DES VALEURS DE CD>;

<LISTE DES CM>::=<NOMBRE>|<NOMBRE>,<LISTE DES CM>|<NOMBRE>-<NOMBRE>|
<NOMBRE>-<NOMBRE>,<LISTE DES CM>

<NOMBRE>::=<CHIFFRE>|<CHIFFRE><NOMBRE>

<LISTE DES VALEURS DE CL>::=<VALEUR DE CL>|
<VALEUR DE CL>,<LISTE DES VALEURS DE CL>

<LISTE DES VALEURS DE CB>::=<VALEUR DE CB>|
<VALEUR DE CB>,<LISTE DES VALEURS DE CB>

<LISTE DES VALEURS DE CD>::=<VALEUR DE CD>|
<VALEUR DE CD>,<LISTE DES VALEURS DE CD>

<VALEUR DE CL>::=<CARACTERE>|<CARACTERE>,<VALEUR DE CL>

<VALEUR DE CB>::=<CARACTERE>|<CARACTERE>,<VALEUR DE CB>

<VALEUR DE CD>::=<CARACTERE>|<CARACTERE>,<VALEUR DE CD>

- Les noms de variable ont toujours trois caractères
- Les valeurs de type (valeur de CL, de CB, de CD) sont limitées à quatre caractères
- Pour les CM, les nombres sont limités à trois chiffres
- Nombre maximum de valeurs de variable : 64
- Nombre maximum de valeurs d'un type : 253.

Les déclarations de valeur prédéfinie de validations, de saturations et de code morphologique sont faites en même temps que la définition d'une règle alors que les autres déclarations sont faites à part.
Un éditeur facilite la définition de ces outils morphologiques en conversationnel. Un compilateur incrémentiel autorise une mise à jour en temps réel des informations correspondantes. De même un éditeur permet l'exécution conversationnelle de la morphologie avec possibilité de modifier les informations définies précédemment même en cours d'exécution [cf. chapitre C].

II.1.10 - Génération morphologique

Comme nous l'avions déjà annoncé, le système est réversible. Cette possibilité est très intéressante, non seulement parce qu'elle évite de définir de nouveaux outils pour réaliser la génération, mais parce qu'elle permet d'exercer un contrôle optimal de l'analyse.
En effet, dès que l'utilisateur a écrit un certain nombre de modèles et de règles, il peut demander une génération afin de vérifier l'écriture des outils qu'il vient de définir. De ce fait, ce générateur est devenu l'outil indispensable pour vérifier l'adéquation du modèle d'analyse.

Principe de la génération

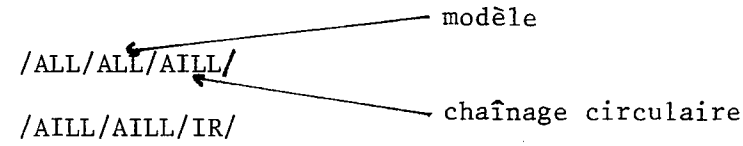
En fonction d'une chaîne $\alpha_1\alpha_2 \ldots\alpha_n$ représentant un mot, on détermine par analyse la racine de cette chaîne. Ayant découvert cette racine, on génère par analyse toutes les formes acceptables par l'analyseur. On peut également ne demander qu'une partie de formes acceptables.

A ce moment là, il suffit de préciser la catégorie et les variables grammaticales des formes que l'on veut obtenir. Dans le cas de plusieurs bases, il faut que toutes ces bases soient chaînées en anneau au niveau du dictionnaire.

Remarque - si la chaîne $\alpha_1\alpha_2 \ldots\alpha_n$ est incorrecte mais que la racine est correcte, on peut générer les formes correspondantes.

Exemple :

Pour générer toutes les formes du verbe "aller" il faut que l'on ait le dictionnaire suivant :



/ALL/ALL/AILL/

/AILL/AILL/IR/

/IR/IR/VA␣/

/VA␣/VA␣/VAS␣/

/VAS␣/VAS␣/VONT␣/

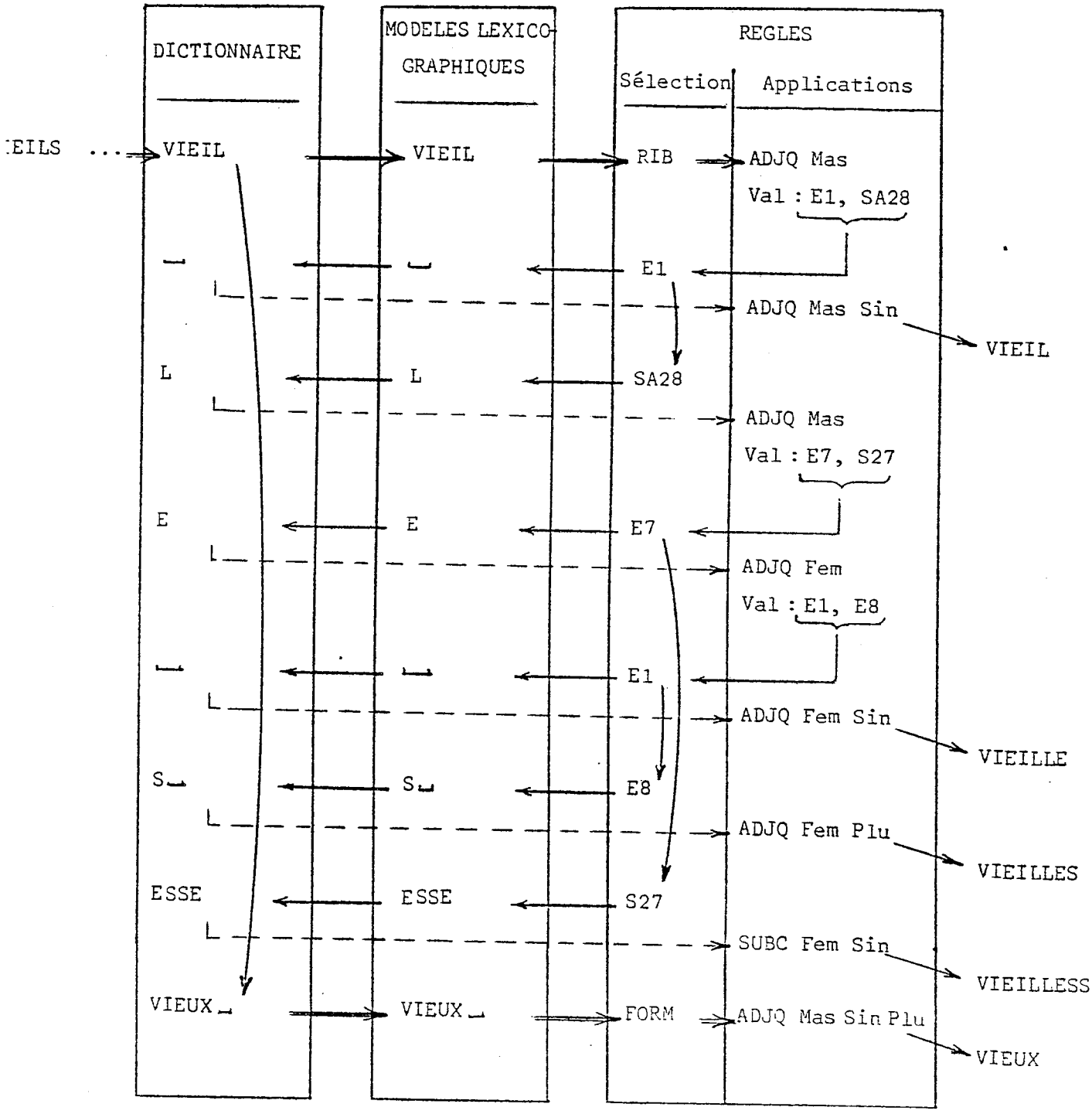
/VONT␣/ VONT␣/ALL/

/VAS␣/VAS␣/VAIS␣/

Dans le dictionnaire, on a prévu des pointeurs logiques en nombre variable (paramétré) qui permettent ces chaînages circulaires. Ce chaînage est géré automatiquement par le système. Par exemple, pour le verbe aller, il suffit de faire toujours référence à la base "ALL" au niveau de ce chaînage. Ceci nous a paru indispensable étant donné que les utilisateurs sont en général des non informaticiens.

On trouve ci-après un schéma illustrant le principe de cette génération.

SCHEMA DE LA GENERATION MORPHOLOGIQUE



B

LE SYSTEME D'ANALYSE SYNTAXIQUE

I - INTRODUCTION

I.1 - ROLE DE LA SYNTAXE

Etant donné que certains auteurs nient l'intérêt d'une syntaxe, il est naturel de se demander si l'analyse syntaxique est nécessaire. Parmi ces auteurs on peut distinguer deux motivations : ceux qui ne s'intéressent pas à ce problème et qui se contentent d'utiliser quand même des analyseurs syntaxiques existants, et ceux qui résolvent ce problème par d'autres méthodes que celles utilisées habituellement. L'intérêt des premiers est orienté vers une autre classe de problèmes telle que la sémantique. Si l'on examine, par exemple, les systèmes "questions réponses" il est bien évident que la préoccupation majeure des écrivains de logiciels doit résider en l'interprétation de la question et en la construction d'un réseau sémantique adapté pour fournir des réponses. La plupart des réalisations actuelles sont fondées soit sur une langue extrêmement limitée, soit sur un langage "quasi naturel" ce qui revient à réduire bon nombre de problèmes linguistiques, voire à supprimer totalement l'analyse morphologique et syntaxique. Par contre, si l'on désire concevoir un système "questions-réponses" sur un domaine limité mais acceptant toute la langue, il faudra bien se préoccuper des problèmes linguistiques en amont des problèmes sémantiques.
Quant à la deuxième catégorie, elle résoud ce problème par l'utilisation par exemple, de grammaires transformationnelles qui permettent non seulement la résolution des problèmes sémantiques mais également des problèmes syntaxiques.

La syntaxe, peu importe les modèles linguistiques utilisés, est fondamentale pour :

- construire une ou plusieurs structures syntaxiques
- lever certaines ambiguités.

La syntaxe est nécessaire pour lever l'amgibuïté contenue dans la phrase suivante : "La maison de l'oncle que nous avons vu" ? En effet, l'ambiguité ne peut être levée que par des considérations grammaticales : à savoir que "vu" étant au masculin, il ne peut se rapporter qu'à "oncle" et non pas à "maison". Naturellement, la syntaxe ne résoud pas tous les problèmes, et par exemple, dans des phrases telles que "le pilote ferme la porte" et "le boucher sale la tranche", l'analyseur syntaxique fournit deux structures correspondant à des interprétations différentes. Il est à noter que ces phrases sont intrinsèquement ambiguës et que seul le contexte des phrases précédentes pourrait peut être permettre le choix entre les deux interprétations possibles.

Pour réaliser l'analyse syntaxique, nous avons défini deux modèles : un filtre de dépendances qui a pour objet de construire toutes les arborescences possibles et un filtre grammatical qui sélectionne les arborescences ainsi fournies. La méthode préconisée est voisine de celle utilisée par CUNIN-SIMONET-VOIRON [CSV1] pour la compilation d'Algol 68 où ils construisent un arbre abstrait qui est ensuite décoré. Nous reviendrons sur ce parallèle dans les chapîtres suivants. Avant d'exposer les modèles que nous avons choisis, nous expliquons ci-dessous pourquoi nous n'avons pu utiliser les méthodes classiques d'analyse syntaxique définies par les écrivains de compilateurs dirigés par la syntaxe et par les pionniers du traitement automatique des langues.

I.2 - LES METHODES UTILISEES

En compilation, les langages de programmation sont, en général, décrits à l'aide d'une grammaire "hors contexte". Les algorithmes d'analyse sont alors élaborés en fonction de cette grammaire. Deux grandes classes d'analyseurs ont alors vu le jour . Ceux qui ont été établis autour des années 1965 que l'on pourrait appeler analyseurs généraux et s'appliquant sur des grammaires "hors contexte" presque quelconques (élimination du symbole vide, de symboles inacessibles, etc...). Leur temps d'exécution étant en général de l'ordre de n^3 où n est le nombre d'éléments de la phrase, on a défini depuis des algorithmes plus performants qui ne s'appliquent que sur une classe particulière de grammaire "hors contexte" (Bounded context, LL(K), LR(K), précédences) [FLOYD, KNU3, KNU4]. La philosophie est la suivante : A partir d'une grammaire vérifiant certaines conditions, on établit des tables ou des matrices qui permettront ensuite à l'exécution d'obtenir des analyseurs déterministes dont le fonctionnement n'est pas sans rappeler les analyseurs d'états finis. Les résultats obtenus sont spectaculaires et tous les compilateurs actuels utilisent maintenant une des méthodes particulières évoquées précédemment. Est-il possible d'utiliser ces analyseurs (généraux ou particuliers) dans le cas des langues naturelles ?

I.2.1. - Les analyseurs généraux

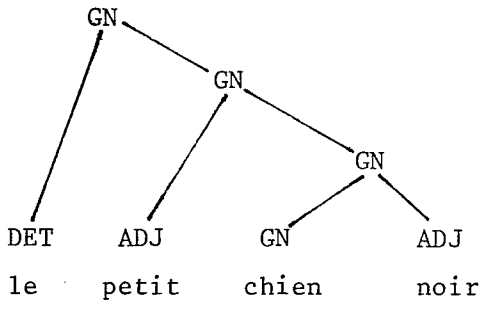
Etant donné que CHOMSKY [CHO] avait défini ces classes de grammaire pour les langues naturelles, de tels algorithmes ont été utilisés dans ce domaine. Par exemple, au C.E.T.A., dans le premier système de traduction automatique, l'algorithme de "COCKE YOUNGER KASAMI" était utilisé [VEIV, VEILL1].

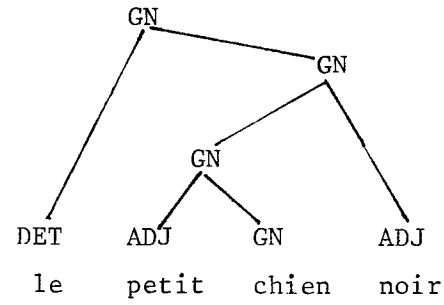
Comme paramètre, on lui fournissait une grammaire "hors contexte" mise sous forme normale de CHOMSKY. L'inconvénient, outre le temps d'exécution assez élevé, résidait dans le nombre de structures proposées. Par exemple :

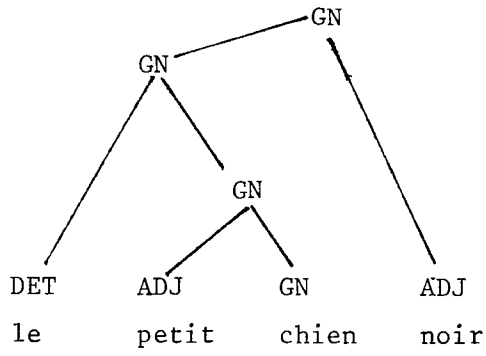
Soit la phrase "Le petit chien noir".
et la grammaire "hors contexte"

DET*GN => GN
GN*ADJ => GN
ADJ*GN => GN

On obtient les trois structures suivantes :





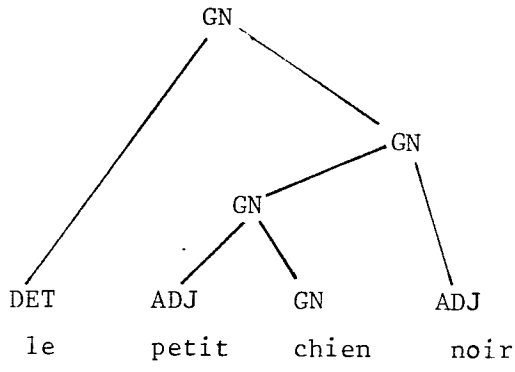


Afin d'éviter cette prolifération de structures, on peut créer de nouveaux symboles intermédiaires, ce qui alourdit considérablement la grammaire ou ajouter comme au C.E.T.A. des règles interdites en cours de cheminement. C'est ce qu'on appelle des saturations qui sont différentes de celles utilisées en morphologie.

Exemple 1

1	DET*GN => GN	2	interdit l'application des règles
2	GN*ADJ => GN	3	2 et 3 après avoir appliqué respectivement les
3	ADJ*GN => GN		règles 1 et 2

Ces saturations permettent l'obtention de la structure unique :



Cet algorithme a donné de bons résultats bien que la maîtrise des grammaires à saturations soit un peu délicate car cela revient à demander aux linguistes de définir le cheminement de l'algorithme d'analyse lorsqu'ils définissent les règles de grammaire correspondantes. De plus, comme nous l'avons déjà indiqué, il se posait le problème de la convergence de la grammaire qui était modifiée en fonction des textes à analyser.

D'autres algorithmes ont été utilisés :

I.2.2. - Les analyseurs particuliers

Comme nous l'avons vu, les analyseurs particuliers ne s'appliquent que sur une certaine classe de grammaires "hors contexte". Le principe est en général le suivant : en fonction d'une grammaire "hors contexte" G, il faut trouver une grammaire "hors contexte" équivalente G' qui vérifie certaines conditions. Comme il n'existe pas d'algorithme qui permette de trouver cette grammaire automatiquement (le problème étant indécidable), on opère, en pratique, par transformations successives en examinant, à la main, les règles qui font que la grammaire ne satisfait pas aux conditions. Ce travail peut être assez long, voire dans quelques cas impossible, mais lorsqu'on a trouvé G', l'algorithme proposé est alors très performant. Dans le cas des langages de programmation, cette approche est très intéressante et est utilisée par la majorité des compilateurs actuels du fait que le langage que l'on désire compiler est bien défini et qu'il suffit de trouver une grammaire équivalente satisfaisant aux critères. Par contre, dans le cas des langues naturelles, le langage n'est pas défini par sa grammaire. Les règles de grammaire ne sont qu'approximatives et on est amené à y apporter sans cesse des modifications à chaque analyse d'un nouveau texte. Comme il n'existe pas d'algorithme de transformation de grammaire, il est impensable, dans ce cas, d'utiliser une méthode particulière.

I.2.3. - Adéquation des grammaires "hors contexte"

Dans le cas où l'on admet que le texte d'entrée est correct, l'écriture des règles ne pose pas de problème . Par contre, si l'on admet que le texte d'entrée peut être incorrect et si l'on désire détecter le maximum d'erreurs, il faut établir ces règles avec soin. En effet, l'emploi de règles récursives ne permet pas toujours la détection des formes erronées. Si l'on reprend l'exemple 1, on peut avoir autant d'adjectifs que l'on veut à droite ou à gauche du groupe nominal alors qu'il ne faudrait autoriser qu'un seul adjectif à droite et au maximum deux adjectifs à gauche. La grammaire de l'exemple 1 ne permet pas de résoudre ce problème car elle n'est pas générative.

En outre, les structures obtenues à partir des grammaires "hors contexte" sont difficilement interprétables : c'est-à-dire inutilisables par d'autres modèles. Il faut donc utiliser un algorithme de transformation de structures qui permette l'obtention de structures plus facilement utilisables (interprétables) telles que les structures de dépendances [VEILL1, VAUQ].

Les grammaires génératives (voire transformationnelles), sont des modèles parfois satisfaisants du point de vue de la théorie linguistique (explication d'un phénomène). Ces modèles sont en général très satisfaisants pour la description et l'analyse des langages artificiels. Par contre dans le cas de l'analyse automatique des langues naturelles, ils posent certains problèmes. En conséquence, nous nous sommes orientés vers d'autres modèles.

I.2.4. - Les grammaires de dépendances [HAYS, GAIF, VEILL1, MACH, VAUQ]

Définition

Une grammaire de dépendances sur un vocabulaire V_N est constituée par la donnée d'une famille de parties C_i de V_N qui constituent des catégories syntaxiques, telles que $\bigcup_i C_i = V_N$ et d'un ensemble de règles de la forme

1) $\quad *(X)$

2) $\quad X(X_1\ X_2 \ \ldots\ X_i\ *\ X_{i+1} \ \ldots\ X_n)$

où X et X_i représentent des noms de catégories syntaxiques.
On dit que :

- X est le gouverneur
- $X_1\ X_2 \ \ldots\ X_n$ sont les dépendants du gouverneur X. Dans la règle 2, l'étoile (*) marque la place du gouverneur par rapport à ses dépendants.

- Les règles du type 1 indiquent les gouverneurs possibles d'une phrase.
- Les règles du type 2 dans le cas où n = 0 (X(*)) sont des règles terminales.

NB C'est une grammaire générative faiblement équivalente à une grammaire "hors contexte" [GAIF].

Génération

Le système de génération est le suivant :

1°) On choisit un gouverneur possible grâce à une règle de type 1.

2°) On choisit ensuite des règles de type 2 jusqu'à obtenir une expression complètement parenthésée (la structure de la phrase, appelée structure de dépendances) uniquement composée de règles terminales).

