Etudes et algorithmes liés à une nouvelle structure de données en T.A : les E-graphes

Marco Antonio Clemente-Salazar

THESE

présentée à

l' Université Scientifique et Médicale de Grenoble

pour obtenir le grade de

DOCTEUR INGENIEUR
"Informatique"

par

Marco Antonio CLEMENTE - SALAZAR

ETUDES ET ALGORITHMES LIES A UNE NOUVELLE STRUCTURE DE DONNEES EN T.A. : LES E-GRAPHES.

Thèse soutenue le 17 mai 1982 devant la Commission d'Examen :

Monsieur	**B. VAUQUOIS**	:	**Président**
Messieurs	**Ch. BOITET**		
	Ph. JORRAND		
	R. PASERO		**Examinateurs**
	G. VEILLON		
	J. VIDART		

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

année scolaire 1980-1981

Président de l'Université : M. J.J. PAYAN

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

PROFESSEURS DE 1ère CLASSE

Mlle	AGNIUS DELORD Claudine	Biophysique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Clinique dermatologie
	AMBROISE THOMAS Pierre	Parasitologie
	ARNAUD Paul	Chimie
	ARVIEU Robert	Physique nucléaire
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale A
	BEAUDOING André	Clinique pédiatrie et puériculture
	BELORISKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
Mme	BERIEL Hélène	Pharmacodynamie
M.	BERNARD Alain	Mathématiques pures
Mme	BERTRANDIAS Françoise	Mathématiques pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques pures
	BEZES Henri	Clinique chirurgicale & traumatologie
	BILLET Jean	Géographie
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET EYMARD Joseph	Clinique Hépato-gastro-entérologie
Mme	BONNIER Jane-Marie	Chimie générale
MM.	BOUCHERLE André	Chimie et toxicologie
	BOUCHET Yves	Anatomie
	BOUCHEZ Robert	Physique nucléaire
	BRAVARD Yves	Géographie

.../...

MM.	BUTEL Jean	Orthopédie
	CABANEL Guy	Clinique rhumatologie et hydrologie
	CARLIER Georges	Biologie végétale
	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHARACHON Robert	Clinique O.R.L.
	CHATEAU Robert	Clinique neurologique
	CHIBON Pierre	Biologie animale
	COEUR André	Chimie analytique et bromotologique
	COUDERC Pierre	Anatomie pathologique
	CRABBE Pierre	C.E.R.M.O.
	DAUMAS Max	Géographie
	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELOBEL Claude	M.I.A.G.
	DELORMAS Pierre	Pneumo-phtisiologique
	DENIS Bernard	Clinique cardiologique
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Electrochimie
	DODU Jacques	Mécanique appliquée IUT 1
	DOLIQUE Jean-Michel	Physique des plasmas
	DUCROS Pierre	Cristallographie
	FONTAINE Jean-Marc	Mathématiques pures
	GAGNAIRE Didier	Chimie physique
	GASTINEL Noël	Analyse numérique
	GAVEND Jean-Micnel	Pharmacologie
	GEINDRE Michel	Electro-radiologie
	GERBER Robert	Mathématiques pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	JEANNIN Charles	Pharmacie galénique
	JOLY Jean-René	Mathématiques pures
	KAHANE André	Physique
	KAHANE Josette	Physique
	KLEIN Joseph	Mathématiques pures
	KOSZUL Jean-Louis	Mathématiques pures
	LACAZE Albert	Hermodynamique
	LACHARME Jean	Biologie cellulaire
	LAJZEROWICZ Joseph	Physique

.../...

Mme	LAJZEROWICZ Jeannine	Physique
MM.	LATREILLE René	Chirurgie thoracique
	LATURAZE Jean	Biochimie pharmaceutiques
	LAURENT Pierre	Mathématiques appliquées
	LE NOC Pierre	Bactériologie virologie
	LLIBOUTRY Louis	Géophysique
	LOISEAUX Jean-Marie	Sciences nucléaires
	LOUP Jean	Géographie
	LUU DUC Cuong	Chimie générale et minérale
	MALINAS Yves	Clinique obstétricale
Mlle	MARIOTTE Anne-Marie	Pharmacognostie
MM.	MAYNARD Roger	Physique du solide
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	NEGRE Robert	Mécanique IUT 1
	MOZIERES Philippe	Spectrométrie physique
	OMONT Alain	Astrophysique
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY PEYROULA Jean-Claude	Physique
	PERRET Jean	Sémeiologie médicale (neurologie)
	PERRIER Guy	Géophysique
	PIERRARD Jean-Marie	Mécanique
	RACHAIL Michel	Clinique médicale B
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
Mme	RENAUDET Jacqueline	Bactériologie
M.	REVOL Michel	Urologie
Mme	RINAUDO Marguerite	Chimie CERMAV
MM.	DE ROUGEMONT Jacques	Neuro-chirurgie
	SARRAZIN Roger	Clinique chirurgicale B
Mme	SEIGLE MURANDI Françoise	Botanique et crytogamie
MM.	SENGEL Philippe	Biologie animale
	SIBILLE Robert	Construction mécanique IUT 1
	SOUTIF Michel	Physique
	TANCHE Maurice	Physiologie
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire

.../...

MM.	VAN CUTSEM Bernard	Mathématiques appliquées
	VAUQUOIS Bernard	Mathématiques appliquées
	VERAIN Alice	Pharmacie galénique
	VERAIN André	Biophysique
	VIGNAIS Pierre	Biochimie médicale

PROFESSEURS DE 2ème CLASSE

MM.	ARNAUD Yves	Chimie IUT 1
	AURIAULT Jean-Louis	Mécanique IUT 1
	BEGUIN Claude	Chimie organique
	BOITET Christian	Mathématiques appliquées
	BOUTHINON Michel	E.E.A. IUT 1
	BRUGEL Lucien	Energétique IUT 1
	BUISSON Roger	Physique IUT 1
	CASTAING Bernard	Physique
	CHARDON Michel	Géographie
	CHEHIKIAN Alain	E.E.A. IUT 1
	COHEN Henri	Mathématiques pures
	COHENADDAD Jean-Pierre	Physique
	COLIN DE VERDIERE Yves	Mathématiques pures
	CONTE René	Physique IUT 1
	CYROT Michel	Physique du solide
	DEPASSEL Roger	Mécanique des fluides
	DOUCE Roland	Physiologie végétale
	DUFRESNOY Alain	Mathématiques pures
	GASPARD François	Physique
	GAUTRON René	Chimie
	GIDON Maurice	Géologie
	GIGNOUX Claude	Sciences nucléaires
	GLENAT René	Chimie organique
	GOSSE Jean-Pierre	E.E.A. IUT 1
	GROS Yves	Physique IUT 1
	GUITTON Jacques	Chimie
	HACQUES Gérard	Mathématiques appliquées
	HERBIN Jacky	Géographie
	HICTER Pierre	Chimie
	IDELMAN Simon	Physiologie animale
	JOSELEAU Jean-Paul	Biochimie
	JULLIEN Pierre	Mathématiques appliquées
	KERCKOVE Claude	Géologie

.../...

MM.	KRAKOWIACK Sacha	Mathématiques appliquées
	KUHN Gérard	Physique IUT 1
	KUPKA Yvon	Mathématiques pures
	LUNA Domingo	Mathématiques pures
	MACHE Régis	Physiologie végétale
	MARECHAL Jean	Mécanique
	MICHOULIER Jean	Physique IUT 1
Mme	MINIER Colette	Physique IUT 1
MM.	NEMOZ Alain	Thermodynamique
	NOUGARET Marcel	Automatique IUT 1
	OUDET Bruno	Mathématiques appliquées
	PEFFEN René	Métallurgie IUT 1
	PELMONT Jean	Biochimie
	PERRAUD Robert	Chimie IUT 1
	PERRIAUX Jean-Jacques	Géologie minéralogie
	PERRIN Claude	Sciences nucléaires
	PFISTER Jean-Claude	Physique du solide
	PIERRE Jean-Louis	Chimie organique
Mlle	PIERY Yvette	Physiologie animale
MM.	RAYNAUD Hervé	Mathématiques appliquées
	RICHARD Lucien	Biologie végétale
	ROBERT Gilles	Mathématiques pures
	ROBERT Jean-Bernard	Chimie physique
	ROSSI André	Physiologie végétale
	SAKAROVITCH Michel	Mathématiques appliquées
	SARROT REYNAUD Jean	Géologie
	SAXOD Raymond	Biologie animale
Mme	SOUTIF Jeanne	Physique
MM.	STUTZ Pierre	Mécanique
	VIALON Pierre	Géologie
	VIDAL Michel	Chimie organique
	VIVIAN Robert	Géographie

CHARGES D'ENSEIGNEMENT PHARMACIE

MM.	ROCHAS Jacques	Hygiène et hydrologie
	DEMENGE Pierre	Pharmacodynamie

PROFESSEURS SANS CHAIRE (médecine)

M.	BARGE Michel	Neuro-chirurgie

.../...

MM.	BOST Michel	Pédiatrie
	BOUCHARLAT Jacques	Psychiatrie
	CHAMBAZ Edmond	Biochimie (hormonologie)
	CHAMPETIER Jean	Anatomie
	COLOMB Maurice	Biochimie
	COULOMB Max	Radiologie
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GROULADE Joseph	Biochimie A
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Gérontologie
	JALBERT Pierre	Histologie
	MAGNIN Robert	Hygiène
	PHELIP Xavier	Rhumatologie
	REYMOND Jean-Charles	Chirurgie générale
	STIEGLITZ Paul	Anesthésiologie
	VROUSOS Constantin	Radiothérapie

MAITRES DE CONFERENCES AGREGES (médecine)

MM.	BACHELOT Yvan	Endocrinologie
	BENABID Alim Louis	Médecine et chirurgie
	BERNARD Pierre	Gynécologie obstétrique
	CONTAMIN Charles	Chirurgie thoracique
	CORDONNIER Daniel	Néphrologie
	CROUZET Guy	Radiologie
	DEBRU Jean-Luc	Médecine interne
	DYON Jean-François	Chirurgie infantile
	FAURE Claude	Anatomie et organogènèse
	FAURE Gilbert	Urologie
	FLOYRAC Roger	Biophysique
	FOURNET Jacques	Hépato-gastro-entérologie
	GAUTIER Robert	Chirurgie générale
	GIRARDET Pierre	Anesthésiologie
	GUIDICELLI Henri	Chirurgie générale
	GUIGNIER Michel	Thérapeutique (réanimation)
	JUNIEN-LAVILLAUROY Claude	Clinique O.R.L.
	KOLODIE Lucien	Hématologie biologique
	MALLION Jean-Michel	Médecine du travail
	MASSOT Christian	Médecine interne
	MOUILLON Michel	Ophtalmologie

.../...

MM.	PARAMELLE Bernard	Pneumologie
	RACINET Claude	Gynécologie-Obstétrique
	RAMBAUD Pierre	Pédiatrie
	RAPHAEL Bernard	Stomatologie
	SCHAEFER René	Cancérologie
	SEIGNEURIN Jean-Marie	Bactériologie-virologie
	SOTTO Jean-Jacques	Hématologie
	STOEBNER Pierre	Anatomie-pathologique

A mes parents.

A mes frères.

A ma femme.

Je tiens à remercier.

Monsieur B. Vauquois, Professeur à l'Université Scientifique et Médicale de Grenoble (USMG) et Directeur du Groupe d'Etudes pour la Traduction Automatique (GETA), pour l'honneur qu'il me fait de présider le Jury.

Monsieur Ch. Boitet, Maître de Conférences à l'USMG de Grenoble, pour ses innombrables conseils et sa constante supervision tout le long de mes recherches.

Monsieur G. Veillon, Professeur à l'Institut National Polytechnique de Grenoble, pour avoir bien voulu juger mon travail et faire partie de ce Jury.

Monsieur J. Vidart, Professeur à l'Université Simón Bolívar de Caracas, pour avoir lu en détail cette thèse, m'apportant ainsi, par ses critiques, une connaissance plus approfondie de mon sujet.

Monsieur Ph. Jorrand, Maître de Recherches au CNRS, est un observateur attentif de mon travail et je suis sensible à l'interêt qu'il lui porte.

Monsieur R. Pasero, Maître Assistant à l'Université d'Aix-Marseille, pour l'interêt qu'il a temoigné à mon travail et les suggestions pertinentes qu'il m'a faites.

Je voudrais remercier aussi tous et chacun des membres du GETA pour son excellent accueil et pour son aide soit directe, soit grâce à leur travail dans le groupe. Finalement, à tous ceux qui d'une façon ou d'une autre ont contribué à la réalisation de cette thèse.

Marco CLEMENTE

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

CHAPITRE II. TRANSFORMATIONS DE LA STRUCTURE.

CHAPITRE III. PROGRAMMATION D'UN PROTOTYPE.

ETUDES ET ALGORITHMES LIES A UNE NOUVELLE STRUCTURE DE DONNEES EN T.A. : LES E-GRAPHES

INTRODUCTION.

Tout le long de son existence, la traduction automatisée s'est heurtée au problème du traitement des ambiguités. Ce traitement comprend deux aspects: (1) l'aspect linguistique, où ce problème est résolu par une stratégie appropriée et (2) l'aspect formel, où ce problème nous mène à la recherche d'une représentation appropriée, tout particulièrement pour le traitement automatique. C'est ce dernier aspect qu'aborde ce travail. Plus précisément, il s'agit de l'étude d'une nouvelle structure de données qui s'efforce d'allier souplesse de représentation et de traitement.

Dans le passé, on a utilisé d'abord les chaînes comme moyen de représentation, puis les arbres (étiquetés, décorés), les graphes de chaînes et les réseaux (sémantiques). Dans ce travail, on étudie un type particulier de structure: les E-graphes. Cette structure veut allier certains avantages des systèmes-Q et des arbres d'ARIANE-78; notamment l'utilisation d'une même structure pour tout le processus de traduction du premier, la décidabilité prévisible, le parallélisme du second.

Ainsi, le Chapitre I fait un bref tour d'horizon de la traduction automatisée et présente une discussion des traits caractéristiques des deux systèmes qui nous ont servi de base, accompagné des motivations du projet européen de traduction: EUROTRA, pour lequel le GETA a proposé les E-graphes. On trouvera ensuite une discussion sur quelques structures en usage, puis à la fin la définition des E-graphes et leurs notions principales: sous-structures, isolabilité, ordre, etc.

Le Chapitre II est une étude des transformations de la structure. On analyse les transformations globales, c'est-à-dire celles où il y a une modification de la structure complète. Puis les transformations locales, i.e. celles où on opère sur de sous-structures. Enfin, on propose une écriture des schémas de E-graphes et des règles de grammaire en vue de la transduction.

Le Chapitre III programme en PROLOG un prototype réduit pour une classe propre de E-graphes qui a la caractéristique de pouvoir être décrite par une écriture polynômiale.

Finalement, on trouvera en Annexe les bases pour une programmation plus efficace des les E-graphes en général. On réalise aussi un système de réécriture qui pour l'instant est une simulation des systèmes-Q.

CHAPITRE I

DEFINITION DU PROBLEME

I N T R O D U C T I O N

Ce chapitre a pour objet, d'une part, la présentation des idées qui sont à la base de ce travail et, d'autre part, la définition formelle de la structure objet.

On précise d'abord quelques termes, puis on passe à un très bref panorama des outils de programmation qui ont existé et qui existent. La nécessité de structures de données puissantes est mise en relief de même que l'importance de la complexité, la décidabilité et l'adéquation.

On trouvera ensuite une présentation des modèles qui ont inspiré ce travail et l'analyse des détails, surtout des détails informatiques qui m'ont semblé rentrer dans le cadre des buts poursuivis.

La dernière partie définit la structure. On débute par un petit rappel de notions essentielles des réseaux pour continuer avec les sous-structures permises, leurs propriétés et terminer par la définition d'un ordre, d'un parcours canonique de la structure et de l'égalité entre E-graphes.

I. TRADUCTION AUTOMATISEE.

1. GENERALITES [13,1,5]

On voit souvent des articles où figurent les termes de "traduction automatique", "traduction assistée par ordinateur" et "aide à la traduction". Il y manque celui de "traduction automatisée" auquel on s'intéresse plus particulièrement pour remplacer celui de "traduction automatique" qui a fait l'objet d'usages abusifs et a tendance à induire en erreur.

Je voudrais, d'abord, replacer ces termes dans le contexte des utilisations possibles de l'informatique pour la traduction en reprenant [5] et faire la distinction entre:

- **Traduction humaine assistée par la machine** (THAM) ou **Traduction assistée**.

  Le traducteur est aidé par un système informatique (traitement de texte, dictionnaire ou thésaurus automatique). C'est toujours le traducteur humain qui traduit.

- **Traduction interactive** (TI).

  C'est la machine qui traduit en se faisant aider par un linguiste. Le système ITS de l'Université de Provoh (Utah) est un exemple de TI.

- **Traduction automatique aidée par l'homme** (TAAH), (que l'on appelle ici "automatisée") ou **Traduction automatique révisée**.

  Il n'y a pas d'interaction homme-machine au cours de la traduction brute. Certaines parties du texte peuvent être refusées par le système et doivent par conséquent être traduites par l'homme. La traduction brute obtenue est révisée si la qualité demandée l'exige. C'est le cas des systèmes METEO, TAUM-AVIATION (Montréal) et des systèmes du GETA (Grenoble) et du SFB 100 (Saarbruecken).

- **Traduction automatique**.

  Tout est traduit par la machine. On ne prévoit pas de révision, car on suppose que la qualité sera assez élevée pour pouvoir s'en passer, ou bien que les utilisateurs potentiels se contenteront de ce qui est produit par le système, bon ou mauvais.

2. BREF HISTORIQUE [1,23,24].

Depuis 1945 [24], des efforts importants ont été engagés sur la TA dans divers pays. Plusieurs voies d'approche ont donné lieu à différentes **générations** [23,1] de systèmes.

Si les méthodes de première génération (Georgetown par exemple) ont obtenu des résultats assez vite, la qualité de traduction est insuffisante pour les besoins des utilisateurs qui voudraient obtenir une traduction brute d'assez bonne qualité pour être révisée par un spécialiste monolingue ou simplement parcourue.

Les méthodes de deuxième génération demandent un travail initial plus grand et, si la qualité de traduction est meilleure, elle ne l'est pas dans le même rapport. Cependant, la possibilité d'améliorer le système en ne modifiant que des données externes, sans toucher à aucun programme est un avantage décisif. Parcourons maintenant quelques méthodes de programmation.

3. OUTILS DE PROGRAMMATION.

Les premiers chercheurs en TA programmaient directement en langage machine [22]. Encore maintenant, les systèmes SYSTRAN sont essentiellement composés de programmes et de tables écrits en macro-assembleur IBM 370, et les traces d'exécution sont réduites à des "dumps" hexadécimaux.

L'étape suivante est celle où des systèmes formels appliquent un algorithme unique sur une entrée et un ensemble non-structuré de règles qui n'ont aucun contrôle sur cet algorithme (grammaires hors-contexte, systèmes-Q). Plus tard, on a vu la création des modèles où les règles peuvent comporter des contrôles sur l'algorithme général, qui simule toujours un modèle non-déterministe: PROLOG [20], ATEF [10], ROBRA [8], les ATN [25] et des modèles dérivés comme REZO [21].

Dans les systèmes actuels, on trouve cependant des parties écrites directement dans des langages de programation plus usuels, et de plus bas niveau: macro-assembleur (SYSTRAN), FORTRAN (SUSY), PASCAL (TAUM/AVIATION). Ceci est sans doute une erreur, dans la mesure où le niveau est trop bas pour que des linguistes programment avec aisance et efficacité et où par conséquent il faut passer par un intermédiaire informaticien qui "traduit" (souvent mal) l'idée du linguiste, ou bien voir les linguistes passer le plus clair de leur temps à résoudre des problèmes de programmation et non des problèmes linguistiques.

4. NECESSITE DE STRUCTURES DE DONNEES PUISSANTES [5].

La source de l'inadéquation des langages de programmation usuels provient du bas niveau de leurs structures de données et de contrôle. On y travaille sur des entiers, des caractères, des tableaux, ... alors qu'on a besoin de travailler directement sur des arbres et des dictionnaires. Par exemple, la construction d'un type "arbre" en PASCAL est, à cet égard, une illusion, car on définit seulement le type "cellule permettant d'implanter un noeud d'un arbre", sans se préoccuper de définir les opérateurs sur le type ni de garantir sa permanence: ainsi, une fausse manoeuvre dans la manipulation des pointeurs peut transformer un arbre en une structure "bouclée".

D'autre part, les traitements lingustiques sont souvent impérativement liés à des modèles non-déterministes, soit parce qu'il faut produire plusieurs solutions (analyse morphologique, syntaxique), soit parce que l'on veut disposer d'un mécanisme de "backtrack" afin de programmer de façon "heuristique".

Beaucoup de langages de programmation pour l'IA ou le traitement de langues naturelles incluent une forme de non-déterminisme, mais très peu sont munis de paralélilisme (ROBRA, systèmes-Q) ou de fonctions de contrôle pour définir des heuristiques (PROLOG, ATEF, ROBRA). Le **principe No 6** de [5] résume ce point de vue.

> Les logiciels de base pour la TAAH seront de plus en plus des langues ou des systèmes de programmation de très haut niveau, aves des types de données puissants et adaptés (chaînes, arbres, graphes, et réseaux décorés), avec des opérateurs spécifiques, ainsi que des primitives de contrôle pour la programmation non-déterministe, parallèle et heuristique.

5. COMPLEXITE, DECIDABILITE, ADEQUATION [5,9].

Si on prend tous les types nécessaires, avec tous les opérateurs possibles et toutes les structures de contrôle, on est pratiquement certain que le modèle de calcul obtenu aura la puissance maximale de calcul, celle d'une machine de Turing, et qu'il sera donc indécidable. Par conséquent, on ne pourra pas, en général, donner d'estimations de la complexité dynamique d'un programme non-trivial écrit dans ce formalisme.

D'autre part, tout ce que l'on veut programmer dans des systèmes de TAAH est certainement subrécursif: une autre approche est de définir des formalismes subrécursifs de complexité appropriée pour les différents types de traitement. C'est ce qui a été initialement tenté au GETA, où les différents modèles algorithmiques ATEF, ROBRA, TRANSF, et SYGMOR ont été conçus pour être décidables et de complexité linéaire.

D'ailleurs, la décidabilité est une contrainte essentielle quand on veut écrire de gros systèmes à plusieurs personnes, et les faire tourner sur des milliers ou des millions de mots. Cependant, imposer certaines contraintes destinées à garantir l'arrêt du système dans tous les cas de figure peut conduire à écrire certaines choses de manière trop lourde alors qu'on prourrait les écrire plus simplement dans un formalisme plus général, quitte alors à prouver la décidabilité cas par cas. D'où l'idée d'ailleurs implémentée dans les ATN, dans ATEF et dans ROBRA, de construire les modèles algorithmiques sous-jacents comme des extensions de modèles décidables, de telle façon qu'on puisse détecter les sources d'indécidabilité de façon statique (à la compilation) et dans ces cas, rechercher des preuves particulières de décidabilité.

Enfin, ces modèles algorithmiques doivent être adéquats, c'est-à-dire qu'il doivent permettre d'écrire des processus linguistiques de façon aisée et concise. Pour concrétiser, reprenons un autre principe de [5]

> Les opérateurs complexes associés aux types de données de base doivent être adéquats et leur complexité bornée (linéaire si possible). La structure de contrôle de leurs appels doit être fondée sur un modèle décidable, de sorte qu'on puisse localiser statiquement les éventuelles sources d'indécidabilité.

II. MODELES DE BASE

1. SYSTEMES-Q.

Je reprends le resumé de Colmerauer [13] concernant les systèmes-Q:

"Un système-Q est un ensemble de règles permettant de faire certaines transformations sur de graphes orientés. Chaque flèche d'un tel graphe est surmontée d'une expression parenthésée. Les transformations peuvent correspondre à une analyse, à une synthèse de phrase, ou à une manipulation formelle de ce genre".

Parmi les originalités des systèmes-Q, on compte,

La Reversibilité. Sous certaines conditions (en particulier: mêmes variables à droite et à gauche mais aussi, pas de partie droite illicite en partie gauche, ex. A*(U*)== A*+U*.) le même système-Q peut être utilisé pour décrire une transformation et la transformation inverse.

L'Enchaînement. Plusieurs systèmes-Q peuvent s'enchaîner les uns à la suite des autres, chacun prenant comme données les résultats du précédent.

Voici maintenant une présentation du formalisme des systèmes-Q.

1.1. Notions fondamentales.

- Caractère. C'est la plus petite unité d'information. Elle peut consister en une lettre, un chiffre, un signe significatif ou un signe non-significatif.
- Etiquette. C'est une suite de caractères dont le premier peut être quelconque mais les suivants doivent être des chiffres ou des lettres.
- Arbre. Elément de base formé d'étiquettes, de parenthèses et de virgules. L'expression parenthésée résultante correspond à un arbre parcouru en préordre. Enfin, une étiquette est aussi un arbre.
- Liste. Une suite d'arbres séparés par des virgules est une liste. Un arbre est une liste.

- Paramètre. Un paramètre est formé d'une lettre, suivie d'un astérisque, éventuellement suivi d'un chiffre. Selon la première lettre, les paramètres représentent des étiquettes, des arbres ou des listes.

- Arbre paramétré. Comme son nom l'indique, c'est un arbre qui comporte des paramètres. Il désigne la classe d'arbres que l'on peut obtenir en substituant à chacun de ses paramètres, conformément à son type, une étiquette, un arbre ou une liste quelconque.

- Relations entre des listes. Les règles des systèmes-Q utilisent des relations binaires d'égalité, d'inégalité, d'inclusion et d'exclusion.

- Chaîne. Une chaîne est une suite non-vide d'arbres séparés par le symbole +.

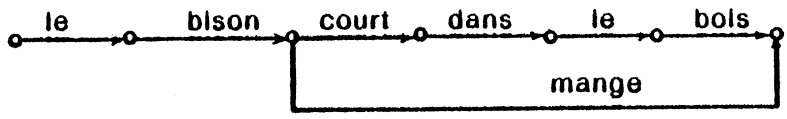
- Graphe de chaînes.

Il est constitué d'un ensemble de flèches (chacune étiquetée par un arbre) qui relient un ensemble de sommets entre eux, et qui satisfait aux deux conditions:

1. Il ne contient pas de circuit.
2. Il ne peut exister qu'une entrée et une sortie.

Pour représenter un graphe de chaînes, on associe à chaque sommet un entier naturel et à chaque flèche le triplet constitué de l'arbre qui l'étiquette encadré par les sommets origine et arrivée.

On permet, pour abréger, l'écriture consistant à ne numéroter que les sommets "carrefour" et à regrouper dans une chaîne les arbres des "chemins sans carrefour". Par exemple, le graphe



aura les deux écritures,

écriture canonique

-1- le -2-
-2- bison -3-
-3- court -4-
-3- mange-7-
-4- dans -5-
-5- le -6-
-6- bois -7-

écriture abrégée

-1- le + bison -2-
-2- court dans le bois -3-
-2- mange -3-

N.B. Les sommets n'ont une numérotation fixe que dans une écriture donnée.

1.2. Systèmes-Q généraux.

Un système-Q général est constitué d'un ensemble de règles de la forme,

<membre gauche> == <membre droit> [/<condition>].

où les membres gauche et droit sont des chaînes d'arbres paramétrés. Le double signe égal se lit "se réécrit en" et le point marque la fin de la règle. La condition peut éventuellement être vide.

Un système-Q définit un transducteur graphe de chaînes -> graphe de chaînes de la puissance d'une machine de Turing, donc indécidable.

Etant donné un graphe de chaînes, il sera transformé en un autre, en deux étapes. La première consiste à ajouter au graphe, en parallèle, la chaîne du membre droit chaque fois que celle du membre gauche se présente dans le graphe et que la condition est vérifiée. Les nouvelles flèches ainsi ajoutées peuvent aussitôt servir à de nouvelles additions si elles font partie d'une chaîne correspondant à un membre gauche d'une règle. L'application d'une règle ne se fait qu'une fois sur le même chemin et le processus s'arrête lorsqu'il n'y a plus de règles à appliquer. Si le processus converge, le résultat est indépendant de l'ordre dans lequel on a appliqué les règles, il est indéfini autrement.

La seconde étape, après convergence de la première, consiste à supprimer d'abord les flèches figurant dans des chemins sur lesquels s'est appliquée au moins une règle, puis celles qui ne sont pas dans un chemin allant de l'entrée à la sortie du graphe.

2. A R I A N E - 7 8.

Le système ARIANE-78 constitue un système de traduction automatisée multilingue de **deuxième génération** [1,23] aidé par l'homme (TAAH), au sens où la traduction purement automatique, peut être suivie d'une révision humaine.

Il offre aux linguistes un environnement interactif ainsi que des meta-langages spécialisés avec lesquels ils écrivent les données et procédures linguistiques (essentiellement dictionnaires et grammaires) utilisées pour construire des systèmes de traduction.

L'unité de traduction n'est pas la phrase, mais plutôt un ou plusieurs paragraphes, de telle sorte que le contexte utilisable, par exemple pour résoudre les anaphores, est plus vaste que dans d'autres systèmes de seconde génération.

2.1. DESCRIPTION GENERALE DU PROCESSUS DE TRADUCTION.

Le processus. Le processus de traduction d'un texte depuis une langue source vers une langue cible est divisé en trois phases séquentielles: analyse, transfert et génération.

La sortie de l'analyse est un descripteur structural du texte d'entrée. Il est transformé en un descripteur structural équivalent dans la langue cible par la phase de transfert, puis ce dernier descripteur est transformé en texte de sortie par la phase de génération.

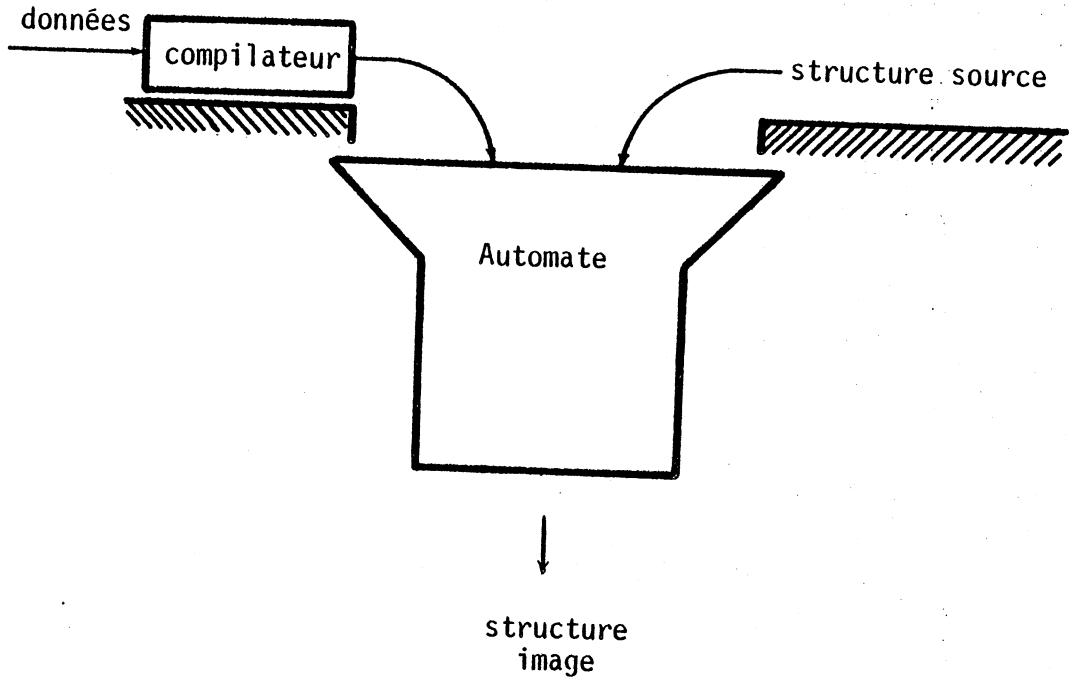
La structure de données. Pour représenter l'unité de traduction depuis l'analyse morphologique jusqu'à la génération morphologique, ARIANE-78 utilise un type unique de structure de données: une structure arborescente étiquetée complexe. Chaque noeud de cet arbre porte un enregistrement (masque) de 32 octets, donnant une valeur pour chacune des variables grammaticales utilisées dans l'étape courante.

Organisation linguistique générale. A chaque étape, les données linguistiques peuvent être de quatre sortes:

1. Les variables grammaticales.
2. Les formats ou classes. Ce sont des masques (ou combinaison de valeurs de variables) constants auxquels on a donné des noms, et que l'on peut utiliser comme références dans les dictionnaires et les grammaires.
3. Les dictionnaires.
4. Les grammaires. Celles-ci contiennent les règles et la stratégie à suivre.

Ces données sont exprimées par un meta-langage, leur syntaxe et leur cohérence sont d'abord vérifiées par le compilateur correspondant qui génère alors un code intermédiaire.

A l'exécution, ce code est utilisé par l'automate pour transformer un ensemble de structures en un autre. Voici un schéma du processus général.



2.2. MODELES INFORMATIQUES.

Les composants algorithmiques auxquels je m'intéresse sont: ATEF, ROBRA, TRANSF et SYGMOR.

2.2.1. ATEF.

Le composant ATEF (Analyse de Textes en Etats Finis) est fondé sur un modèle de transduction d'états finis non-déterministe.

1. Données.

Elles comprennent,

- Des déclarations de variables et de formats.
- Des dictionnaires. Un maximum de six et un dictionnaire de tournures. Chaque dictionnaire contient des articles qui sont composés d'un segment (ou morphe) comme entrée, ainsi que d'informations concernant l'utilisation des valeurs des variables (informations le plus souvent condensées dans deux formats), et éventuellement d'une valeur de la variable spéciale UL (unité lexicale).

- Une grammaire. Chaque règle comprend une liste de "formats d'appel", une liste de conditions et une liste d'actions.

2. L'automate.

ATEF analyse successivement chaque mot du texte, en examinant à priori toutes les possibilités. Chaque étape d'une analyse particulière consiste à découper un segment dans "ce qui reste" de la "forme" à analyser, et à appliquer l'une des règles appelée par le format "morphologique" associé à ce segment dans l'article correspondant trouvé dans le dictionnaire.

Les conditions peuvent concerner les résultats de l'analyse d'un contexte (borné au quatre "formes" précédentes), les chaînes accessibles et les résultats partiels stockés par cette analyse.

Il existe trois types d'action.

1. Affectation de valeurs au masque "C" défini par le système. Ce masque contient le résultat courant de l'analyse.

2. Transformation de la partie de la "forme" non encore segmentée.

3. Appel de fonctions spéciales qui permettent de:

- Stocker le résultat courant (utile pour le traitement des mots composés).
- Créer de nouvelles unités lexicales à partir de la forme en cours.
- Décider qu'une borne de phrase est atteinte.

Il existe un mode interactif dans lequel le système s'arrête quant il rencontre un mot inconnu. Il demande dans ce cas à l'utilisateur s'il veut changer l'entrée ou indexer de nouveaux articles dans les dictionnaires. La sortie d'ATEF est une structure arborescente étiquetée où chaque sommet porte un masque de variables.

2.2.2. ROBRA

ROBRA [8], successeur de CETA [10,11], est un langage permettant d'écrire des transducteurs arbre --> arbre [4] et qui peut être considéré comme un langage algorithmique de très haut niveau.

1. Données.

Elles comprennent essentiellement,

- Les variables.
- Les formats.
- Le graphe de contrôle et l'ensemble de règles de transformation.

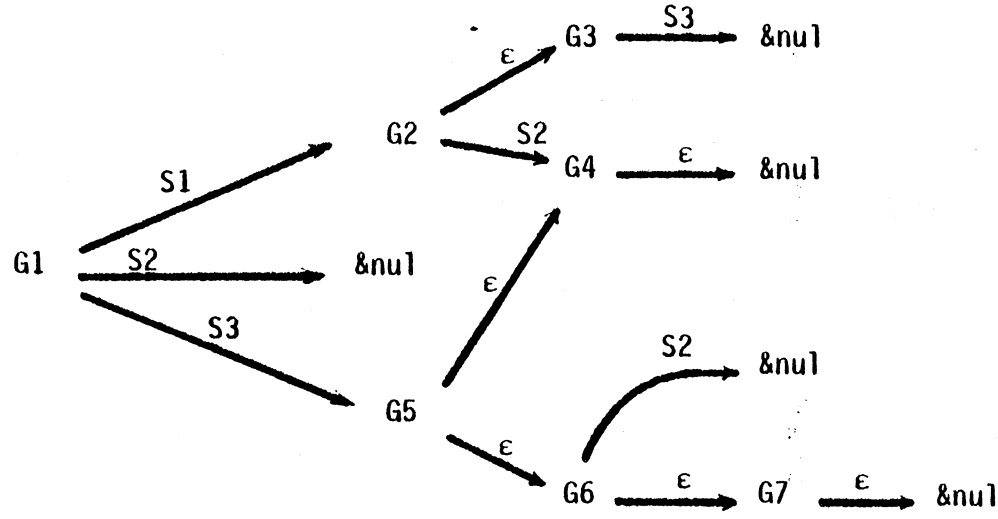
2. L'automate.

Je reprends ici la description faite en [8] avec quelques modifications. Comme on l'a dit plus haut, le langage ROBRA permet d'écrire des transducteurs arbre --> arbre. Un tel transducteur est structuré en système transformationnel (ST) qui opère sur une arborescence objet A. Un ST est un graphe "de contrôle" dont les noeuds portent des grammaires transformationnelles (GT) et les arcs des conditions. Une GT, à son tour, est un ensemble ordonné de règles transformationnelles (RT). Une règle transformationnelle Ri s'écrit Ri= (Si,Ti) où Si, la partie gauche, est un schéma de sous-arborescence et où la partie Ti définit la transformation associée. Ti s'écrit enfin Ti= (Ii,Xi) où Ii est la sous-arborescence image remplaçant Si et où Xi décrit le transfert dans B (arbre transformé) des sommets dépendants de Si dans A-Si.

Un ST est un graphe orienté, sans cycle, à une seule entrée où:

- Chaque noeud porte une GT, ou le symbole de sortie "&nul".
- Tout noeud portant "&nul" est sans successeur.
- Chaque arbre porte un schéma de transition S ou bien ϵ (schéma vide).
- Les arcs issus d'un même noeud sont ordonnés.

Exemple.



Le ST fonctionne de manière heuristique en cherchant, pour un arbre d'entrée A0 donné, le premier chemin conduisant à &nul. Ceci est réalisé au moyen d'un mécanisme de backtrack. Par exemple, si le schéma S1 apparaît dans G1(A0)= A1 on calculera G2(G1(A0))= A2, puis G3(G2(G1(A0)))=A3. Si S3 apparaît dans A3, le système s'arrête et produit A3. Sinon, il revient en arrière, à A2. Si S2 apparaît dans A2, on produit G4(A2). Sinon, on revient à A1, etc.

L'application d'une GT consiste en une (mode "unitaire") ou plusieurs (mode "exhaustif") application(s) élémentaire(s). Pour une application élémentaire d'une GT, GT= R1,R2,...,Rn, le système recherche toutes les occurrences indépendantes des Si, 1<=i<=n. S'il y a conflit entre Si et Sk, i<k, la priorité est donnée à Si. On obtient ainsi un ensemble de transformations enracinées sur les points Pj. En mode C (Coupe) les Pj sont sur une coupe de A. En mode T (Total) les transformations ne sont pas nécessairement enracinées sur une coupe. On ne donnera pas ici de détails concernant les règles.

Quant à la décidabilité du modèle, ni ROBRA, ni CETA ne sont décidables que sous certaines conditions [4].

Enfin, remarquez que pour ROBRA, le fait de travailler sur un seul arbre objet et non sur une collection (comme en PROLOG ou en systèmes-Q) provient de l'expérience linguistique: au lieu de séparer toutes les homophrases ambiguës, qu'on peut ensuite difficilement comparer, on cherche au contraire à garder les ambiguïtés dans une même structure de travail.

2.2.3. TRANSF

Ce modèle est essentiellement fondé sur la consultation d'un dictionnaire dont les articles sont des règles.

1. Données.

Du point de vue algorithmique, le modèle est constitué d'un dictionnaire bilingue de règles simples de transfert, accessibles à partir de l'UL source. Chaque règle est une suite de triplets,

‹condition› / ‹sous-arbre image› / ‹affectation› ,

la dernière condition étant vide (nil).

2. L'automate.

Il parcourt l'arbre d'entrée en préordre et fabrique l'arbre de sortie comme suit:

- On accède au dictionnaire par l'UL du sommet courant.
- On choisit le premier triplet de l'article dont la condition est vérifiée.
- Le sous-arbre image (qui est en général dégénéré) est ajouté à la sortie avec les valeurs de variables calculées par la partie affectation. Ainsi, l'arbre de sortie est très semblable à l'arbre d'entrée.

La possibilité de transformer un sommet d'entrée en un sous-arbre de sortie permet la création de mots composés ou de sommets auxiliaires qui seront utilisés dans l'étape du transfert de structure en vue du traitement de tournures idiomatiques.

2.2.4. SYGMOR [22].

Ce modèle provient de la composition de deux transducteurs. Le premier, un transducteur arbre --> chaîne, fournit le mot des feuilles de l'arbre objet; le second transforme la chaîne (composée de masques de variables) en une chaîne de caractères sous le contrôle des données linguistiques.

1. Données.

Elles sont composées de:

- Déclaration de variables, de formats et de procédures, de conditions et d'affectations.
- Dictionnaire (huit au maximum).
- La grammaire.

Chaque article du dictionnaire donne une liste de triplets.

‹condition› / ‹affectation› / ‹chaîne›

le dernier ayant une condition vide (nil).

Une règle de grammaire est constituée par:

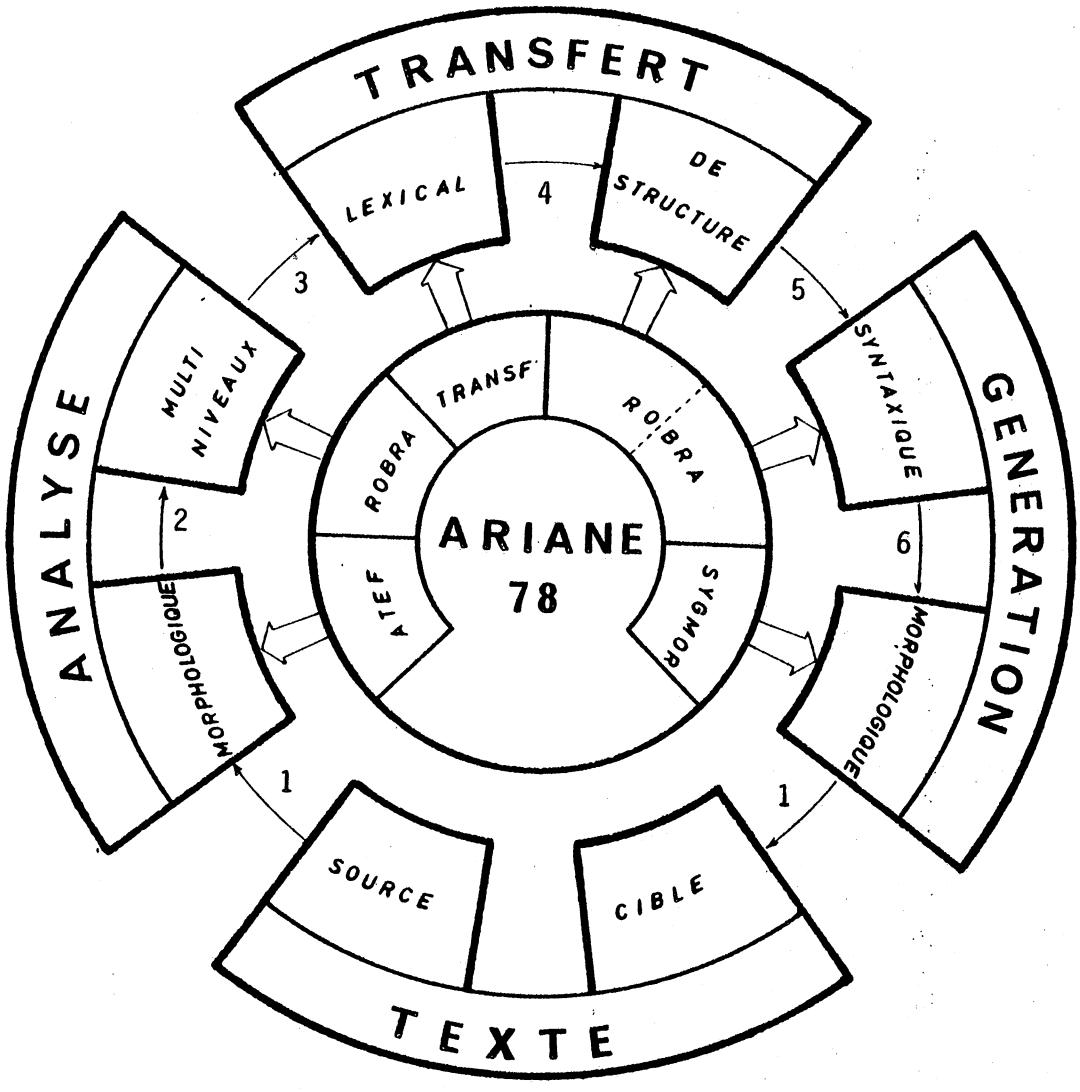
- Une condition d'application sur les masques et les chaînes accessibles.
- Une partie pour la consultation d'un ou plusieurs dictionnaires et pour la manipulation de chaînes accessibles.
- Une partie affectation.
- Une partie de transformation de chaîne permettant de réaliser des substitutions de sous-chaînes.
- Enfin, une partie où l'on indique la suite des transitions à suivre après l'application de la règle.

2. L'automate.

Pour le traitement d'un masque, SYGMOR cherche la première règle applicable (au moins une doit avoir une condition vide), l'applique, et suit les transitions indiquées, à moins qu'on ne trouve une règle obligatoire non-applicable. Dans ce cas, le système exécute la règle particulière "MOTINC" (mot inconnu) ou une action par défaut si cette règle est absente.

Il faut remarquer que, contrairement à ATEF, SYGMOR réalise un automate d'états finis **déterministe**.

2.2.5. SCHEMA GENERAL.



1 - Chaîne de caractères.
2 - Arborescence étiquetée.
3 - Structure intermédiaire source.
4 - Structure source avec UL cibles.
5 - Structure intermédiaire cible.
6 - Représentation de surface du texte cible.

3. EUROTRA: UN PROJET ACTUEL A GRANDE ECHELLE [24].

3.1. MOTIVATIONS.

Avec 6 langues et avec la probabilité d'avoir jusqu'à 9 langues, les institutions de la Communauté Européenne doivent faire face à un besoin énorme de traduction: d'où l'appel à toutes sortes d'outils informatiques pour la traduction (depuis le plus simple dictionnaire automatique jusqu'à la traduction entièrement automatisée).

Etant donné qu'il n'y avait aucun système commercial adéquat pour une traduction multilingue, la Communauté a lancé en 1978 un projet européen de traduction: EUROTRA.

Le but du projet est de développer un système de traduction multilingue, aidé par l'homme (TAAH). Le système devra être capable de traiter toutes les langues qui sont maintenant présentes dans la Communauté et de pouvoir s'étendre à d'autres langues. D'où les exigences suivantes:

- Fournir une traduction automatisée de haute qualité de façon pratique et à un prix raisonnable.
- Etre extensible à des domaines nouveaux.
- Etre continuellement améliorable.
- Constituer un outil de recherche et d'enseignement.
- Etre portable et fortement modulaire pour permettre une souplesse d'utilisation.

3.2. FONCTIONNEMENT DU SYSTEME.

Le processus de traduction est décomposé en trois parties principales: analyse, transfert et génération. Tout le long de ces trois phases, les aspects linguistiques et computationnels du système sont considérés comme étant pratiquement et conceptuellement différents, afin de permettre l'amélioration et le développement des aspects linguistiques du système, tout en retenant le cadre du logiciel comme un outil de base. De cette façon de nouveaux couples de langues peuvent être ajoutés avec un minimum de perturbations du système de même que l'on peut continuer à raffiner les couples que l'on traite à ce moment-là.

Tout le système utilise une et une seule structure de données: les E-graphes. Des cas particuliers de la structure de données peuvent être utilisés à des niveaux particuliers du système mais ils seront exprimés à tout niveau avec le même formalisme.

Toute la phase d'analyse a comme but la production d'une représentation intermédiaire de l'entrée du texte en langue source sous forme d'un ensemble d'arbres avec plusieurs niveaux d'étiquetage pour tenir compte de l'information morphologique, syntaxique et logico-sémantique qui a été déterminée pendant la phase d'analyse.

L'étape de transfert prend comme entrée les arbres étiquetés et réalise essentiellement deux types d'opérations sur eux. Les unités lexicales de la langue source sont transformées en unités lexicales de la langue cible et la forme structurale du texte d'entrée est transformée en une forme structurale appropriée pour la langue cible. La sortie du transfert est aussi un autre ensemble d'arbres étiquetés, encore une fois avec plusieurs niveaux d'étiquetage, mais représentant cette fois la structure sous-jacente du texte cible.

La phase de génération prend cette structure sous-jacente et produit la sortie finale, donnant une version traduite du texte d'entrée.

Toutes les parties du système ont accès aux dictionnaires. Pendant l'analyse et la génération les dictionnaires sont monolingues, pendant le transfert, ils sont bilingues. Les dictionnaires utilisés contiendront de l'information morphologique, syntaxique, sémantique et statistique.

Le système doit être conçu comme faisant partie d'un système "intégré" de traitement de texte. Ainsi, des modules particuliers peuvent être utilisés à des fins autres que la traduction automatisée. Aussi un tel système fournira-t-il plusieurs niveaux de traduction et de choix de connexion de modules, définis par l'utilisateur. Ceci veut dire que l'utilisateur sera capable de décider le niveau de traduction dont il a besoin ou l'utilisation de modules particuliers pour ses propres fins. Enfin, le système permettra l'utilisation "on-line" interactive et le traitement par lots (batch processing).

III. DISCUSSION SUR QUELQUES STRUCTURES DE DONNEES.

Maintenant qu'on a fait le tour des idées génératrices, regardons brièvement l'évolution d'outils informatiques et les structures utilisées comme "descripteurs structurels".

Les premiers outils, les "analyseurs syntaxiques" des années 60, présentés sur un modèle hors-contexte, un modèle de dépendance ou bien une autre classe de grammaires formelles sont considérés peu à peu comme périmés et d'autres outils, comme les transducteurs sont perçus comme plus adéquats pour l'écriture de modèles linguistiques calculables.

En ce qui concerne les structures utilisées comme "descripteurs structurels", le développement des grammaires formelles, de la théorie des automates, de la linguistique formalisée et des compilateurs de langages de programmation de haut niveau a centré tous les efforts sur un descripteur structurel représenté par un arbre étiqueté. En effet, les grammaires de constituants (hors-contexte et hors-contexte augmenté) et les grammaires de dépendance conduisent à des structures syntaxiques de ce dernier type, même si les niveaux d'interprétation ne sont pas les mêmes. Ainsi, des arbres qui reflètent l'organisation syntaxique de la phrase ont été considérés comme de descripteurs structuraux adéquats.

L'approche du GETA, fondée sur l'expérience passé, le travail réalisé à d'autres endroits pour des raisons d'échanges scientifiques, a choisi de fusionner plusieurs niveaux d'interprétation sur une seule structure (i.e. sur un seul arbre étiqueté). Il y a plusieurs raisons pour suivre cette approche. D'abord, une stratégie séquentielle pour obtenir le niveau désiré d'interprétation le plus profond force le système à porter toutes les ambiguïtés à chaque niveau, en attendant la solution au niveau suivant. Aussi, si une structure unique peut porter différents niveaux d'interprétation au moyen d'étiquettes appropriées, alors une stratégie parallèle peut être adoptée pour toute transduction de cette arborescence.

Or, l'utilisation d'arborescences comme descripteurs présente des inconvénients (dans la référence aux anaphores, par exemple). En effet, on ressent souvent (cf. [17]) la nécessité d'une structure de graphe.

Cette structure ne saurait être un graphe sans contrainte, étant donné que l'on cherche à long terme l'efficacité, la rapidité et le moindre coût dans le traitement de textes comportant un grand nombre d'occurrences (pour donner une idée du nombre d'occurrences manipulables, le bilan effectué dans [23], réalisé sur quatre textes différents, comptait environ 15.000 mots). D'autre part, on veut éviter des structures bouclées où la moindre erreur peut rendre un algorithme indécidable, d'où l'idée d'une généralisation des systèmes-Q et la définition schématique d'un E-graphe: un graphe sans circuit, avec un sommet d'entrée, un sommet de sortie et des arbres sur les arcs.

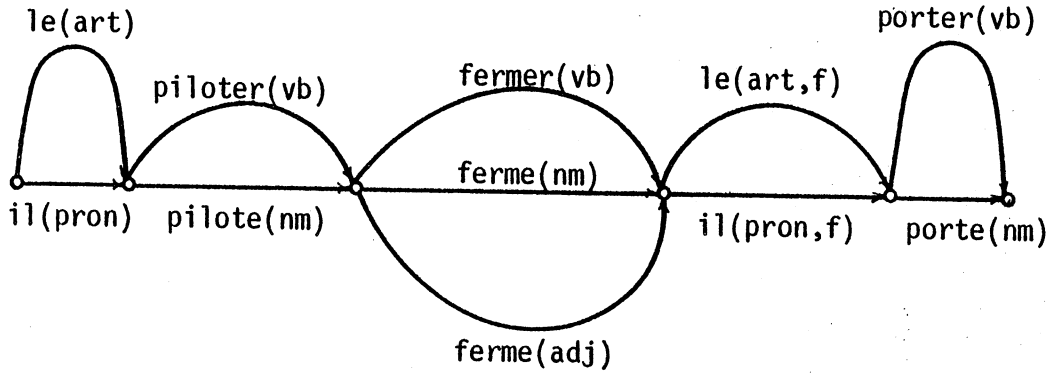
Cette structure est utilisée [26] implicitement par les analyseurs morphologiques de SUSY et du GETA (avec des arbres dégénérés sur les arcs) et explicitement par REZO et les systèmes-Q. Elle se prête bien à la représentation des ambiguïtés, soit

- avec des arcs et de chemins, utilisant des arbres peu profonds.
- avec des arbres, utilisant des arbres plus profonds et moins d'arcs.

Afin d'illustrer la façon de représenter les ambiguïtés prenons une simplification de l'exemple donné en [26] et considérons la phrase,

"Le pilote ferme la porte"

On pourrait exprimer les ambiguïtés qu'elle présente au moyen du E-graphe suivant



IV. L E S E - G R A P H E S.

Cette partie formalise les notions un peu intuitives données plus haut, ainsi que d'autres notions qui deviendront la base de la définition pratique.