Exemple

Soient :

a) les catégories SUBC, PRAR, VERB définies par :

SUBC = {chien, soupe}

PRAR = {le, la}

VERB = {mange}

b) Les règles

*(SUBC)

*(VERB)

SUBC(PRAR *)

VERB(SUBC * SUBC)

VERB(*)
SUBC(*)
PRAR (*)

On peut générer :

a) *(SUBC)

*(SUBC(PRAR *))

*(SUBC(PRAR(*) *))

*(SUBC(*)(PRAR(*) *))

ce qui donne (chien (le *))
ou (soupe (la *))

d'où les structures de dépendances :

b) *(VERB)

*(VERB(SUBC * SUBC))

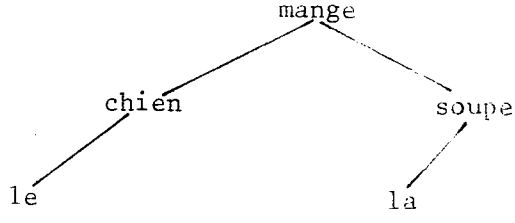
*(VERB(SUBC(PRAR *) * SUBC(PRAR *)))

*(VERB(*) (SUBC(*) (PRAR(*) *) * SUBC(*) (PRAR(*) *)))

ce qui donne :

(mange (chien (le *) * soupe (la *)))

d'où la structure



Adéquation des grammaires de dépendances

A partir des règles de dépendances, on peut définir un algorithme d'analyse [HAYS, VAUQ] permettant d'obtenir des structures de dépendances qui sont interprétables. Pour illustrer cette interprétation, nous donnons ci-après un exemple réalisé par JOLOBOFF à partir de structures de dépendances élaborées par P.I.A.F.

Validation et interprétation d'une structure de dépendances

Dans le cas où un contexte précis fige la sémantique, on peut constituer un "dictionnaire sémantique". Ainsi, à chaque mot du vocabulaire, on peut associer les concepts auxquels se réfèrent les éventuels dépendants de ce mot et le concept auquel il se réfère lui-même. Dans un cours d'enseignement assisté sur la loi des chocs interviennent des concepts de temps, de vecteurs,de réels par exemple. En outre, on sait : que la norme est une opération portant sur un seul vecteur et que le résultat est un réel que la somme de deux vecteurs est un vecteur et que la somme de deux réels est un réel. On sait également que les expressions "avant" et "après" introduisent un concept de temps.

On peut donc constituer le dictionnaire suivant :

norme(vecteur) → réel

somme(réel,réel) → réel

somme(vecteur,vecteur) → vecteur

vitesse(objet, instant) → vecteur

avant (.....) → instant

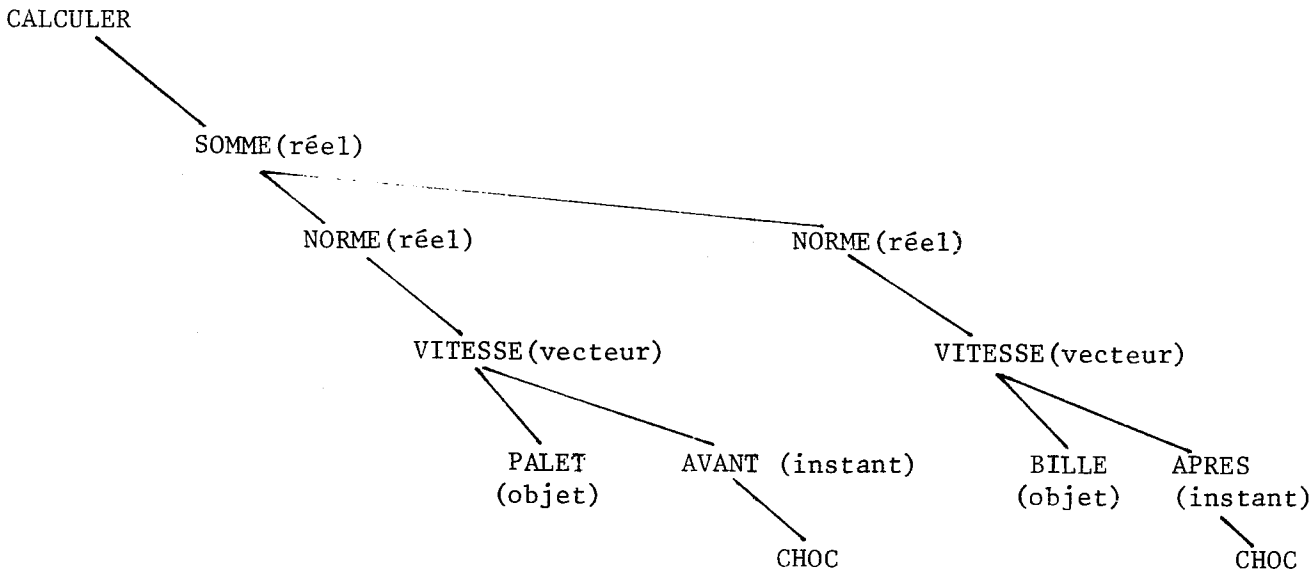
après (.....) → instant

palet (vide) → objet

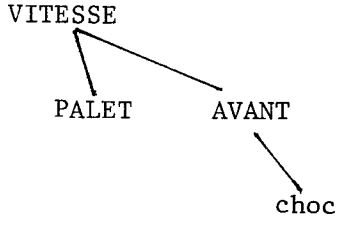
bille (vide) → objet.

Celà signifie que la norme est un nombre réel qui a un seul dépendant qui doit être un vecteur. De même la somme qui est un nombre réel doit avoir deux dépendants qui sont des nombres réels ou la somme qui est un vecteur doit avoir deux dépendants qui sont des vecteurs.
... signifie qu'il n'y a aucune contrainte de dépendances.
vide signifie que cet élément n'a pas de dépendants.

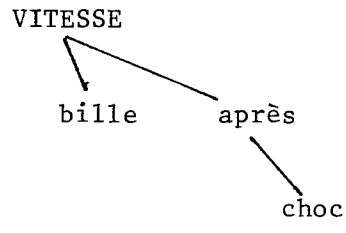
Soit la phrase : "Calculer la somme de la norme de la vitesse du palet avant le choc et de la norme de la vitesse de la bille après le choc". Le système P.I.A.F. propose plusieurs structures de dépendances qui sont ensuite filtrées par le programme de Joloboff, en ayant préalablement supprimé les informations inutiles telles que les articles, à l'aide du dictionnaire sémantique précédent.
Il obtient la seule structure :



laquelle est interprétée facilement si l'on appelle V1 la vitesse du palet avant le choc et V'2 la vitesse de la bille après le choc.



V1

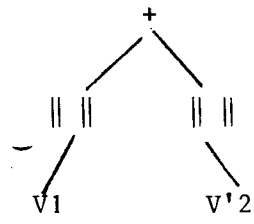


V'2

||V1|| et ||V'2|| les sous-structures : NORME et NORME

V1 V'2

On obtient :



qui donne ||V1|| + ||V'2||.

L'inconvénient majeur des grammaires de dépendances provient du fait qu'il faut décrire toutes les combinaisons possibles entre un gouverneur et ses dépendants. Par exemple, pour le groupe nominal, il faut écrire la grammaire suivante :

*(SUBC)

SUBC(PRAR *)

SUBC(PRAR ADJQ *)

SUBC(PRAR ADJQ ADJQ *)

SUBC(PRAR * ADJQ)

SUBC(PRAR ADJQ * ADJQ)

SUBC(PRAR ADJQ ADJQ * ADJQ)

Il est à noter qu'il existe d'autres types de grammaires conduisant à des structures interprétables telles que les grammaires systémiques de WINOGRAD [WIN].

Nous avons choisi les structures de dépendances mais, afin de réduire le nombre de règles, nous proposons au chapitre suivant un système de relations de dépendances et un algorithme d'analyse qui permettent d'obtenir des structures interprétables en acceptant un risque raisonnable. (celui d'accepter des phrases mal formées).

B1 ANALYSEUR DE DEPENDANCES

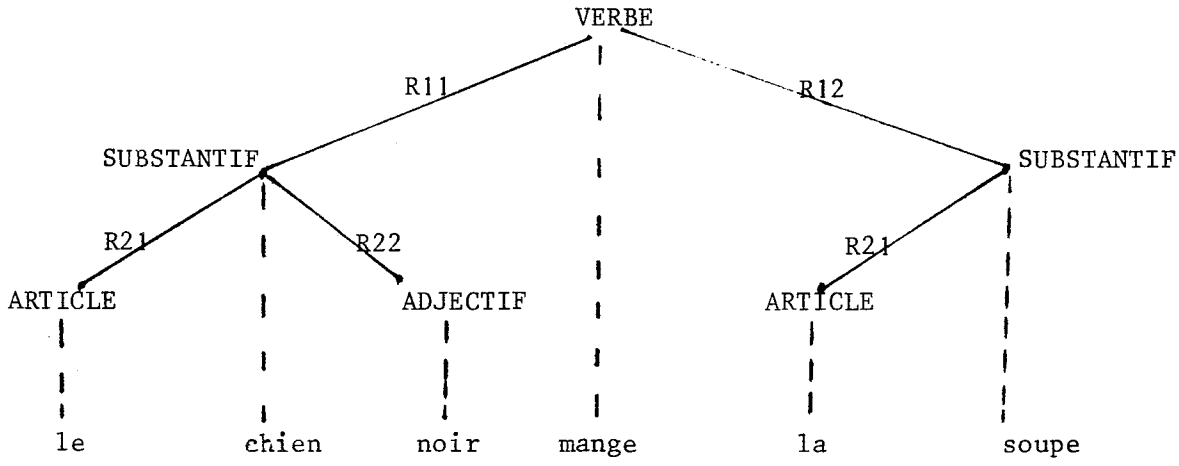
I - INTRODUCTION

Le paramètre de cet analyseur ne sera pas constitué, comme dans les analyseurs traditionnels ("hors contexte"), d'une liste de règles de grammaire mais par une liste de relations de dépendances entre les catégories comme le font KULAGINA [VAKU] et TSEITINE [TSE]. Ils décrivent toutes les combinaisons possibles entre un gouverneur et un dépendant alors que nous proposons ci-après un moyen d'éviter cette combinatoire. De plus, on peut noter que l'algorithme proposé par KULAGINA n'est pas général car il tient compte des particularités de la langue, que celui de TSEITINE n'est qu'un algorithme de prétraitement syntaxique et que tous les deux ne possèdent pas de métalangage pour écrire leurs relations. Nous étudions tout d'abord la forme de nos relations de dépendances et ensuite l'algorithme d'analyse que nous préconisons.

II - LES RELATIONS DE DEPENDANCES

II.1 - PRINCIPE

Prenons un exemple : soit la phrase :
"Le chien noir mange la soupe", en fonction de cet énoncé et de relations entre les catégories, on veut obtenir la structure suivante :



Pour cette structure nous avons quatre relations :

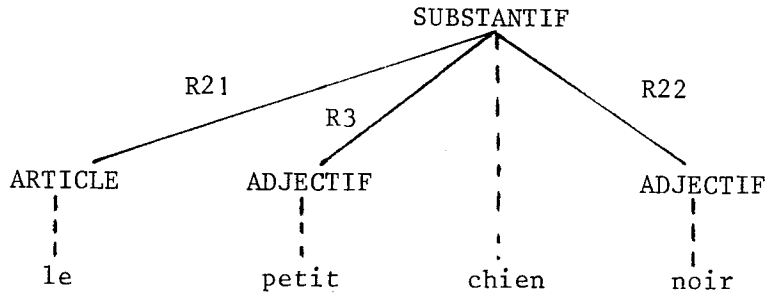
: R11 qui indique que le substantif ("chien") est dépendant gauche du verbe ("mange" qui est le gouverneur de la phrase)

: R12 qui indique que le substantif ("soupe") est dépendant droit du verbe (mange)

: R21 qui indique que l'article ("le" ou "la") est respectivement dépendant gauche du substantif ("chien" ou "soupe")

: R22 qui indique que l'adjectif ("noir") est dépendant droit du substantif ("chien")

Il faudra donc que ces relations puissent indiquer si le dépendant est à droite ou à gauche de son gouverneur.
De plus, si l'on examine la structure de la phrase
"le petit chien noir"



on s'aperçoit que cette indication est insuffisante, car non seulement il faut que la relation R3 indique que l'adjectif ("petit") doit se trouver à gauche de son gouverneur , le substantif ("chien"), mais que par rapport à l'article ("le") qui est aussi un dépendant gauche du même gouverneur, il doit se trouver moins à gauche.

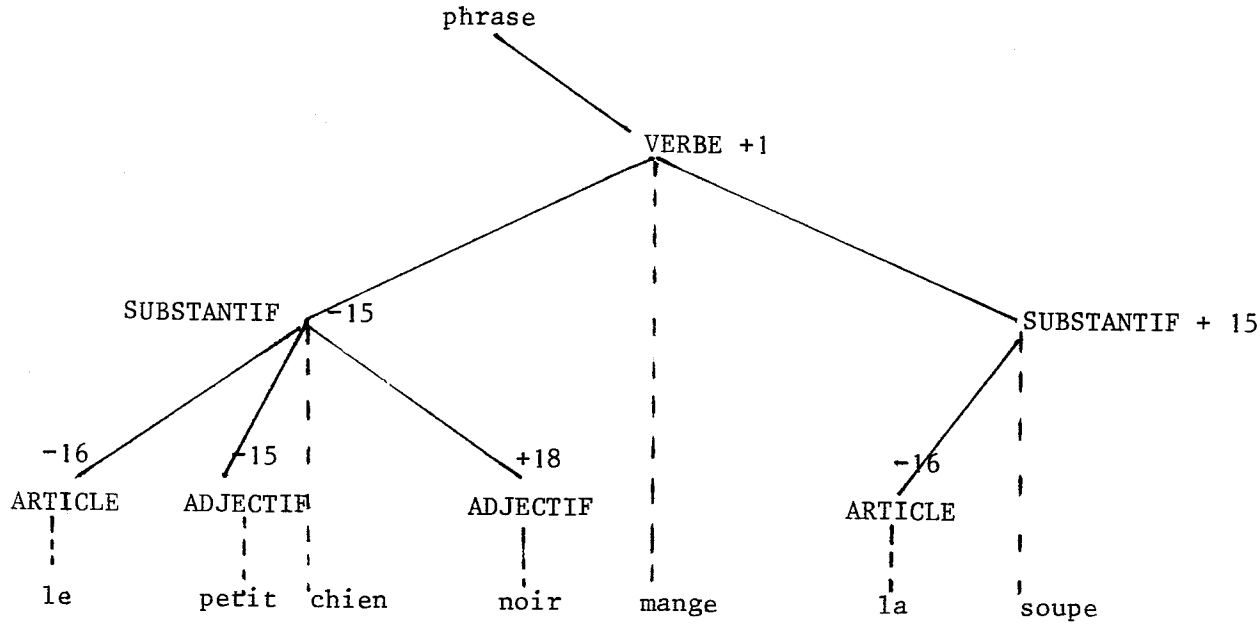
Pour établir ces relations, nous avons utilisé la notion de poids qui avait été introduite dans le système de traduction automatique du C.E.T.A. pour effectuer la génération du français [VEILL1].

Poids

A chaque sommet de la structure de dépendances, on affecte un entier : le poids. Les poids négatifs correspondent aux mots placés à gauche du gouverneur : les dépendants gauches, et les poids positifs à ceux placés à droite : les dépendants droits.

Afin de pouvoir déterminer les gouverneurs possibles d'une phrase, nous avons ajouté une nouvelle catégorie syntaxique "phrase". D'autre part, le poids doit également représenter la place de tous les dépendants par rapport à un même gouverneur. Enfin, d'éviter les problèmes liés à la récursivité (cf. introduction), tous les dépendants d'un même gouverneur doivent avoir des poids différents.

Exemple :



Les relations de dépendances sont donc les suivantes :

(1) PHRA★VERB:=+1 ; cette relation indique que le gouverneur principal est un verbe.

(2) VERB★SUBC:=-15,+15 ; cette relation précise qu'il peut y avoir, à gauche (-15) ou à droite (+15) d'un verbe, un substantif.

(3) SUBC★ARTD:=-16 ; cette relation précise qu'à gauche d'un substantif, on peut trouver un article

(4) SUBC★ADJQ:=-15,+18 ; cette relation précise qu'à gauche et à droite d'un substantif, on peut trouver un adjectif qualificatif.

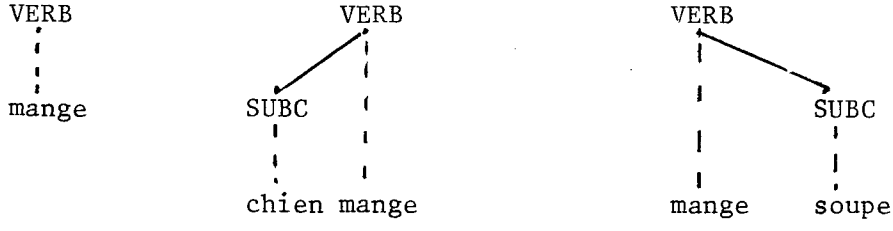
NB - Une relation de dépendances :

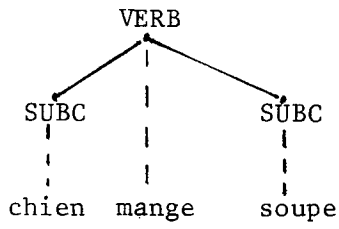
GOUVERNEUR★DEPENDANT:= $x_1, x_2, \ldots, x_n$; avec

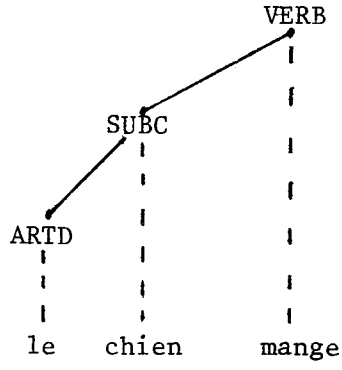
$- \infty < x_1 < x_2 < \ldots x_j < 0 < x_{j+1} < \ldots < x_n < + \infty$

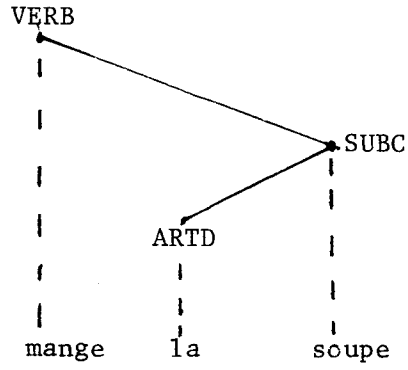
signifie qu'il peut y avoir 0, 1, 2 ou n dépendants de la catégorie "DEPENDANT" du gouverneur "GOUVERNEUR" de poids respectifs $x_1, x_2, \ldots, x_n$. C'est le langage que nous mettons à la disposition de l'utilisateur.

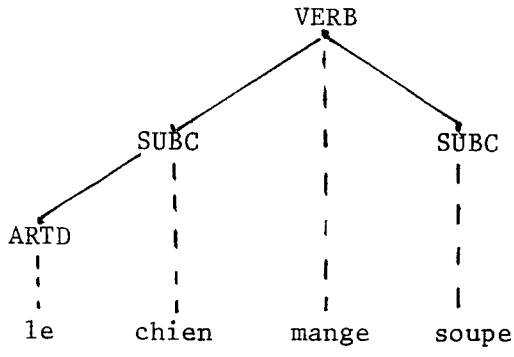
En conséquence, cet ensemble de quatre relations précédentes permet d'obtenir l'analyse et la construction des arborescences suivantes, reconnues comme des phrases :

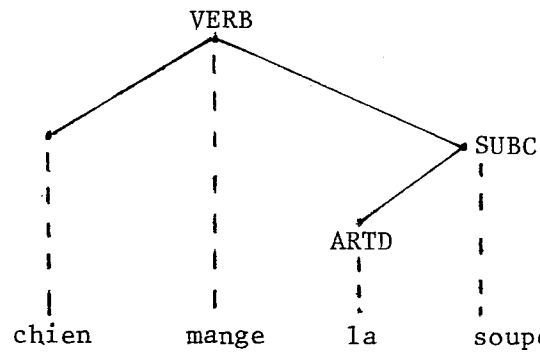



VERB
SUBC
SUBC
chien
mange
soupe

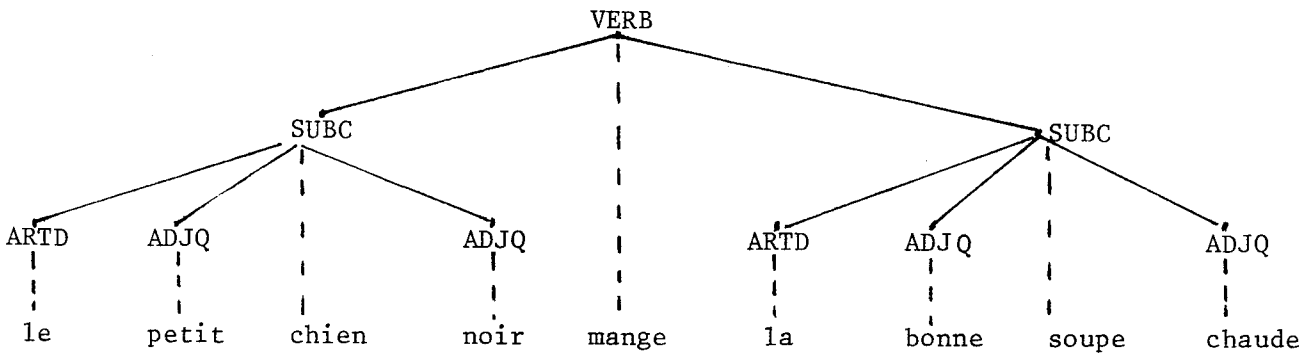

VERB
SUBC
ARTD
le
chien
mange


VERB
SUBC
ARTD
mange
la
soupe


VERB
SUBC
SUBC
ARTD
le
chien
mange
soupe


VERB
SUBC
ARTD
chien
mange
la
soupe

etc...