1. DEFINITION FORMELLE

Un E-graphe G est intuitivement un réseau [19], c'est-à-dire un multigraphe sans sommet isolé, sans circuit et avec exactement un sommet d'entrée et un sommet de sortie. Avant d'aborder la définition formelle, il est nécessaire d'introduire certaines notions qui nous serviront dans la suite. Quant à la notation utilisée, je reprends celle due à [1] avec de légères modifications.

1.1. ELEMENT DE RESEAU.

Définition.

Un élément de réseau (ER) est une fonction ρ de N_+ dans N_+. Un ER est fini ssi Dom ρ l'est. Notons $\Pi 1$ et $\Pi 2$ les projections canoniques de N_+ sur N_+. On définit alors:

σ	$= \Pi 1 \rho$	la fonction source
τ	$= \Pi 2 \rho$	la fonction but
$A(\rho)$	$=$ Dom ρ	l'ensemble des arcs
$S(\rho)$	$= \Pi 1(\text{Im } \rho) \cup \Pi 2(\text{Im } \rho)$	l'ensemble des sommets

Si $\alpha \in A(\rho)$, on dira que l'arc α relie $\sigma(\alpha)$ à $\tau(\alpha)$. Inversement, on peut définir des applications de N_+ dans $\wp(N_+)$

$\underline{\sigma}(x)$	$= \{\alpha \mid \sigma(\alpha) = x\}$	l'ensemble des arcs issus du sommet x
$\underline{\tau}(x)$	$= \{\alpha \mid \tau(\alpha) = x\}$	l'ensemble des arcs de but x

On définit enfin les <u>prédécesseurs</u> et les <u>successeurs</u> d'un sommet $x \in S(\rho)$ respectivement comme

$$\hat{\sigma}(x) = \sigma\underline{\tau}(x)$$
$$\hat{\tau}(x) = \tau\underline{\sigma}(x)$$

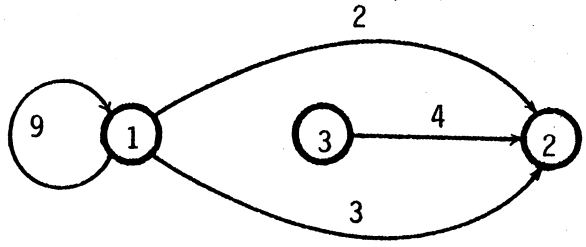
L'ensemble de sommets réliés à x sera noté

$$\Gamma(x) = \hat{\sigma}(x) \cup \hat{\tau}(x)$$

On notera $\hat{\sigma}^*$ et $\hat{\tau}^*$ les extensions transitives de $\hat{\sigma}$ et $\hat{\tau}$.

Il est évidemment possible de représenter un ER fini par un ensemble de triplets de la forme $(\alpha, \rho(\alpha))$. Exemple

$$\rho = \begin{cases} (2,1,2) \\ (3,1,2) \\ (4,3,2) \\ (9,1,1) \end{cases}$$



1.2. NOEUDS PARTICULIERS, CHEMINS.

On utilisera indifféremment les termes "sommet" et "noeud". Définissons-en quatre ensembles particuliers.

Définition.

Les sommets <u>initiaux</u> (resp. finaux) sont ceux qui n'ont pas de prédécesseur (resp. de successeur). Les sommets <u>origines</u> (resp. extrémités) sont ceux qui n'ont pas d'autre prédécesseur (resp. successeur) qu'eux mêmes.

On aura pour ces sommets les notations.

$$I(\rho) = \{x \mid \hat{\sigma}(x) = 0\} \qquad O(\rho) = \{x \mid \hat{\sigma}(x) \subset \{x\}\}$$
$$F(\rho) = \{x \mid \hat{\tau}(x) = 0\} \qquad E(\rho) = \{x \mid \hat{\tau}(x) \subset \{x\}\}$$

Définition.

Un <u>chemin</u> γ d'un ER ρ est une application d'un segment initial de N_+ dans $A(\rho)$, c'est-à-dire une suite finie ou infinie d'arcs telle que:

$$(\forall i \in \mathrm{Dom}\ \gamma - \{0\})[\tau\gamma(i-1) = \sigma\gamma(i)]$$

On notera $C(\rho)$ l'ensemble des chemins de ρ. Le <u>segment recouvert</u> par γ est la suite des sommets rencontrés, soit $\hat{\gamma}(N)$, avec

$$\hat{\gamma}(0) = \sigma\gamma(0) \text{ et } \hat{\gamma}(x+1) = \tau\gamma(x)$$

L'ensemble des chemins reliant deux sommets x et y sera noté: $\bar{\gamma}(x.y)$. On a donc

$$\bar{\gamma}(x.y) = \bigcup_{n\in N} \{\gamma \mid \text{Dom } \gamma=[0.n] \ \&\ \sigma\gamma(0)=x \ \&\ \tau\gamma(n)=y\}$$

Un chemin γ est <u>simple</u> s'il est injectif. Un chemin γ est <u>élémentaire</u> si $\hat{\gamma}$ est injectif. Un <u>circuit</u> est un chemin γ fini d'origine et d'extrémité identiques: $\sigma\gamma(0) = \tau\gamma(n)$.

Une <u>boucle</u> est un arc de source et de but identiques. C'est encore un circuit à un élément. Une <u>chaîne</u> désignera un chemin simple ou élémentaire.

1.3. RESEAUX.

En reprenant l'exemple du 1.1, on voit bien que la définition "par triplets" doit être définie à l'ordre et à la numérotation près.

On peut parfois définir une fonction f de F^E dans $F'^{E'}$ en utilisant deux applications f1:E -> E' et f2:F -> F' (mais on ne peut définir tous les f de cette façon). Dans ce cas, on a

$$(f(\alpha))\,(f1(e)) = f2(\alpha(e))$$

pour $e\in E$ et $\alpha\in F^E$. Ceci nous conduit à la

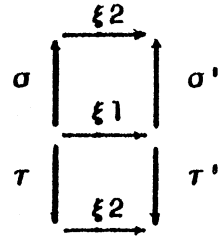
Définition.

Deux ER ρ et ρ' sont isomorphes s'il existe un couple $(\xi 1, \xi 2)$ de bijections de N sur N tel que

(1) Dom $\rho' = \xi 1(\text{Dom } \rho)$
(2) $\xi 2\sigma = \sigma'\xi 1$ & $\xi 2\tau = \tau'\xi 1$

Si de plus $\xi 1$ et $\xi 2$ sont récursives, on dira que ρ et ρ' sont récursivement isomorphes. Si $\xi 1$ est croissante, l'isomorphie est ordonnée.

La propriété (2) signifie que l'image de la source (du but) est la source (le but) de l'image, ou encore que le diagramme suivant est commutatif



Il est évident que l'isomorphie et l'isomorphie récursive sont des relations d'équivalence sur l'ensemble des ER.

Définition.

Un réseau est une classe d'isomorphie récursive d'éléments de réseau. Un réseu ordonné est une classe d'isomorphie récursive ordonnée d'ER.

Toutes les proprietés et définitions introduites pour les ER se transposent immédiatement aux réseaux. En particulier, les chemins, les circuits, les sommets initiaux, finals, ... sont conservés par isomorphie. Ceci provient de ce que $\xi 1$ et $\xi 2$ sont des bijections. Introduisons enfin la notion d'étiquetage.

Définition.

Un ER étiqueté sur E et W (E pour les arcs, W pour les sommets) est un couple (ρ, ϵ) où:

(1) ρ est un ER
(2) $\epsilon: A(\rho) \oplus S(\rho) \longrightarrow E \oplus W$ est une fonction définie par

- **l'étiquetage des arcs $\quad \epsilon 1: A(\rho) \rightarrow E$**
- **l'étiquetage des sommets $\quad \epsilon 2: S(\rho) \rightarrow W$.**

Cette définition permet de considérer une seule fonction d'étiquetage ϵ pour les arcs et pour les sommets.

Définition.

Deux ER ρ et ρ' étiquetés sont isomorphes s'ils sont isomorphes en tant que ER et s'il existe un couple d'isomorphie $(\xi 1, \xi 2)$ tel que

$$\text{(3)} \quad \epsilon 1' \xi 1 = \epsilon \;\&\; \epsilon 2' \xi 2 = \epsilon 2$$

Ceci impose évidemment que les ensembles d'étiquettes soient identiques. D'autre part, tous les couples d'isomorphie vérifient (3) dès que l'un d'eux la vérifie. Il est clair que l'isomorphie d'ER étiquetés est une relation d'équivalence.

Definition.

Un <u>réseau étiqueté</u> sur (E,W) est une classe d'isomorphie récursive d'ER étiquetés sur (E,W). Un <u>réseau étiqueté</u> sur (E,W) <u>ordonné</u> est une classe d'isomorphie récursive ordonnée d'ER étiquetés sur (E,W).

1.4. GRAPHES DE CHAINES. E-ARBRES.

Définition.

Un graphe de chaînes est un réseau fini sans circuits comprenant exactement un sommet initial et un sommet final.

Définition.

Un arbre est un réseau sans circuits, comprenant un et un seul sommet initial et où chaque sommet est relié au sommet initial par un chemin élémentaire.

Un arbre décoré ou E-arbre est un arbre ordonné dont seuls les sommets sont étiquetés par: une étiquette et une décoration (qui est elle-même un arbre décoré) éventuellement vide.

Afin d'éviter la possibilité d'avoir des chaînes ou des décorations infinies, de même que des boucles (un arbre qui apparaît comme décoration d'un de ses noeuds), on définit proprement un E-arbre en reprenant la définition donnée en [26].

Définition.

Soit L un ensemble énumérable d'étiquettes, et soit T l'ensemble de tous les arbres ordonnés et étiquetés sur L. Alors l'ensemble E des E-arbres étiquetés sur L est construit comme suit.

(1) $T \subset E$

(2) Si $t = et_1(...et_n) \in T$ est un arbre à n noeuds et si $e_1.....e_n \in E$ sont des E-arbres, alors $e = et_1[e_1](...et_n[e_n])$ est aussi un E-arbre (sur L).

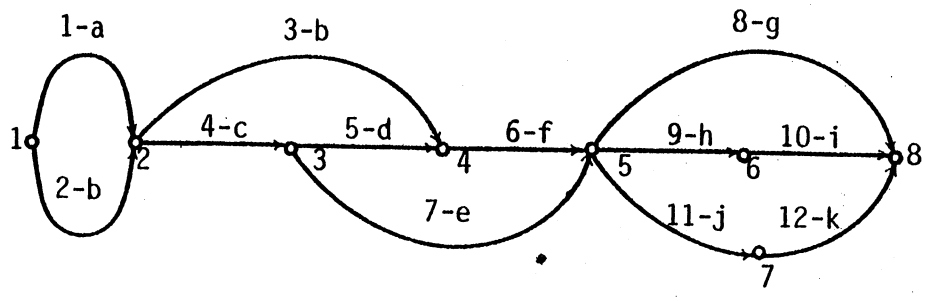
1.5. E-GRAPHE.

Définition.

Un E-graphe G est un réseau fini, sans circuits, comprenant exactement un sommet initial, un sommet final et où seuls les arcs sont étiquetés. Il est complètement déterminé par le 7-uplet

$$G = (A.S.\rho.I.O.E.\epsilon)$$

On utilisera indifféremment les termes "sommet initial" et "entrée", de même que "sommet final" et "sortie", avec les notations I et O respectivement. Voici un exemple de E-graphe.



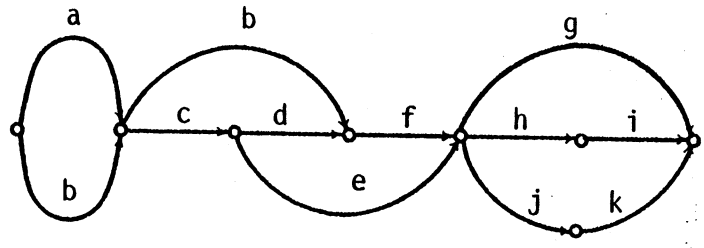
ρ= {(1,1,2),(2,1,2),(3,2,4),(4,2,3) ,(5,3,4) ,(6,4,5),
(7,3,5),(8,5,8),(9,5,6),(10,6,8),(11,5,7),(12,7,8)}

A(ρ)= {1,....,12}; S(ρ)= {1,....,8}; I(ρ)= {1}= I; F(ρ)= {8}= O

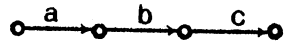
E= {a,b,c,d,e,f,g,h,i,j,k}

ϵ= {(1,a),(2,b),(3,b),(4,c) ,(5,d) ,(6,f),
(7,e),(8,g),(9,h),(10,i),(11,j),(12,k)}

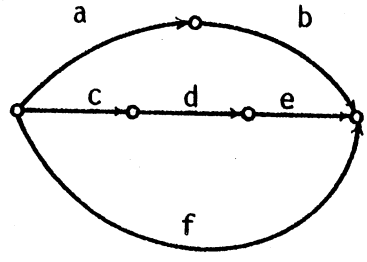
Pour des raisons de simplicité, on confondra dans la représentation de G les arcs et les E-arbres. Exemple



Il y a deux formes "géométriques" qui reçoivent un nom. Un E-graphe qui consiste en un seul chemin de l'entrée à la sortie est un E-graphe <u>séquentiel</u>. Exemple

G = 

Un E-graphe à n chemins élémentaires entre l'entrée et la sortie est un <u>faisceau</u>. Exemple

G = 

1.6. CLASSES DE E-GRAPHES.

Il y a trois classes intéressantes de E-graphes: les E-graphes simples, les E-graphes récursifs et les E-graphes réguliers.

1.6.1. E-graphes simples.

Définition.

Un E-graphe simple est un E-graphe où chaque élément de l'ensemble E d'étiquettes est un E-arbre.

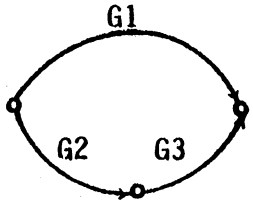
Les E-graphes donnés comme exemple au 1.5 sont des E-graphes de cette classe.

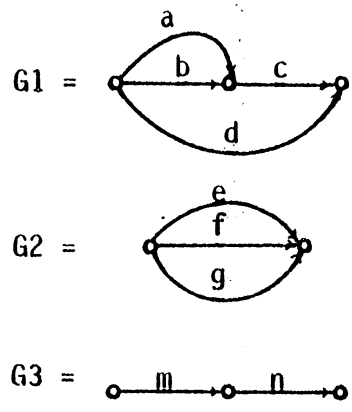
1.6.2. E-graphes récursifs.

Définition.

Un E-graphe récursif est un E-graphe où chaque élément de l'ensemble E est un E-graphe simple ou récursif.

Exemple.





1.6.2. E-graphes réguliers.

Soit k un ensemble fini d'étiquettes et soit le vocabulaire V= k U {(,),+} d'étiquettes et des trois symboles: parenthèse gauche, parenthèse droite et plus. Considérons maintenant le langage engendré sur V par la grammaire GR suivante

<e-graphe>::= <arbre> | <faisceau> | <seq>
<faisceau> ::= <seq> + <rfaisceau>
<rfaisceau> ::= <seq> + <rfaisceau> | <seq>
<seq> ::= <arbre> | <arbre><rseq> | (<faisceau>)<rseq>
<rseq> ::= <seq> | <vide>
<arbre> ::= <sommet> | <sommet>(<liste d'arbres>)
<liste d'arbres>::= <arbre> | <arbre>,<liste d'arbres>
<sommet> ::= <étiquette>
<étiquette> ::= <caractère> | <caractère><étiquette>
<caractère>::= k1 | k2 | ... | kp , si p=|k|

Une chaîne $\underline{d}$ de ce langage est par exemple

d= a((b+c+d)e+f)(g+h)

Maintenant, on va associer un E-graphe G à une chaîne $d= d_1 \ldots d_s$ du langage de la façon suivante (pour la chaîne d, on note $d_{kt}= d_k\, d_{k+1} \ldots d_t$ (k<=t))

- A chaque sous-chaîne d_{kt} (1<=k<=s)(1<=t<=s) ne contenant pas (,) ou +, on associe un chemin G' de G, formellement

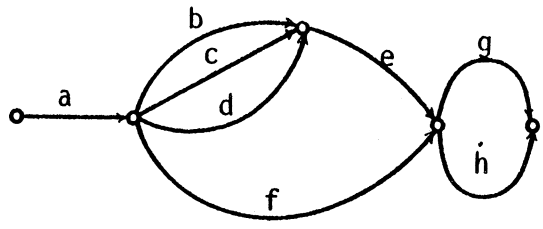
$d_{kt}^{(i)}$ associé à G'_i <=> $(\exists \gamma_i)[(\forall j)[\epsilon\gamma_i(j-k)= d_j]$
avec γ_i élémentaire, $l <= C(\rho)$, (k<=j<=t)

- Et pour chaque niveau de parenthésage, on associe un faisceau

$(\exists l)[(\forall p,q)[p<q \;\&\; d_p= (\;\&\; d_q=)] \;\&\; (\exists k)[d_{kl}= +]] \Rightarrow$
$(\exists \gamma_m)[[\sigma\gamma_1(0)= \ldots = \sigma\gamma_m(0)] \;\&\; [\tau\gamma_1(n)= \ldots = \tau\gamma_m(n)] \;\&\; m= l+1]$

avec γ_m élémentaire, (1<=p<=s), (1<=q<=s) et (p<k<q)

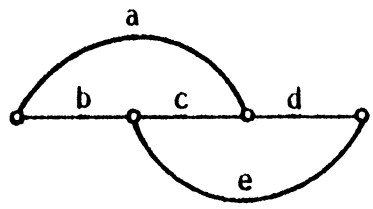
Voici une représentation graphique du E-graphe associé à la chaîne $\underline{d}$.



Définition.

Soit G un E-graphe. S'il existe une chaîne $\underline{d}$ du langage L(GR) que l'on peut associer à G, alors G est régulier. Autrement on dira que G est non-régulier.

Cette classe est malheureusement propre puisqu'il n'existe pas de chaîne pour tout E-graphe. Par exemple,



n'a pas de chaîne associée.

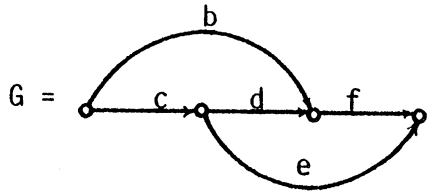
2. SOUS-STRUCTURES.

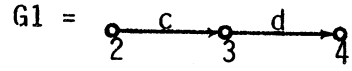
Il peut y avoir plusieurs types de sous-structures dans un E-graphe, mais pour des raisons d'homogénéité, on s'intéresse particulièrement à des sous-structures qui sont elles mêmes des E-graphes:les sous-E-graphes (seg).

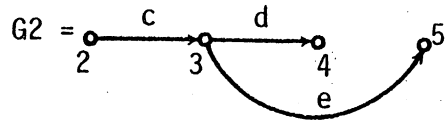
Définition.

Soient G' et G deux E-graphes. G' est un seg de G ssi $\rho' \subset \rho$ et $\epsilon' \subset \epsilon$.

Considérons le E-graphe G. G1 est un seg de G et G2 est un sous graphe de G mais non un sous-E-graphe.







3. ISOLABILITE.

L'isolabilité est une caractéristique qui détermine pour chaque E-graphe G, de classes de seg. Un seg **isolé** est intuitivement un seg qui n'a pas d'arcs qui "entrent dans" ou qui "sortent de" G. Pour la distinguer de ses sous-classes, on l'appellera isolabilité faible.

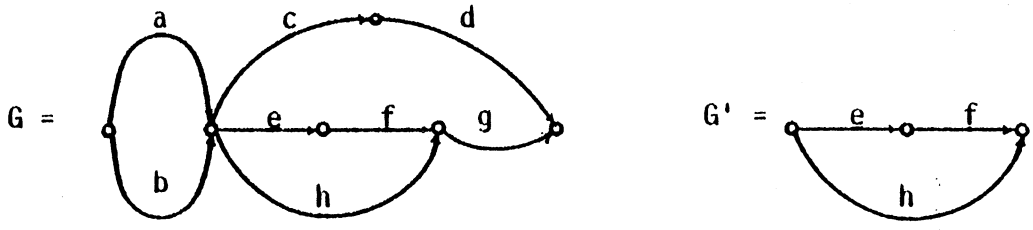
3.1. ISOLABILITE FAIBLE.

Définition.

Soit G un E-graphe. Un seg G' de G est isolable faiblement (segif), ssi tout arc α de G qui entre ou qui sort d'un sommet x autre que l'entrée I' ou la sortie O', est un arc de G'

$$G' \text{ segif de } G \iff [(\forall\alpha)[\sigma(\alpha)\in(S'-\{I'\}) \vee \tau(\alpha)\in(S'-\{O'\})] \Rightarrow \alpha\in A']$$

Exemple: soit G le E-graphe suivant, G' est un segif de G



3.2. ISOLABILITE MOYENNE.

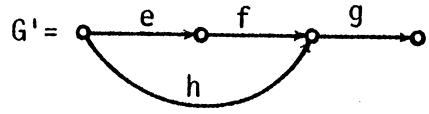
Une sous-classe des segif est la classe des seg qui ne laissent pas de "bouts pendants" lorsque l'on les efface du E-graphe principal.

Définition.

Soit G un E-graphe et G' un seg de G. G' est moyennement isolable (segmi), ssi G' est un segif de G et il existe au moins un chemin γ, n'appartenant pas à G, de source I' et de but O'.

$$G' \text{ segmi de } G \iff (G' \text{ segif de } G) \;\&\; (\exists\gamma)[\hat{\gamma}(0)= I' \;\&\; \hat{\gamma}(n)= O' \;\&\; \gamma\notin G']$$

Exemple: G' est un segmi de G.



3.3. ISOLABILITE FORTE.

Cette sous-classe comporte tout segif qui peut être considéré intuitivement comme une unité dans le E-graphe, i.e. tel que certaines fonctions puissent le considérer éventuellement comme un élément de la structure. Ceci est traduit par le fait que l'entrée (la sortie) du segif est le seul sommet où entrent (sortent) des arcs qui ne sont pas dans la sous-structure.

Définition.

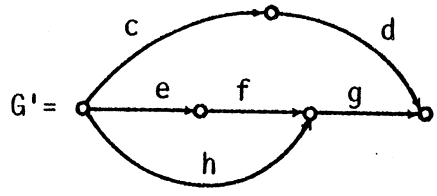
Soit G un E-graphe. Un seg G' de G est un seg fortement isolable (segfi) ssi

- **tout arc α, entrant dans S'-{I'}**
- **ou sortant de S'-{O'}**

appartient à A',

G' segfi de G $\Leftrightarrow$ $(\forall \alpha \in A)[\sigma(\alpha) \in (S'-\{O'\}) \vee \tau(\alpha) \in (S'-\{I'\})] \Rightarrow \alpha \in A']$

Exemple: G' est un segfi de G.

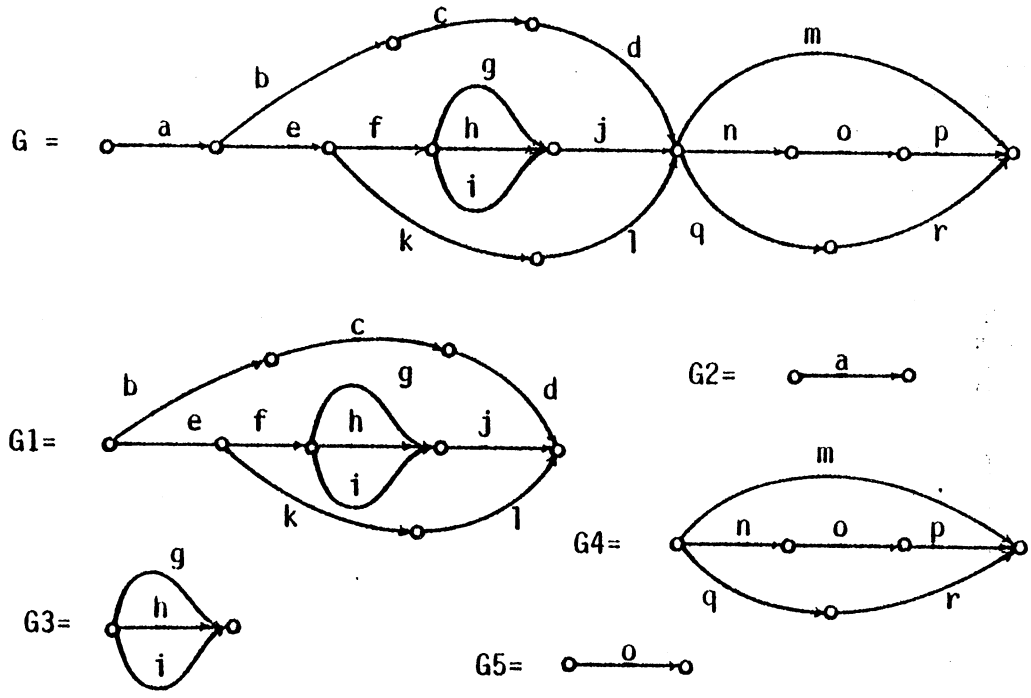


Une sous-classe intéressante de segfi est celle des segfi-élémentaires. Intuitivement, un segfi-élémentaire G' pourra être remplacé dans G par un arc étiqueté par G'.

Définition.

Soit G' un segfi de G. G' est un segfi-élémentaire de G ssi G' n'est pas un arc et soit il n'existe pas de segfi contenu strictement dans G', soit chaque segfi contenu strictement est un arc. Si G ne contient pas de segfi-élémentaire, on dira que G est élémentaire.

Exemple, soit le E-graphe G et les segfi suivants.



Seuls G3 et G4 sont des segfi-élémentaires.

Une sous-classe qui est "apparue" lors de l'implémentation et qui peut avoir un intérêt est la classe des segfi-externes.

Définition.

Soit G' un segfi de G. G' est un segfi-externe de G ssi il n'existe pas de segfi qui contient strictement G'.

Un E-graphe élémentaire est bien sûr externe. L'inverse est en général faux. Par exemple, le segfi G4 est externe, G1 et G2 sont aussi externes.

4. O R D R E.

Dans cette section on introduit plusieurs relations d'ordre sur E-graphe. Puisque tout ensemble fini partiellement ordonné admet un (ou plusieurs) élément(s) z maximal ou minimal, on notera max(z) ou min(z) respectivement cet (ces) éléments.

4.1. ORDRE VERTICAL.

L'ordre vertical, ou ordre total strict sur les arcs issus d'un même noeud, va être obtenu au moyen d'une énumération arbitraire ρ de l'ensemble $\underline{\sigma}(x)$ des arcs issus de x. Graphiquement, cet ordre est représenté de haut en bas.

Définition.

Soient deux arcs α et β issus de x. Supposons que l'énumération de $\underline{\sigma}(x)$ produise

$$\rho(\underline{\sigma}(x)) = \{\mu_1 \ . \ .... \ \mu_n\}$$

on dit que "α précède verticalement β" ssi

$$(\exists i,j)[\mu_i=\alpha \ \& \ \mu_j=\beta \ \& \ i<j] \qquad (1<=i<=n),\ (1<=j<=n)$$

on note $\alpha<v\beta$.

La relation équivalente "β succède verticalement α" sera notée $\beta v>\alpha$.

4.2. ORDRE HORIZONTAL.

L'ordre horizontal est l'ordre entre deux arcs situés sur un chemin quelconque du E-graphe.

Définition.

Soient α et β deux arcs de A. On dit que "α précède horizontalement β" s'il existe un chemin $\gamma(N)$ tel que $\gamma(0)=\alpha$ & $\gamma(n)=\beta$. On note $\alpha<h\beta$.

Cette relation sur l'ensemble A est une relation d'ordre partiel strict puisque G est sans circuits.

5. TRAJECTOIRES ET LEUR ORDRE.

Définition.

Une trajectoire dans un E-graphe G est un chemin allant de l'entrée I à la sortie O.

Ordre sur les trajectoires.

Définition.

Soient $\gamma(N)$ et $\lambda(N)$ deux trajectoires distinctes de G. On dit que "γ succède à λ", noté $\gamma>\lambda$ ssi

$$(\exists i)[(\forall j)[0<=j<i \Rightarrow \gamma(j)=\lambda(j)] \ \& \ [\lambda(i)<v\gamma(i)]] \qquad (0<=i<=n)$$

6. PARCOURS CANONIQUE.

Le parcours canonique va être donné par l'énumération séquentielle des arcs de chaque trajectoire et ceci sans repétition des arcs déjà produits.

A cette fin, on définit une opération, notée Θ, sur deux listes A et B, consistant à concaténer à A la sous-liste de B formée des éléments hors de A.

Définition.

Soient $A=(a1 \ldots an)$ et $B=(b1 \ldots bm)$ deux listes, alors

$$A \ \Theta \ B = (a1 \ldots an bj_1 \ldots bj_k)$$

$$\text{avec} \quad 1<=j_1<\ldots<j_k<=m, \ \text{et} \ (\forall p,i)[bj_p \neq ai] \qquad (1<=i<=n), \ (1<=p<=k)$$

Cette opération est associative. Pour le démontrer définissons la fonction suivante

$$\zeta_R(r)= \begin{cases} \epsilon \ \text{(suite vide)} & \text{si } r \epsilon R \\ r & \text{sinon} \end{cases}$$

et notons $\Sigma(A)$ l'ensemble formé des éléments de A (A peut avoir des répétitions), i.e. $\Sigma(A)=\{a1 \ldots an\}$. Alors si $B=(b1 \ldots bm)$,

$$A \ \Theta \ B= (a1 \ldots an \zeta_{\Sigma(A)}(b1) \ldots \zeta_{\Sigma(A)}(bm))= A.\zeta_{\Sigma(A)}(B)$$

$$(A \oplus B) \oplus C = (A.\zeta_{\Sigma(A)}(B)).\zeta_{\Sigma(A \oplus B)}(C)$$
$$= A.\zeta_{\Sigma(A)}(B).[\zeta_{\Sigma(A)}(\zeta_{\Sigma(B)}(C))]$$
$$= A.\zeta_{\Sigma(A)}(B.\zeta_{\Sigma(B)}(C))$$
$$= A \oplus (B \oplus C) \quad \text{q.e.d.}$$

notez les proprietés suivantes de Σ et ζ.

- $\Sigma(A \oplus B) = \Sigma(A) \cup \Sigma(B)$
- $\zeta_{A \cup B} = \zeta_A.\zeta_B$
- $\zeta_R(Y) \cdot \zeta_R(Z) = \zeta_R(Y \cdot Z)$

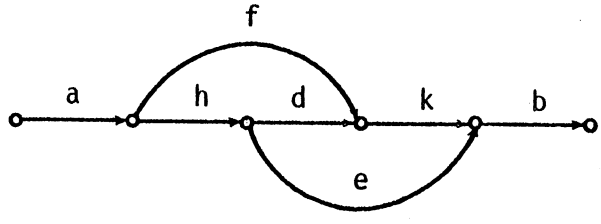
Définition.

Ainsi. si γ_i est la liste des arcs de la i-ème trajectoire. alors le parcours canonique est donné par

$$\gamma_1 \oplus \gamma_2 \oplus ... \oplus \gamma_t$$

avec t= nombre des trajectoires.

Exemple: soit le E-graphe suivant



les listes de chaque trajectoire sont

γ_1= afkb; γ_2= ahdkb; γ_3= aheb

d'où le parcours

$\gamma_1 \oplus \gamma_2 \oplus \gamma_3$= afkbhde

7. EGALITE DE DEUX E-GRAPHES.

Définition.

Un E-graphe G" est égal à un E-graphe G' ssi ils ont le même ensemble de chemins et le même ordre vertical, i.e.

$$G" = G' \Leftrightarrow [C(\rho")=C(\rho')] \;\&\; [(\forall\gamma.\lambda \text{ de } G')[\gamma<v\lambda] \Rightarrow (\exists\gamma'.\lambda' \text{ de } G")[\gamma'<v\lambda'] \;\&\; (\gamma=\gamma' \;\&\; \lambda=\lambda')]$$

V. C O N C L U S I O N.

On a vu dans ce chapitre: d'une part, un bref tour d'horizon de "l'environnement informatique" dans lequel se présente le travail. Il se situe donc dans la seconde génération, dans le cadre de la TAAH. D'autre part, une formalisation de toutes les notions concernant la structure objet et les seules sous-structures permises: les sous-E-graphes.

Une fois situé le travail et définie la structure, on se pose le problème de l'étude des transformations des E-graphes. Pour cette étude, je vais me borner uniquement à la structure objet, en laissant de côté les manipulations possibles sur les E-arbres. Voilà l'objet du chapitre suivant.

TABLE BIBLIOGRAPHIQUE.

[1] Bolt (77b)

[2] Bolt (79a)

[3] Bolt (81)

[4] Bolt*(77a)

[5] Bolt*(77b)

[6] Bolt*(78)

[7] Bolt*(80)

[8] Chau (74)

[9] Chau (75)

[10] Chau*(72)

[11] Colm (70)

[12] Daun (77b)

[13] Glad (69)

[14] Hofm (78)

[15] Kula*(69)

[16] Kunt (72)

[17] Rous (75)

[18] Stew (75)

[19] Thou (76)

[20] Vauq (75)

[21] Vauq (79)

[22] Wood (70)

[23] *EURO(80)

CHAPITRE II

TRANSFORMATIONS DE LA STRUCTURE

INTRODUCTION.

Le problème que l'on se pose dans ce chapitre est la manipulation de la structure. On définit donc des transformations intéressantes et aussi des fonctions qui vont devenir la base du processus de transduction de E-graphes. Divisons-les pour l'instant en deux types: globales et locales.

Les premières opèrent sur la structure objet en la considérant comme un élément. On trouvera donc des opérations comme la concaténation, l'alternation (ou mise en parallèle), l'expansion, la factorisation et enfin, brièvement la minimisation.

Les secondes opèrent sur des sous-structures isolables. On étudie leur remplacement, leur effacement et leur ajout.

Finalement on présente les mécanismes de base d'une transduction: la reconnaissance au moyen de schémas de E-graphes et les règles qui expriment la transformation voulue.

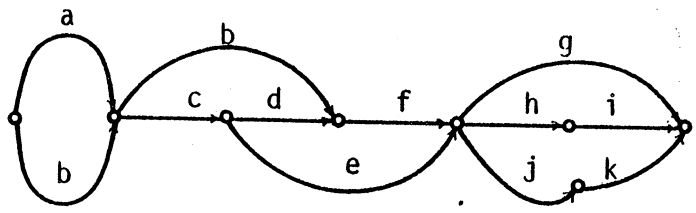
I. TRANSFORMATIONS GLOBALES.

1.CONCATENATION ET ALTERNATION.

1.1. CONCATENATION.

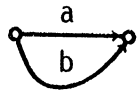
Si G' et G" sont deux E-graphes, alors la concaténation de G' et de G", notée G'·G" ou G'G" est obtenue en identifiant l'entrée de G" avec la sortie de G'. Par exemple, le E-graphe

G =

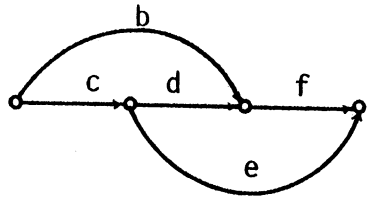


peut être exprimé comme le résultat de la concaténation des trois E-graphes G1,G2,G3

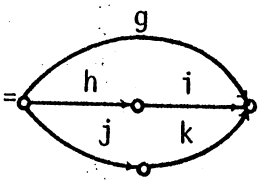
G1 =



G2 =



G3 =



c'est-à-dire, G=G1.G2.G3. On notera la non-commutativité de cette opération.

Définition.

Le E-graphe G, résultat de la concaténation d'un E-graphe G" à un autre G' est défini par.

$$\rho = \rho' \cup (\rho''-M) \cup \{(\alpha.O', \tau(\alpha)) \mid (\alpha.\rho''(\alpha)) \in M\}$$

$$\text{avec } M = \{(\alpha.\rho''(\alpha)) \mid \alpha \in A'' \ \&\ \sigma(\alpha)=I''\}$$
$$\text{et la condition } A' \cap A''=0 \ \&\ S' \cap S''=0$$

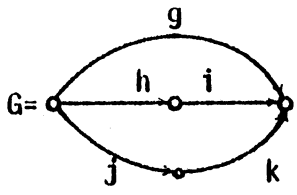
$$S = S' \cup (S''-\{I''\}); \qquad I=I'; \qquad O=O''.$$

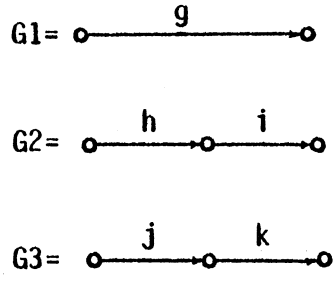
on note G= G'G" ou G= G'.G"

Etant donné qu'on privilégie les étiquettes et non les noms des sommets ou des arcs, si la condition d'intersection vide entre arcs et sommets des deux E-graphes n'est pas satisfaite, on peut toujours renuméroter A" ou S" par une fonction arbitraire afin de pouvoir réaliser l'opération.

1.2. ALTERNATION

Soit G' et G" deux E-graphes quelconques. Alors l'alternation entre G' et G", notée G'+G", est obtenue en confondant les entrées et les sorties de G' et G". par exemple, si





alors G est le résultat de l'alternation de G1, G2 et G3 , i.e. G= G1+G2+G3. On notera encore la non-commutativité de cette opération puisque les E-graphes sont ordonnés.

Définition.

Le E-graphe G, résultat de l'alternation d'un E-graphe G" à un autre G' est défini par.

$$\rho = \rho' \cup (\rho" - (M1 \cup M2)) \cup \{(\alpha, l', \tau(\alpha)) \mid (\alpha, \rho"(\alpha)) \in M1\} \cup \{(\alpha, \sigma(\alpha), O') \mid (\alpha, \rho"(\alpha)) \in M2\}$$

avec $M1 = \{(\alpha, \rho"(\alpha)) \mid \alpha \in A" \ \& \ \sigma(\alpha) = I"\}$

$M2 = \{(\alpha, \rho"(\alpha)) \mid \alpha \in A" \ \& \ \tau(\alpha) = O"\}$

et la condition $(A' \cap A") = 0 \ \& \ (S' \cap S") = 0 \ \& \ (\forall \alpha \in A")(\forall \beta \in A')[\beta < v\alpha]$

$S = S' \cup (S" - \{I", O"\})$; $I = I'$; $O = O'$

on note G= G'+G"

Remarque. Les E-graphes réguliers peuvent aussi être définis comme la classe de E-graphes obtenue par concaténation et alternation d'arcs simples.

1.3. PROPRIETES

I. Associativité. Aussi bien l'opération de concaténation que celle d'alternation sont des opérations associatives, car on a

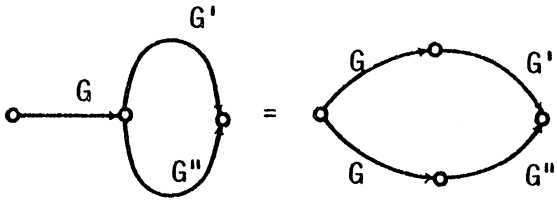
$(G.G').G" = G.(G'.G")$ et
$(G+G')+G" = G+(G'+G")$

respectivement.

II. Distributivité. La concaténation est distributive à gauche et à droite par rapport à l'alternation. Par exemple

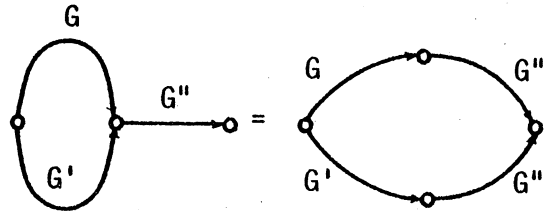
à gauche

$G.(G'+G") = G.G'+G.G"$



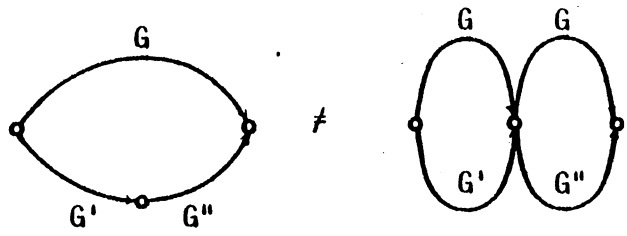
à droite

$(G+G').G" = G.G"+G'.G"$



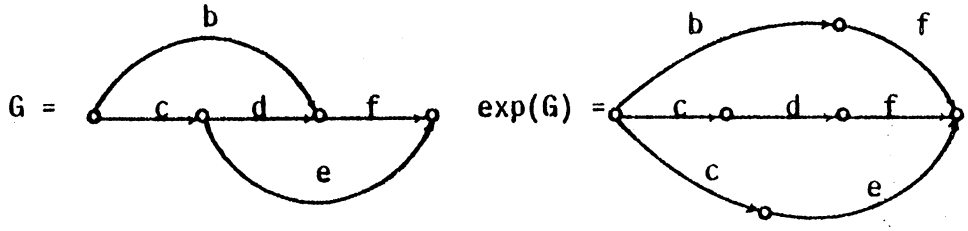
Par contre, l'alternation n'est pas distributive par rapport à la concaténation. Par exemple

$$G+(G'.G'') \neq (G+G').(G+G'')$$



2. EXPANSION.

L'expansion d'un E-graphe G est l'opération qui produit un faisceau exp(G) à partir de toutes les trajectoires de G. Ainsi, l'expansion de G1 produit exp(G1).



Définition

Soit G un E-graphe et soit $\bar{\gamma}_G(I,O) = \{\gamma 1, \ldots, \gamma n\}$ l'ensemble des chemins reliant I et O. S'il existe Gi (1<=i<=n), seg de G, tels que,

$$(\forall i)\,[\bar{\gamma}_{Gi}(I,O) = \{\gamma i\}],$$

alors l'expansion de G est donnée par l'expression.

$$\exp(G) = G1 + \ldots + Gn = \sum_{i} Gi$$

3. FACTORISATION ET MINIMISATION.

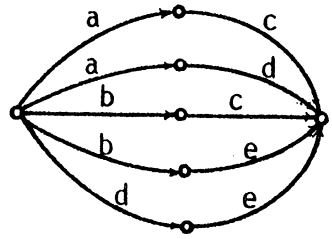
Le problème que l'on se pose est celui de factoriser les parties communes ou, ce qui est équivalent, de se rapprocher d'une écriture polynômiale minimale en un certain sens.

Les propriétés de la concaténation et de l'alternation suggèrent une opération de factorisation et, puisque celle-ci réduit le nombre de concaténations, on peut se demander s'il est possible de trouver un E-graphe minimal, i.e. un E-graphe où le nombre de concaténations et le nombre d'alternations soient minimum. On cherchera alors une minimisation.

Ces deux opérations sont distinctes et peuvent éventuellement coïncider quant au résultat mais non pas quant à la méthode de traitement, comme on va le voir. Donnons maintenant quelques exemples, avant de chercher à formaliser les opérations qui nous intéressent.

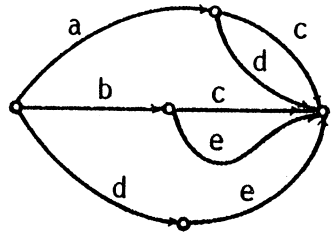
Soit le E-graphe G= ac+ad+bc+be+de

G=



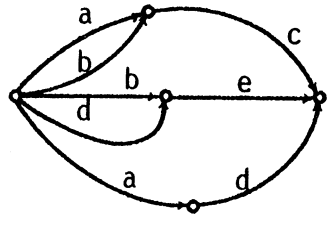
ce E-graphe-ci peut être factorisé "à gauche", donnant alors comme résultat G1= a(c+d)+b(c+e)+de.

G1=



Intuitivement, on a appliqué la distributivité à gauche de "." par rapport à "+". La factorisation "à droite" donne un résultat différent, G2= (a+b)c+(b+d)e+ad

G2=

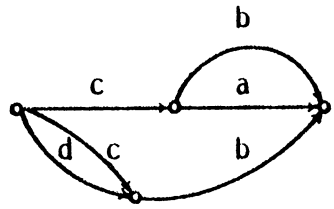


Maintenant, on a appliqué la distributivité à droite. Remarquez que l'expansion peut s'interpréter comme une "défactorisation complète".

Dans ces deux exemples, on s'est ramené d'une expression avec 4 alternations et 5 concaténations, notée (4+,5.), à une expression (4+,3.). Deux arcs identiques ont été fusionnés. On comprend dès maintenant que, sur des E-graphes un peu plus complexes, on peut factoriser à gauche à certains endroits, puis à droite sur d'autres, etc. On s'intéressera, pour la simplicité, à des méthodes uniformes.

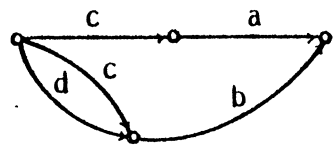
Considérons maintenant, le E-graphe

G3= c(b+a)+(c+d)b



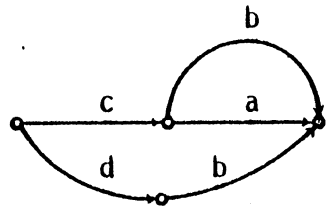
Il est évident que l'on peut minimiser G3 de deux manières: en éliminant l'arc b de la première trajectoire

G4= ca+(c+d)b



ou en éliminant l'arc c qui est en parallèle avec d

G5= c(b+a)+db

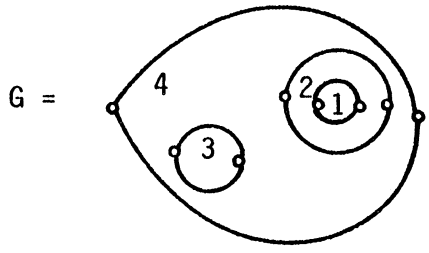


On notera que, cette fois, aucune séquence d'expansions et de factorisations ne nous permettera de retrouver G3. On s'intéressera donc aux conditions et méthodes pour la minimisation.

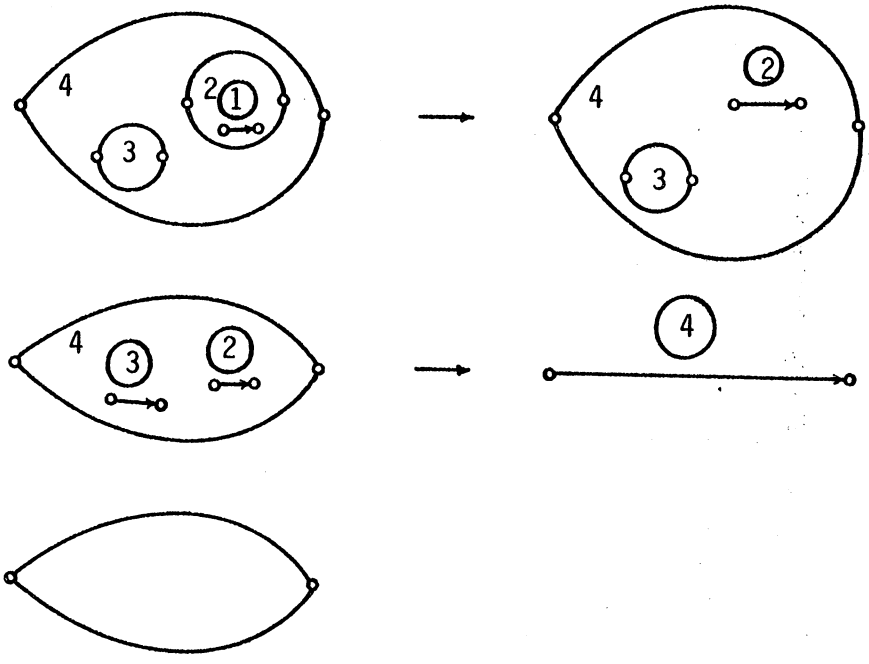
3.1. FACTORISATION.

La factorisation d'un E-graphe est une opération que l'on peut diviser en trois parties: (1) un processus qui réalise les transformations propres à l'opération, (2) une "factorisation élémentaire" qui agit sur des segfi-élémentaires et enfin (3) une "factorisation générale" qui opère sur tout le E-graphe en cherchant les factorisation élémentaires à effectuer. Intuitivement, le processus de factorisation est le suivant: c'est la factorisation générale (FG) qui démarre le processus. Une FG cherche un segfi-élémentaire, puis on le remplace par un arc étiqueté du segfi-élémentaire factorisé. On recommence le processus jusqu'à épuisement des segfi-élémentaires. A la fin on aura un E-graphe étiqueté par des E-graphes factorisés qu'il faudra "défactoriser" pour avoir le résultat.

Exemple: Dans cet exemple, on numérote les segfi, schématisés par un rond. Soit maintenant le E-graphe schématisé suivant



après recherche et remplacement



et 4 factorisé.

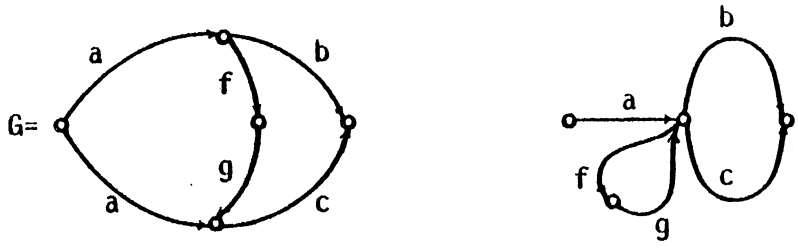
3.1.1. Factorisations gauche et droite

I. Factorisation gauche.

Intuitivement, cette opération, notée $f_g(G)$ (ou fG s'il n'y a pas d'ambiguïté), parcourt de manière canonique un à un tous les sommets x à plus d'un successeur. A chaque sommet x, on identifie les arcs qui portent les mêmes arbres parmi l'ensemble des arcs issus de x, puis on applique une transformation afin de ne garder qu'un seul arc.

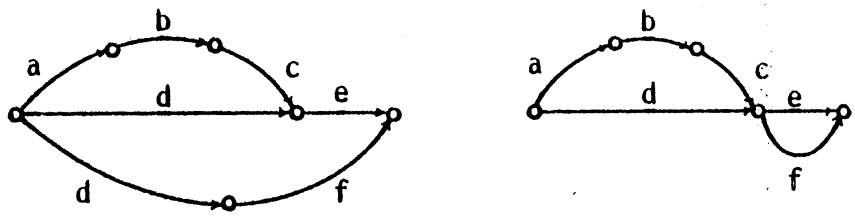
Une condition suffisante pour que la factorisation gauche puisse s'appliquer en un sommet x est que tout élément de l'ensemble des successeurs de x ait un seul arc incident. Autrement, on pourrait avoir

- Une boucle. Soit par exemple



Une factorisation de G produirait la boucle fg.

- Un chemin parasite.



A cause du chemin bc incident à $\tau(d)$, une factorisation de G produirait le chemin abcf qui n'existait pas auparavant.

- L'idempotence. Si on factorisait

ceci impliquerait la propriété: a+a=a ce qui en général est faux.

Avant de continuer, donnons une nouvelle notation sur l'ensemble A des arcs: "les arcs α et β sont étiquetés par le même E-arbre", notée

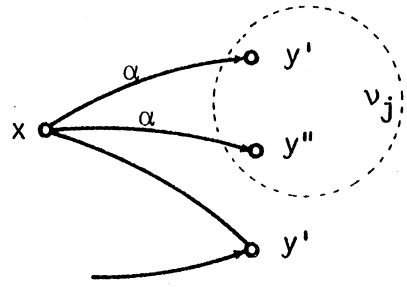
$$\alpha \ \mu \ \beta \ .$$

Il est évident que la relation μ est une relation d'équivalence.

Soit maintenant un sommet x. De manière plus formelle, la factorisation gauche en x consiste à trouver l'ensemble quotient $\underline{\sigma}(x)/\mu$, c'est-à-dire, l'ensemble des classes d'équivalence des arcs issus de x et portant le même arbre. Supposons qu'il y ait j classes d'équivalence (évidemment $1 <= j <= |\underline{\sigma}(x)|$) et désignons par $\nu_j(x)$ l'ensemble des sommets successeurs de la j-ième classe. Soit α et β deux arcs, alors

$$\nu_j(x) = \{ y \mid (\exists \alpha)[(\alpha, \sigma(\alpha)=x, \tau(\alpha)=y) \in Cl_j] \ \& \ (\nexists \beta)[\beta \neq \alpha \ \& \ \tau(\beta)=y] \}$$

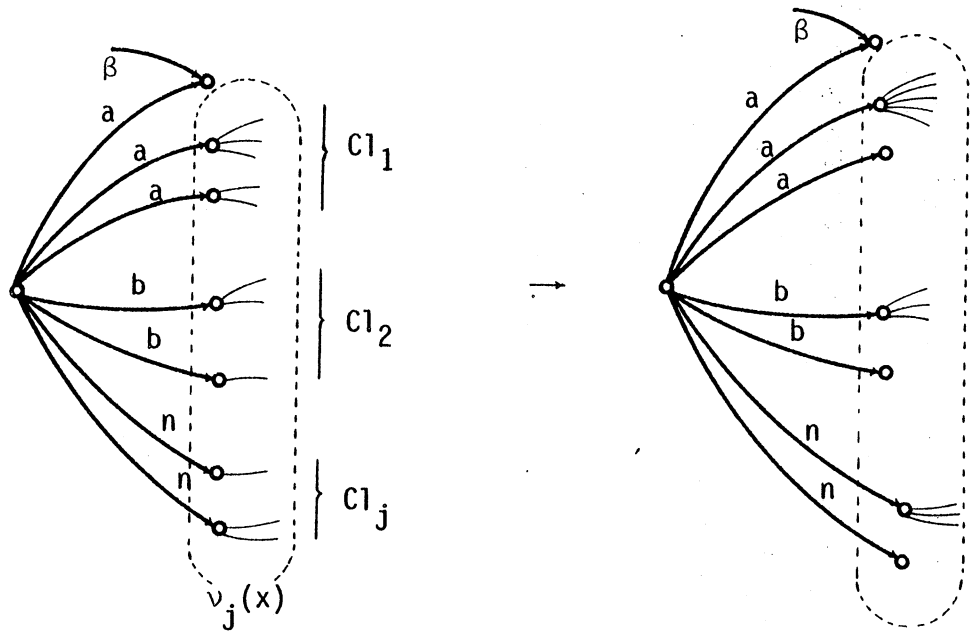
graphiquement



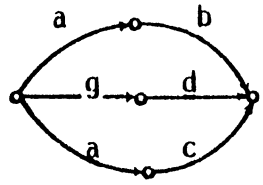
On choisit alors un **représentant**, i.e. un sommet z_j dans ν_j , puis on effectue la transformation

$$\sigma(\underline{\sigma}(\nu_j(x)) = z_j \ .$$

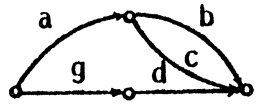
c'est-à-dire que tous les arcs dont la source appartient à $\nu_j(x)$ auront z_j comme nouvelle source.