VERB
SUBC
SUBC
ARTD
ADJQ
ADJQ
ARTD
ADJQ
ADJQ
le
petit
chien
noir
mange
la
bonne
soupe
chaude

II.2 - CONSTRUCTION DES RELATIONS DE DEPENDANCES

Supposons que nous ayons déjà défini les relations suivantes :

(1) PHRA*SUBC:=+1 ;

(2) SUBC*ARTD:=-16 ;

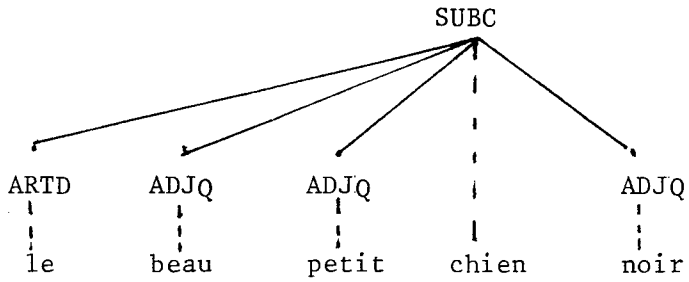
(3) SUBC*ADJQ:=+18 ;

qui permettent l'analyse des phrases nominales telles que

"chien", "chien noir", "le chien", "le chien noir".

et supposons que nous voulions analyser une phrase telle que :

"le beau petit chien noir" dont la structure est :



il faut compléter la relation (3) en tenant compte de la relation (2). Si x et y sont les poids des adjectifs "beau" et "petit", ils doivent vérifier la relation :

$$-16 < x < y < 0$$

On choisit un couple x,y vérifiant cette relation, par exemple x = -15 et y = -14. D'où l'écriture de la relation (3) :

SUBC*ADJQ:=-15, -14, +18 ;

on obtient donc les relations :

1) PHRA*SUBC := +1 ;

2) SUBC*ARTD := -16 ;

3) SUBC*ADJQ := -15, -14, +18 ;

La grammaire de dépendances équivalente aux trois relations précédentes serait la suivante :

*(SUBC)	équivalente de PHRA*SUBC := +1 ;
SUBC(ARTD *)	équivalente de SUBC*ARTD := -16 ;
SUBC(ADJQ *) SUBC(* ADJQ) SUBC(ADJQ ADJQ *) SUBC(ADJQ ADJQ * ADJQ) SUBC(ADJQ * ADJQ)	équivalentes de SUB*ADJQ := -15,-14,+18 ;
SUBC(ARTD ADJQ *) SUBC(ARTD * ADJQ) SUBC(ARTD ADJQ ADJQ *) SUBC(ARTD ADJQ ADJQ * ADJQ) SUBC(ARTD ADJQ * ADJQ)	résultent du poids -16 de l'article et des poids respectifs des adjectifs.
SUBC(*) ARTD(*) ADJQ(*)	règles terminales

III - ALGORITHME D'ANALYSE

Soit une phrase $X_1\ X_2 \ldots X_n$ où X_i est une catégorie syntaxique obtenue par l'analyseur morphologique. En fonction des relations de dépendances, on construit un tableau à deux dimensions A(n+1,n+1) où n est le nombre de catégories de la phrase.

GOUVERNEUR / DEPENDANT	PHRA X_o	X_1	X_2	.....	X_n
PHRA X_0	$\emptyset$				
X_1	P_{10}	$\emptyset$	P_{12}		P_{1n}
X_2	P_{20}	P_{21}	$\emptyset$		P_{2n}
⋮					
X_n	P_{n0}	P_{n1}	P_{n2}		$\emptyset$

où P_{ij} : ensemble des poids déterminé par les relations, entre le dépendant X_i et le gouverneur X_j.

Propriétés de ce tableau

α) $P_{ii} = \emptyset \qquad \forall i \in [0,n]$

β) Etant donné qu'on ne traite que des structures de dépendances projectives, on a les propriétés suivantes :

β1) $\forall i, \forall j \in [1,n]$ avec $i > j$, si l'ensemble P_{ij} n'est pas vide, il ne peut être constitué que de poids strictement positifs.

β2) $\forall i, \forall j \in [1,n]$ avec $i < j$, si l'ensemble P_{ij} n'est pas vide, il ne peut être constitué que de poids strictement négatifs.

Le premier pas de l'algorithme consiste donc en la suppression des poids positifs ou des poids négatifs des ensembles P_{ij} suivant que $i < j$ ou $i > j$.

III.1 - ETUDE DE PLUSIEURS DEPENDANTS A DROITE D'UN GOUVERNEUR

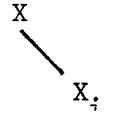
III.1.1. - Principe

Etant donné qu'il y a indépendance entre les dépendants droits et les dépendants gauches d'un gouverneur, nous étudions les conditions de dépendance (à droite par exemple) de n éléments. Il est évident que le problème est identique pour les dépendants gauches.

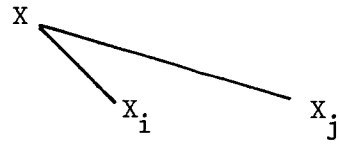
Soit X un gouverneur et X_1 X_2 ... X_n n dépendants droits éventuels de X avec : X, X_1, X_2 ,..., X_n apparaissant dans cet ordre dans la phrase.

On a donc n ensembles de poids P_1, P_2 ,...,P_n entre le gouverneur X et les dépendants éventuels X_i avec $i \in [1,n]$. Il s'agit ici d'étudier toutes les combinaisons possibles entre X et les X_i.

Une première solution (évidente) est de considérer séparément les dépendants :



ensuite, il faut examiner la possibilité de deux dépendants



avec

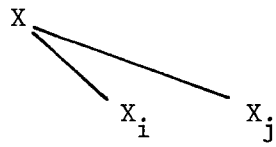
$$1 \leq i < j \leq n$$

puis de trois, etc... jusqu'à n.

Nous donnons maintenant le principe qui permet de déterminer la possibilité de k dépendants ($2 \leq k \leq n$)

1°) $k = 2$.

Pour obtenir

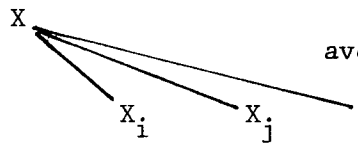


avec $1 \leq i < j \leq n$

il faut qu'il existe un poids de P_j qui soit supérieur à un poids de P_i.

2°) $k = 3$

De même pour obtenir



avec $1 \leq i < j < 1 \leq n$

il faut trouver un poids de P_1 qui soit supérieur à un poids de P_j, lui-même supérieur à un poids de X_i.

Le principe est donc le suivant :

soit m_1 le plus petit poids de P_1 que nous notons :

$m_1 = MIN(P_1)$

soit m_2 l'élément de P_2 qui soit tel que

a) $m_2 > m_1$

b) Il n'existe pas d'élément $m_2' \in P_2$ tel que $m_2 > m_2' > m_1$

On définit de même m_3, m_4 ,...,m_n.
On a donc :

1°) $m_1 = MIN(P_1)$

2°) $m_j > m_{j-1}$ $m_j \in P_j$ et $m_{j-1} \in P_{j-1}$ $j \in [2,n]$

et tel qu'il n'existe pas d'éléments $m_j' \in P_j$ tel que

$$m_j > m_j' > m_{j-1}$$

Si on a pu construire k éléments m_i, c'est que le gouverneur X peut admettre k dépendants à droite.

Pour les dépendants gauches, on applique le même principe en changeant les relations MIN et "<" par MAX et ">".

Propriétés

Si un gouverneur X admet un n-uple X_1 X_2 ... X_n (n > 1) comme dépendants droits, il admet n(n-1) uples formé à partir des éléments X_1, X_2 ,...,X_n.

L'algorithme que nous proposons est donc le suivant : soit X un gouverneur dont on veut déterminer toutes les combinaisons possibles de dépendants droits de rang compris entre rang(X)+1 à m inclus où rang(X) est une fonction entière donnant le rang de X dans la phrase et m un rang quelconque supérieur à rang(X).

Nous calculons les 1-uple possibles, puis les combinaisons de 2-uples possibles que l'on détermine à partir des 1-uples etc... des n-uples possibles construits à partir des (n-1) uples.

III.1.2. - Schémas d'algorithme

Nous utilisons les fonctions suivantes :

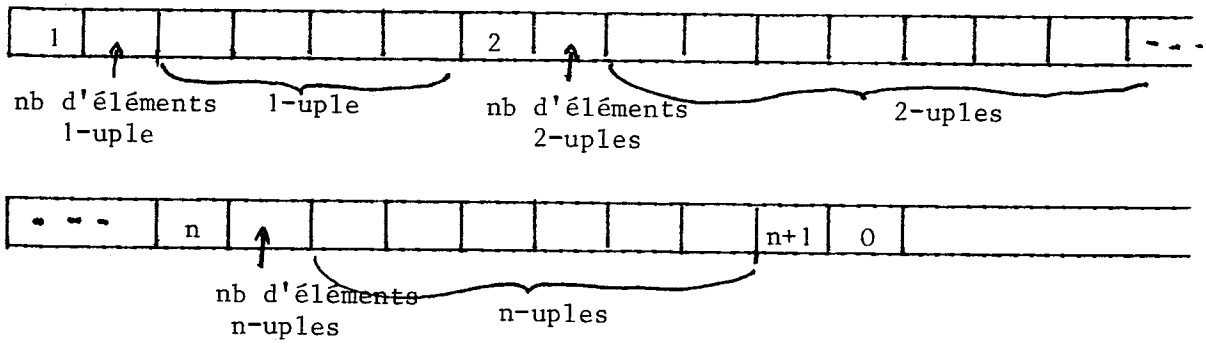
RANG(X) : fonction entière déterminant le rang de X dans la phrase

CAT(j) : fonction donnant la catégorie X de l'élément de rang j
On a donc RANG(CAT(j)) = j
CAT(RANG(X)) = X

Poids(X,Y) : fonction ayant comme résultat un vecteur contenant la valeur des poids entre le gouverneur de catégorie X et le dépendant de catégorie Y. Ce vecteur est ordonné suivant l'ordre croissant des valeurs de poids.

vide(P) : où P est un vecteur. Le prédicat délivre la valeur VRAI si le vecteur est vide, FAUX dans le cas contraire.

Cet algorithme construit un vecteur RESULTAT ayant la structure suivante :



```
ACTION  DEPENDANTDROIT(ENTREE : X,R, SORTIE : RESULTAT)
    CO R ≥ RANG(X) CO

    RX ← RANG(X)
    RESULTAT(1) ← 1
    IR ← 2          CO indice de RESULTAT CO
    IM ← 0          CO indice du vecteur M des minimas CO
    J ← RX+1
    NUPLE ← 1
    CO CONSTRUCTION DES 1-UPLE POSSIBLES CO
    TANTQUE J ≤ R
       FAIRE
          SI NON VIDE(P ← POIDS(X, CAT(J)))
             ALORS
                RESULTAT(IR ← IR+1) ← J
                M(IM ← IM+1) ← P(1)
             FINSI
          J ← J+1
       FINFAIRE
    RESULTAT(2) ← IR-2 CO nombre de 1-uple CO
    CO CONSTRUCTION DES N-UPLES SUIVANTS CO
    FUTURIM ← NBUPLEPRECEDENT ← RESULTAT(INUPLEPRECEDENT ← 2)
    IM ← 0

    TANTQUE NBUPLEPRECEDENT ≠ 0
       FAIRE
          RESULTAT(IR ← IR+1) ← NUPLE ← NUPLE+1
          IR ← IR+1
          J ← INUPLEPRECEDENT+NUPLE-1
          TANTQUE J ≤ NBUPLEPRECEDENT*(NUPLE-1)+IN UPLEPRECEDENT
             FAIRE
                POI ← M(IM ← IM+1)
                K ← J
                TANTQUE RANGPOSSIBLEAPRES(RESULTAT(K),K)
                   FAIRE
                      K' ← RESULTAT(K)
                      MP ← ELEMENTMIN(POIDS(X,CAT(K')),POI)
                      SI MP ≠ 0
```

```
                                        ALORS
                                          |  REMPLIR(J,NUPLE,K',RESULTAT,IR)
                                          |  M(FUTURIM ← FUTURIM+1) ← MP
                                        FINSI
                                    FINFAIRE
                                    J ← J+NUPLE-1
                            FINFAIRE
                        NBUPLEPRECEDENT ← (IR-(J-NUPLE+3))%NUPLE
                        RESULTAT(IN UPLEPRECEDENT ← J-NUPLE+3) ← NBUPLEPRECEDENT
                FINFAIRE
FINACTION
```

```
PREDICAT  RANGPOSSIBLEAPRES (ENTREE : J , SORTIE: K)
    CO parcours des 1-uples CO
    NBUPLE1← RESULTAT(2)
    L ← 3
    TANTQUE L < NBUPLE1+2 ET J > RESULTAT(L)
       FAIRE
          |  L ← L+1
       FINFAIRE
    SI L = NBUPLE1+2
       ALORS
          |  RANGPOSSIBLEAPRES ← FAUX
       SINON
          |  RANGPOSSIBLEAPRES ← VRAI
          |  K ← L+1
       FINSI
FINPREDICAT
```

```
FONCTION    ELEMENTMIN (ENTREE : P, POI)
    |
    |   N ← CARDINAL(P)
    |   SI POI ≥ P(N)
    |      ALORS
    |         |   ELEMENTMIN ← 0
    |      SINON
    |         |   J ← 1
    |         |   TANTQUE POI ≥ P(J)
    |         |      FAIRE
    |         |         |   J ← J+1
    |         |      FINFAIRE
    |         |   ELEMENTMIN ← P(J)
    |      FINSI
FINFONCTION
```

```
ACTION  REMPLIR(ENTREE : J, NUPLE, K, ENTREE-SORTIE : RESULTAT, IR)
    |   L ← J-NUPLE+1
    |   I ← 1
    |   TANTQUE I ≤ NUPLE-1
    |      FAIRE
    |         |   RESULTAT(IR ← IR+1) ← RESULTAT(L ← L+1)
    |         |   I ← I+1
    |      FINFAIRE
    |   RESULTAT(IR ← IR+1) ← K
FINACTION
```

III.2 - PRINCIPE DE L'ALGORITHME

Le principe de l'algorithme est de construire tous les arbres possibles compatibles avec les relations de dépendances. Ayant déterminé un candidat gouverneur, on a deux possibilités :

- Soit déterminer un sous-arbre gauche et un sous-arbre droit. Ensuite par une technique de retour arrière "backtracking" construire tous les sous-arbres possibles. Cet algorithme, qui s'écrit récursivement, nécessite en plus l'utilisation d'une pile d'états mémorisant pour chaque noeud le contexte de la solution choisie afin de permettre le retour arrière.

- Soit déterminer tous les sous-arbres gauches puis tous les sous-arbres droits. Ensuite un algorithme de sélection permet de construire les arbres correspondants.

Pratiquement, nous avons choisi la première possibilité. Pour des raisons de clarté, nous donnons dans les pages qui suivent les schémas d'algorithmes correspondant à la deuxième possibilité car cet algorithme, excepté le stockage des solutions partielles, s'écrit complètement récursivement.

III.2.1. - Schémas d'algorithme

Afin de ne pas trop rentrer dans les détails, nous disposons des actions suivantes :

- GOUVERNEUR PHRASE(CATEGORIES, N, GOUVERNEUR, NBGOUV)
  qui détermine les NBGOUV gouverneur possibles stockés dans le vecteur GOUVERNEUR.

- PREMIERDEPENDANT (RESULTAT, DEP)

  S' il existe au moins un dépendant,
  - DEP contient la première possibilité
  - DERNIER(RESULTAT) prend la valeur VRAI

Sinon

- DERNIER(RESULTAT) prend la valeur FAUX
- Le contenu de DEP est non significatif.

- SUCCESSEUR (RESULTAT, DEP)

S' il existe encore une combinaison de dépendants,

- DEP contient la première solution restante
- DERNIER(RESULTAT) prend la valeur VRAI

Sinon

- DERNIER(RESULTAT) prend la valeur FAUX
- Le contenu de DEP est non significatif.

- PREMIERELEMENT (DEP, P), SUIVANT(DEP,S), DERNIER(DEP) permettent comme précédemment d'énumérer tous les dépendants d'une combinaison choisie stockée dans DEP.

```
ACTION  ANALYSE(CATEGORIES, N)
    |
    |   CO
    |   |  CATEGORIES : VECTEUR CONTENANT LA CATEGORIE DE CHAQUE UNITE
    |   |  LEXICALE DE LA PHRASE.
    |   |  N : NOMBRE D'UNITES LEXICALES
    |   CO
    |
    |   GOUVERNEUR PHRASE (CATEGORIES, N, GOUVERNEUR, NBGOUV)
    |   I ← 1
    |   TANTQUE I ≤ NBGOUV
    |      FAIRE
    |         |  SI BRANCHEGAUCHE(GOUVERNEUR(I),1)
    |         |     ALORS
    |         |        |  BD ← BRANCHEDROITE(GOUVERNEUR(I),N)
    |         |     FINSI
    |         |  I ← I+1
    |      FINFAIRE
    |
FINACTION
```

```
PREDICAT   BRANCHEGAUCHE(ENTREE : X,R)
|
|     CO X : CATEGORIE, R : RANG D'UNE CATEGORIE Y
|     |      tel que RANG(X) ≥ (RANG(Y) = R)
|     CO
|     SI R = RANG(X)
|        ALORS
|        |    BRANCHEGAUCHE ← VRAI
|        SINON
|        |    SOLUTION ← FAUX
|        |    DEPENDANTGAUCHE(X,R,RESULTAT)
|        |    PREMIERDEPENDANT(RESULTAT,DEP)
|        |    TANTQUE NON DERNIER(RESULTAT)
|        |       FAIRE
|        |       |    PREMIERELEMENT(DEP,P)
|        |       |    SI BRANCHEGAUCHE(P,R)
|        |       |      ALORS
|        |       |      |    CPLE ← VRAI
|        |       |      |    SUIVANT(DEP,S)
|        |       |      |    TANTQUE NON DERNIER(DEP) ET CPLE
|        |       |      |       FAIRE
|        |       |      |       |    CPLE ← COUPLE(P,S)
|        |       |      |       |    P ← S
|        |       |      |       |    SUIVANT(DEP,S)
|        |       |      |       FINFAIRE
|        |       |      |    SI CPLE
|        |       |      |       ALORS
|        |       |      |       |    SOLUTION ← SOLUTION OU
|        |       |      |       |    BRANCHEDROITE(P,RANG(X)-1)
|        |       |      |       FINSI
|        |       |      FINSI
|        |       |    SUCCESSEUR(RESULTAT,DEP)
|        |       FINFAIRE
|        |    BRANCHEGAUCHE ← SOLUTION
|        FINSI
FINPREDICAT
```

```
PREDICAT  BRANCHEDROITE(ENTREE : X, R)
    CO X : CATEGORIE, R : RANG D'UNE CATEGORIE Y
    |  tel que RANG(X) ≤ (RANG(Y) = R)
    CO
       SI R = RANG(X)
          ALORS
          |   BRANCHEDROITE ← VRAI
          SINON
          |   SOLUTION ← FAUX
          |   DEPENDANTDROIT(X,R,RESULTAT)
          |   PREMIERDEPENDANT(RESULTAT,DEP)
          |   TANTQUE NON DERNIER(RESULTAT)
          |      FAIRE
          |      |   PREMIERELEMENT(DEP,P)
          |      |   SI BRANCHEGAUCHE(P,RANG(X)+1)
          |      |      ALORS
          |      |      |   CPLE ← VRAI
          |      |      |   SUIVANT(DEP,S)
          |      |      |   TANTQUE NON DERNIER(DEP) ET CPLE
          |      |      |      FAIRE
          |      |      |      |   CPLE ← COUPLE(P,S)
          |      |      |      |   P ← S
          |      |      |      |   SUIVANT(DEP,S)
          |      |      |      FINFAIRE
          |      |      |   SI CPLE
          |      |      |      ALORS
          |      |      |      |   SOLUTION ← SOLUTION OU
          |      |      |      |   BRANCHEDROITE(P,R)
          |      |      |      FINSI
          |      |      FINSI
          |      |   SUCCESSEUR(RESULTAT,DEP)
          |      FINFAIRE
          |   BRANCHEDROITE ← SOLUTION
          FINSI
FINPREDICAT
```

```
PREDICAT   COUPLE(X,Y)
    CO  X : CATEGORIE, Y : CATEGORIE tels que
    |   RANG(X) < RANG(Y)
    CO
    SI  RANG(X) = RANG(Y)-1
        ALORS
        |   COUPLE ← VRAI
        SINON
        |   CP ← BRANCHEDROITE(X, R ← RANG(Y)-1)
        |   R ← R-1
        |   TANTQUE R ≥ RANG(X)
        |       FAIRE
        |       |   CP ← CP OU (BRANCHEDROITE(X,R)
        |       |                   ET BRANCHEGAUCHE(Y,R+1)
        |       |   R ← R-1
        |       FINFAIRE
        |   COUPLE ← CP
        FINSI
FINPREDICAT
```

III.2.2. - Exemple

Soit la phrase :

"Le beau petit chien jaune et noir mange la soupe de poisson".
L'analyse morphologique permet d'obtenir les catégories suivantes:

ARTD ADJQ ADJQ SUBC ADJQ COCO ADJQ VERB ARTD SUBC PREP SUBC

On dispose des relations suivantes :

PHRA *SUBC := +1 ;
PHRA*COCO := +1 ;
PHRA*VERB := +1 ;

VERB*SUBC := -20, +20 ;
VERB*PREP := +30 ;

SUBC*ARTD:= -16 ;
SUBC*ADJQ := -15, -14, +18 ;
SUBC*COCO := +18 ;
SUBC*PREP := +20 ;

COCO*ADJQ := -10, +10 ;
PREP*SUBC := +7 ;

TABLEAU DE LA PHRASE EN TENANT COMPTE DE LA PROJECTIVITE

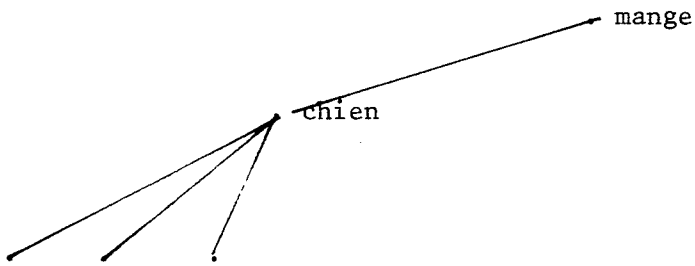
	PHRA	ARTD le	ADJQ beau	ADJQ petit	SUBC chien	ADJQ noir	COCO et	ADJQ jaune	VERB mange	ARTD la	SUBC soupe	PREP de	SUBC poisson
PHRA	0												
ARTD		0			-16						-16		-16
ADJQ			0		-15 -14		-10				-15 -14		-15 -14
ADJQ				0	-15 -14		-10				-15 -14		-15 -14
SUBC	+1				0				-20				
ADJQ					+18	0	-10				-15 -14		-15 -14
COCO	+1				+18		0						
ADJQ					+18		+10	0			-15 -14		-15 -14
VERB	+1								0				
ARTD										0	-16		-16
SUBC	+1								+20		0		
PREP					+20				+30		+20	0	
SUBC	+1								+20			+7	0

Prenons par exemple, le gouverneur VERB ("mange).

. mange

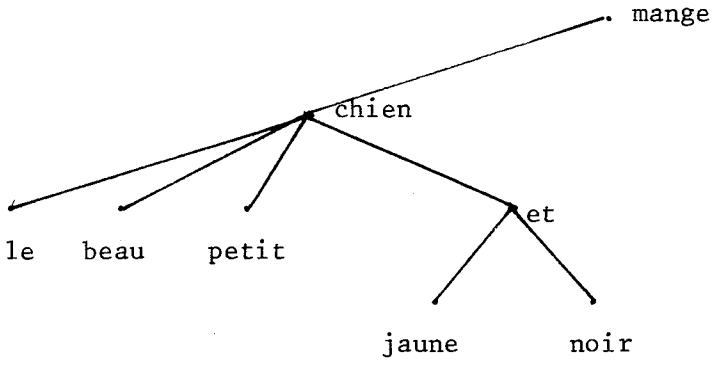
à gauche, on construit l'unique solution :

à gauche de "chien" on construit l'unique solution :



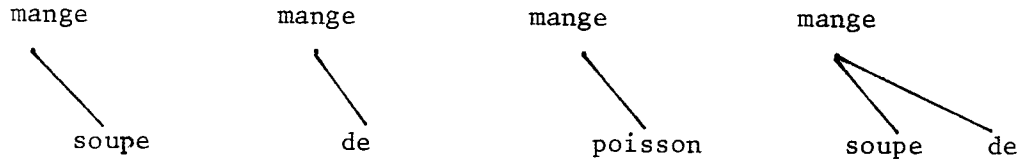
à droite de "chien" on construit les solutions :

La première et la troisième ne permettent pas la construction d'un sous-arbre avec le "et" d'où la solution :

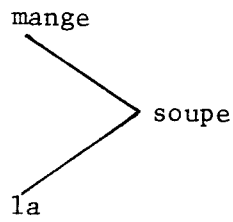


Le sous-arbre gauche du VERB ("mange") étant terminé, on examine maintenant la construction des sous-arbres droits du VERB.

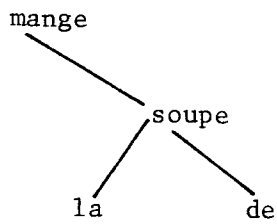
A droite, on obtient 4 solutions :



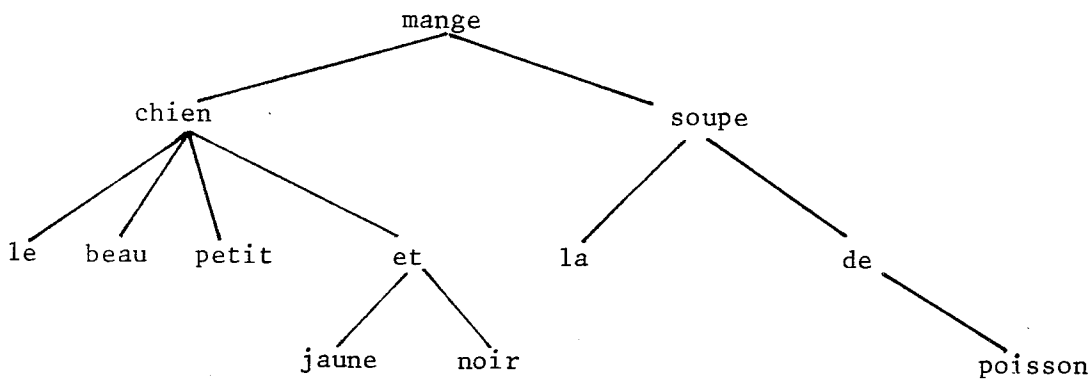
A gauche de "soupe" on obtient l'unique solution



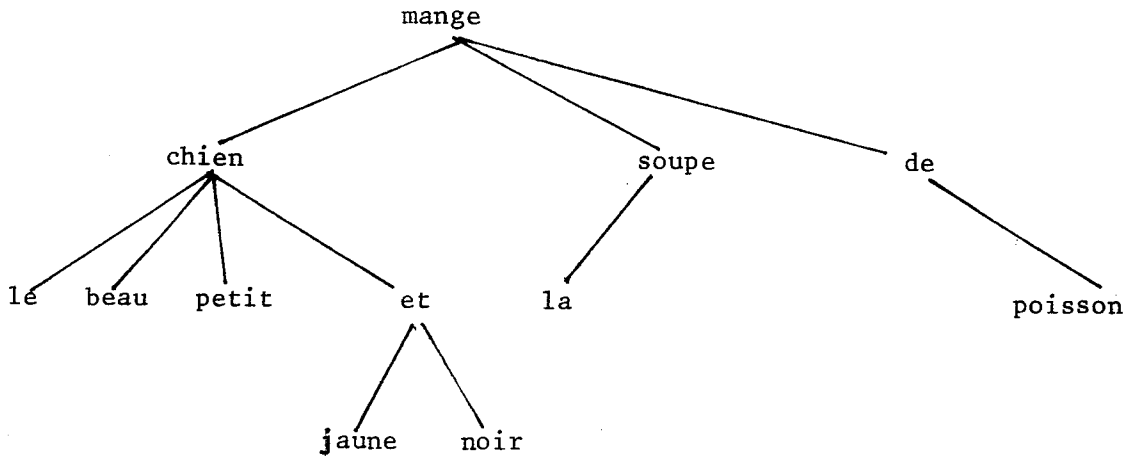
à droite de "soupe", on obtient l'unique solution



à droite de "de", on obtient une solution, d'où première structure possible :



Les 2ème et 3ème solutions de "mange" ne permettent pas la construction d'un sous-arbre gauche. Par contre la dernière permet l'obtention d'une deuxième solution :



Cette dernière solution, qui est fausse, ne peut être supprimée que par un modèle d'un autre type : on peut manger de bon appétit mais pas de poisson.

III.3 - PROBLEMES LINGUISTIQUES LIES AUX STRUCTURES DE DEPENDANCES

Nous nous intéressons ici aux problèmes posés par les compléments de noms, de relatives et les conjonctions de coordination.

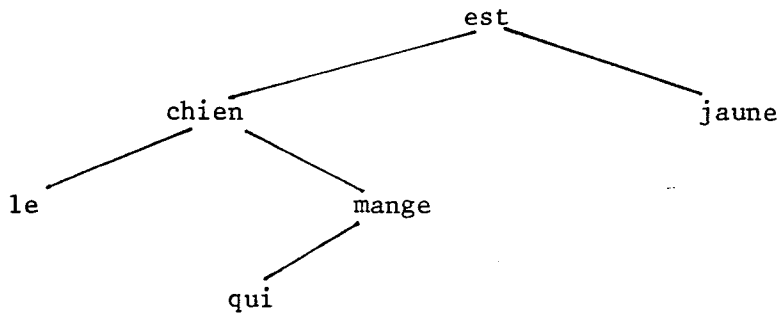
III.3.1. - Les compléments de noms

Dans le cas des compléments de noms reliés entre eux par la préposition "de", il est aisé de constater, comme sur l'exemple précédent, que l'algorithme propose toujours les deux solutions : soit le "de" est dépendant du verbe, soit le "de" est dépendant du substantif.
Pour lever cette indétermination, un modèle supplémentaire est nécessaire, et il ne peut être que d'une autre classe. Dans le cas des compléments de verbe, le problème est le même, mais il peut être résolu par un modèle précisant pour chaque verbe quels sont les compléments autorisés.
L'étude très complète de GROSS [GRO] serait une bonne base linguistique de ce modèle.

III.3.2. - Les relatives

Dans le cas d'une relative, le substantif est gouverneur du verbe de la relative.
Par exemple, pour la phrase :

"le chien qui mange est jaune", on doit obtenir la structure suivante :



Etant donné que pour les propositions principales, le verbe est gouverneur du substantif, l'introduction de la relation inverse implique un très grand nombre d'essais improductifs. Pour résoudre ce problème, on modifie automatiquement la catégorie du verbe de la relative en VERS. De ce fait, on dispose de la relation SUBC*VERS := ... ; quant aux autres relations "VERS*CAT := ...", elles sont réécrites automatiquement à partir des relations existantes VERB*CAT := ... ;

III.3.3. - Les conjonctions de coordination

Une conjonction de coordination peut coordonner deux adjectifs, deux substantifs, deux verbes etc..., mais elle ne peut jamais coordonner deux unités lexicales de nature différente par exemple un adjectif et un substantif. Il faut donc ajouter ce contrôle au niveau de la construction des arbres si on veut éviter de coordonner deux unités lexicales de nature différente. L'initiative de ces contrôles est laissée à l'utilisateur (cf. B.3.I.1).

III.3.4. - Homographies, priorités

Pour traiter les homographies sur les catégories (par exemple "la" article défini ou pronom personnel), on ne considère plus, au niveau algorithmique, qu'en entrée, on dispose d'un vecteur de catégories mais d'un vecteur de vecteurs de catégories. Ceci a pour effet de compliquer l'algorithme d'analyse. Linguistiquement, on peut résoudre certaines homographies en créant par exemple une catégorie pronom-article puisque dans le cas des articles définis, ils sont également pronom personnel.

Afin d'accélérer l'analyse, on peut, par un système de priorités, réduire le nombre de gouverneurs de la phrase. Les gouverneurs de phrase sont les substantifs, les verbes (sauf ceux des relatives que nous avons traités à part) et les conjonctions de coordination. Nous avons établi les règles suivantes :

- S'il y a un verbe dans la phrase, celà signifie que les substantifs ne peuvent être gouverneur de cette phrase. D'où la suppression de tous les essais avec les substantifs dans ce cas qui est le plus fréquent (Etant donné qu'on a déjà traité les verbes des relatives).

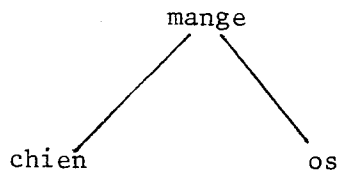
- Après avoir appliqué la règle précédente, si parmi les candidats gouverneurs possibles, il y a une conjonction de coordination qui gouverne deux candidats gouverneurs, on élimine ces deux candidats gouverneurs au profit de la conjonction de coordination. C'est le cas de deux phrases reliées par une conjonction de coordination.

Exemple

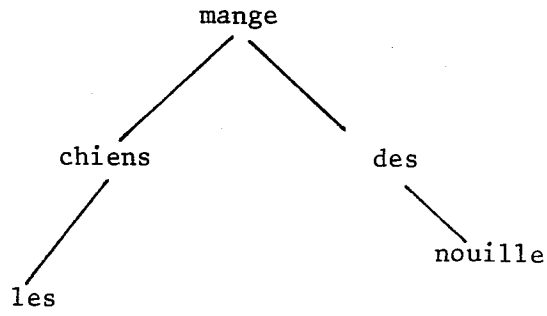
"Le chien et le chat"
"Le chien mange sa soupe et le chat boit du lait".

III.3.5 - Résultats obtenus

L'analyseur de dépendances permet de traiter des phrases correctes et incorrectes. En effet, dans le cas par exemple de la phrase "chien mange os" l'algorithme propose la structure suivante :

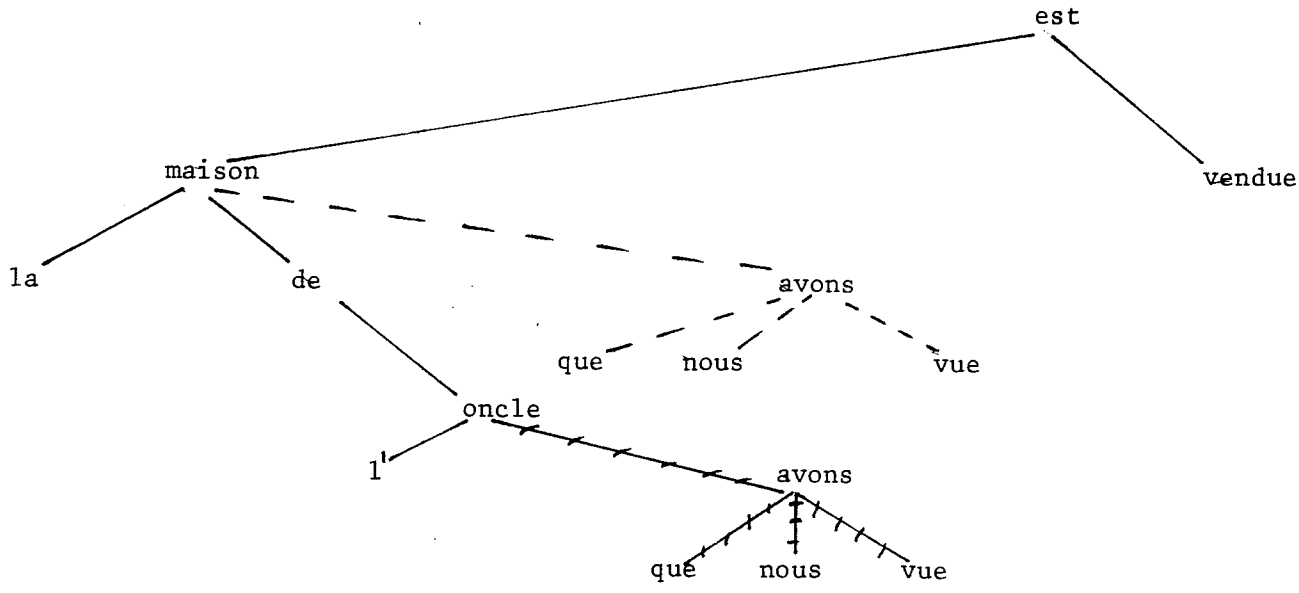


De même dans le cas de phrases incorrectes grammaticalement, on obtient toujours des solutions puisque l'algorithme ne se sert pas de variables grammaticales pour élaborer des structures. Par exemple, pour la phrase : "Les chiens mange des nouille" l'algorithme élabore la structure :



Quant aux phrases correctes, il peut aussi proposer des structures fausses telles que pour la phrase :

"La maison de l'oncle que nous avons vue est vendue.
On obtient deux structures :



"avons" est soit dépendant de "oncle" soit de "maison". Etant donné que le participe passé "vue" est au féminin, "avons" doit être dépendant uniquement de "maison". Pour lever cette indétermination, on a défini un deuxième modèle d'analyse qui a pour objet de valider ou d'infirmer une structure (ou sous-structure) de dépendances. Ce contrôle est effectué grâce à une grammaire binaire de type hors contexte qui utilise les variables grammaticales définies au niveau de la morphologie (pour vérifier les accords de variables) et des variables syntaxiques utilisées pour des contrôles

plus syntaxiques telles que : un substantif doit toujours être précédé d'un déterminant (article, pronom), un verbe ne peut avoir qu'un sujet, etc... L'étude de ce modèle est réalisée dans le chapitre suivant.

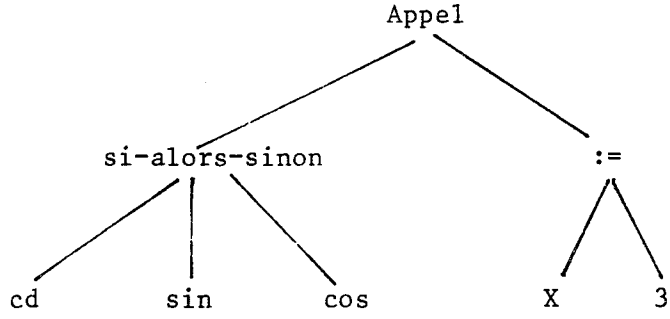
B2 - UTILISATION D'UNE GRAMMAIRE HORS CONTEXTE

I - UTILISATION D'UNE GRAMMAIRE HORS CONTEXTE

Comme nous l'avons évoqué précédemment, ce deuxième filtre permet d'effectuer des contrôles sur les structures (ou sous-structures) produites par le filtre précédent. La philosophie de la stratégie employée est très proche de celle utilisée par CUNIN-SIMONET-VOIRON [CSV1,CSV2] pour la compilation du langage algol 68. En fonction d'une grammaire LL(1) et d'un système d'attributs, les auteurs construisent un arbre abstrait. Celui-ci est ensuite décoré à l'aide des attributs qui peuvent être modifiés en fonction du contexte. Par exemple, soit l'instruction :

; si cd alors sin sinon cos fsi (X:=3) ;

l'arbre abstrait obtenu est le suivant :



La décoration consiste

1°) à préciser la nature des objets utilisés, par exemple que cd a un résultat de type booléen, que sin et cos sont des "réel" procédures dont le paramètre doit être un réel etc...

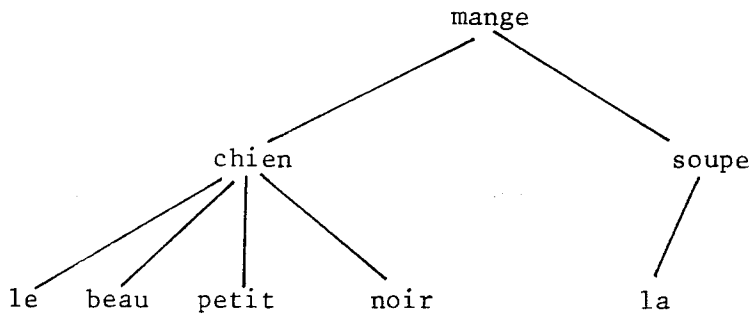
2°) En fonction de ces attributs, plusieurs parcours de l'arbre abstrait ainsi décoré permettent de vérifier la validité de l'instruction et d'effectuer certaines transformations sur les attributs. Par exemple, on "élargit" le 3 en un nombre réel.