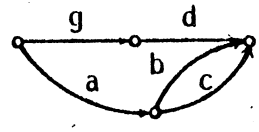


On supprime enfin tous les arcs de $\underline{\sigma}(\nu_1(x))$ sauf ceux dont le but est le représentant z_1. Il est important de remarquer que le choix des représentants peut induire un nouvel ordre. Par exemple, en factorisant



on peut avoir les deux résultats suivants

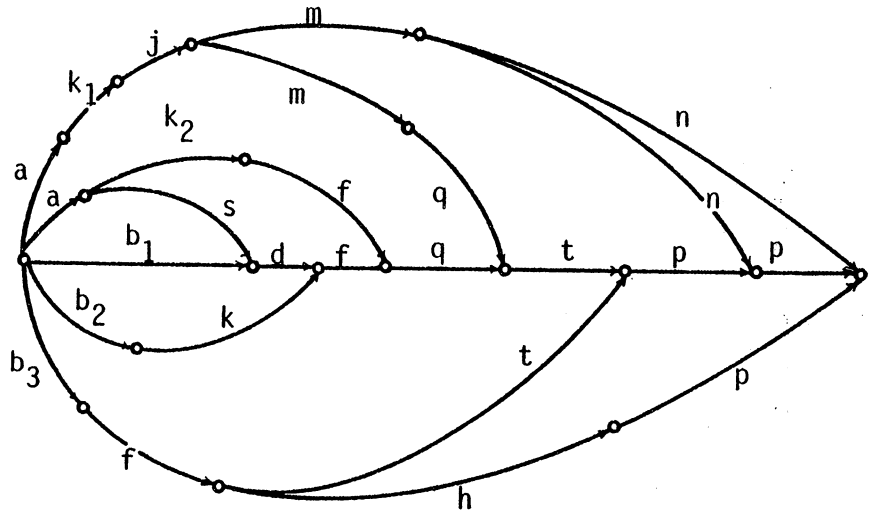




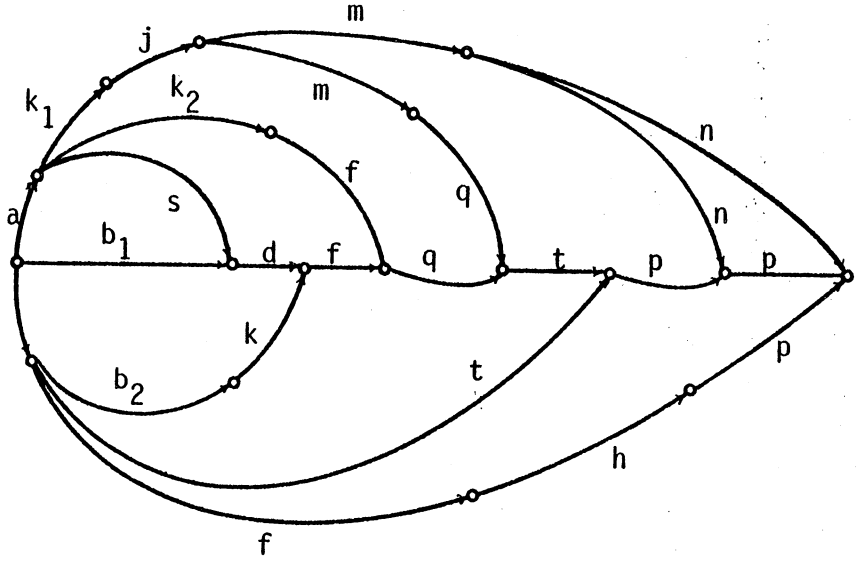
On montre maintenant un exemple de factorisation gauche (sauf mention explicite, le représentant sera le plus petit élément de chaque classe dans l'ordre vertical).

EXEMPLE. Soit G le E-graphe à factoriser à gauche (la trace des transformations est donnée ci-dessous)

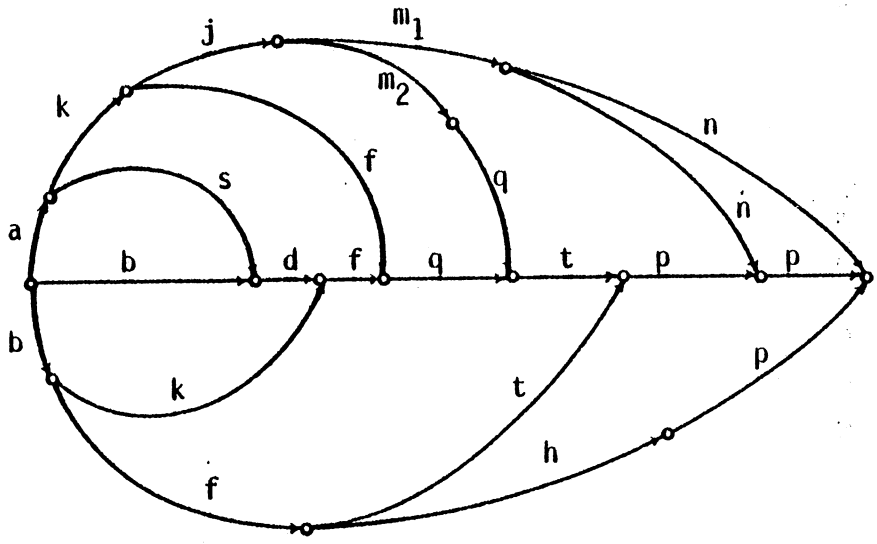
G =



G1 =



G2 =



G3 =

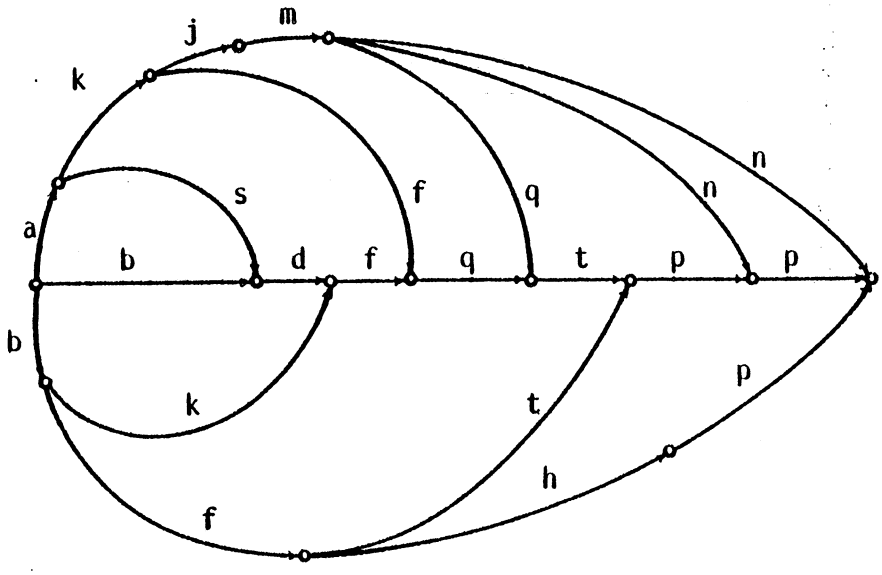


Tableau de la trace de la factorisation gauche.

sommet visité	remarques	avant $\underline{\sigma}(x)$	avant $\underline{\sigma}(x)/\mu$	E-graphe
1		{a,a,b1,b2,b3}	{a,a},{b2,b3}	G
2		{k1,k2,s}	{k1,k2},{s}	G1
3	s.c.			
4		{m1,m2}	{m1,m2}	G2
5	s.c.			
6	n.m.			
7	n.m.			
8	–			
9	n.m.			
10	n.m.			
11	–			
12	–			
13	n.m.			
14	n.m.			
15	n.m.			
16	s.c.			
17	–			
18	s.c.			
19	n.m.			
		après		
1		{a,b1,b2}	{a},{b2}	G1
2		{k1,s}	{k1},{s}	G2
4		{m1}	{m1}	G3

N.B. s.c. = sans changement
– = disparu
n.m. = non multisuccesseur

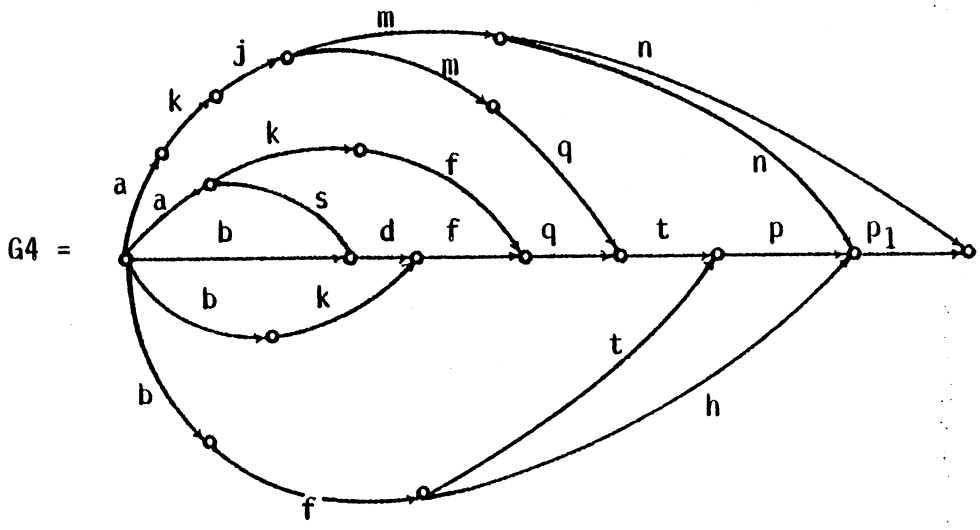
II. Factorisation droite.

La définition est la même que pour une factorisation gauche mais en utilisant la substitution s suivante

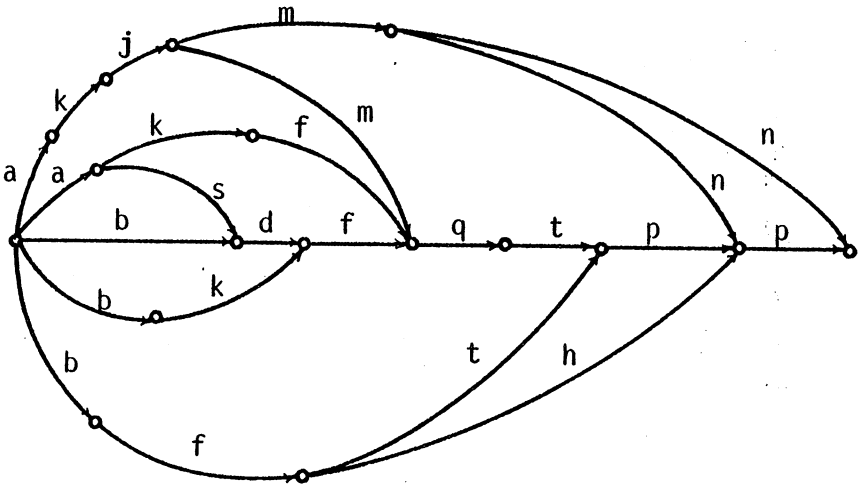
s = { successeurs/prédécesseurs, sortant/entrant, entrant (ou incident)/sortant, σ/τ, τ/σ}

Le parcours est pris comme un parcours canonique en considérant l'entrée comme la sortie et inversement.

EXEMPLE. Soit encore G le E-graphe à factoriser à droite (la trace est donnée dans le tableau ci-dessous)



G5 =



G6 =

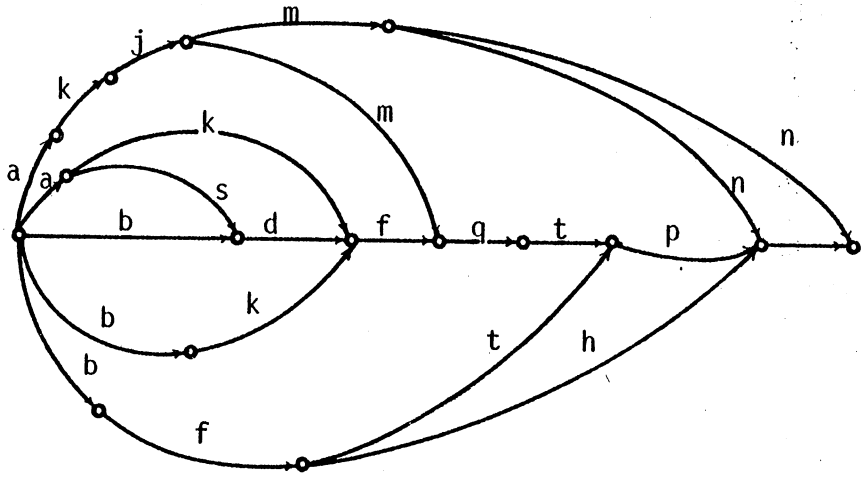


Tableau de la trace de la factorisation droite.

sommet visité	remarques	a v a n t $\underline{\sigma}(x)$	$\underline{\sigma}(x)/\mu$	E-graphe
6		{n.p1.p2}	{n}.{p1.p2}	G
5	n.m.			
4	n.m.			
3	n.m.			
2	n.m.			
1	n.m.			
7	s.c.			
10	s.c.			
9		{q.q}	{q.q}	G4
8		{m.f.f}	{m}.{f.f}	G5
12	s.c.			
11	n.m.			
14	s.c.			
16	n.m.			
17	n.m.			
18	n.m.			
19	–			
13	–			
15	–			
		a p r è s		
6		{n.p1}	{n}.{p1}	G4
9		{q}	{q}	G5
8		{m.f}	{m}.{f}	G6

N.B. s.c. = sans changement
– = disparu
n.m. = non multisuccesseur

3.1.2. Factorisation élémentaire.

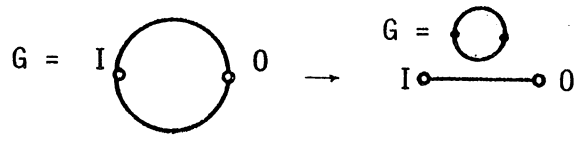
Une factorisation élémentaire d'un E-graphe G, notée $\rho(G)$ est la composée $\pi o \delta$ d'une factorisation δ (gauche ou droite) appliquée à G ou à un segfi-élémentaire de G, i.e.

$$\delta(G) = \begin{cases} f_{[\,]}\ (G) & \text{si G est élémentaire} \\ f_{[\,]}\ (\text{le 1er. G', s'il existe}) & \text{sinon} \end{cases}$$

avec G' = segfi-élémentaire de G
[ ] = g.d

et d'une transformation π de G en un E-graphe récursif consistant en un seul arc étiqueté par G lui-même.

$$\pi(G) = (G, I, O),$$



3.1.3. Factorisation générale (FG)

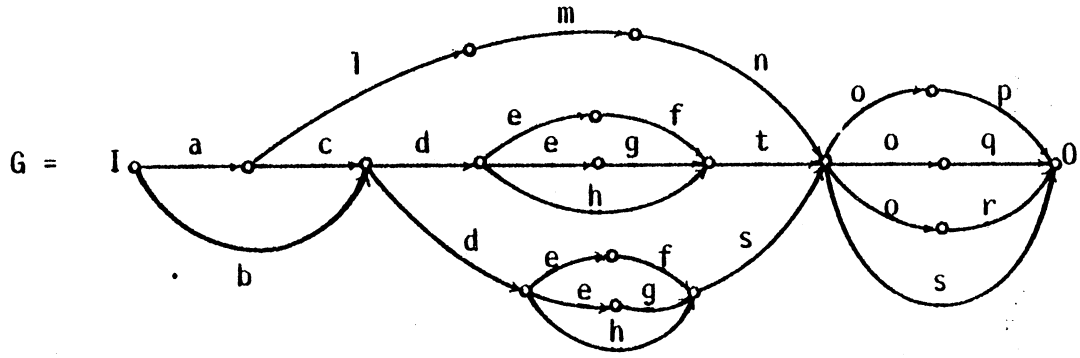
Une factorisation générale Ψ d'un E-graphe G est définie à partir de ρ comme

$$\Psi(G) = T^{-1} o\ \rho^{(n)}\ (G) \qquad \text{où} \quad \rho^{(n)}\ (G) \text{ est le 1er non-factorisable}$$

$$\text{avec} \quad T^{-1}\ (G) = \begin{cases} G & \text{si G n'a qu'un arc étiqueté par un E-graphe} \\ T^{-1}\ (G') & \text{sinon et si G' est obtenu à partir de G en remplaçant un arc récursif par le E-graphe qu'il porte} \end{cases}$$

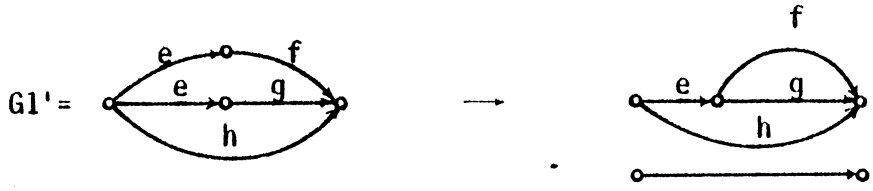
C'est-à-dire qu'on applique à G des factorisations élémentaires jusqu'à ce qu'il ne soit plus factorisable, i.e. jusqu'à ce que $\rho(G)$ ne produise plus aucune transformation sur G. A ce moment, on aura un E-graphe récursif étiqueté par des segfi factorisés. On transforme enfin ce E-graphe récursif en un E-graphe simple qui sera le résultat.

EXEMPLE. Soit G le E-graphe sur lequel on va appliquer une FG avec des factorisations gauches

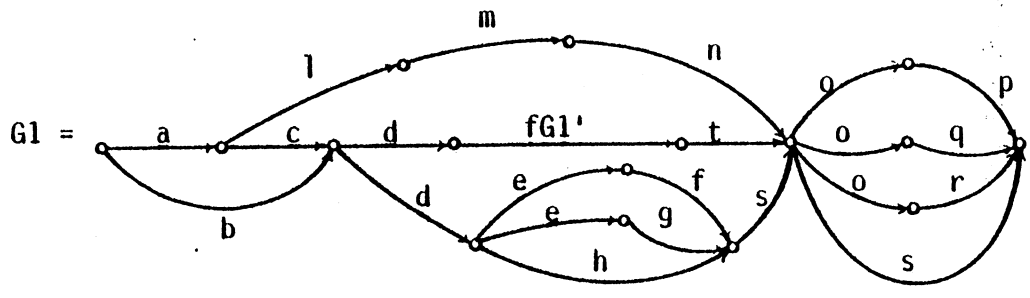


$$\psi(G) = T^{-1} \circ \rho^{(1)}(G)$$

Pour calculer $\rho^{(1)}(G)$, admettons que le premier segfl-élémentaire rencontré soit



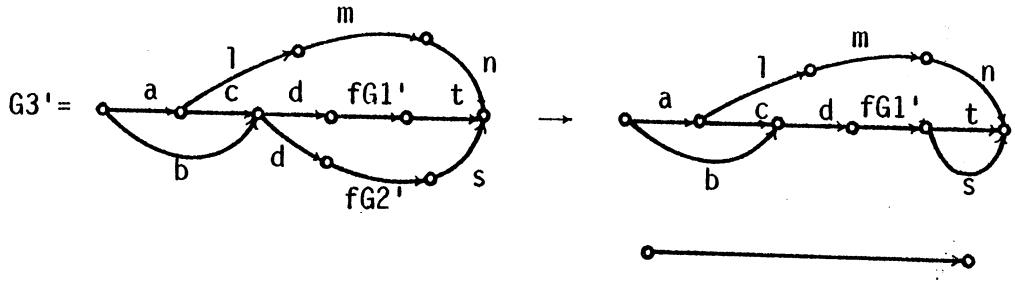
on aura alors



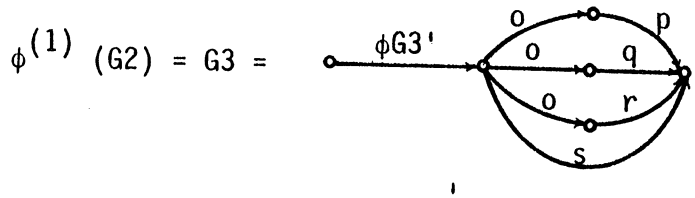
On a maintenant: $\psi(G) = T^{-1} \circ \rho^{(2)}(G) = T^{-1} \circ \rho^{(1)}(G1)$.

puisque G1 est encore factorisable. Pour calculer $\rho^{(1)}(G1)$, admettons que le premier segfl-élémentaire rencontré dans G1 soit G2=G1' et on refait une transformation analogue.

On obtient maintenant: $\psi(G) = T^{-1} \circ \rho^{(3)}(G) = T^{-1} \circ \rho^{(2)}(G1) = T^{-1} \circ \rho^{(1)}(G2)$, puisque G2 est encore factorisable. Admettons cette fois que G3' soit la prochaine sous-structure à factoriser (remarquez que f(G1')= f(G2'), alors



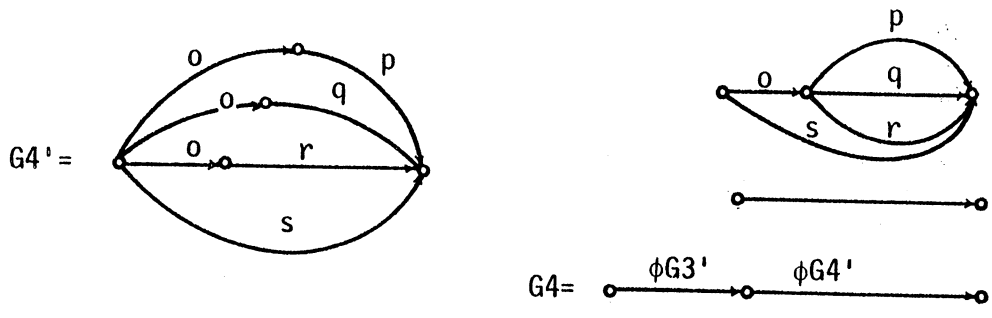
La factorisation de G2 produit



Le pas suivant est la factorisation de G3

$$\psi(G) = T^{-1} \circ \rho^{(4)}(G) = T^{-1} \circ \rho^{(1)}(G3)\ ,$$

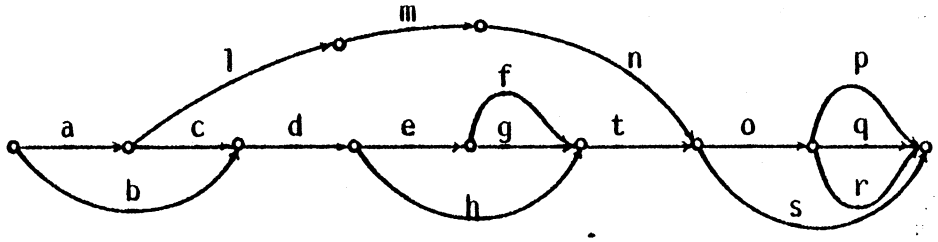
cette fois, le seul segfi-élémentaire de G3 est G4' et G4 le résultat de la transformation



Ce E-graphe n'est plus factorisable, ce qui produit finalement

$$\psi(G) = T^{-1}(G4),$$

d'où le résultat final

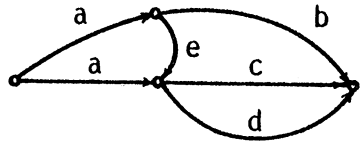

m
l
n
p
f
a
c
d
e
g
t
o
q
b
h
s
r

3.2. MINIMISATION.

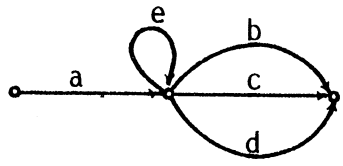
Si on admet l'idempotence de l'alternation, i.e. la propriété

$$\alpha + \alpha = \alpha$$

on arrive naturellement au problème de trouver un E-graphe minimal, c'est-à-dire un E-graphe où il n'existe pas de sommet dont deux arcs issus soient étiquetés par le même arbre. Des difficultés apparaissent lorsque l'on essaie d'identifier (ou d'éliminer) cette classe d'arcs, par exemple dans le E-graphe suivant

G = 

On ne peut identifier les deux arcs portant a, sous peine de créer une boucle avec l'arc e (d'ailleurs, cette identification ne serait pas une application correcte de l'idempotence de +)



Ce problème n'est pas entièrement résolu. Voici cependant un bref tour d'horizon de quelques méthodes proposées pour minimiser un E-graphe.

3.2.1. Méthode de fusion.

Cette méthode, inspirée des travaux de [12] sur la minimisation de machines séquentielles, consiste à faire l'analogie machine-E-graphe de la façon suivante:

machine		E-graphe
ensemble d'états	------>	ensemble de sommets
ensemble d'entrées	------>	ensemble d'étiquettes
ensemble de transitions	------>	ensemble d'arcs

La méthode est fondée sur la construction d'une matrice M, |A|x|A|, telle que

$$m[i,j] = \begin{cases} 1 & \text{si } \alpha j \text{ est adjacent à } \alpha i \\ - & \text{sinon} \end{cases}$$

On réalise alors sur M des manipulations sur les lignes et les colonnes, jusqu'à ce qu'on arrive à avoir une matrice à une seule ligne et une seule colonne. Ces manipulations sont de trois types et on va les illustrer à l'aide de trois exemples.

1. Elimination des 1 (ou recherche des sous-trajectoires).

Soit le E-graphe a.b.c, avec la matrice

	a	b	c
a	-	1	-
b	-	-	1
c	-	-	-

Les manipulations sont les suivantes

	a	b	c
a	-	1	-
b	-	-	1
c	-	-	-

--->

	a	bc
a	-	1
bc	-	-

--->

	abc
abc	-

2. Fusion de lignes et de colonnes (ou recherche des faisceaux).

Soit le E-graphe a+b+c, avec la matrice

	a	b	c
a	-	-	-
b	-	-	-
c	-	-	-

Les manipulations sont les suivantes

	a	b	c
a	–	–	–
b	–	–	–
c	–	–	–

--->

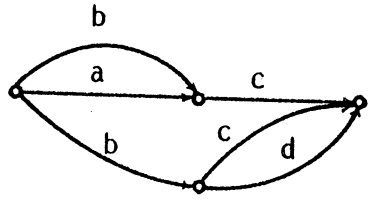
	a	b+c
a	–	–
b+c	–	–

--->

	a+b+c
a+b+c	–

3. "Eclatement" d'un arbre.

Soit le E-graphe (b+a)c+b(c+d)



avec la matrice

	a	b	c	d
a	–	–	1	–
b	–	–	1	1
c	–	–	–	–
d	–	–	–	–

Etant donné l'impossibilité d'appliquer aucune des opérations précédentes, on éclate b et on tente de nouveau les mêmes manipulations, i.e.

	a	b1	b2	c	d
a	–	–	–	1	–
b1	–	–	–	1	–
b2	–	–	–	–	1
c	–	–	–	–	–
d	–	–	–	–	–

--->

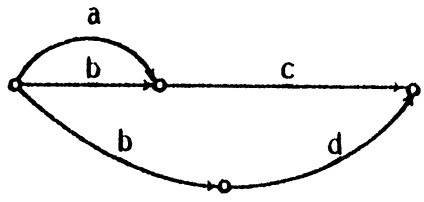
	a	b1	bd	c
a	–	–	–	1
b1	–	–	–	1
bd	–	–	–	–
c	–	–	–	–

--->

	a+b1	bd	c
a+b1	–	–	1
bd	–	–	–
c	–	–	–

	(a+b1)c	bd			(a+b)c+bd
(a+b1)c	-	-	--->	(a+b)c+bd	-
bd	-	-			

et le résultat



Cette méthode produit un E-graphe avec un nombre d'arcs inférieur ou égal à celui du E-graphe de départ. En fait, selon l'ordre des opérations, elle peut conduire à plusieurs résultats. Parmi le sous-ensemble de ceux qui ont le plus petit nombre d'arcs, on en choisit un. Cependant, on montre [8,9] que ce résultat n'est pas optimal en général.