La stratégie est donc la même : on produit une structure interprétable qui est ensuite vérifiée par un parcours de l'arborescence en tenant compte des informations liées à chaque noeud de la structure. Dans le cas de la langue naturelle, les informations sont les variables morphologiques et syntaxiques. Pour effectuer cette vérification, on effectue un parcours postfixé de la structure (ou de la sous-structure) en utilisant une grammaire "hors contexte" dont les règles sont définies par l'utilisateur. Cette stratégie peut se trouver complétée par l'apport sémantique impliquant des transformations plus profondes (cf. programme de Joloboff).

Exemple :

Soit la phrase : "Le beau petit chien noir mange la soupe".

La structure de dépendances proposée est la suivante :



En effectuant un parcours postfixé, on vérifie :

1°) le genre et le nombre del'article "le" et du substantif "chien". La réécriture consiste à indiquer, par une variable syntaxique, que "chien" a pris son article et qu'il est à la troisième personne.

2°) le genre et le nombre des adjectifs ("beau", "petit", "noir") et du substantif "chien".

3°) la personne et le nombre du substantif "chien" qui a pris son article et du verbe "mange". La réécriture consiste à indiquer par une variable syntaxique que le verbe a pris son sujet.

4°) le genre et le nombre de l'article "la" et du "substantif" soupe. La réécriture consiste comme précédemment à indiquer que "soupe" a pris son article.

5°) que le verbe a pris son sujet, qu'il n'a pas encore eu de complément d'objet, que le substantif "soupe" a pris son article. La réécriture consiste à indiquer que le verbe a un complément d'objet.

Nous avons donc défini un langage d'écriture de règles dont chacune d'elle est subdivisée en deux parties : la première, qui consiste en un critère d'applicabilité de la règle, est conditionnelle et porte sur les variables morphologiques et syntaxiques ; la seconde est une transduction d'informations sur le gouverneur. En outre, à chaque catégorie morphologique, est associée une catégorie syntaxique soit à l'aide de règles terminales, soit à l'aide de déclarations d'équivalences. Nous avons choisi la deuxième solution. Par exemple, aux catégories morphologiques ARTD (article défini), SUBC (substantif commun), VERB (verbe) sont associées respectivement les catégories syntaxiques ARTDE, GNOMO, VERBO. De plus, on doit également définir des variables syntaxiques telles que VBO qui est une variable de verbe et qui ne peut prendre que certaines valeurs comme SET et OBJ qui signifient que le verbe correspondant a pris son sujet (SET) ou qu'il a un complément d'objet (OBJ). Toutes ces informations sont définies par l'utilisateur. Dans le chapitre sur la réalisation pratique des deux filtres syntaxiques nous donnons en détail le langage d'écriture de la grammaire "hors contexte".

II - UTILISATION DES DEUX FILTRES SYNTAXIQUES

Suivant les applications, on peut soit utiliser l'analyseur de dépendances seul, soit utiliser les deux modèles syntaxiques. On peut aussi utiliser le module "hors contexte" seul (algorithme de COCKE).

Par exemple, dans le cas d'une application extra-linguistique où la langue a été réduite, l'analyseur de dépendances peut suffire à l'application envisagée. Ce peut être le cas d'un système questions-réponses.

Par contre, dans le cas d'une analyse de textes, on utilise les deux modules en parallèle : dès que le filtre de dépendances a construit une sous-structure, elle est validée ou infirmée par le contrôleur de variables. Les deux analyseurs étant complémentaires, cette utilisation en parallèle permet l'obtention plus rapide de structures que dans le cas d'une utilisation en séquentielle où le contrôleur de variables doit vérifier toutes les structures proposées par l'analyseur de dépendances. Dans le cas où aucune solution n'a été retenue par le contrôleur, on utilise à ce moment-là les deux filtres en séquentielles. En effet, dans la majorité des cas, le premier filtre construit une ou plusieurs structures. Le contrôleur peut alors localiser et détecter les erreurs sur les structures proposées. Le choix de la structure de contrôle est laissé à l'initiative de l'utilisateur grâce à un éditeur général qui sera décrit ultérieurement.

L'intérêt de définir plusieurs modèles complémentaires et incomplets est que

1°) la tâche de l'écrivain de paramètres linguistiques est simplifiée du fait que les modèles sont moins complexes et plus facilement maîtrisables.

2°) l'adjonction de nouveaux modèles (tels que le modèle défini par JOLOBOFF pour la sémantique) pouvant intervenir en parallèle permettrait d'obtenir des structures encore plus rapidement.

3°) L'utilisation séquentielle de ces modèles permet la détection et la localisation des erreurs.

4°) Pour une application donnée, on n'utilise que les modèles réellement nécessaires, ce qui permet de rendre le système plus général et adaptatif aux applications.

B3 - REALISATION PRATIQUE

I - LES REGLES DE DEPENDANCES

Afin que les paramètres linguistiques soient indépendants des programmes, on a prévu non seulement de définir les relations de dépendances mais également les priorités et les traitements particuliers tels que le verbe de la relative. Ces informations sont définies en dehors des règles de dépendances.

I.1 - DEFINITIONS DES TRAITEMENTS PARTICULIERS

Actuellement on dispose des cinq traitements suivants :

a) Déclarations des nouvelles catégories syntaxiques

Il s'agit essentiellement de déclarer la catégorie PHRA et la catégorie VERS qui a le même comportement que la catégorie VERB en tant que gouverneur. Cette déclaration est explicitée par :

```
CATEGORIES SYNTAXIQUES := PHRA, VERS(VERB) ;
```

b) Priorité du gouverneur de phrase VERB

Cette priorité est explicitée par :

```
GOUV := VERB ;
```

c) Traitement particulier de la conjonction de coordination

Ce traitement est explicité par :

```
ET := COCO ;
```

d) Définitions des catégories n'ayant qu'un seul dépendant possible

Le mot clé DEP1 permet cette définition en écrivant par exemple :

DEP1 := PHRA ;

e) Définitions des catégories qui ne sont jamais des feuilles

On donne ici la liste des catégories qui admettent toujours au moins un dépendant. Par exemple :

NTERM := PEPC, COCO ;

On peut étendre, si on le désire, les traitements particuliers, par exemple : définir les catégories admettant au plus deux dépendants, trois dépendants etc afin d'optimiser l'algorithme.

I.2 - COMPILATEUR DES REGLES DE DEPENDANCES

Le langage de définitions des règles de dépendances est extrêmement simple, puisque chaque règle est de la forme :

GOUVERNEUR * DEPENDANT := liste de poids ;

Exemple : SUBC*ADJQ := -15, -14, +7 ;

Remarque : La liste de poids peut très bien ne pas être triée en ordre croissant.

Un langage de commande permet la mise à jour de ces relations. Les commandes sont les suivantes :

¬ GOUVERNEUR*DEPENDANT	: suppression d'une règle
? GOUVERNEUR*DEPENDANT	: interrogation d'une règle
$	: liste des règles.

Pour modifier une règle de dépendances, on peut soit la redéfinir, soit modifier sa liste de poids, après avoir interrogé la règle, par la commande suivante :

/ancien poids/nouveau poids/

Le compilateur contrôle toutes les commandes et imprime les messages appropriés. Naturellement, vu sa simplicité, il est incrémentiel et peut être appelé en cours d'analyse d'un texte pour modifier les règles défaillantes.

II - LE LANGAGE DE LA GRAMMAIRE "HORS CONTEXTE"

Comme nous l'avons vu, il se subdivise en deux parties : la partie déclaration des variables et des catégories utilisées et la partie écriture des règles "hors contexte".

II.1 - ECRITURE DES DECLARATIONS

II.1.1. - Déclarations des variables et des catégories syntaxiques

Comme pour la morphologie, l'utilisateur doit définir les symboles qu'il utilise. Nous avons déjà vu qu'il doit définir les catégories syntaxiques, les valeurs de variables et les noms de variables syntaxiques. Par exemple : VBO est un nom de variables syntaxiques qui ne peut prendre que deux valeurs : SET et OBJ. Le formalisme d'écriture est le suivant :

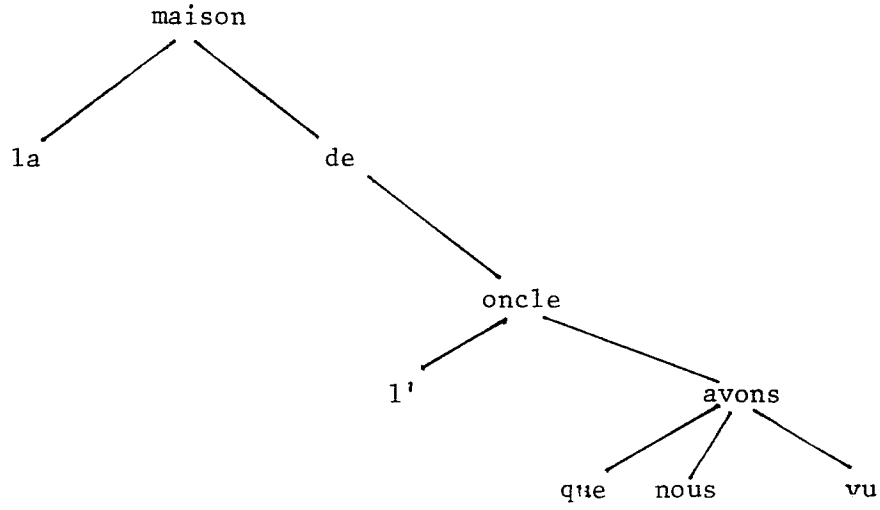
<DECLARATIONS SYNTAXIQUES>	::=<DECLARATIONS DES VARIABLES> <DECLARATIONS DES CATEGORIES>
<DECLARATIONS DES VARIABLES>	::=VARIABLES SYNTAXIQUES :=<DECLARATIONS>
<DECLARATIONS>	::=<DECLARATION SIMPLE> |<DECLARATIONSIMPLE><DECLARATIONS>
<DECLARATION SIMPLE>	::=<NOM DE VARIABLES SYNTAXIQUES> := <LISTE DE VALEUR DE VARIABLES>;
<LISTE DE VALEURS DE VARIABLES>	::=<VALEUR DE VARIABLES SYNTAXIQUES> |<VALEUR DE VARIABLES SYNTAXIQUES>, <LISTE DE VALEURS DE VARIABLES>
<NOM DE VARIABLES SYNTAXIQUES>	::=<TROIS CARACTERES>
<VALEURS DE VARIABLES SYNTAXIQUES>	::=<TROIS CARACTERES>
<TROIS CARACTERES>	::=<CARACTERE><CARACTERE><CARACTERE>
<CARACTERE>	::=<LETTRE>|<CHIFFRE>
<DECLARATIONS DES CATEGORIES>	::=CATEGORIES SYNTAXIQUES:=<LISTE DES> CATEGORIES SYNTAXIQUES>;
<LISTE DES CATEGORIES SYNTAXIQUES>	::=<SYMBOLE DE CATEGORIE> |<SYMBOLE DE CATEGORIE>,<LISTE DES CATEGORIES SYNTAXIQUES>
<SYMBOLE DE CATEGORIE>	::=<LETTRE><QUATRE CARACTERES>
<QUATRE CARACTERES>	::=<CARACTERE><TROIS CARACTERES>
<LETTRE	::=A|B|...|Z|
<CHIFFRE	::=0|1|...|9|

Exemple :

VARIABLES SYNTAXIQUES :=
GNO := ART, PEC ;
VBO := SET, OBJ ;
CATEGORIES SYNTAXIQUES := VERBO, INFIO,
PAPRT, PAPAS, ADJGO, GNOMO, GNOPR, PREPO ;

II.1.2. - Variables véhiculaires [VEILL1]

Elles sont indispensables dans le cas où la vérification de variables doit se faire entre deux syntagmes de niveau supérieur ou égal à deux. Par exemple, dans la structure



Il est nécessaire de vérifier que le genre et le nombre de oncle et de vu sont identiques. Pour ce faire, on transmet ("véhicule") le genre et le nombre du participe passé sur l'auxiliaire avoir à l'aide de la variable véhiculaire GNV que l'on pourra ensuite vérifier lors de l'utilisation d'une règle entre "oncle" et "avons".

Les variables véhiculaires sont déclarées comme les variables syntaxiques. Par exemple, on définit GNV comme étant égal à :

GNV := MAV, FEV, NEV ;

II.1.3. - Déclarations particulières

Dans la partie conditionnelle des règles, on est amené à vérifier l'ensemble des valeurs d'un nom de variable. Par exemple, pour contrôler l'accord en genre, il faut écrire un prédicat qui doit délivrer la valeur vrai s'il y a au moins une valeur en commun entre le syntagme de gauche et le syntagme de droite de la règle. Pour ce faire, on déclare au préalable ces prédicats de contrôle de variables à l'aide de symbole que l'on appelle "SYMBOLE VIV" où VIV signifie : "valeurs identiques de variables".

Exemple :

WGNRR := GNR(G).GNR(D)
WGNRV := GNR(G).GNV(D)

De même pour simplifier l'écriture des règles, il est possible de déclarer un symbole regroupant plusieurs noms de variables ayant le même comportement et subissant le même traitement syntaxique. On les appelle "symboles de macro-variables".

Exemple :

GNB := (GNR,NBR) genre et nombre
PTM := (PRS, TPS, MOD) personne, temps, mode

ce qui fait qu'à l'utilisation écrire PTM := PTM(D) ; évite d'écrire PRS := PRS(D) ; TPS := TPS(D) ; MOD := MOD(D) ;

La syntaxe de ces déclarations particulières est la suivante :

```
<DECLARATIONS PARTICULIERES>          ::=<DECLARATION VIV><DECLARATION MACROVAR>
<DECLARATION VIV>                     ::=VIV:<LISTE DE VIV>;
<LISTE DE VIV>                        ::=<DEFINITION VIV>
                                        |<DEFINITION VIV>,<LISTE DE VIV>
<DEFINITION VIV>                      ::=<SYMBOLE VIV>≒<PARTIE DROITE VIV>
<SYMBOLE VIV>                         ::=W<NOM DE VARIABLES>
                                          <DERNIERE LETTRE D'UN NOM DE VARIABLES>
<NOM DE VARIABLES>                    ::=<NOM DE VARIABLES MORPHOLOGIQUES>
                                        ¦<NOM DE VARIABLES SYNTAXIQUES>
<NOM DE VARIABLES MORPHOLOGIQUES>     ::=UN DES NOMS DE VARIABLES MORPHOLOGIQUES
                                          DECLARES
<NOM DE VARIABLES SYNTAXIQUES>        ::=UN DES NOMS DE VARIABLES SYNTAXIQUES
                                          DECLARES
<DERNIERE LETTRE D'UN NOM DE VARIABLES>::=LA DERNIERE LETTRE D'UN NOM DE
                                          VARIABLES DE MEME NATURE QUE LE PRECEDENT
<PARTIE DROITE VIV>                   ::=<NOM DE VARIABLES INDICE>.
                                          <NOM DE VARIABLES INDICE>
<NOM DE VARIABLES INDICE>             ::=<NOM DE VARIABLES>(<INDICE>)
<INDICE>                              ::=G|D
<DECLARATION MACROVAR>                ::=MACROVAR:<LISTE DE MACROVAR>;
<LISTE DE MACROVAR>                   ::=<DEFINITION MACROVAR>
                                        |<DEFINITION MACROVAR>,<LISTE DE
                                          MACROVAR>
<DEFINITION MACROVAR>                 ::=<NOM DE MACRO-VARIABLES>≒
                                          <PARTIE DROITE MACROVAR>
<NOM DE MACRO-VARIABLES>              ::=<TROIS LETTRES>
<PARTIE DROITE MACROVAR>              ::=(<LISTE DE NOMS DE VARIABLES>)
<LISTE DE NOMS DE VARIABLES>          ::=<NOM DE VARIABLES>
                                        |<NOM DE VARIABLES>,<LISTE DE NOMS DE
                                          VARIABLES>
<TROIS LETTRES>                       ::=<LETTRE><LETTRE><LETTRE>
<LETTRE>                              ::=A|B|...|Z|
```

II.2 - ECRITURE DES REGLES

II.2.1. - Exemple d'une règle

```
G001 : ARTDE*GNOMO ⇒ GNOMO
          SI   WGNRR & WNBRR ALORS
               GNB := GNB(G).GNB(D) ;
               GNO := ART ; PRS := TRE $
```

A partir de cet exemple, on constate qu'une règle est composée :

- d'une étiquette : "G001"
- d'une proposition syntaxique : "ARTDE*GNOMO ⇒ GNOMO" où le syntagme de gauche "ARTDE" concaténé au syntagme de droite "GNOMO" se réécrit GNOMO.
- d'une proposition conditionnelle : "SI WGNRR & WNBRR ALORS" qui est la condition d'applicabilité de la règle G001
- d'une partie affectation : "GNB := GNB(G).GNB(D) ;
  GNO := ART ; PRS := TRE"
  permettant de compléter le syntagme résultat GNOMO.
- d'un indicateur de fin de règle : le symbole"$".

II.2.2. - Syntaxe d'écriture

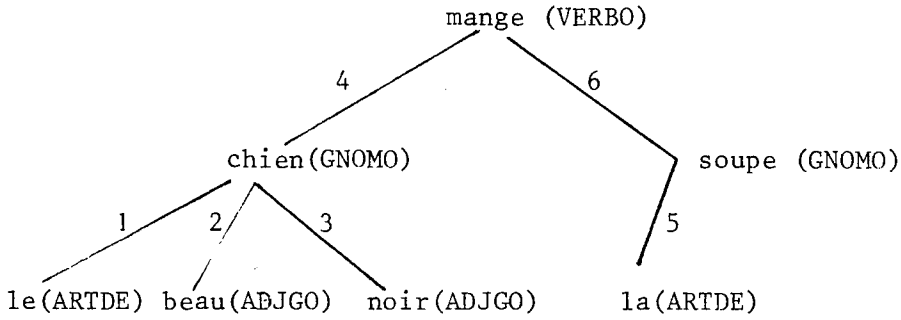
On donne ci-après, la syntaxe d'écriture des règles permettant d'infirmer ou de confirmer les structures de dépendances obtenues par l'algorithme précédent.

```
<REGLE SYNTAXIQUE>            ::=<ETIQUETTE><PROPOSITION SYNTAXIQUE>
                                 <PROPOSITION CONDITIONNELLE>
                                 <EXPRESSION D'AFFECTATION>
<ETIQUETTE>                   ::=<LETTRE><CHIFFRE><CHIFFRE><CHIFFRE>
<PROPOSITION SYNTAXIQUE>      ::=<SYMBOLE DE CATEGORIE><SYMBOLE DE CATEGORIE>=>
                                 <SYMBOLE DE CATEGORIE>
<PROPOSITION CONDITIONNELLE>  ::=SI<EXPRESSION CONDITIONNELLE>ALORS
                                |vide
<EXPRESSION CONDITIONNELLE>   ::=<EXPRESSION LOGIQUE>
                                |<EXPRESSION CONDITIONNELLE><OPERATEUR BINAIRE>
                                 <EXPRESSION LOGIQUE>
<EXPRESSION LOGIQUE>          ::=<EXPRESSION LOGIQUE SIMPLE>
                                |<EXPRESSION LOGIQUE><OPERATEUR BINAIRE><EXPRESSION
                                                    LOGIQUE SIMPLE>
                                |(<EXPRESSION CONDITIONNELLE>)
<EXPRESSION LOGIQUE SIMPLE>   ::=<CONDITION>(<INDICE>)|<SYMBOLE VIV>
                                |¬<SYMBOLE VIV>
<CONDITION>                   ::=<TERME>|<CONDITION><OPERATEUR BINAIRE><TERME>
                                |(<EXPRESSION LOGIQUE>)
<TERME>                       ::=<VALEUR DE VARIABLES>|¬<VALEUR DE VARIABLES>
                                |(<CONDITIONS>)
<OPERATEUR BINAIRE>           ::= & ou |
<INDICE>                      ::= G|D
<SYMBOLE DE CATEGORIE>        ::=UN SYMBOLE DE CATEGORIE SYNTAXIQUE DECLARE
<VALEUR DE VARIABLES>         ::=<VALEUR DE VARIABLES SYNTAXIQUES>
                                |<VALEUR DE VARIABLES MORPHOLOGIQUES>
<VALEUR DE VARIABLES SYNTAXIQUES>::=UNE DES VALEURS DE VARIABLES SYNTAXIQUES
                                    DECLAREES
<VALEUR DE VARIABLES MORPHOLOGIQUES>::=UNE DES VALEURS DE VARIABLES MORPHO-
                                    LOGIQUES DECLAREES
```

```
<EXPRESSION D'AFFECTATION>     ::=<AFFECTATION SIMPLE>
                                 |<AFFECTATION SIMPLE>;<EXPRESSION D'AFFECTATION>
<AFFECTATION SIMPLE>           ::=<PARTIE GAUCHE>:=<EXPRESSION>
<PARTIE GAUCHE>                ::=<NOM DE VARIABLES>|<NOM DE MACRO-VARIABLES>
<EXPRESSION>                   ::=<EXPRESSION INDICEE>|<EXPRESSION SIMPLE>
                                 |<EXPRESSION INDICEE><OPERATEUR LOGIQUE>
                                  <EXPRESSION SIMPLE>
<EXPRESSION INDICEE>           ::=<NOM DE VARIABLES INDICEES>
                                 |<NOM DE VARIABLES INDICEES><OPERATEUR LOGIQUE>
                                  <NOM DE VARIABLES INDICEES>
<NOM DE VARIABLES INDICEES>    ::=<NOM DE VARIABLES>(<INDICE>)
<EXPRESSION SIMPLE>            ::=<VALEUR DE VARIABLES>
                                 |<EXPRESSION SIMPLE><OPERATEUR LOGIQUE>
                                  <VALEUR DE VARIABLES>
<NOM DE VARIABLES>             ::=<NOM DE VARIABLES MORPHOLOGIQUES>
                                 |<NOM DE VARIABLES SYNTAXIQUES>
<NOM DE VARIABLES MORPHOLOGIQUES>::=UN DES NOMS DE VARIABLES MORPHOLOGIQUES
                                    DECLARES
<NOM DE VARIABLES SYNTAXIQUES>::=UN DES NOMS DE VARIABLES SYNTAXIQUES DECLARES
<SYMBOLE VIV >                 ::=UN DES SYMBOLES DE VIV DECLARES
<NOM DE MACRO-VARIABLES>       ::=UN DES NOMS DE MACRO-VARIABLES DECLARES
<LETTRE>                       := A|B|...|Z|
<CHIFFRE>                      ::= 0|1|...|9|
<OPERATEUR LOGIQUE>            ::= .|+|-
```

II.2.3. - Exemple d'utilisation

Soit la structure de dépendances :



Les syntagmes :

le	: ARTDE	MAS	SIN	MANGE	: VERBO	IND	PRE	UND	TRE	SIN
beau	: ADJGO	MAS	SIN							
noir	: ADJGO	MAS	SIN	SOUPE	: GNOMO	FEM	SIN			
chien	: GNOMO	MAS	SIN	la	: ARTDE	FEM	SIN			

Les règles :

GOO1 : ARTDE*GNOMO => GNOMO
SI WGNRR & WNBRR ALORS
GNB := GNB(G).$_{GNB}$(D) ; GNO := ART ; PRS := TRE $

GOO6 : ADJGO*GNOMO => GNOMO
SI WGNRR & WNBRR ALORS
GNB := GNB(G).GNB(D) ; GNO := GNO(D) ; PRS := TRE $

GOO7 : GNOMO*ADJGO => GNOMO
SI WGNRR & WNBRR ALORS
GNB := GNB (G).GNB(D) ; GNO := GNO(G) ; PRS := TRE $

V001 : GNOMO* VERBO => VERBO
SI ¬ SET(D) & ¬ IMP(D) & ART(G) & WNBRR & WPRSS
ALORS
TM := TM(D) ; NBR:=NBR(G).NBR(D) ;
PRS:=PRS(D).PRS(G) ; VBO:=SET $

V002 : VERBO*GNOMO => VERBO
SI (SET|IMP)(G) & ART(D) & ¬ OBJ(G)
ALORS
PTM:=PTM(G) ; NBR:=NBR(G) ;
VBO:=VBO(G)+OBJ $

On applique :

1°) La règle G001
car la condition est vérifiée (accord en genre et en nombre)
Les variables de GNOMO sont égales à :
GNR := MAS ; NBR := SIN ; GNO := ART ; PRS := TRE

2°) La règle G006
car la condition est vérifiée. Les variables de GNOMO sont égales à :
GNR := MAS ; NBR := SIN ; GNO := ART ; PRS := TRE

3°) La règle G007 qui donne les mêmes résultats

4°) La règle V001
car la condition est vérifiée. Les variables de VERBO sont égales à :
PRS := TRE ; TPS := PRE ; MOD := IND ; NBR := SIN ; VBO := SET.

5°) La règle G001
car la condition est vérifiée. Les variables de GNOMO sont égales à :
GNR := FEM ; NBR := SIN ; GNO := ART ; PRS := TRE

6°) La règle VOO2

car la condition est vérifiée. Les variables du VERBO sont égales à :

PRS := TRE ; TPS := PRE ; MOD := IND ; NBR := SIN ; VBO := SET+OBJ.

C

LE SYSTEME P.I.A.F.

LES APPLICATIONS

Comme nous l'avons mentionné dans l'introduction, le système P.I.A.F. (Programme Interactif d'Analyse du Français) est intéractif afin de permettre un dialogue entre l'utilisateur et la machine. Il est composé de plusieurs modules à savoir : le transducteur général d'états finis, les filtres de dépendances et "hors contexte". De plus, deux éditeurs : un éditeur lexicographique pour l'utilisation du transducteur général d'états finis et un éditeur général pour l'utilisation du système complet, autorisent une intéraction complète tant au niveau de la définition des paramètres linguistiques qu'au niveau de l'exécution des différents programmes.

L'articulation entre les différents composants du système est illustrée par le schéma ci-après. (cf. page suivante).

I - LES EDITEURS

Ils permettent :

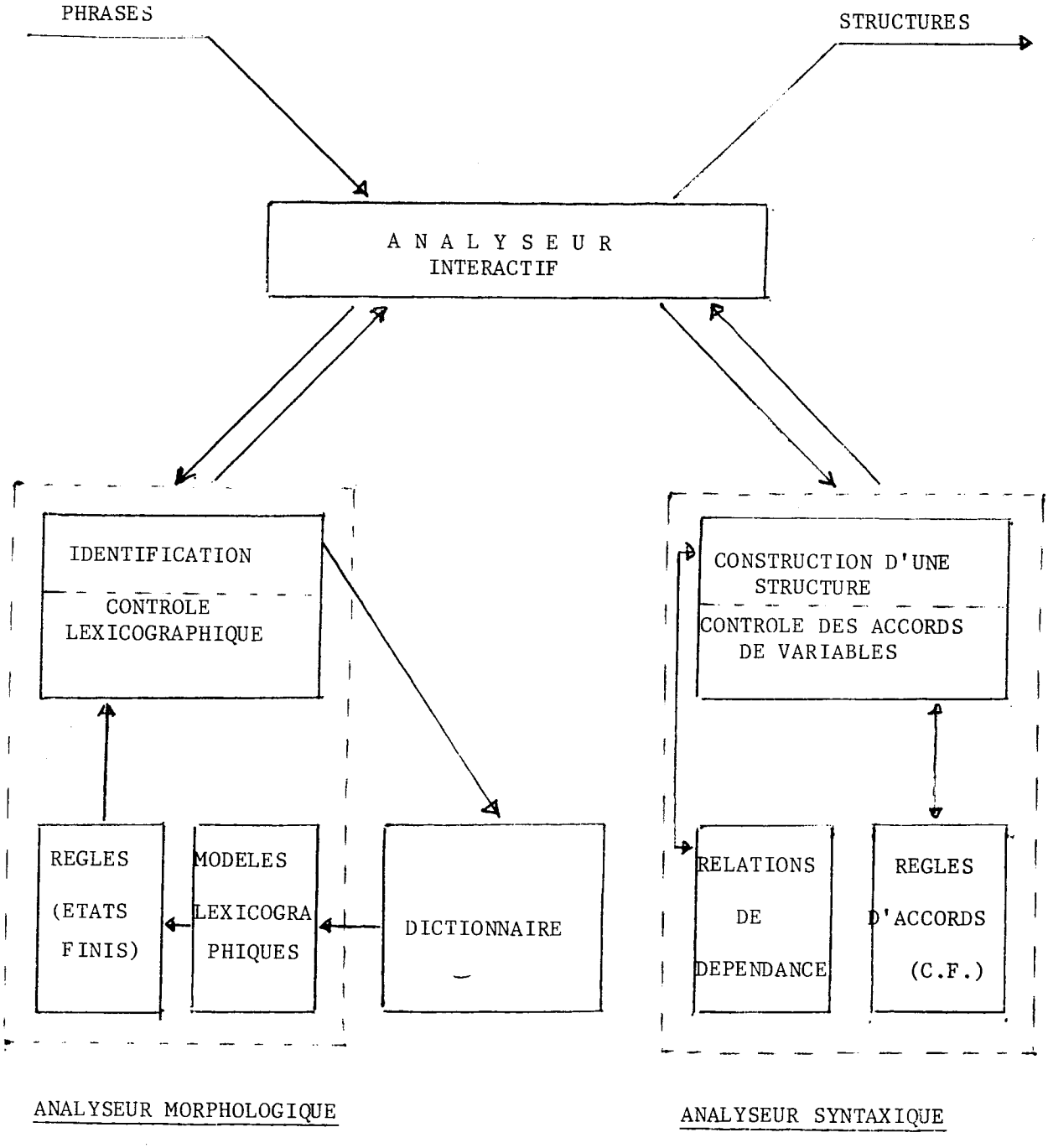
- La création en mode conversationnel des paramètres linguistiques : règles de grammaires morphologiques, dictionnaire, liste de paradigmes (modèles), relations de dépendances, grammaire "hors contexte".

- L'exécution intéractive des différents modules d'analyse : c'est-à-dire qu'en cours d'exécution, on peut remettre en cause les paramètres linguistiques et ainsi les modifier.

- La détection et la correction manuelle des erreurs.

- L'utilisation en parallèle ou en série des deux filtres syntaxiques, ce qui permettra la localisation des erreurs syntaxiques.

SCHEMA FONCTIONNEL DU SYSTEME P.I.A.F.



- La correction automatique de certaines classes d'erreurs liée à la généralité et à la réversibilité des modèles proposés.

Etant parfaitement conscientsdes difficultés posées par l'analyse automatique des langues, nous proposons un traitement automatique assisté du français et d'autres langues naturelles telles que l'anglais ou l'italien.

I.1 - L'éditeur lexicographique

Comme nous l'avons déjà indiqué, il permet l'utilisation en mode conversationnel du transducteur général d'états finis. On trouve dans [COUGD] le détail de toutes les commandes accessibles à ce jour. Il permet :

- L'analyse d'un texte en pas à pas ou en continu (STOP, NSTOP).

- D'obtenir des résultats détaillés ou non :les résultats détaillés permettent de contrôler la grammaire du transducteur (BAVARD, NBAVARD, TRACE).

- l'Accès aux paramètres linguistiques en cours ou non d'exécution (GRAM, DICT).

- La correction du texte en cours d'exécution (ORTHO)

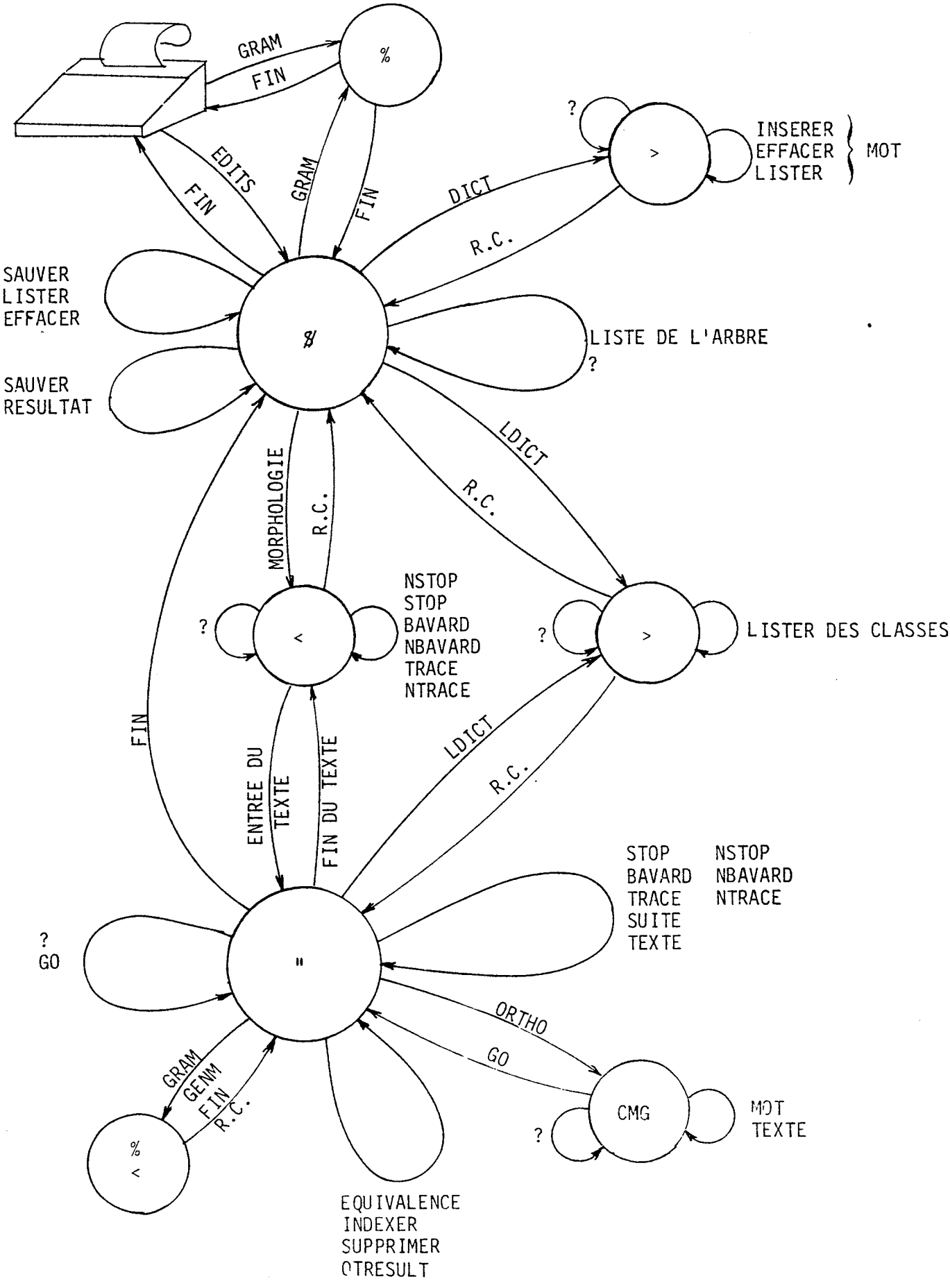
- La mise à jour du dictionnaire soit directement (DICT) soit par l'intermédiaire d'un système d'équivalence permettant l'obtention du résultat désiré.

- L'impression au terminal ou à l'imprimante des paramètres linguistiques et du texte d'entrée.

- L'accès à l'éditeur particulier de la grammaire et des modèles (GRAM).

On trouve ci-après le diagramme de cet éditeur.

DIAGRAMME DE L'EDITEUR LEXICOGRAPHIQUE



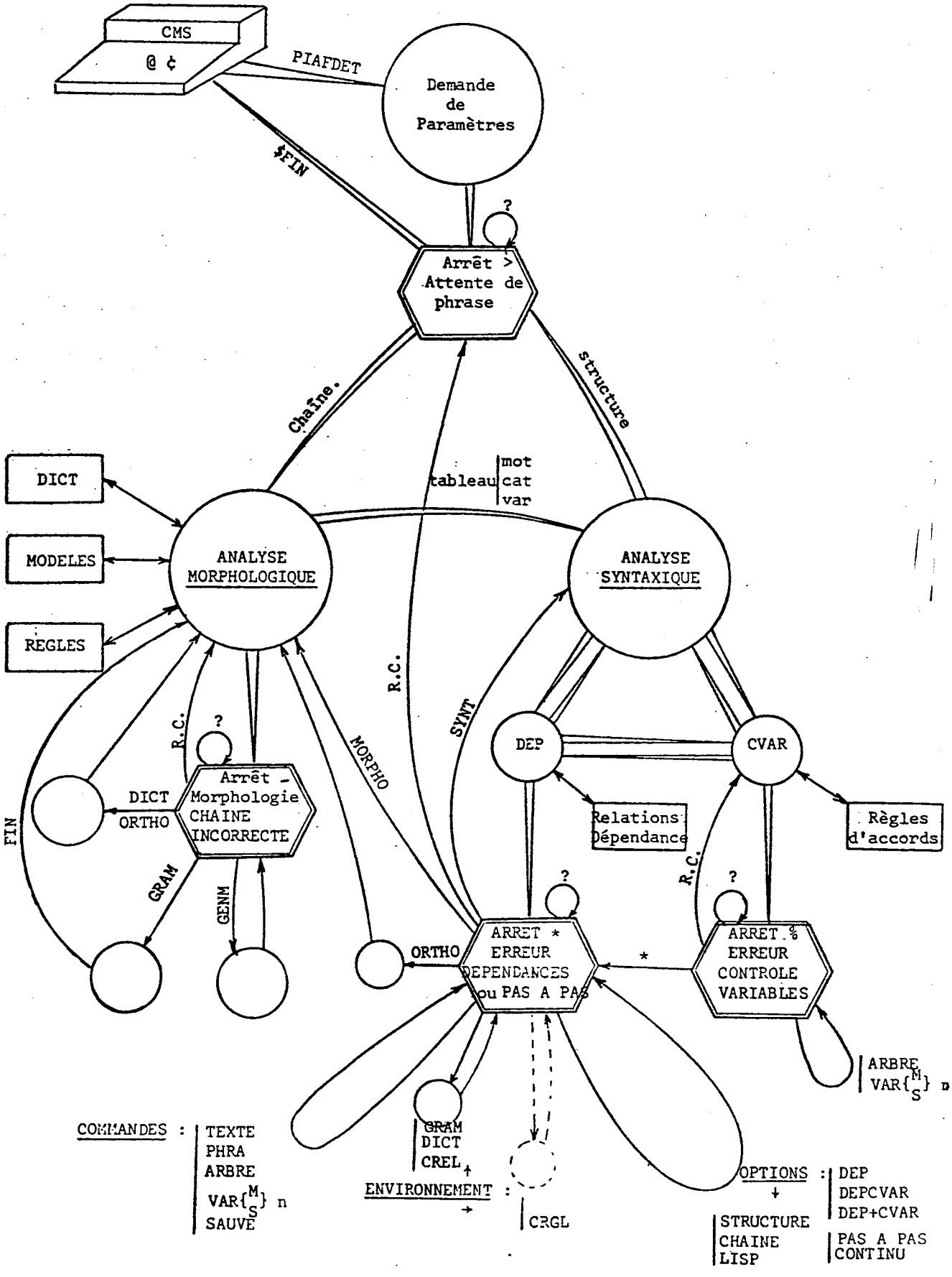
I.2 - L'EDITEUR GENERAL

Ses fonctions principales sont les suivantes :

- Choisir le type d'analyse désirée :
  - analyse de dépendances seule (DEP)
  - analyse de dépendances et hors contexte en série ou en parallèle (DEP+CVAR ou DEPCVAR).
- Avoir accés aux résultats du transducteur général d'états finis (VARM n)
- Mettre à jour les paramètres linguistiques : grammaires et modèles morphologiques (GRAM), relations de dépendances (CREL), dictionnaire (DICT).
- Interroger les variables syntaxiques (VARS n)
- Corriger le texte d'entrée (ORTHO)
- Choisir le type de résultats désirés :
  - structures arborescentes (STRUCTURE)
  - structures parenthésées (LISP)
  - accepteur (réponse par oui-non) (CHAINE)
- Utiliser le générateur morphologique (GENM).

On trouve ci-après le diagramme de cet éditeur.

DIAGRAMME DE L'EDITEUR GENERAL



II - LES APPLICATIONS

Le système P.I.A.F. étant entièrement paramétré, interactif et utilisant les redondances de la langue, la modification des paramètres et l'agencement des différents modules permettent d'obtenir de nombreuses applications.

II.1 - APPLICATIONS DU TRANSDUCTEUR GENERAL D'ETATS FINIS

II.1.1. - Détection automatique des erreurs lexicales

Les erreurs lexicales sont détectées automatiquement par l'analyseur morphologique. On peut ensuite, grâce à l'éditeur, les corriger "on line". On peut même dans certains cas les corriger automatiquement en utilisant soit une transcription phonétique soit la génération morphologique.