3.2.2. D'autres méthodes.

Ginsburg [8] a abordé le problème en réduisant les états superflus, mais la quantité de calculs est beaucoup trop importante pour qu'on puisse envisager des applications pratiques. Sa technique pour réduire une machine donnée à une machine à nombre minimum d'états a l'inconvénient de ne pas être entièrement automatique. Je ne donnerai donc pas plus de détails sur ces procédés.

Une autre méthode consisterait à donner aux E-graphes une structure d'anneau et à appliquer les résultats obtenus par Chatelin [3] concernant la diminution du nombre de produits effectifs sur le calcul d'un ensemble de formes bilinéaires sur un anneau non-commutatif. Ceci pose le problème des éléments neutre et symétrique pour les lois de composition $\underline{+}$ et $\underline{.}$.

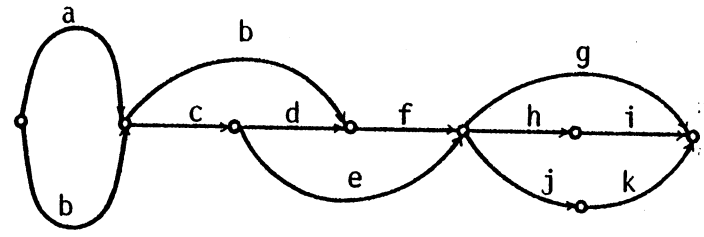
Pour l'instant, le problème reste ouvert, et on se contente donc de méthodes approchées.

4. ARBORESCENCE ASSOCIEE A UN E-GRAPHE

Il est utile d'avoir une opération qui transforme un E-graphe en une arborescence étiquetée et ordonnée.

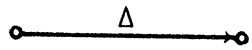
Il existe, bien sûr, plusieurs façons de construire un arbre à partir d'un E-graphe mais on cherche un arbre qui conserve l'ordre vertical (ordre sur les arcs issus d'un même sommet) et aussi l'ordre horizontal (ordre sur les arcs d'une trajectoire. Par exemple, soit

G =



la transformation donnerait comme résultat G1, étiqueté par l'arbre Δ

G1 =



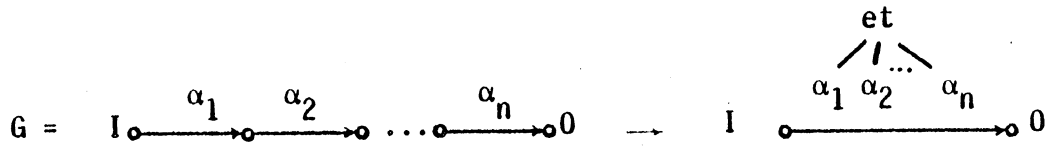
avec

Δ= et(ou(a,b), alt(et(b,f), et(c,d,f), et(c,e)), ou(g, et(h,i), et(j,k)))

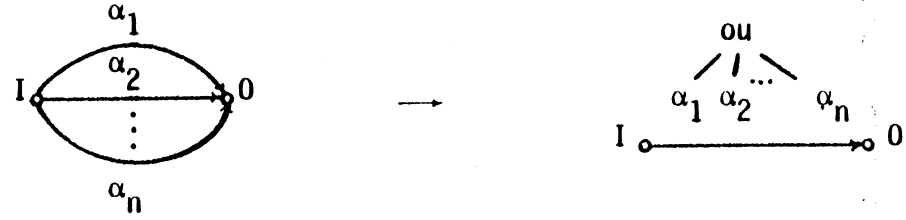
L'arborescence Δ satisfait cette restriction avec l'avantage de refléter la structure de G: un E-graphe avec trois segfi séquentiels (représentés par les trois fils du premier niveau) dont le premier et le dernier sont des alternations régulières (i.e. pouvant être décrits par une écriture régulière) et le second une alternation non-régulière.

Le processus de génération de Δ sera analogue à celui de la factorisation quant à la méthode de recherche et à la substitution de chaque segfi-élémentaire par un arc étiqueté jusqu'à ce qu'on arrive à un seul arc. Ce qui est différent, ce sont les opérations réalisées sur chaque segfi-élémentaire.

On procède à une division des segfi-élémentaires en deux classes: ceux qui sont réguliers et ceux qui sont non-réguliers. Pour tous les segfi de la première classe, on fait deux opérations. La première $\theta 1$ transforme toute trajectoire du segfi en question en un arc étiqueté par un arbre ordonné de racine et et dont les fils seront les E-arbres de la trajectoire. Par exemple, graphiquement cette opération produit

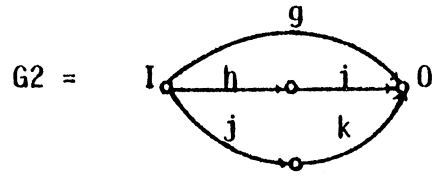


La seconde opération, $\theta 2$, transforme toute alternation en un arc étiqueté par un arbre ordonné de racine ou et dont les fils seront les E-arbres qui étiquettent l'ensemble des arcs issus de l'entrée de l'alternation. Graphiquement on a

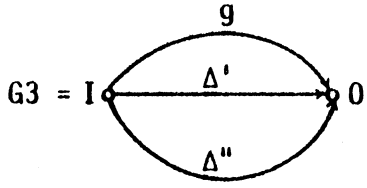


Pour les segfi de la deuxième classe, les non-réguliers, on fait d'abord leur expansion, et on les traite ensuite comme des segfi réguliers. Cependant, au lieu de $\theta 2$ on utilise une opération, $\theta 3$, qui réalise la même transformation que $\theta 2$ mais dont la racine sera alt pour marquer la non-régularité du segfi.

Reprenons l'exemple donné plus haut. Considérons ainsi le segfi-élémentaire de G suivant

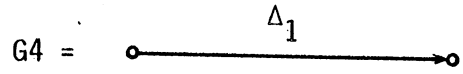


après $\theta 1$, G2 devient



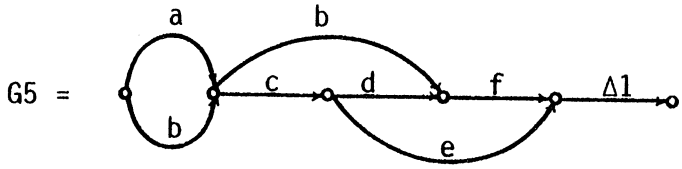
Δ'= et(h,i)
Δ"= et(j,k)

et après $\theta 2$ on a

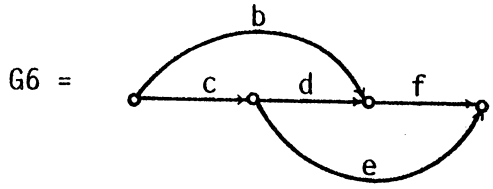


Δ_1= ou(g,Δ',Δ'')

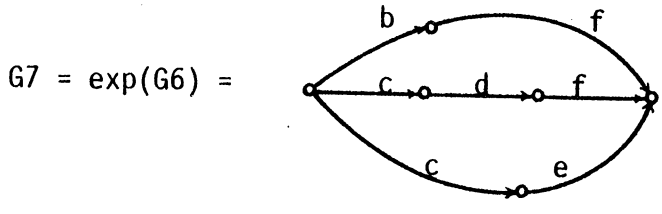
ce qui transforme G en



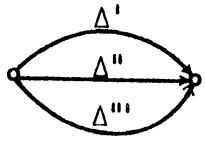
Considérons un autre segfi-élémentaire,



puisqu'il est non-régulier, on réalise son expansion,

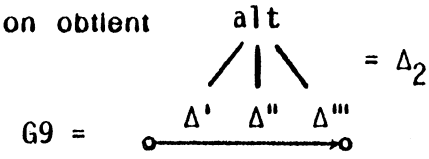


après $\theta 1$, G7 devient,

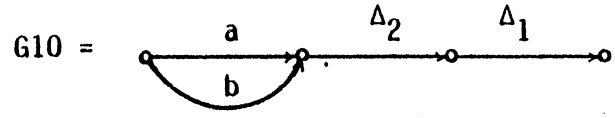


Δ'= et(b,f)
Δ''= et(c,d,f)
Δ'''= et(c,e)

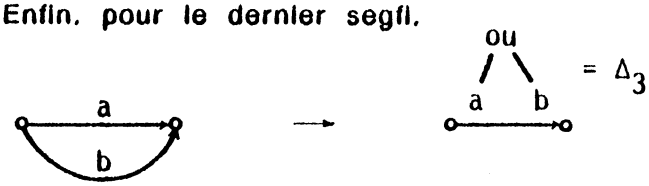
et après $\theta 3$, on obtient



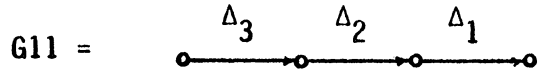
G5 devient ainsi,



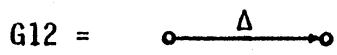
Enfin, pour le dernier segfi,



ce qui transforme G10 en,



et sur lequel l'application de $\theta 2 \circ \theta 1$ produit finalement,



où Δ est l'arborescence cherchée.

Passons maintenant à la formalisation. Définissons d'abord les transformations essentielles.

Opération $\theta 1$.

Définition.

Cette transformation, applicable uniquement à un E-graphe G élémentaire, est définie par

$$\theta 1(G)= \begin{cases} R^n(Gn) & \text{si G élémentaire.} \\ 0 & \text{sinon.} \end{cases}$$

avec $n \leq C(\rho)$. Gn le n-ième seg séquentiel et

$$R^I(GI)= \begin{cases} GI & \text{si GI est (réduit à) un arc} \\ GI' & \text{sinon} \end{cases}$$

où $GI'= (\rho=(E.I.O).\ E= \{et(\alpha_1 \ \ldots \ \alpha_{|AI|})\})$

Opération $\theta 2$.

Définition.

Cette transformation est applicable à un E-graphe G régulier et élémentaire et qui de plus vérifie la propriété suivante: G ne contient aucun seg séquentiel Gs tel que $|As|>1$. Maintenant, $\theta 2$ est définie par.

$$\theta 2(G)= \begin{cases} W^n(Gn) & \text{si G régulier et élémentaire.} \\ 0 & \text{sinon} \end{cases}$$

avec $n \leq$ nombre des seg de G. Gn le n-ième faisceau de G et.

$W^I(GI)= GI'$. avec $GI'= (\rho=(E.I.O).\ E=\{ou(\alpha_1. \ \ldots \ .\alpha_q)\})$

avec $\alpha i \in \sigma(Ii)$ & $(\forall i)(\forall j)[\sigma(\alpha i)=\sigma(\alpha j)$ & $\tau(\alpha i)=\tau(\alpha j)]$
$(1 \leq i \leq q.\ 1 \leq j \leq q)$
et q= nbre d'arcs du faisceau.

Opération $\theta 3$.

Définition.

On a pour $\theta 3$ la même définition que pour $\theta 2$, sauf pour le E-arbre E qui cette fois sera

$$E = \{alt(\alpha_1, \ldots, \alpha_q)\}$$

Ayant défini $\theta 1$, $\theta 2$ et $\theta 3$, on peut maintenant formaliser la notion d'arborescence associée à un E-graphe par la.

Définition.

Soit G un E-graphe. L'arborescence associée à G, notée $\Delta(G)$, est donnée par l'expression

$\Delta(G) = \chi^n(G)$. où $\chi^n(G)$ est un arc étiqueté par un E-arbre.

avec

$$\chi^1(G) = \begin{cases} \theta 2 \circ \theta 1(G') & \text{si } G' \text{ régulier.} \\ \theta 3 \circ \theta 1(exp(G')) & \text{sinon} \end{cases}$$

avec G' le premier segfi-élementaire de G.

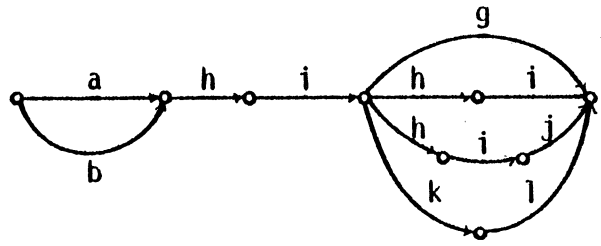
$$\chi^{l+1}(G) = \chi(\chi^l(G))$$

II. TRANSFORMATIONS LOCALES.

1. REMPLACEMENT

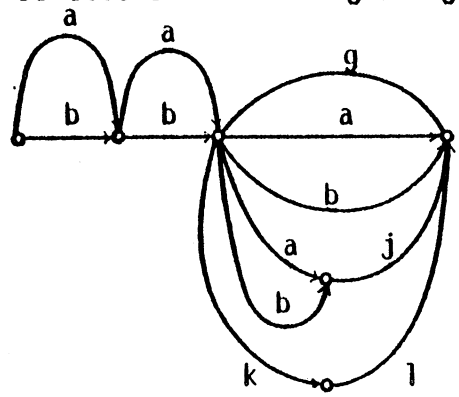
Etant donné deux E-graphes G et G", cette opération réalise la substitution d'une sous-structure isolable G' de G par G". Par exemple, soit

G =



et supposons G"= a+b et G'= h.i. Le remplacement de G' par G" sur G comprend ici les trois cas de sous-structures: segif, segmi et segfi et donne comme résultat

G1 =



Définition.

Soient G et G" deux E-graphes et soit G' une sous-structure isolable de G. Alors, le E-graphe Gr, résultat du remplacement de G' par G" dans G est,

$$\rho r = (\rho - \rho') \cup (\rho" - (M1 \cup M2)) \cup \{(\alpha.I', \tau(\alpha)) \mid (\alpha.\rho"(\alpha)) \in M1\} \cup \{(\alpha.\sigma(\alpha).O') \mid (\alpha.\rho"(\alpha)) \in M2\}$$

avec $M1 = \{(\alpha.\rho"(\alpha)) \mid \alpha \in A" \ \& \ \sigma(\alpha) = I"\}$
$M2 = \{(\alpha.\rho"(\alpha)) \mid \alpha \in A" \ \& \ \tau(\alpha) = O"\}$

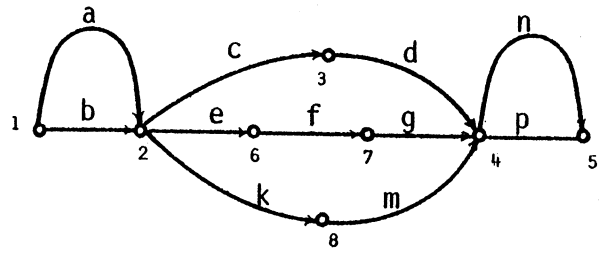
et la condition $(A \cap A"=0) \ \& \ (S \cap S")=0$

$Sr = (S - (S' - \{I', O'\})) \cup (S" - \{I", S"\})$

2. EFFACEMENT

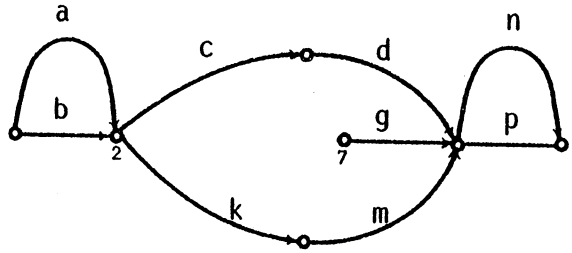
Si G est un E-graphe, cette opération élimine de G une de ses sous-structures. Les caractéristiques de cette sous-structure, appelons-la G', ne sont pas arbitraires et le fait de travailler avec des sous-structures isolables ne produit pas toujours un E-graphe. Par exemple, soit

G =



si on efface de G le segif G'= e.f, on obtient

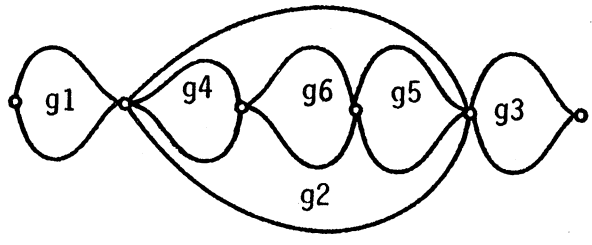
G1 =



qui n'est plus un E-graphe.

Quant à l'effacement des segfi, trois cas peuvent se présenter: (1) I∈S', (2) O∈S' et (3) I,O∉S'. Pour les cas (1) et (2), il n'y a pas de problèmes d'altération de structure. Le cas (3) altère la structure d'une façon particulière. Prenons un exemple, soit le E-graphe schématique suivant

G =



La suppression de g6 produit deux sommets non-connexes qu'on choisit de confondre . Le cas de g2 est analogue.

On notera que cette méthode n'est pas applicable aux segfi en général: confondre 2 et 7 dans G1 conserve la structure, mais si G'= e.f.g. confondre 2 et 4 produirait une boucle. Pour cette raison, les seules structures effaçables seront les segfi et les segmi. Pour toute autre structure, on supprimera le plus petit segmi qui la contient. On passe maintenant à la

Définition.

Soit G un E-graphe et G' un seg de G. Le E-graphe Ge, résultat de l'effacement de G' est.

- **si G' est un segfi et ($I \notin S'$ & $O \notin S'$)**

  $\rho e = (\rho - M1) - \rho' \cup \{(\alpha, \sigma(\alpha), O') \mid (\alpha, \rho(\alpha)) \in M1\}$

  avec $M1 = \{(\alpha, \rho(\alpha)) \mid \alpha \in A \ \& \ \tau(\alpha) = I'\}$

  $Se = S - (S' - \{O'\})$

- **si G' est un segfi et $I \in S'$**

  $\rho e = \rho - \rho'$; $Ie = O'$; $Se = S - (S' - \{O'\})$

- **si G' est un segfi et $O \in S'$**

  $\rho e = \rho - \rho'$; $Oe = I'$; $Se = S - (S' - \{I'\})$

- **si G' est un segmi**

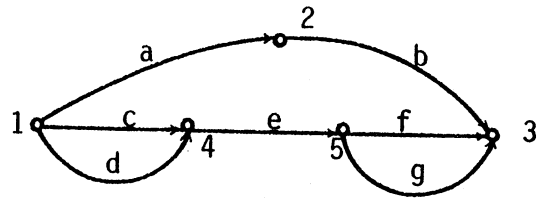
  $\rho e = \rho - \rho'$; $Se = S - (S' - \{I', O'\})$

- **si G' est un segif, alors on efface G", le plus petit segmi qui le contient.**

3. A J O U T.

Cette opération réalise une addition d'un E-graphe G', dans un autre G. Ceci est fait en localisant deux sommets distincts de G pour les confondre avec l'entrée et la sortie de G'. Le choix de ces deux sommets n'est pas arbitraire et il faudra définir le nouvel ordre des arcs issus du sommet confondu avec l'entrée de G'. Considérons les exemples suivants, soit

G =

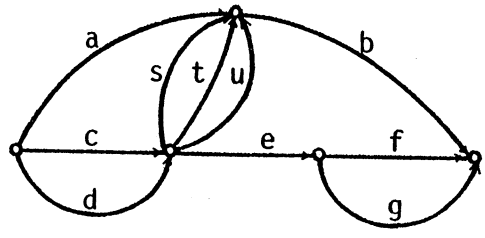


Supposons G'= s+t+u. Il y a différentes manières d'ajouter G' à G. On peut cependant les regrouper en trois classes. En effet, chaque couple (x,y) de noeuds de G satisfait exactement une des 3 conditions

(1) $\bar{\gamma}(x,y) = 0$
(2) $\bar{\gamma}(x,y) = 0 \ \& \ \bar{\gamma}(y,x) \neq 0$
(3) $\bar{\gamma}(x,y) = \bar{\gamma}(y,x) = 0$

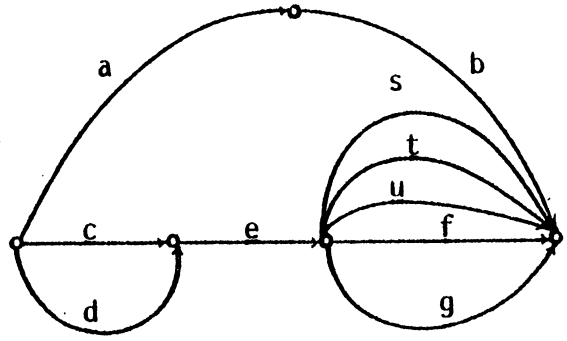
On note que seule la seconde classe ne permet pas l'ajout puisque ceci créerait des circuits. Les deux autres posent le problème du nouvel ordre parmi les arcs issus du sommet confondu avec l'. Pour des raisons d'homogénéité, l'ajout sera fait soit "en haut" soit "en bas". Par exemple, pour (3), si on prend (4,2) et confondons l' avec 4 et O' avec 2, on aura

G1 =



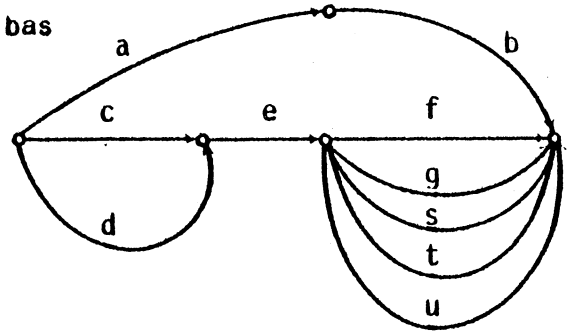
On remarquera que les deux ajouts possibles donnent la même représentation graphique. Pour (1), si on prend (5,3), on aura pour l'ajout en haut

G2 =



et pour l'ajout en bas

G3 =



On notera que si l'on prend (I,O) comme couple de noeuds, on retrouve l'alternation. Sauf indication contraire, tout ajout sera réalisé par le bas.

Définition.

Soit deux E-graphes G et G', l'ajout de G' à G sur un couple de sommets (x,y) de S produit un nouveau E-graphe Ga défini par

$$\rho a = \rho \cup (\rho' - (M1 \cup M2)) \cup \{(\alpha, x, \tau(\alpha)) \mid (\alpha, \rho'(\alpha)) \in M1\} \cup \{(\alpha, \sigma(\alpha), y) \mid (\alpha, \rho'(\alpha)) \in M2\}$$

avec $M1 = \{(\alpha, \rho'(\alpha)) \mid \alpha \in A' \ \& \ \sigma(\alpha) = I'\}$

$M2 = \{(\alpha, \rho'(\alpha)) \mid \alpha \in A' \ \& \ \tau(\alpha) = O'\}$

et la condition $(A \cap A' = 0) \ \& \ (S \cap S'' = 0) \ \& \ \bar{\gamma}(y, x) = 0$

$Sa = S \cup (S' - \{I', O'\})$

III. T R A N S D U C T I O N.

1. RECONNAISSANCE

Un problème essentiel pour toute transformation d'un E-graphe est celui de décrire une sous-structure et de pouvoir localiser une ou plusieurs de ses occurrences. On aborde maintenant le problème de la description.

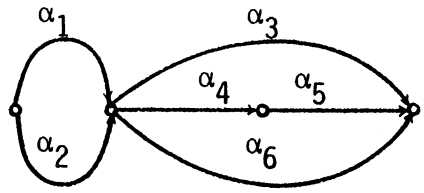
Schémas de E-graphe. De façon analogue à ce qui est fait dans les systèmes CETA [4,5] et ROBRA [1], l'écriture d'un schéma comporte deux éléments.

- Une description structurelle.
- Une condition portant sur les étiquettes (ou éventuellement sur l'état du système).

1.1. DESCRIPTION STRUCTURELLE

Cette partie du schéma correspond à la description de la forme "géométrique" du E-graphe. On peut employer la notation utilisée pour les E-graphes, cependant, cette fois, chaque arc est étiqueté par un symbole purement formel unique: le **nom** de l'arc. Par exemple, soit le schéma suivant

S1 =



il aura comme description,

en notation séquentielle		en notation régulière
S1=	(α1,1,2)	S1= (α1+α2)(α3+α4.α5+α6)
	(α2,1,2)	
	(α3,2,3)	
	(α4,2,4)	
	(α5,4,3)	
	(α6,2,3)	

Afin de rendre plus fine la description structurelle, on introduit des éléments de description structurelle.

1.2. ELEMENTS DE DESCRIPTION STRUCTURELLE.

Ces éléments doivent nous permettre d'exprimer des contraintes de contigüité (verticale ou horizontale), d'accessibilité, de position ou bien de résoudre les conflits possibles.

1.2.1. Contigüité.

Deux arcs formels α et β sont contigus verticalement ssi

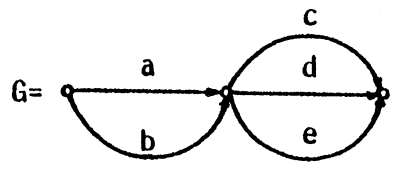
$\alpha <_v \beta$ ou $\beta <_v \alpha$,et on note $\alpha * \beta$.

En notation séquentielle, on fait suivre un arc d'un descripteur de contigüité consistant en un n-uplet d'arcs contigus verticalement, par exemple

S2= (α1,1,2) $\langle \alpha 1 * \alpha 2 \rangle$
(α2,1,2)

en notation régulière, on fait suivre l'opérateur + de *: S2= (α1 +* α2)

Maintenant, si on applique ce schéma au E-graphe



on trouve les occurrences suivantes

α1-a	α1-c	α1-d
α2-b	α2-d	α2-e

Remarquez la notation de l'élément minimal α et de l'élément maximal β de l'ensemble d'arcs issus d'un sommet x.

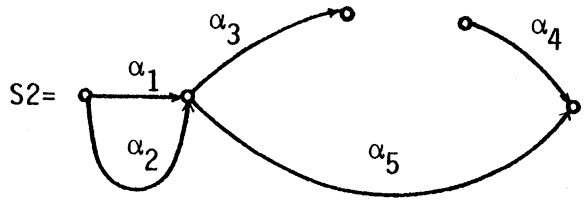
$\min(\underline{\sigma}(x)) = {}^*\alpha$
$\max(\underline{\sigma}(x)) = \beta^*$

1.2.2. Accessibilité.

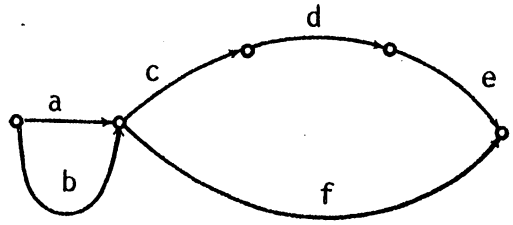
Un arc formel β est accessible depuis un arc formel α ssi $\alpha<h\beta$. De même que pour la contiguïté, on fait suivre un arc d'un descripteur d'accessibilité qui consiste en un n-uplet d'arcs séparés par le symbole .. Par exemple, soit le schéma

S2= (α1,1,2)
(α2,1,2)
(α3,2,3) <α3..α4>
(α4,3,4)
(α5,2,4)

ou en notation régulière: S2= (α1+α2)(α3..α4+α5)



qui, appliqué au E-graphe suivant



produit les trois occurrences

(1)	(2)	(3)
α1 - a	α1 - a	α1 - a
α2 - b	α2 - b	α2 - b
α3 - c	α3 - d	α3 - c
α4 - d	α4 - e	α4 - e
α5 - f	α5 - f	α5 - f

1.2.3. Conflits.

Afin de préciser l'occurrence à choisir lorsque plusieurs sont possibles, on définit plusieurs indicateurs de position. Les précisions concernant la position peuvent porter sur l'ordre horizontal (profondeur) ou sur l'ordre vertical (hauteur).

I. Profondeur relative. On appellera **superficielle** l'occurrence qui est la plus près du sommet d'entrée. De même, on appellera **profonde** l'occurrence qui est la plus loin du sommet d'entrée. On peut noter par

$S et $P

respectivement.

II. Position verticale relative. Pour un sommet x quelconque, on considérera comme le **haut** l'arc $\min(\underline{\sigma}(x))$ et comme le **bas** l'arc $\max(\underline{\sigma}(x))$. On peut noter

$H et $B

respectivement.

1.3. CONDITIONS

1.3.1. Conditions locales.

Le deuxième élément du schéma est une condition portant sur une ou plusieurs étiquettes. Chaque condition est une expression booléenne que l'étiquette doit satisfaire pour pouvoir être considérée comme un élément du schéma. Des relations booléennes intéressantes peuvent être: l'égalité, l'inclusion et leur négation. Exemple, soit le schéma S3 avec la description structurelle et les conditions locales suivantes

S3= $\alpha 1+\alpha 2/\alpha 1$:arbre=a,$\alpha 2$:arbre=b/

S3 est applicable à G1= a+b, mais ne le sera pas à G2=b+a puisque l'ordre vertical n'est pas le même.

1.3.2. Conditions globales.

Le troisième élément du schéma est une condition entre plusieurs arcs du E-graphe. De même que pour les conditions locales, la condition est une expression booléenne. On conserve les mêmes relateurs et opérateurs. Exemple, soit encore le schéma S3, avec la description structurelle et les conditions locales et globales suivantes.

S3= $\alpha 1+\alpha 2$ / $\alpha 1$:arbre=a / arbre($\alpha 1$)=arbre($\alpha 2$) /

S3 est applicable à G= a+a.

2. R E G L E S.

Une transformation est réalisée au moyen d'une règle de réécriture. Celle-ci se compose de deux parties principales.

- **La partie gauche**. Au moyen d'un schéma de E-graphe, elle définit le seg sur, lequel va s'appliquer la transformation.

- **La partie droite**. Elle définit le E-graphe objet, lui-même décrit par un schéma de transformation.

Notez qu'il s'agit d'un système de substitution. Par conséquent, il faut une fonction de transfert, ou bien il faut exiger que l'occurrence du schéma soit toujours un segmi. On choisit cette dernière convention ("condition d'occurrence") pour une première approche.

Schéma de transformation. Ce schéma définit les modifications formelles du schéma en partie gauche. Le schéma consiste en

- **Une description structurelle**. De même que pour la partie gauche, cet élément est un schéma purement formel de E-graphe.

- **Un ensemble d'affectations des étiquettes**. Pour chaque arc du schéma, on doit préciser la provenance de l'étiquette associée et ses modifications éventuelles.

Exemple, soit la règle suivante

R: $\alpha 1+\alpha 2$/$\alpha 1$:arbre=a,$\alpha 2$:arbre=b/ == $\beta 1+\beta 2+\beta 3$/$\beta 1$:=$\alpha 1$,$\beta 2$:=$\alpha 2$,$\beta 3$:=c/

l'application sur G= a+b donne comme résultat G= a+b+c.

IV. C O N C L U S I O N.

Dans ce chapitre on a discuté et défini formellement les opérations principales, et présenté les problèmes de reconnaissance et d'écriture des règles en vue de la transduction.

Il manque évidemment le traitement du problème de la transduction à proprement parler, notamment celle qui va être utilisée pour les étapes principales (analyse, transfert et génération) du processus de traduction: la transduction E-graphe --> E-graphe. Puisque l'on cherche l'homogénéité de la structure interne tout le long du processus, on voudrait que l'analyse de la chaîne d'entrée et la génération de celle de sortie soient réalisables par cette transduction. Pour l'instant, on se contente d'utiliser dans une première approche les systèmes-Q. C'est notre point de départ pour une transduction peut-être plus "adéquate" pour les E-graphes.

L'étape suivante de notre travail est la programmation. Celle-ci a été divisée en deux parties: le chapitre suivant et l'annexe. Dans le chapitre suivant, la programmation a été réalisée en PROLOG étant donnée sa puissance d'expression, pour concrétiser rapidement les notions et les problèmes concernant les E-graphes.

Dans l'annexe, on a utilisé le pseudo-PASCAL afin de "suivre" plus en détail les transformations et d'envisager déjà les éventuelles problèmes posés par une programmation plus efficace.

TABLE BIBLIOGRAPHIQUE.

[1] Bolt*(78)

[2] Brai (68)

[3] Chat (79)

[4] Chau (74)

[5] Chau (75)

[6] Colm (70)

[7] Colm (75)

[8] Gins (59a)

[9] Gins (59b)

[10] Gins (59c)

[11] Hopc (69)

[12] Huff (54)

[13] Knut (73)

[14] Neth (59)

[15] Wirt (76)

[16] *EURO(80)

C H A P I T R E I I I

PROGRAMMATION D'UN PROTOTYPE

I N T R O D U C T I O N.

Ce chapitre est consacré à la description d'un système de traitement des E-graphes qui concrétise les notions formelles vues aux chapitres précédents. Pour des raisons de simplicité et dans une première approche, on a retenu le choix des E-graphes réguliers comme structure de travail, puisqu'il existe toujours une chaîne pour représenter un E-graphe de cette classe. En outre, pour les mêmes raisons, on n'a pas cherché l'efficacité quoiqu'elle ne soit pas exclue: c'est l'annexe qui présente les bases pour une programmation plus efficace.

On commence par expliquer brièvement PROLOG, le langage utilisé, puis le système. Enfin, on montre les listes et des exemples d'exécution.

I. L E L A N G A G E : PROLOG.

Je reprends pour ce numéro-ci la description faite dans [9] avec de légères modifications.

La définition de PROLOG est née d'un besoin de disposer d'un formalisme permettant simultanément la description et la résolution de problèmes sur calculateur. C'est à la base un démonstrateur automatique qui permet de traiter aisément de données formelles telles que arbres, listes, formules algébriques, etc. Son support théorique est la logique des prédicats du premier ordre.

PROLOG travaille sur des fichiers qui sont des ensembles de "clauses". Programmes, données et résultats ont tous cette structure et un ensemble de commandes simples permet de lire ou d'écrire ces fichiers. D'autre part, les programmes peuvent être non-déterministes et récursifs.

1. TERMINOLOGIE DE BASE.

Sauf indication contraire, on suppose qu'à chaque symbole s est associé un entier I>=0 appelé son ordre. On note: ordre(s)=I.

Soit F un ensemble de symboles dits fonctionnels et soit V un ensemble dénombrable de variables. Toute formule construite comme suit est appelée terme sur F:

- Si V_I est une variable, alors V_I est un terme.
- Si $f \epsilon F$ et ordre(f)=0, alors f est un terme.
- Si $f \epsilon F$ et ordre(f)=n et $t_1,....,t_n$ sont des termes, alors $f(t_1,....,t_n)$ est un terme.

On note $\hat{H}(F)$ ou $\hat{H}$ l'ensemble des termes, et H(F) ou H l'ensemble de termes ne contenant pas de variable. H est souvent appelé univers d'Herbrand.

Soit R un autre ensemble de symboles dits maintenant relationnels, on qualifie d'atomique toute formule construite comme suit:

- Si $r \epsilon R$ et ordre(r)=0, alors r est atomique.
- Si $r \epsilon R$ et ordre(r)=n et $t_1,....,t_n$ sont des termes, alors $r(t_1,....,t_n)$ est atomique.

Si p est atomique, alors +p et -p sont des littéraux. Une clause est un ensemble de littéraux.

Une interprétation (d'Herbrand) I est un ensemble de formules atomiques sans variables. A chaque symbole relationnel r d'ordre n, elle fait correspondre la relation n-aire ρ entre les éléments de l'univers d'Herbrand.

$\rho(t_1,....,t_n)$ ssi $r(t_1,....,t_n) \in I$ $(\forall t_1,....,t_n \in H)$. Dans le cas où n=0, ρ se réduit à la valeur booléenne ρ ssi $r \in I$. Une interprétation I est <u>plus petite</u> qu'une interprétation J ssi $I \subset J$.

On considère que,

- Un ensemble de clauses est une conjonction de clauses.
- Les variables d'une clause sont quantifiées universellement en tête de celle-ci.
- Une clause est une disjonction de littéraux.
- Le signe + marque l'affirmation et le signe - la négation.

De ce fait, on définit la notion de <u>satisfaction</u> comme suit: une interprétation I,

- Satisfait un ensemble de clauses ssi elle satisfait chaque clause de cet ensemble. L'ensemble vide de clauses est considéré comme toujours satisfait.
- Satisfait une clause ssi elle satisfait chaque valeur de cette clause.
- Satisfait une clause sans variable ssi elle satisfait au moins un littéral de cette clause. La clause vide n'est jamais satisfaite.
- Satisfait un littéral sans variable $+p$ ssi $p \in I$.
- Satisfait un littéral sans variable $-p$ ssi $p \notin I$.

Entre deux ensembles de clauses A et B, on définit la relation $I=$ par:

A $I=$ B $\Leftrightarrow$ toute interprétation qui satisfait A satisfait B.

2. ENSEMBLE REGULIER DE CLAUSES. PLUS PETITE INTERPRETATION LE SATISFAISANT.

On a coutume de considérer un "programme" comme la définition d'une certaine fonction f. La "machine" qui l'exécute permet de "calculer" cette fonction en donnant le résultat f(x) pour tout x que l'on fournit comme donnée.

Soit E un ensemble de clauses où apparaît un certain symbole relationnel n-aire r. Supposons qu'il existe une plus petite interprétation I qui satisfasse E. Cette interprétation associe donc au symbole r une relation n-aire ρ. On peut donc considérer E comme un "programme" qui définit la relation ρ, à condition que l'on dispose de règles de déduction, jouant le rôle d'une "machine" permettant de "calculer" cette relation n-aire en énumérant tous les n-uplets qui la satisfont. Dans cette optique, les programmes seront des ensembles de clauses d'un type particulier, dits "réguliers".

Définition.

Une clause est dite <u>régulière</u> ssi elle contient un et un seul littéral positif. Un ensemble de clause est dit <u>régulier</u> ssi il ne contient que des clauses régulières.

<u>Proprieté 1</u>. Si E est un ensemble régulier de clauses, alors il existe une plus petite interprétation, notée Imin(E) qui le satisfait.

<u>Proprieté 2</u>. Soit E un ensemble de clauses ayant une plus petite interprétation Imin(E) qui le satisfait. Pour toute formule atomique sans variables, on a

$$E \models \{\{+p\}\} \quad \text{ssi} \quad p \in Imin(E)$$

Les règles de déduction dont on a besoin pour calculer des relations seront donc celles que l'on utilise en démonstration automatique.

3. REGLES DE DEDUCTION.

Les règles de déduction de PROLOG sont une simplification du principe de résolution de Robinson [20].

Soit L l'ensemble de tous les littéraux. On appellera clause ordonnée toute suite de littéraux $a_1,\ldots,a_n$ avec $n>0$. Quand $n=0$, on note cette suite $\hat{0}$. L'ensemble de clauses ordonnées est noté L^*.

Pour tout $x,y\in L^*$, on conviendra que

$xy = a_1\ldots a_n b_1\ldots b_n$ si $x = a_1\ldots a_n$ et si $y = b_1\ldots b_n$

$x\hat{0} = \hat{0}x = x$

Soit E un ensemble régulier de clauses et soit E_{ord} un ensemble de clauses ordonnées obtenu en remplaçant chaque clause $\{p_0,p_1,\ldots,p_n\}$ de E par une clause ordonnée

$$+p_0\ -p_1\ -p_2\ \ldots -p_n$$

où le littéral positif figure en tête.

Définition.

Pour tout $x,y\in L^*$, on note

$x \vdash_{E_{ord}} y$ **ssi** (a) $(\exists +p\in L)(\exists u,v\in L^*)$ tels que $x = u\ -p\ v$

(b) $(\exists s\in E_{ord})$ et $+qt$ est une variante de s obtenue en renommant les variables de s de façon à n'en avoir aucune de commune avec x.

(c) $y=[utv]\sigma$ où σ est le plus grand unificateur (au sens de Robinson) de l'ensemble $\{p,q\}$.

$x \vdash^{n}_{E_{ord}} y$ **ssi** $\exists u_0, u_1, \ldots u_n \in L^*$ tels que

$$x = u_0 \vdash_{E_{ord}} u_1 \vdash_{E_{ord}} \ldots \vdash_{E_{ord}} u_n = y$$

On énonce seulement le théorème et le corollaire donnés en [9].

Théorème. Pour toute formule atomique p

$$E\ I=\{\{+r\}\}$$

et r est une valeur de p ssi il existe n>0 et il existe une formule atomique q tels que:

$$-p\ +p\ \vdash^{n}_{E_{ord}}\ +q$$

et r est une valeur de q.

Corollaire. Pour toute formule atomique p,

$$r \in \mathrm{Imin}(E)$$

et r est une valeur de p ssi il existe n>0 et il existe une formule atomique q tels que,

$$-p\ +p\ \vdash^{n}_{E_{ord}}\ +q$$

et r est une valeur de q

4. MECANISMES GENERAUX DE PROLOG.

PROLOG est donc un langage où chaque instruction est un énoncé logique et l'exécution d'un programme consiste à faire des déductions.

D'une façon plus précise, un programme PROLOG consistera en une suite ordonnée de clauses. Chaque clause, est une suite ordonnée de littéraux et se termine soit par un point, soit par un point d'exclamation. Dans le premier cas, on enregistre la clause, dans le second, on l'exécute. Par exemple, soit le progamme de concaténation,

```
+CONC(NIL,*X,*X).
+CONC(.(*E,*X),*Y,.(*E,*Z)) -CONC(*X,*Y,*Z).
-CONC(.(A,NIL),.(B,NIL),*X)!
```

Remarquons au passage que les variables sont précédées d'un astérisque. L'exécution consiste à prendre la troisième clause comme la clause de départ x et à calculer successivement des clauses y_1, y_2, ... telles que

$$x \underset{E_{ord}}{\vdash} y_1 \underset{E_{ord}}{\vdash} y_2 \cdots$$

où E_{ord} représente l'ensemble des deux premières clauses. La fonction de sélection est celle qui choisit toujours le littéral le plus à gauche. Si pour un y_i, il existe plusieurs clauses $c_1, c_2, \ldots$ (enregistrées dans cet ordre dans E_{ord}) qui peuvent être utilisées pour construire un y_{i+1} tel que

$$y_i \underset{E_{ord}}{\vdash} y_{i+1}$$

alors le système choisit d'abord c_1, et c'est seulement lorsqu'il aura totalement fini d'explorer cette première voie qu'il choisira c_2 pour l'explorer, et ainsi de suite. L'ordre dans lequel les clauses sont enregistrées est donc important.

PROLOG a un certain nombre de symboles relationnels prédéfinis (ou littéraux évaluables [22]) au niveau de l'interpréteur. Sans d'autres précisions, ils permettent

- La gestion des entrées-sorties.
- La création de clauses et de symboles.
- Le contrôle de la stratégie.
- Le traitement des caractères et des entiers.

II. L E S Y S T E M E.

1. GENERALITES.

Le système utilise un langage très élémentaire comme moyen de communication avec l'utilisateur. Considérons le vocabulaire V de 1.6.2. et construisons le vocabulaire V1 pour le langage de commande.

$$V1 = V \cup \{b\} \cup \{,\} \cup \{;\}$$

et la grammaire GU sur ce vocabulaire.

<unité>::= <opération> b <liste-arg>;
<opération>::= ARBE | FACTD | FACTG | PARC | TRAJ | ALTE | CONC | EFAC | AJUT | SUBS
<liste-arg>::= <e-graphe><re-graphe>
<re-graphe>::= ,<e-graphe> | ,<e-graphe><re-graphe> | <vide>

Par exemple, la chaîne

ARBE a(b+f);

fait partie du langage L(GU).

Le moyen de représentation choisi est la chaîne, puisqu'il en existe toujours une que l'on peut associer à un E-graphe régulier. Introduisons les notations suivantes:

1. nil est la chaîne vide, x(n) une chaîne de longueur n.

2. Si $x = x_1 \ldots x_n$ et $y = y_1 \ldots y_m$ sont des chaînes, alors on note

   x.y

   la concaténation de x et de y.

3. Si $x = x_1 x_2 \ldots x_n$ est une chaîne non-vide, alors

   - tête(x)= x_1 est le premier élément de x.
   - queue(x)= $x_2 \ldots x_n$ est x sans le premier élément.
   - cola(x)= x_n est le dernier élément.

4. La chaîne $x=x_1...x_n$ est une sous-chaîne de la chaîne $y=y_1...y_m$, ssi il y a une application

$$F: \{1,2,.....n\} \dashrightarrow \{1,2,.....m\}$$

telle que $F(i)= k$ seulement si x_i est y_k et F est une fonction monotone strictement croissante. i.e.

$$(F(i)= u \ , \ F(j)= v \ \& \ i<j) \Rightarrow u<v$$

5. Pour une chaîne $x= x_1x_2...x_n$, x_{kt} est $x_kx_{k+1}...x_t$ si $k<=t$.

6. Etant donnée une sous-chaîne x_{kt} de x(n), l'opération d'extraction (ou d'effacement s'il n'y a pas d'ambiguïté avec l'opération sur les E-graphes) est

$$x - x_{kt} = x_1x_2...x_{k-1}x_{t+1}...x_n$$

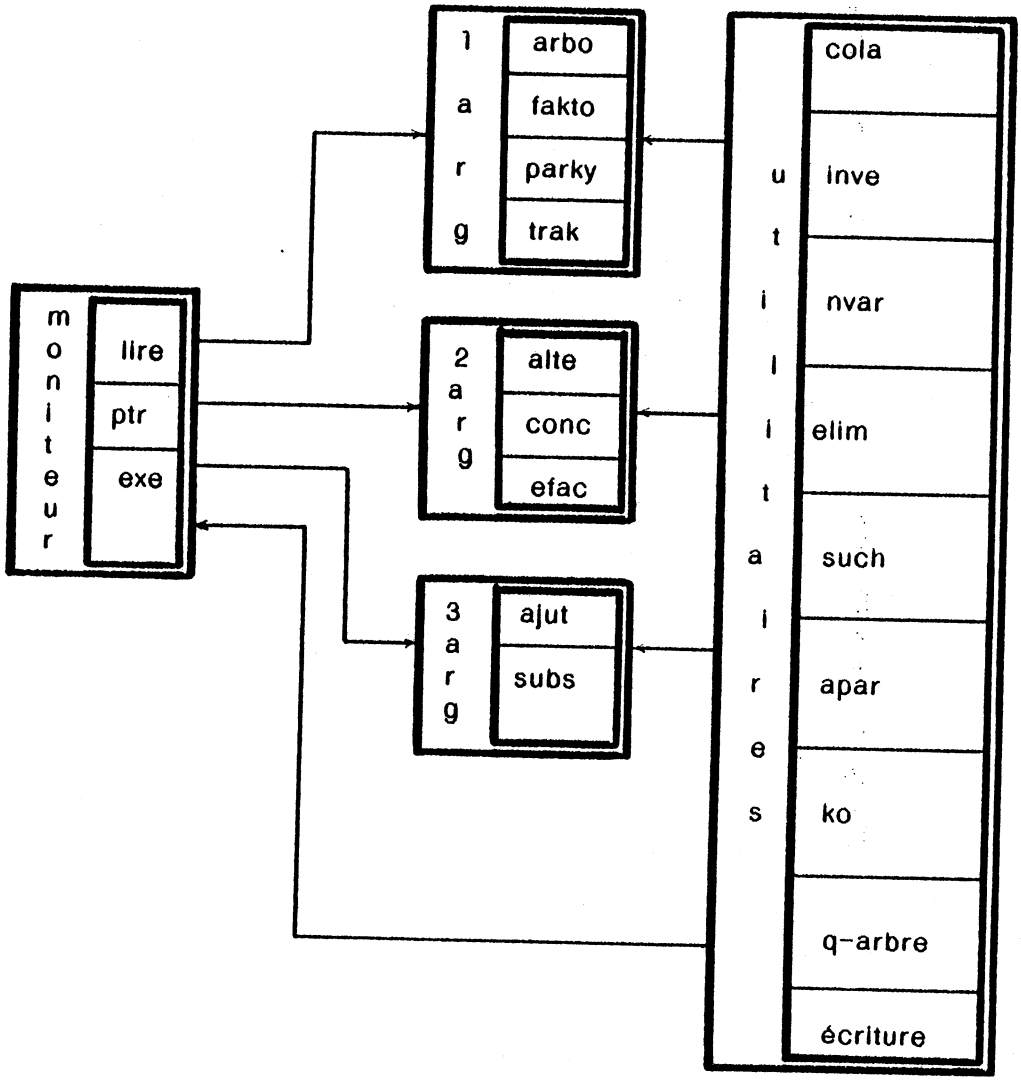
c'est-à-dire qu'on aura la chaîne x sans la sous-chaîne x_{kt}.

7. On dira qu'une chaîne x appartient à une autre chaîne y ssi x est une sous chaîne de y.

8. Etant donnée une sous-chaîne x_{kt} de x(n), on appellera <u>sous-chaîne tête</u> la sous-chaîne $x_{1(k-1)}$ et <u>sous-chaîne queue</u> la sous-chaîne $x_{(t+1)n}$.

2. ORGANISATION DU SYSTEME.

Le système dans son état actuel comporte essentiellement trois parties: (1) un moniteur, (2) des utilitaires et (3) l'ensemble des programmes de traitement qui réalisent les opérations décrites plus haut et qui sont classées selon leur nombre d'arguments en trois groupes. Voici un schéma de l'ensemble.



(1) Moniteur.

Les fonctions du moniteur sont: la reconnaissance des commandes tapées par l'utilisateur, un prétraitement en vue de générér une forme normalisée et enfin l'exécution de l'opération demandée.

(2) Utilitaires.

On a classé parmi les progammes utilitaires tout sous-programme qui est utilisé par plusieurs programmes de traitement. Si x est une chaîne, alors

- COLA: trouve l'élément cola de x.
- INVE: fait l'inversion de x.
- NVAR: teste si x est la chaîne vide.
- ELIM: extrait une sous-chaîne de x.
- SUCH: trouve une sous-chaîne de x.
- APAR: teste l'appartenance d'une sous-chaîne à x.
- KO : concatène à x une chaîne y.
- QARB: construit le q-arbre de x.
- ECRITURE: réalise les modifications nécessaires pour la sortie.

(3) Traitement.

Ce sont les sous-programmes qui réalisent les opérations sur les E-graphes. Si G est un E-graphe, alors

- ARBO: construit l'arborescence associée de G.
- FAKTO: factorise G.
- PARKY: fournit le parcours canonique de G.
- TRAK: produit toutes les trajectoires de G.
- ALTE: met en parallèle (par le bas) un E-graphe sur G.
- CONC: concatène à G un autre E-graphe.
- EFAC: efface une sous-structure de G.
- AJUT: met en parallèle (par le bas) un E-graphe sur une sous-structure de G.
- SUBS: remplace une sous-structure de G par un E-graphe.

3. P R O G R A M M E S.

3.1. MONITEUR.

Le moniteur du système comporte trois sous-programmes: LIRE, PTR (prétraitement) et EXE (exécution). Il lit la chaîne d'entrée (LIRE), vérifie que celle-ci soit une chaîne bien formée, tout en produisant une forme normalisée (PTR) à partir de laquelle EXE pourra trouver l'opération à effectuer et passer à son exécution.

3.1.1. LIRE(*Q).

Ce sous-programme lit la chaîne,caractère par caractère. Tout caractère reservé de PROLOG est ici transformé en un code alphabétique. LIRE détecte aussi la fin de "l'unité" (caractère ";") de même que la fin de traitement (caractère "?"). Le résultat est unifié à *Q.

3.1.2. PTR(*U,*V).

C'est un sous-programme de prétraitement. Au moyen d'une petite grammaire de métamorphose, PTR reconnaît l'opération à effectuer et le(s) E-graphe(s) donné(s) comme argument(s) et contenu(s) dans la chaîne *U. Le résultat est unifié à *V.

3.1.3. EXE(*C,*LAR).

*C et *LAR sont deux paramètres d'entrée. Ils sont unifiés avec l'opération à effectuer ("commande") et une liste de E-graphes ("liste d'arguments"), respectivement.

EXE cherche, avec ces deux paramètres, la première clause qui puisse être unifiée à *C et *LAR. Une fois trouvée, cette clause appelle le sous-programme qui réalisera l'opération, puis elle imprime le résultat et éventuellement s'appelle elle-même avec *LAR diminué d'autant d'arguments que le sous-programme qui a réalisé l'opération en a utilisé. Si la commande n'existe pas ou si le nombre de paramètres est erroné, on sort un message.

3.2. U T I L I T A I R E S.

3.2.1. COLA(*T,*Q,*GR) (trouver l'élément COLA).

COLA parcourt une chaîne *T= T_{1k} et unifie T_k à *Q. Le résultat d'avoir extrait T_k à *T est unifié à *GR. formellement

$$COLA(*T,*Q,*GR) \iff [*T=T_{1k} \;\&\; *Q=T_k \;\&\; *GR=T_{1(k-1)}] \;\vee\; [*T=T1 \;\&\; *Q=T1 \;\&\; *GR=T1]$$

3.2.2. INVE(*G,*R) (inverser une chaîne).

Etant donné une chaîne *G=G_{1k}, INVE réalise l'inversion de *G et le résultat est unifié à *R. On doit tenir compte des parenthèses puisqu'on doit les transformer. Voici la définition.

$$INVE(*G,*R) \iff (\forall i<=n)[[G_i \notin \{(,)\} \iff R_{n-i+1} = G_i] \;\&\; [G_i = (\;] \iff [R_{n-i+1} =)] \;\&\; [G_i =)] \iff [R_{n-i+1} = (\;]]$$

3.2.3. NVAR(*T) (non-variable).

NVAR vérifie que *T n'est pas la chaîne vide.

3.2.4. SUCH(*A,*T,*Q,*G) (sous-chaîne).

Etant données deux chaînes *T et *G, SUCH divise *G en trois parties: la sous-chaîne tête *A, la sous-chaîne *T et la sous-chaîne queue *Q.

$$SUCH(*A,*T,*Q,*G) \iff (*G = *A.*T.*Q) \;\&\; (\forall *A',*Q')[*A' \text{ est une sous-chaîne de } *A) \Rightarrow (*G \neq *A'.*T.*Q')]$$

3.2.5. ELIM(*T,*G,*R) (éliminer).

Si *G est une chaîne, ELIM efface une sous-chaîne *T de *G pourvu que *T≠ *G.

ELIM(*T,*G,*R) <=> SUCH(*A,*T,*Q,*G) & *R= *A.*Q

3.2.6. APAR(*T,*G) (appartenir).

APPART teste si *T appartient à *G.

3.2.7. KO(*Y,*X,*Z) (concaténer).

KO effectue la concaténation de deux termes *Y et *X et unifie le résultat à *Z. *Y ou *X peuvent être de variables. Evidemment, si *Y et *X sont toutes les deux des variables, le résultat sera aussi une variable.