II.1.2. - Corrections automatiques des erreurs lexicales

Dans le cas des erreurs de type phonétique (par exemple : chevo, ocurence, genoux) et dans celui des erreurs provenant de la mauvaise utilisation d'un système de désinences (par exemple : allerons, chevals, genous etc...), on peut proposer des éléments de correction en utilisant dans le premier cas une transcription phonétique et dans le deuxième cas le générateur morphologique.

II.1.2.1. - Le traducteur graphique-phonétique

En changeant les paramètres linguistiques du transducteur général d'états finis, on peut obtenir, à partir d'une chaîne graphique quelconque, toutes les transcriptions phonétiques possibles. Pour ce faire, on utilise le code dérivation "CD" qui contient en fin d'analyse une transcription phonétique possible.

Exemple :

soient :

Les règles

RIB : CD:=CD(D) ; VAL:=( ) ; SAT:=( ).
SUI : CD:=CD(G) || CD(D) ; VAL:=( ) ; SAT:=( ).
FIN : CD:=CD(G) || CD(D) ; VAL:=( ) ; SAT:=( ) ; FIM.

Les modèles

/CH/ : REG:=(RIB) ; CD:=K ; VAL:=(SUI) ; SAT:=( ).
/CH1/ : REG:=(RIB) ; CD:=CH ; VAL:=(SUI) ; SAT:=( ).
/E/ : REG:=(SUI) ; CD:=E ; VAL:=(SUI,FIN) ; SAT:=( ).
/E1/ : REG:=(SUI) ; CD:=E1 ; VAL:=(SUI) ; SAT:=( ).
/V/ : REG:=(SUI) ; CD:=V ; VAL:=(SUI) ; SAT:=( ).
/A/ : REG:=(SUI) ; CD:=A ; VAL:=(SUI) ; SAT:=( ).
/L/ : REG:=(SUI) ; CD:=L ; VAL:=(SUI,FIN) ; SAT:=( ).
/E␣/ : REG:=(FIN) ; VAL:=( ) ; SAT:=( ).
/␣/ : REG:=(FIN) ; VAL:=( ) ; SAT:=( ).

NB : E1 signifie les sons é, è, ê.

VALOO := (RIB)

Le dictionnaire

/A/A/	/E␣/E␣/	/L/L/
/CH/CH/	/␣/␣/	/V/V/
/CH/CH1/	/E/E/	/E/E1/

L'analyse de "CHEVALE␣" permet d'obtenir les transcriptions suivantes :

CH	E	V	A	L	E
CH	E	V	A	L	
CH	E1	V	A	L	E
CH	E1	V	A	L	
K	E	V	A	L	E
K	E	V	A	L	
K	E1	V	A	L	E
K	E1	V	A	L	

II.1.2.2. - Le_correcteur_graphique_phonétique

En utilisant le traducteur précédent sur le dictionnaire graphique, on obtient un dictionnaire graphique phonétique contenant :

- les éléments graphiques avec leurs traductions phonétiques

- les traductions phonétiques.

Si on a pris soin de chaîner circulairement entre eux tous les éléments graphiques possédant la même traduction phonétique, on peut, à partir d'une chaîne phonétique, l'analyser par le transducteur général d'états finis et n'engendrer que des chaînes graphiques correctes car elles sont contrôlées par la morphologie.

Par exemple : à partir de "CHEVALE␣",on engendre les huit transcriptions précédentes qui, elles-mêmes, permettent l'obtention de "CHEVAL␣".

De même à partir de "BAU␣", on produit "BO" qui permet d'obtenir "BEAU␣" et "BEAUX ".

Ensuite, pour pouvoir choisir entre plusieurs solutions proposées telles que "Beau␣" ou "Beaux␣", on utilise les filtres syntaxiques qui permettent la sélection des bonnes solutions.

II.1.2.3. - Correction et génération morphologique

Dans le cas où la racine est correcte, on peut utiliser également le générateur morphologique. Ces erreurs proviennent du mauvais usage des irrégularités de la langue. Par exemple :

"CHEVALS" à la place de "CHEVAUX"
"ALLERONS" à la place de "IRONS".

L'algorithme est le suivant :

Ayant détecté une erreur sur un terme dont la racine est correcte, on analyse la chaîne en supposant que ce qui suit la base peut être analysé à partir de l'état initial de l'automate. Ceci permet d'obtenir des variables grammaticales. Quant à la catégorie, elle est fournie par la racine.

Exemples

-"CHEVALS␣" est une chaîne incorrecte, mais la racine "CHEV" donne la catégorie substantif masculin.
Ensuite l'analyse de la désinence "ALS " à partir de l'état initial permet l'obtention de la marque du pluriel.

-"ALLERONS␣" est une chaîne incorrecte dont la base "ALL" permet d'induire la catégorie verbe.
Ensuite l'analyse de "ERONS␣" fournit les renseignements suivants : première personne du pluriel du futur de l'indicatif.

Pour proposer des éléments de correction, on fait exécuter le générateur morphologique à partir de la racine et des renseignements linguistiques déterminés par l'analyse précédente.

Pour les exemples précédents on exécute la génération à partir de la base "CHEV" avec les renseignements linguistiques "substantif masculin pluriel", ce qui permet d'obtenir la chaîne "CHEVAUX␣". De même, à partir de la base "ALL" et des variables première personne du pluriel du futur de l'indicatif, on obtient la chaîne "IRONS␣".

Naturellement, il faut que les paramètres linguistiques soient établis en fonction de cette correction. Par exemple pour "oeil" et "yeux", il ne faut pas les considérer comme des formes mais comme des racines auxquelles on ajoute la désinence "␣". De même pour "genoux", il faut considérer que l'on a la racine "genou" à laquelle on ajoute la désinence "x␣".

II.1.3 - Traducteur graphie-braille

Basé sur le même principe que le traducteur graphie-phonétique, le transducteur général d'états finis permet également la traduction d'un texte, écrit en français, en langage braille abrégé. Actuellement, nous étudions la possibilité d'un système intégré graphie-braille sur mini-ordinateur.

II.1.4. - Aide à la traduction d'un langage de programmation vers un autre langage de la même famille.

Le problème qui s'est posé était de transporter sur IRIS/80 le système P.I.A.F. écrit en PL /360. Il fallait donc, entre autre, traduire tous les programmes écrits en PL/360 en LP/70 ou en LP/80. Claudine CHASSAGNE s'est penchée sur la traduction PL/360 -LP/80. Pour ce faire, elle a utilisé le transducteur général d'états finis. Les premiers résultats sont encourageants. Il faut attendre qu'elle ait terminé ce travail pour pouvoir étudier les différences de performance entre un algorithme programmé directement en LP/80 et le même algorithme programmé en PL/360 et traduit à l'aide de P.I.A.F. Naturellement, il n'est pas question de proposer un traducteur automatique mais une aide à la traduction, ce qui permettra d'alléger, dans un premier temps, le travail de transport sur une autre machine.

II.1.5. - Détection des variations orthographiques sur les noms propres

Le problème consiste à rapprocher des fichiers relatifs à un état civil ancien (actes de naissances, actes de mariages, actes de décés, etc...), en vue de reconstituer d'une part l'historique de chaque individu, d'autre part le graphe généalogique complet de la population.

Il est à noter que ces fichiers contiennent des informations floues : c'est-à-dire qu'il y a de nombreuses omissions et des erreurs de codification. Il s'ensuit que :

- soit on est incapable d'effectuer automatiquement le rapprochement : erreurs sur les noms patronymiques par exemple,
- soit on aboutit à des ambiguités qui conduisent à des solutions multiples : dans le cas des omissions par exemple.

Pour résoudre ce problème, Y. CHIARAMELLA [CHIA] a été amené à définir plusieurs modèles permettant l'extraction d'invariants à partir des formes initiales. L'un de ces modèles est constitué par l'extraction d'invariants phonétiques où il a utilisé le transducteur général d'états finis.

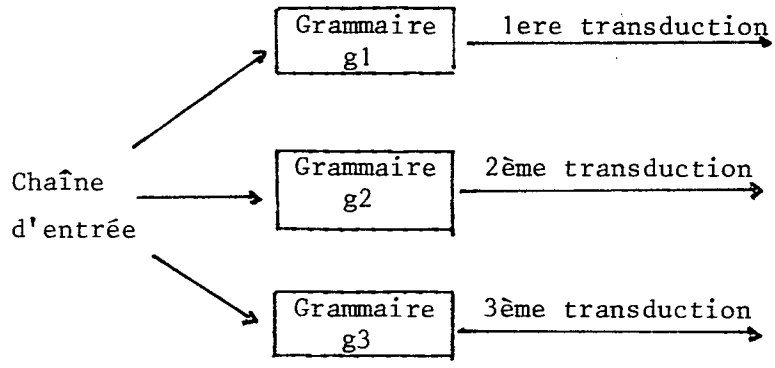
Extraction d'invariants phonétiques

Il a défini trois niveaux de transduction correspondant à une interprétation phonétique de moins en moins stricte de la forme orthographique originale. Le premier niveau correspond à la transcription phonétique habituelle. Le second tient compte de l'accent du terroir : on confond, par exemple, les sons "in" et "an". Quant au troisième, on ne garde que les consonnes.

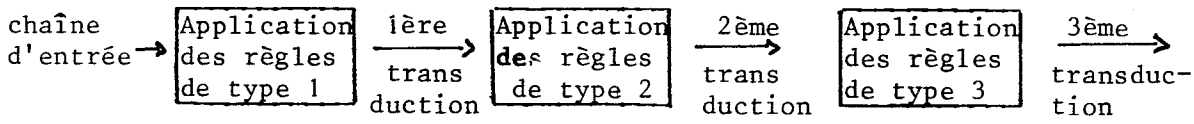
Pour réaliser ces trois niveaux, Y. CHIARAMELLA avait le choix entre deux stratégies : la première qui consiste à définir trois grammaires indépendantes, la seconde qui consiste à définir une grammaire en cascade : c'est-à-dire définir trois types de règles. Le premier type est utilisé pour le premier niveau qui permet l'obtention d'une

chaîne qui sert d'entrée pour le second niveau où on utilise le deuxième type de règles permettant l'obtention d'une chaîne servant elle même d'entrée pour le troisième type de règles.

La première stratégie correspond au schéma suivant :



Quant à la deuxième stratégie, elle correspond au schéma suivant :



CHIARAMELLA a choisi la deuxième stratégie, car elle conduit à des résultats cohérents ce que n'assure pas la première stratégie.

Pour ce faire, on procède de la façon suivante :

Appelons A1, A2,... AN les règles de type 1

B1, B2,... BM les règles de type 2

C1, C2,... CK les règles de type 3

Les validations des règles de type I (I = 1,2,3) ne portent respectivement que sur les règles de même type. Par contre, au niveau des validations initiales, elles sont égales à l'union des validations initiales des règles des trois types.

Du point de vue informatique, il a fallu modifier légèrement le transducteur de telle sorte que lorsque l'on change de type de règles, on prenne comme entrée le résultat de la sortie obtenue avec les règles précédentes.

Cette application est opérationnelle et utilisée dans le cadre de recherches en démographie historique (système Mercure).

II.1.6. - Application aux problèmes de documentation automatique

Nous travaillons actuellement en liaison étroite avec la Documentation Française qui constitue une base de données documentaires dans le domaine de la vie politique française. Les informations sont constituées par des textes brefs relatant les évènements politiques dans leur ordre chronologique. La Documentation Française utilise actuellement le système "Mistral" [MIST]. L'utilisation de ce système pose les problèmes suivants :

- Chaque forme linguistique dérivée d'une même racine est un mot clé. De ce fait, l'interrogation doit soit faire appel à différentes formes de ce mot clé, soit utiliser un opérateur de troncature trop imprécis.

- Il est nécessaire de procéder à une préédition manuelle du texte pour la prise en charge de groupes de mots tels que "Assemblée Nationale" ou pour permettre l'indexation automatique de certains groupes de mots.

- La recherche sur chaîne de caractères oblige à connaître la citation exacte recherchée sans pouvoir tenir compte des variations linguistiques.

- Toute chaîne de caractères comprise entre deux blancs est un mot clé.

Le transducteur permet :

- de constituer un dictionnaire des racines de mots clés afin de réduire l'ensemble des formes de chaque mot clé à une seule forme canonique.

- de réaliser des relations de synonymie en chaînant entre eux les mots clés équivalents.

- d'indexer des mots composés ou de suites de mots.

- de traiter les requêtes en utilisant des formes linguistiques libres.

Pour les besoins de la Documentation Française, il a été nécessaire d'adapter certains modules de P.I.A.F. Par exemple, au niveau du dictionnaire, nous avons défini plusieurs indications pour :

- distinguer les mots vides des mots clés

- choisir le synonyme préférentiel

- décomposer une suite de mots en ces différents constituants

- reconnaître un élément appartenant à une suite de mots.

II.2 - APPLICATIONS DU SYSTEME COMPLET

II.2.1. - Analyse de texte-validations de données linguistiques

On utilise les deux modules d'analyse syntaxique en parallèle ce qui permet l'obtention le plus rapidement possible de structures arborescentes. On peut également utiliser le système pour contrôler la validité d'un texte sans production de structures (dès qu'une structure a été trouvée, on passe à l'analyse de la phrase suivante).

II.2.2. - Détection automatique des erreurs et correction

Les erreurs syntaxiques sont détectées par l'utilisation en série des deux filtres syntaxiques : si aucune solution n'a été trouvée par l'utilisation des deux modules d'analyse en parallèle, l'utilisation en série est automatiquement effectuée. On peut donc localiser les erreurs. On peut ensuite :

- les corriger manuellement en utilisant l'éditeur général
- ou demander des propositions de corrections au système qui utilise.:
  - . soit le transducteur phonétique et l'analyseur morphologique qui proposent des chaînes ayant le même invariant phonétique si l'erreur est de type orthographique.
  - . soit le générateur morphologique qui propose des chaînes dérivées d'une même base ou de bases chaînées entre elles (par exemple verbe aller). Ces propositions sont ensuite filtrées par le système d'analyse syntaxique.

II.3 - AUTRES APPLICATIONS

Le système P.I.A.F. est également utilisé par des chercheurs extérieurs au laboratoire. Parmi toutes les applications, on peut citer :

- La construction d'une grammaire de l'anglais de la classe de 6ème effectuée par un professeur du second degré (Mme Caillol).
- L'analyse du contenu effectuée par Bouché de L'Université de Lyon ;
- L'aide au psychodiagnostic étudiée en collaboration entre l'INSA de Lyon (M.Lavorel) et la Faculté de Médecine de Lyon (M. Cosnier),
- Le regroupement d'index documentaire réalisé par le centre E.D.F. de Clamart.

- La communication avec un robot étudiée par Somalvico de Milan .

- L'utilisation du transducteur général d'états finis par Rouault pour ses études sur la langue.

III EXEMPLES DE RESULTATS

III.1 Analyse morphologique

```
EDITS

$morphologie texte stop nbavard

<les poules du couvent couvent .

LES                    1   CL3    ART   MAS FEM PLU
LES                    2   CL4    POPL  MAS FEM PLU TRE ACC

"go

POULES                 1   E8     SUBC  FEM PLU
"go

DU                     1   CL14   ARTC  MAS SIN PAT .

"go

COUVENT                1   E1     SUBC  MAS SIN

COUVENT                2   PR4    VERB  PLU TRE PRE IND

COUVENT                3   SU1    VERB  PLU TRE PRE SUB

"go

COUVENT                1   E1     SUBC  MAS SIN

COUVENT                2   PR4    VERB  PLU TRE PRE IND
COUVENT                3   SU1    VERB  PLU TRE PRE SUB

"go
.                      1   FORM   PNT

"supprimer /couvent/

"supprimer /couv/
"go
```

```
<les poules du couvent couvent .

LES                     1   CL3   ART   MAS FEM PLU
LES                     2   CL4   POPL  MAS FEM PLU TRE ACC

".go

POULES                  1   E8    SUBC  FEM PLU

"go

DU                      1   CL14  ARTC  MAS SIN PAT

"go

DECOUPAGE PARTIEL       COUVENT COUVENT

"texte

LES POULES DU COUVENT COUVENT .
              |

"/couvent/arbre/

COUVENT                 1   E1    SUBC  MAS SIN

****1   COUVENT COMME ARBRE   ****

"indexer 1

COUVENT                 1   E1    SUBC  MAS SIN

"go

COUVENT                 1   E1    SUBC  MAS SIN

"/couvent/coulent/

COUVENT                 1   PR4   VERB  PLU TRE PRE IND

****1   COUV COMME COUL   ****

COUVENT                 2   SU1   VERB  PLU TRE PRE SUB

****1   COUV COMME COUL   ****

"indexer 1

COUVENT                 1   E1    SUBC  MAS SIN
COUVENT                 2   PR4   VERB  PLU TRE PRE IND
COUVENT                 3   SU1   VERB  PLU TRE PRE SUB

"otresult 1

COUVENT                 1   PR4   VERB  PLU TRE PRE IND
COUVENT                 2   SU1   VERB  PLU TRE PRE SUB
```

```
"sui

<les poulent .

LES                    1   CL3    ART   MAS FEM PLU
LES                    2   CL4    POPL  MAS FEM PLU TRE ACC

"go

DECOUPAGE PARTIEL        POULENT .

"bavard

POULE                  0   RIB    SUBC  FEM

P                      0   RIB    VERB

"ortho

CMG poules

CMG go

POULE                  0   RIB    SUBC  FEM

     S                 1   E8     SUBC  FEM PLU

" nbavard

POULES                 1   E8     SUBC  FEM PLU

" go

.                      1   FORM   PNT
```

III.2 Analyse syntaxique

```
 piafdet

***** LE SYSTEME P.I.A.F. ( DETECT ) EST A¯ VOTRE SERVICE *****
* EQUIPE ALGORITH ET INTELLIG ARTIFICI *
- LABO INFORMAT U.S.M.G. B.P.53 - 38041 GRENOBLE CEDEX -
OLD FILE NOT FOUND
EXECUTION BEGINS...
DONNEES ? (TERM OU NOM DE FICHIER)
>
 terminal
RESULTATS ? ( TERM OU IMPR OU NOM DE FICHIER )
>
 terminal
FORME DU RESULTAT ? (STRUCTURE OU CHAINE OU LISP)
>
 structure
MODE ? ( PAS A PAS OU CONTINU )
>
 pas a pas
ALGORITHMES ? ( DEP OU DEPCVAR OU DEP+CVAR )
>
 dep                                          analyse de dépendances seule
 CORRECTION AUTOMATIQUE ? ( GENM,PHON,GENS)
>


 >
 la maison de l'oncle que nous avons vu .     phrase à analyser

          |LA -(artd)
      |MAISON |(subc)
          |DE -(pepc)
                 |       |L'-(artd)
                 |ONCLE -(subc)                    première stucture
                 |       |QUE -(prl )                proposée
                 |       |NOUS -(popl)
                 |AVONS -(vers)
                         |VU -(ppas)
%

           |LA -(artd)
      |MAISON |(subc)
          |DE -(pepc)
          |       |       |L'-(artd)
          |       |ONCLE -(subc)                    deuxième strucure
          |       |QUE -(prl )                        proposée
          |       |NOUS -(popl)
          |AVONS -(vers)
                  |VU -(ppas)
%
```

```
 *
 depc              analyse de dépendances et de contrôle des variables

*
 syntaxe

                |LA -(artd)
          |MAISON |(subc)
                |DE -(pepc)
                        |        |L'-(artd)              une seule structure
                        |ONCLE -(subc)
                        |        |QUE -(prl )
                        |        |NOUS -(popl)
                        |AVONS -(vers)
                                 |VU -(ppas)
%

 *
 ortho
>
 /vu/vue/          modification de la phrase

                |LA -(artd)
          |MAISON |(subc)
                |DE -(pepc)
                |        |        |L'-(artd)
                |        |ONCLE -(subc)              une seule structure
                |        |QUE -(prl )
                |        |NOUS -(popl)
                |AVONS -(vers)
                         |VUE -(ppas)
%

 *

 >
 il le lui dit .          nouvelle phrase à analyser

               |IL -(popl)
               |LE -(popl)
               |LUI -(popl)
         |DIT -(verb)
%

 *
 crel       appel au compilateur des relations de dépendances
 >
 ?verb*popl        interrogation de la relation

VERB *  POPL := -32,-16,-8,1;
>
 /-32,-16,-8,1/-32,-16,1/       modification de la liste de poids
VERB *  POPL := -32,-16,1;
VERB *  POPL := -32,-16,-8,1;
REMPLACEMENT ?
>
```

```
 oui
VERS *  POPL := -32,-16,-8,1;
REMPLACEMENT ?
>
 non
 >

 *
 syntaxe
ERREUR DEPENDANCE ?        étant donné la modification précédente ,
*                          l'algorithme ne peut plus construire de structure
 texte
IL LE LUI DIT
*
 crel
 >
 verb*popl:=1,-8,-16,-32;
VERB *  POPL := -32,-16,1;      modification de la liste de poids
REMPLACEMENT ?
>
 oui
 VERS *  POPL := -32,-16,-8,1;
REMPLACEMENT ?
>
 non
 >

 *
 syntaxe
                |IL -(popl)
                |LE -(popl)          structure proposée
                |LUI -(popl)
        |DIT -(verb)
%

 *

 >
 il lui le dit .      nouvelle phrase à analyser

DIT
 IL
 LUI        erreur détectée par le contôleur de variables entre le
 LE         gouverneur "dit" et ses dépendants "il", "lui", "le"
 %

 ERREUR CONTROLE VARIABLES ?
*
 dep                      analyse de dépendances seule
 *
 syntaxe
                |IL -(popl)
                |LUI -(popl)              structure proposée sans vérifier
                |LE -(popl)                        les accords
        |DIT -(verb)
%
```

```
 >
 les chevals sont beau .                    nouvelle phrase à analyser
CHAINE INCORRECTE : CHEVALS SONT BEAU .     détection de l'erreur sur "chevals"
-
 ortho
>
 /chevals/chevaux/          correction manuelle de l'erreur

                          |LES -(artd)
                  |CHEVAUX  (subc)
          |SONT -(verb)                              strucure proposée
                  |BEAU -(adjq)
%
 *
 texte
 LES CHEVAUX SONT BEAU      le texte d'entree a été modifié
*
 depc       analyse de dépendances et contrôle des variables
*
 syntaxe
SONT        détection de l'erreur entre "sont" et "beau"
 BEAU
 %

 ERREUR CONTROLE VARIABLES ?
*
 ortho
>
 /beau/beaux/       correction manuelle de l'erreur
                          |LES -(artd)
                  |CHEVAUX  (subc)                  strucure proposée
          |SONT -(verb)
                  |BEAUX -(adjq)
%

 *
 >
 le colloque international de terminologie .      nouvelle phrase

CHAINE INCORRECTE : COLLOQUE INTERNATION          le mot "colloque" n'est pas
-                                                 dans le dictionnaire
 dict
 >
 ?/colloque/
/COINCIDENT/VRAI/       *COMME /SI /
/C/C/
 >
 /colloque/homme/                    introduction du mot "colloque" dans le dictio-
>                                    nnaire

                  |LE -(artd)
          |COLLOQUE  (subc)
                  |INTERNATIONAL  (adjq)                    structure proposée
                  |DE -(pepc)
                          |TERMINOLOGIE  (subc)
%
```

III.3 Transduction phonétique et génération graphique

```
 correct

EXECUTION BEGINS...
BAVARD(O/N)?
- o

 > chevale

  1   K    E    V    A    L
  2   K    E    V    A    L    E
  3   K    E1   V    A    L
  4   K    E1   V    A    L    E
  5   CH   E    V    A    L
  6   CH   E    V    A    L    E
  7   CH   E1   V    A    L
  8   CH   E1   V    A    L    E
CHEVAL
>
  ocurance

  1   O    K    U    R    AN   K    E
  2   O    K    U    R    AN   S    E
  3   O    S    U    R    AN   K    E
  4   O    S    U    R    AN   S    E
OCCURRENCE
OCCURRENCES
>
  occurense

  1   O    KS   U    R    AN   S    E
  2   O    K    U    R    AN   S    E
OCCURRENCE
OCCURRENCES
>
  jou

  1   J    OU
JOUE
 JOUES
JOUENT
J'OU
 >
  manje

  1   M    AN   J    E
MANGE
 MANGENT
MANGES
>
```

transduction phonétique

génération graphique

III.4 Exemples de générations graphiques avec contôle syntaxique après une traduction phonétique des chaînes incorrectes

```
  les bau chevo boivent .                          phrase à analyser

CHAINE INCORRECTE : BAU
CHAINE INCORRECTE : CHEVO                          détection des erreurs
                          |LES -(artd)
                          |BEAUX -(adjq)
                |CHEVAUX  (subc)                   solution proposée
        |BOIVENT  (verb)
%
 *

 > le chevale qui mangent et le chyen ki jou manje .   nouvelle phrase

CHAINE INCORRECTE : CHEVALE
CHAINE INCORRECTE : CHYEN
CHAINE INCORRECTE : KI
CHAINE INCORRECTE : JOU                            détection des erreurs lexicales
CHAINE INCORRECTE : MANJE
ERREUR CF. ?
-ERREUR       ' CHEVAL ' ET  'MANGENT '            détection de l'erreur syntaxique
CORRECTION PROPOSEE
****MANGE
                          |CHEVAL |LE -(artd)
                          |        |(subc)
                          |        |       |QUI -( prl)
                          |        |MANGE -(vers)
                  |ET -(coco)
                  |       |        |LE -(artd)          solution proposée
                  |       |CHIEN -(subc)
                  |                |       |QUI -( prl)
                  |                |JOUE -(vers)
          |MANGENT  (verb)
 %

 > ge tape le chevals .                            nouvelle phrase

CHAINE INCORRECTE : GE
CHAINE INCORRECTE : CHEVALS                        détection des erreurs
                |JE -(popl)
        |TAPE -(verb)                              solution proposée
                |       |LE -(artd)
                |CHEVAL |(subc)
%
```

III.5 Exemples de corrections à l'aide du générateur morphologique

```
>
nous allerons sur les bels chevals de bois .        phrase à analyser

CHAINE INCORRECTE : ALLERONS SUR LES BEL            détection première erreur
 IRONS                  VERB    PLU UNO FUT IND     solution proposée
-
 ortho
>                                                   correction manuelle
 /allerons/irons/

CHAINE INCORRECTE : BELS CHEVALS DE BOIS            détection deuxième erreur
 BEAUX                  ADJQ    MAS PLU             solution proposée
-
 ortho
>                                                   correction manuelle
 /bels/beaux/

CHAINE INCORRECTE : CHEVALS DE BOIS   .             détection troisième erreur
 CHEVAUX                SUBC    MAS PLU             solution proposée
-
 ortho
>                                                   correction manuelle
 /chevals/chevaux/
```

```
        |IRONS -(verb)|NOUS -(popl)
                      |SUR -(pepc)
                             |       |LES -(artd)
                             |       |BEAUX -(adjq)      première structure proposée
                             |CHEVAUX  (subc)
                                     |DE -(pepc)
                                             |BOIS -(subc)
%

          |NOUS -(popl)
        |IRONS -(verb)
                |SUR -(pepc)
                |       |       |LES -(artd)        deuxième structure proposée
                |       |       |BEAUX -(adjq)
                |       |CHEVAUX  (subc)
                |DE -(pepc)
                        |BOIS -(subc)
%
```

III.6 Exemples en documentation automatique

```
 piafdoc
LE SYSTEME PIAFDOC EST A VOTRE SERVICE
LABO INFORMAT USMG BP53 38041 GRENOBLE
EXECUTION BEGINS...
ACTIVATION PROGRAMME ? (GRAM,DICT,BASEDOC,INTERRODOC)
$
 basedoc
DONNEES ? (TERMINAL OU NOM DE FICHIER)
%
 terminal
FORME? (TEXTE OU MISTRAL (ENTREE))
%
 texte
RESULTATS? (TERMINAL OU NOM DE FICHIER)
%
 terminal
FORME? (SRACINE OU MISTRAL (SORTIE))
%
 sracine
>
 les etudiantes privees d'emploi,                    texte d'entrée

ETUDIANT$PRIVE D'EMPLOI $CHOMEUR$                    mots clés et synonyme préférentiel
*

 >
 vous tiendrez compte des projets de reforme,        texte d'entrée

TENIR COMPTE $PROJET DE REFORME $PROJET DE LOI $PROJET$REFORME$
*

 >
 vous tintes compte des etudiantes sans emploi,      texte d'entrée

TENIR COMPTE $ETUDIANT$SANS EMPLOI $CHOMEUR$   mots clés , synonyme préférentiel
*

 >
 les reformes entreprises,                           texte d'entrée

1 ENTREPRISE
2 ENTREPRENDRE                                       participe passé ou substantif ?
CHOIX HOMOGRAPHE ? (MCL: N)
-
 mcl: 2                                              choix de l'utilisateur

REFORME$ENTREPRENDRE$                                mots clés
*

 >
```

CONCLUSION

Le système P.I.A.F., comme nous l'avons vu, permet de nombreuses applications obtenues :

- en modifiant les paramètres linguistiques,
- en changeant la structure de contrôle,
- en utilisant la réversibilité du transducteur général d'états finis.

L'utilisation de plusieurs modèles indépendants et complémentaires permet non seulement de réduire la combinatoire de l'analyse mais également de définir des modèles moins stricts et incomplets, donc plus faciles à élaborer. Actuellement, P.I.A.F. utilise deux filtres pour l'analyse syntaxique. Cette limitation n'est qu'artificielle, et il n'est pas impensable d'envisager d'y inclure d'autres filtres tels que le programme réalisé par V. Joloboff où d'en définir de nouveaux contrôlant par exemple les liens horizontaux ou verticaux d'une structure. Les différents filtres indépendants opèreraient en parallèle sur une structure de données commune qui est ici l'ensemble des relations de dépendances d'une phrase donnée.

En outre, P.I.A.F. est une première approche d'un logiciel orienté vers l'analyse de données ambigües. En fonction des applications que nous avons traitées, on peut dégager les notions suivantes :

- la notion d'erreur ou d'ambiguité est toujours relative à un modèle donné.

- les différents aspects de la redondance sont concrétisés par la définition d'invariants indépendants.

D'autres applications pourraient être envisagées telles que la détection et la correction d'erreurs liées à l'insuffisance d'un appareillage utilisé : lecteur optique, reconnaisseur de parole.

Dans toutes les applications que nous avons expérimentées et dans toutes celles que nous envisageons, l'interactivité du système est essentielle pour laisser le contrôle définitif à l'utilisateur ; car, comme nous l'avons déjà indiqué, on ne peut proposer dans ce domaine qu'un traitement assisté.

Le système P.I.A.F. est opérationnel sur l'ordinateur IBM 370/67 sous CP/CMS du Centre Interuniversitaire de Calcul de Grenoble.

BIBLIOGRAPHIE

[AHUL] AHO A.V. - ULLMAN J.D.

The theory of parsing, translation, and compiling
Volume 1, Parsing, Volume 2 - Compiling
Prentice Hall 1973.

[APRE] APRESJAN JU.D

Eléments sur les idées et les méthodes de la linguistique structurale contemporaine.
Monographies de linguistique mathématique n° 5
DUNOD 1973.

[BOIT] BOITET C.

Un essai de réponse à quelques questions théoriques et pratiques liées à la traduction automatique.
Définition d'un système prototype.
Thèse de doctorat d'Etat - Université de Grenoble 1976.

[BWKRL] BOBROW D.G. - WINOGRAD T. - KRL RESEARCH GROUP

Expérience with KRL-0 : One cycle of a knowledge representation language.
5 th International Joint Conference on Artificial Intelligence 1977.

[CAR3] CARBONNEL

Natural semantics in artificial intelligence
Proc. of IJCA I 3 - Stanford Août 1973.

[CHI] CHAUCHE J.

Transducteurs et arborescences. Etudes et réalisations de systèmes appliquées aux grammaires transformationnelles.
Thèse de Doctorat d'Etat - Université de Grenoble 1974.

[CHCGV] CHIARAMELLA Y. - COURTIN J. - GRANDJEAN E. - VEILLON G.

Utilisation des techniques de recherche en parallèle au contrôle de données ambiguës. Applications aux sciences humaines.
Congrès A.F.C.E.T. : Panorama de la nouveauté informatique en France.
Paris Novembre 1976.

[CHIA] CHIARAMELLA Y.

Détection automatique des variations orthographiques sur des noms propres. Définition d'un transducteur morpho-phonétique intéractif.
Conférence internationale sur le traitement automatique des langues.
OTTAWA Juin/juillet 1976.

[CHO] CHOMSKY N.

Aspects of theory of syntax.
The M.I.T. Press 1965.

[COLM1] COLMERAUER A. et al.

Un système de communication homme-machine en Français.
Université d'Aix-Marseille, octobre 1972.

[COLM2] COLMERAUER A.

Les systèmes -Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur.
Projet TAUM. Université de Montréal, Janvier 1971.

[COU1] COURTIN J.

Organisation d'un dictionnaire pour l'analyse morphologique.
Séminaire de théorie des automates et de traitement automatique des langues. Grenoble, Février 1973.

[COU2] COURTIN J.

Un analyseur syntaxique interactif pour la communication homme-machine.
Conférence internationale sur le traitement automatique des langues.
PISE 27 août/1er septembre 1973.

[COU3] COURTIN J.

Un système d'analyse des langues naturelles : application à la correction intéractive de textes.
Séminaire de programmation - Grenoble, 18 avril 1975.

[COU4] COURTIN J.

Utilisation des redondances pour l'analyse et le contrôle automatiques d'énoncés en langue naturelle.
Conférence internationale sur le traitement automatique des langues.
OTTAWA juin/juillet 1976.

[COU5] COURTIN J.

Langages analysables de gauche à droite.
Construction d'un analyseur pour langages LR (1)
Thèse de 3ème cycle - Université de Grenoble - 29 juin 1968.

[COUD] COURTIN J. - DUJARDIN D.

Paramètres linguistiques de la morphologie française dans le système P.I.A.F.
Document interne - Grenoble - décembre 1976.

[COUGD] COURTIN J. - DUJARDIN D. - GRANDJEAN E.

Editeur lexicographique pour les langues naturelles.
Document interne. Grenoble 1976.

[COURS] COURTIN J. - RIEU J.L. - SGALL P.

Un métalangage pour l'analyse morphologique.
Document interne C.E.T.A. Grenoble - septembre 1969.

[COUS] COUSOT P.

Structures de données et de programmes.
Cours Université de Grenoble 1975.

[COVO] COURTIN J. - VOIRON J.

Modèle pour l'apprentissage de la programmation.
R.A.I.R.O. Informatique vol. 11, n° 1, 1977 p. 3 à 20.

[CSV1] CUNIN P.Y. - SIMONET M. - VOIRON J.

Méthodologie d'écriture des compilateurs.
Une expérience du langage Algol 68.
Thèse de docteur Ingénieur - Université de Grenoble, avril 1976.

[CSV2] CUNIN P.Y.- DELAUNAY M. - SIMONET M. - VOIRON J.

Construction d'un arbre abstrait "décoré" dans un compilateur.
Séminaire I.R.I.A. 18 avril 1975.

[DSK] DELL'ORCO P. - SPADAVECCHIA V.N. - KING M.

Using knowledge of a data base world in interpreting natural language queries.
I.F.I.P. Congress 1977.

[DUJ] DUBREIL J. - DUBREIL - JACOTIN M.L.

Leçons d'algèbre moderne.
Dunod 1961.

[FEN] FENNEL R.D.

Multiprocess software architecture for ai problem solving
Carnegie Mellon University, May 1975.

[FLOYD] FLOYD R.W.

Bounded context syntactic analysis
Comm. A.C.M. 7 : 2, 62-67, 1964.

Syntactic analysis and operator precedence
J.A.C.M. 10 : 3, 316-333, 1963.

[GAIF] GAIFMAN H.

Dependency system and phrase structure system.
Information and Control. Vol. 8, n° 3 juin 1965.

[GRAN1] GRANDJEAN E.

Compilateur syntaxique
Document C.E.T.A. GTC 17
Programme d'analyse syntaxique
Document C.E.T.A. GCT 18.

[GRAN2] GRANDJEAN E.

Conception et réalisation d'un dictionnaire pour un analyseur interactif de langues naturelles.
Mémoire C.N.A.M. Université de Grenoble, 1975.

[GRAN3] GRANDJEAN E.

Système P.I.A.F. Détecteur de fautes d'orthographe : P.I.A.F.D.E.T.
Document interne - Grenoble 1976.

[GROSS] GROSS M.

Méthodes en syntaxe.
Hermann, 1975.

[HAYS] HAYS D.G.

Dependency theory : a formalism and some observations.
Memorandun R.M. 4087 P.R. The rand corporation - 1964.

[HEND] HENDRIX G.

Human engineering for applied natural language processing.
5 th International Joint Conference on Artificial Intelligence 1977.

[HOUL] HOPCROFT J.E. - ULLMAN J.D.

Formal languages and their relation to automata
Addison Wesley, 1969.

[KNU1] KNUTH D.E.

The art of computer programming.
Volume 1, 3. Addison Wesley 1973.

[KNU2] KNUTH D.E.

Optimum binary search trees
Acta Informatica - p. 14-25, 1971.

[KNU3] KNUTH D.E.

On the translation of languages from left to right
Information and control 8 : 6, 607-639, 1965.

[KNU4] KNUTH D.E.

Top down analysis
International summer school on computer programming.
Copenhague - Août 1967.

[MACH] MACHOVA S.

Fonctional generative description of Czech. Dependency phrase structure model.
Explizite beschreiburg der spache und automatische textbearbeitung
Vol. 3, Praha 1977.

[MAHL] MAHL R.

Algorithmique et structures de données
Cours d'informatique, Ecole des Mines de Saint-Etienne.

[MIST] MISTRAL : recherche documentaire
Manuel d'utilisation : C.I.I. 1974.

[OTT] Preprint de la conférence internationale sur le traitement automatique des langues. OTTAWA, juin/juillet 1976.

[PAIR1] PAIR C.

Les structures d'information et leur représentation en mémoire.
Cours de l'Ecole d'été d'Informatique de l'A.F.C.E.T., Alès 1971.

[PAIR2] PAIR C.

Compilation
Cours de l'Ecole d'été d'Informatique de l'A.F.C.E.T.
Neuchatel, 1972.

[PETR] PETRICK S.R.

On natural language based computer systems.
I.B.M. Journal of Research and Development - vol. 20, n° 4, juillet 1976.

[PISE] Prepint de la conférence internationale sur le traitement automatique des langues.
PISE - 27 août/1er septembre 1973

[PLAT] PLATH W.J.

REQUEST : A natural language question - answering system
I.B.M. Journal of Research and Development - vol. 20, n° 4, juillet 1976.

[SACER] SACERDOTI E.

Language access to distributed data with error recovery
5 th International Joint Conference on Artificial Intelligence. 1977.

[SALBER] SALTON G. - BERGMARK D.

Clustered file generation and its application to computer science taxonomies.
I.F.I.P. Congress 1977.

[SALT] SALTON G.

The SMART System. Experience in Automatic Document Processing
Prentice Hall Inc. 1971.

[SCHA] SCHANK R.C.

Identification of Conceptualizations Underlying natural language.
Computer Models of Thought and Language.
Freemann San Francisco 1973.

[SIMM] SIMMONS R.F.

Semantic Networks : Their Computation and use for understanding english sentences.
Computer Models of Thought and Language.
Freemann San Francisco 1973.

[TE] TESNIERES

Eléments de syntaxe structurale
Klinchsieck 1959.

[THOM] THOMPSON F.B. - THOMPSON B.H.

Pratical natural language processing : The REL system as prototype ,
Advances in computers 13, Académic Press, New York, 1975.

[TSE] TSEITINE G.S.

Algorithme d'analyse syntaxique simplifiée.
Pb. Cyb. n° 24, p. 227-242, 1971.

[VAKU] VAKULOVSKAIA G.V. - KULAGINA O.S.

Sur un algorithme d'analyse syntaxique de textes russes.
Pb. Cyb. n° 18, p. 217-301, 1967.

[VAUQ] VAUQUOIS B.

La traduction automatique à Grenoble.
Documents de linguistique quantitative n° 24.
DUNOD 1975.

[VEILL1] VEILLON G.

Modèles et algorithmes pour la traduction automatique.
Thèse d'Etat - Université de Grenoble, 1970.

[VEILL2] VEILLON G.

Algorithmique
Cours Université de Grenoble, 1974.

[VEILL3] VEILLON G.

La notion de niveau dans les systèmes d'analyse automatique des langues naturelles. Une approche informatique.
Colloque sur les modèles logiques et niveaux d'analyse linguistique.
METZ, Novembre 1974.

[VEIV] VEILLON G. - VEYRUNES J.

Etude de la réalisation pratique d'une grammaire Context free et de l'algorithme associé.
Document C.E.T.A. G.001-1 - Avril 1964.

[WALK] WALKER D.E.

Speech understanding research.
Final report. Project 4762.
Artificial Intelligence Center. Stanford 1976.

[WIL] WILKS Y.

An Artificial Intelligence Approach to Machine Translation.
Computer Models of Thought and Language
Freemann San Francisco 1973.

[WIN] WINOGRAD T.

A procedural model of language understanding . Computer models of thought and language.
W.H. Freemann - San Francisco. 1973.

[WOOD] WOODS W.

Transition network grammars for natural language analysis
Comm. A.C.M. vol. 13, n° 10, octobre 1970.

[WOODS] WOODS et al

Speech understanding system
Final technical progress report. B B N
Report n° 3438 Cambridge 1976.