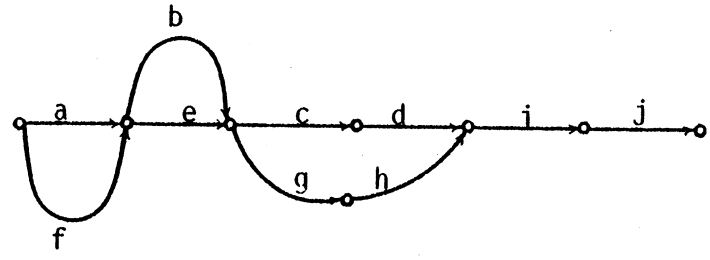
3.2.8. QARB(*G,*R) (quasi-arborescence).

Ce sous-programme construit un arbre *R, appelé une quasi-arborescence, à partir d'un E-graphe *G. Cet arbre est une forme "normalisée" qui nous servira de support pour d'autres opérations. Le nom de quasi-arborescence est dû au fait que c'est une arborescence associée à un E-graphe, la seule différence étant que l'on garde les symboles + qui jouent cette fois-ci le rôle de sentinelles.

Etant donné que l'algorithme est essentiellement le même que celui de ARBO où on construit une arborescence associée, je ne m'attarde pas à donner des détails et je montre plutôt un exemple.

Exemple. Soit G= (a+f)((b+e)c+gh)dij

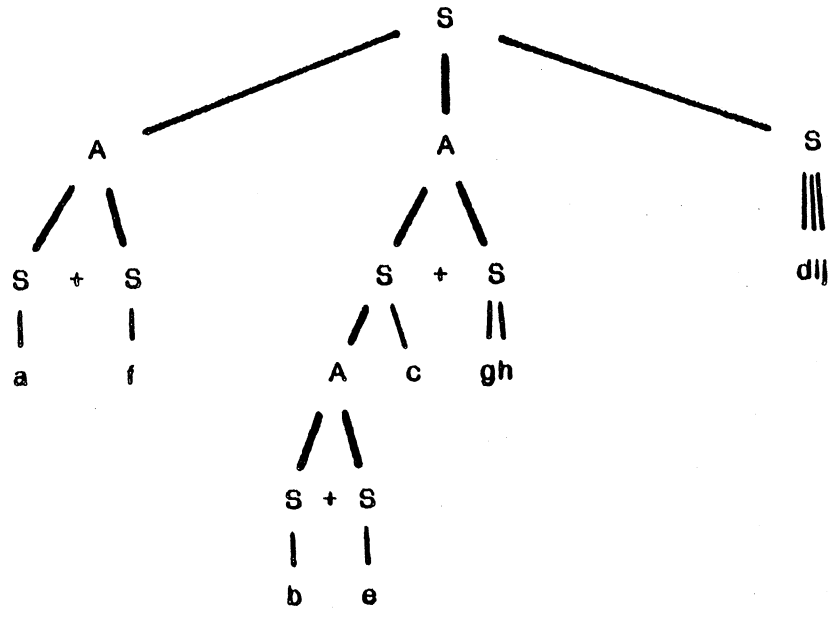
G =



la transformation realisée par QARB produira l'arbre

S(A(S(a)+S(f))A(S(A(S(b)+S(e))c)+S(gh))S(dij))

graphiquement.



3.3. T R A I T E M E N T.

3.3.1. ARBO(*G,*R) (arbre associé)

ARBO construit une arborescence associée à un E-graphe *G. Cette arborescence est unifiée à *R. L'algorithme utilise une grammaire de métamorphose pour effectuer la transformation.

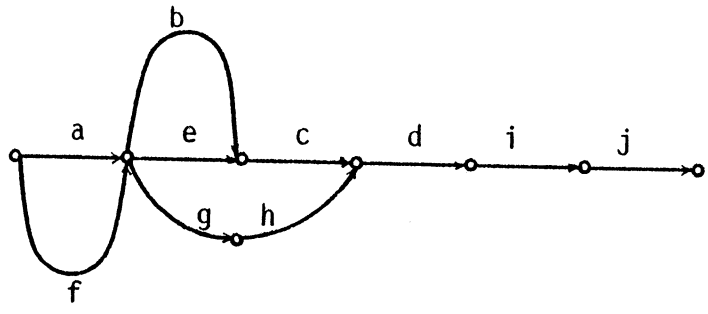
ARBO est composé de trois parties: (1) la définition d'un E-graphe régulier; (2) la recherche de seg séquentiels et (3) l'analyse d'un faisceau.

(1) Définition d'un E-graphe régulier (AGCO). Cette partie analyse la chaîne d'entrée (AEG) pour vérifier que c'est un E-graphe régulier. Au fur et à mesure que l'analyse progresse, ARBO recherche les seg séquentiels au moyen d'ASEK qui fournit, pour chaque seg séquentiel rencontré, la liste de ses arcs. Puis ARBO transforme cette liste en feuilles d'un arbre de racine et (fonction Θ1). Il recherche enfin les faisceaux.

(2) Recherche des seg séquentiels (ASEK). On parcourt le E-graphe et chaque seg séquentiel est transformé en une liste dont les éléments sont toutes les etiquettes des arcs du seg séquentiel.

(3) Analyse d'un faisceau. C'est la réalisation de la fonction Θ2. Elle transforme le faisceau en un arbre de racine ou dont les fils seront les étiquettes des arcs du faisceau.

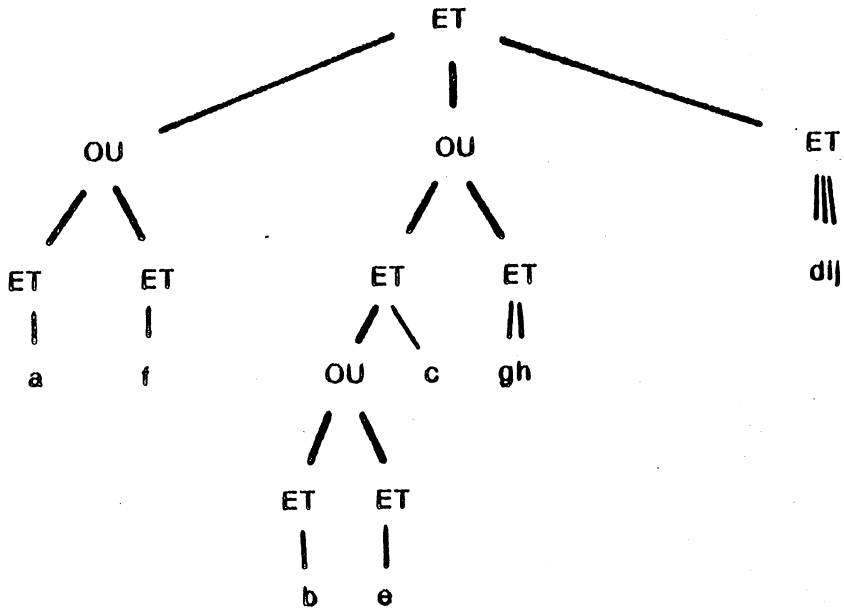
Exemple: Soit G= (a+f)((b+e)c+gh)dij



la transformation réalisée par ARBO produira l'arbre

ET(OU(ET(a)ET(f))OU(ET(OU(ET(b)ET(e))c)ET(gh))ET(dij))

graphiquement.



3.3.2. FAKTO(*G,*R,gauche) (factorisation).

Soit *G un E-graphe, alors FAKTO fournit la factorisation gauche de *G. Le résultat est unifié à *R. Une factorisation droite,

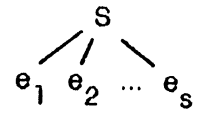
FAKTO(*G,*R,droite)

est obtenue en inversant la chaîne *G, en appliquant sur elle l'algorithme de factorisation gauche et en effectuant de nouveau une inversion pour retrouver le résultat.

L'algorithme comporte trois parties principales une fois construit le q-arbre de *G (QARB): (1) une factorisation générale sur un arbre, (2) une factorisation élémentaire sur des sous-arborescences et (3) un aplatissement du q-arbre résultant.

(1) Factorisation générale sur un arbre. Cette partie recherche, pour chaque niveau, des sous-arborescences "factorisables". Si $u_1,...u_k$ sont les sommets à un niveau j du q-arbre, alors si u_i=S (i= 1,...,k) est un sous-arbre de la forme

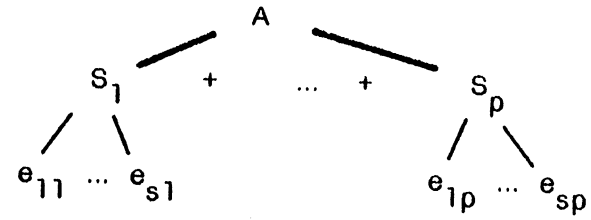
$\Delta 1$ =



i.e. tel que fils(u_i) sont des étiquettes, alors on passe au sous-arbre u_{i+1} (NINIV), avec i+1<=k, sinon on visite dans l'ordre les fils de u_i (DUTY).

Si u_i=A est de la forme

$\Delta 2$ =

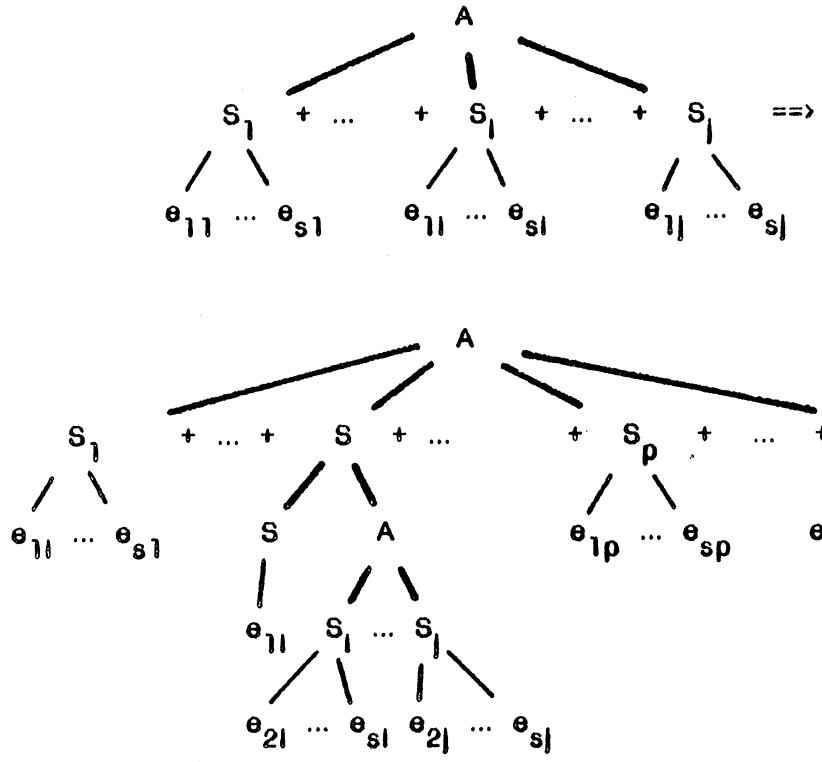


c'est-à-dire, (∀x)[x=fils(u_i)] => x=S & (∀y)[y=fils(fils(u_i))] => y∈E, alors on réalise sur u_i une factorisation élémentaire, sinon on visite dans l'ordre les fils de u_i.

(2) Factorisation élémentaire.

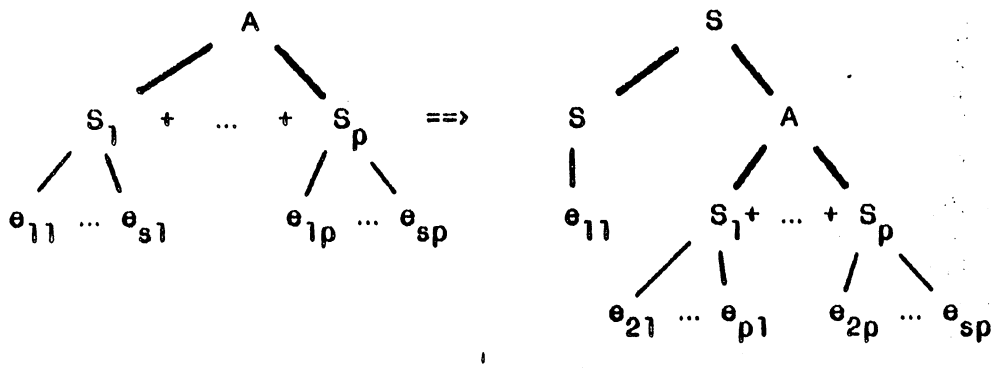
Si l'arborescence localisée par la factorisation générale n'est pas factorisable, elle reste évidemment inchangée. Autrement cette arborescence peut subir deux types de transformation:

I. Partielle. Graphiquement, la transformation est la suivante



avec $e_{11} = e_{1k} = \ldots = e_{1j}$ et $e_{1p} \neq e_{11}$ et $e_{1q} \neq e_{11}$ ($1<=k<=j$, $1<=p<=j$, $1<=q<=j$).

II. Totale. De façon analogue.

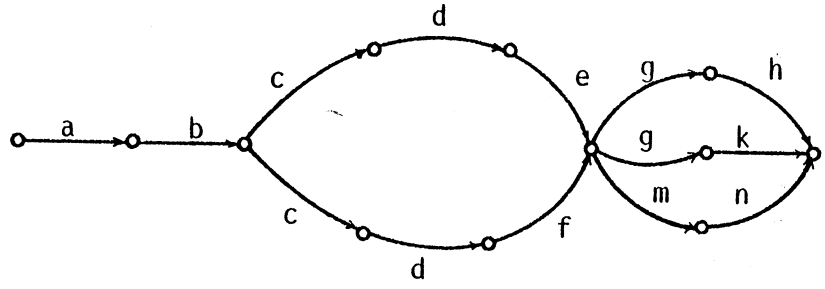


avec $e_{11} = \ldots = e_{1p}$.

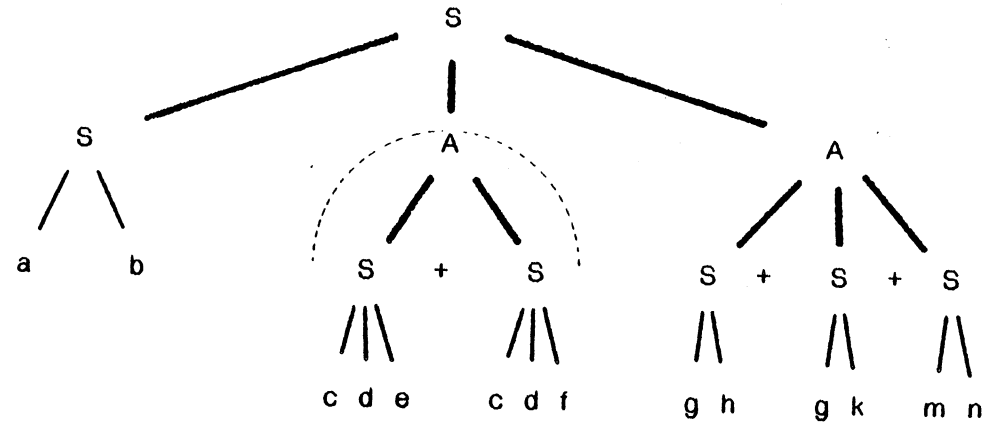
(3) Aplatissement du q-arbre. APLA(*G,*R)

Au moyen d'une petite grammaire de métamorphose, on réalise la transformation inverse de celle du q-arbre, i.e. APLA produit une chaîne *R étant donné un q-arbre *G.

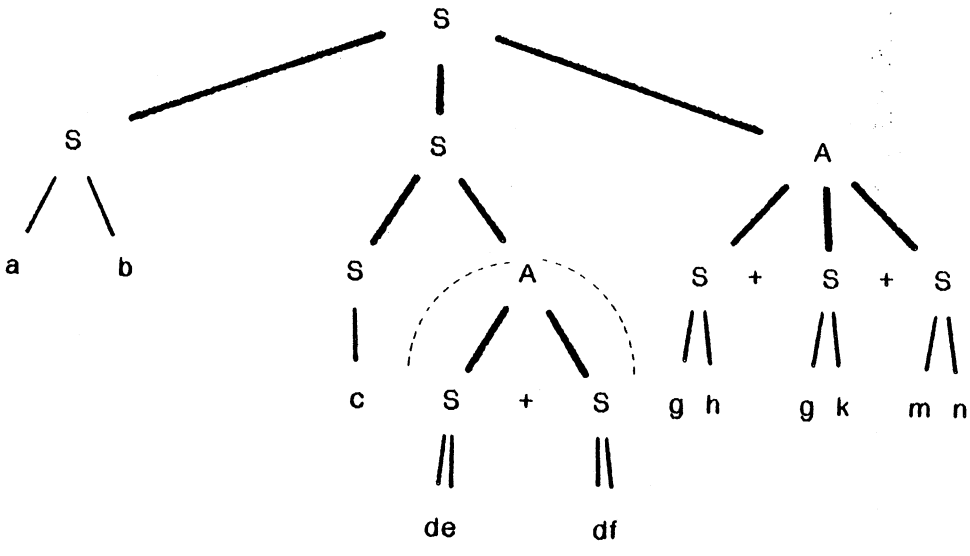
Exemple: Soit le E-graphe suivant à factoriser (la partie "active" de l'arbre est entourée).



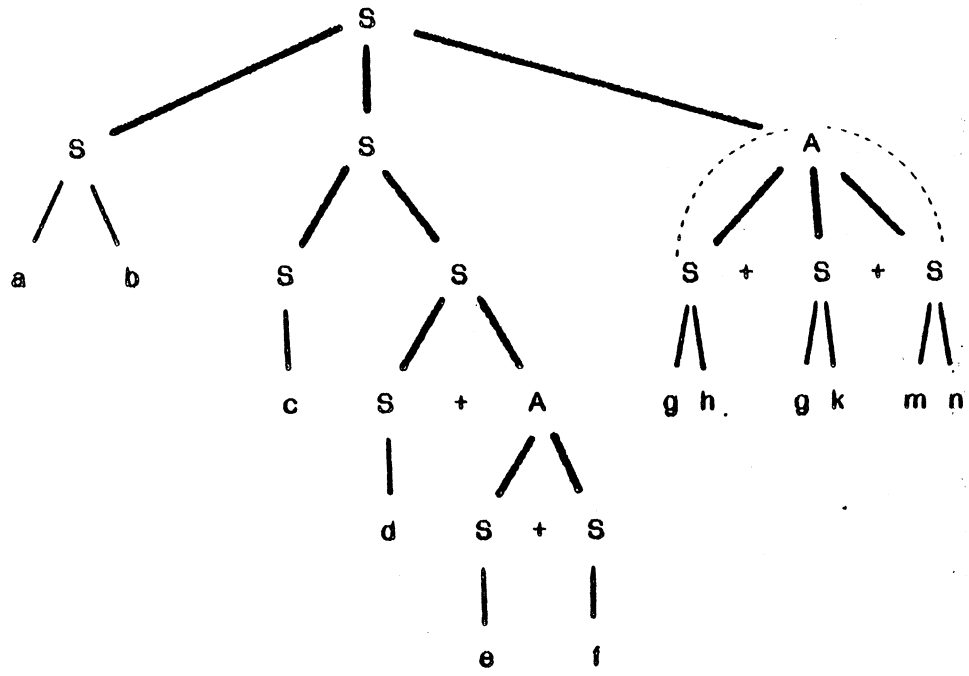
son q-arbre sera



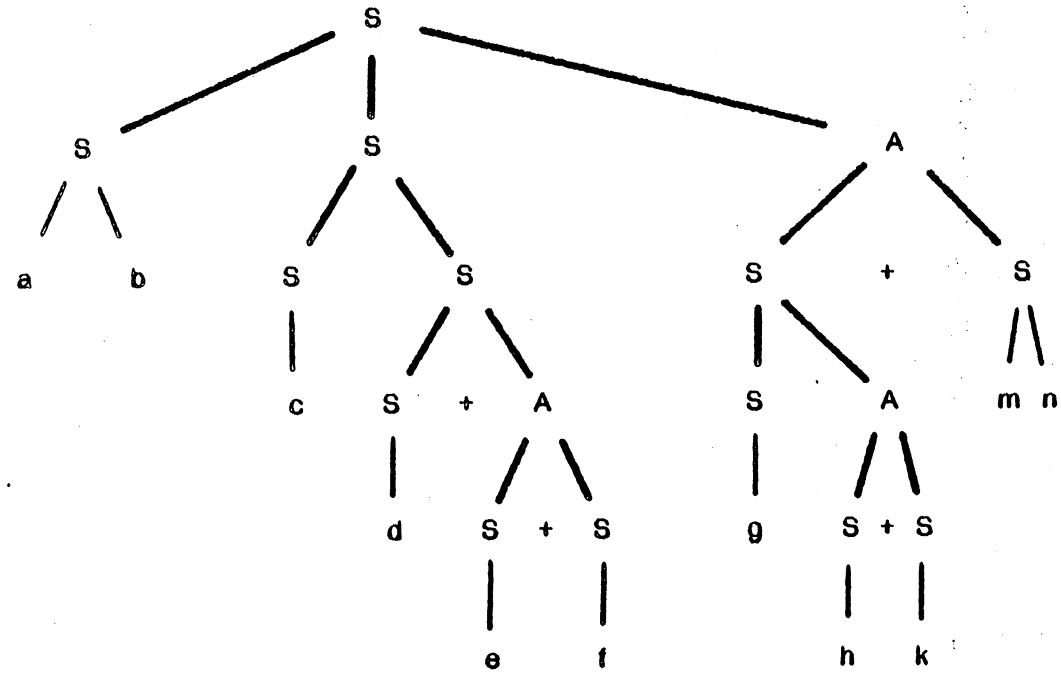
première transformation



deuxième transformation

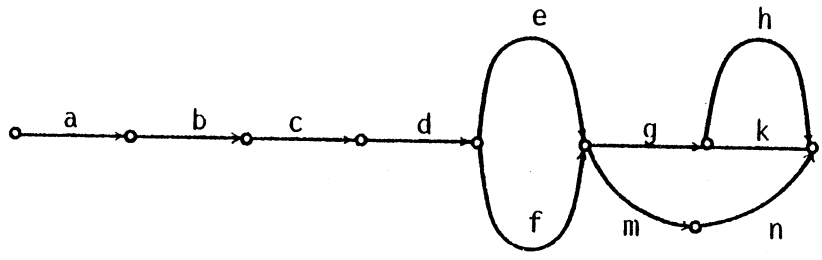


q-arbre final



et l'aplatissement du q-arbre produit finalement comme résultat

abcd(e+f)(g(h+k)+mn)



3.3.3. PARKY(*G,*R) (parcours).

Si *G est un E-graphe, PARKY unifie à *R la chaîne dont les éléments constituent le parcours canonique de *G. L'algorithme comporte trois parties: (1) génération de l'expansion de *G, (2) contrôle de la fusion de deux trajectoires et (3) fusion de deux trajectoires.

La partie (1) produit tout simplement exp(*G). La partie (2) "calcule" l'expression

$(\ldots((\gamma 1 \Theta \gamma 2) \Theta \gamma 3) \Theta \ldots \gamma n)$ (n= nbre de traj.)

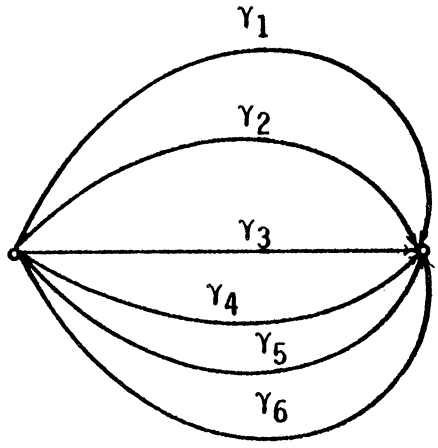
La partie (3) réalise l'opération de fusion Θ entre deux listes A et B. On se souviendra que cette opération concatène à A la sous-liste de B formée d'éléments hors de A.

Exemple: Soit le même E-graphe G=(a+f)((b+e)c+gh)dij. La partie (1) produit

$$\exp(G) = \Sigma\ \gamma i$$

avec
$\gamma 1$= abcdij $\gamma 4$= fbcdij
$\gamma 2$= aecdij $\gamma 5$= fecdij
$\gamma 3$= aghdij $\gamma 6$= fghdij

graphiquement



La partie (2) fournit le parcours

$\gamma 1 \Theta \gamma 2 \Theta \gamma 3 \Theta \gamma 4 \Theta \gamma 5 \Theta \gamma 6$ = abcdijeghf

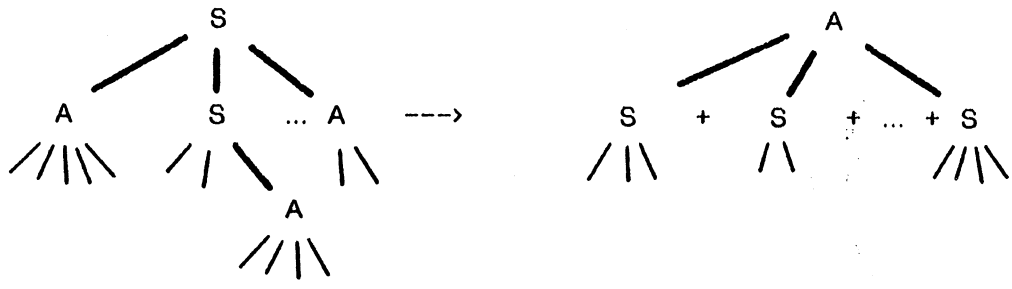
qui sera unifié à *R.

La partie (3) produit, par exemple pour $\gamma 1$ et $\gamma 2$

$\gamma 1 \Theta \gamma 2$ = abcdije

3.3.4. TRAK(*G,*T) (trajectoires).

Si *G est un E-graphe, TRAK construit un faisceau *T avec toutes les trajectoires de *G. L'algorithme construit d'abord un q-arbre et au moyen de transformations sur des sous-arborescences on produit un arbre à trois niveaux dont l'énumeration des feuilles en préordre fournit le résultat, i.e. de façon schématique,



On cherche donc à construire une arborescence Δ avec les propriétés suivantes:

- racine(Δ)= A
- (∀x)[x=fils(A)] => x∈{S,+}
- (∀y)[y=fils(fils(A))] => y∈E

Une fois qu'elle est obtenue, on l'aplatit pour avoir le E-graphe résultant.

TRAK est constitué de trois parties: (1) une recherche (par niveaux) de sous-arborescences à traiter, (2) une transformation d'un niveau donné et (3) un aplatissement du q-arbre.

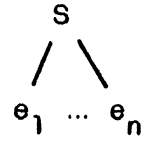
(1) Recherche par niveaux.

Cette partie cherche (NIVEL) un niveau j du q-arbre tel que tous les sous-arbres $u_1,...u_k$ à ce niveau satisfont la condition

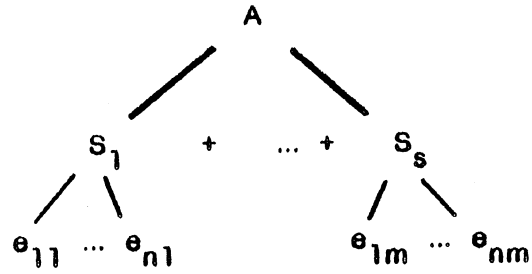
$$[racine(u_j)=S \ \& \ [(\forall x)[x=fils(u_j)] \Rightarrow x\in E]] \ V$$

$$[racine(u_j)=A \ \& \ [(\forall y)[y=fils(u_j)] \Rightarrow y\in\{S,+\}] \ \& \ [(\forall x)[x=fils(fils(u_j))] \Rightarrow x\in E]]$$

graphiquement, les sous-arbres doivent avoir soit la forme



soit la forme



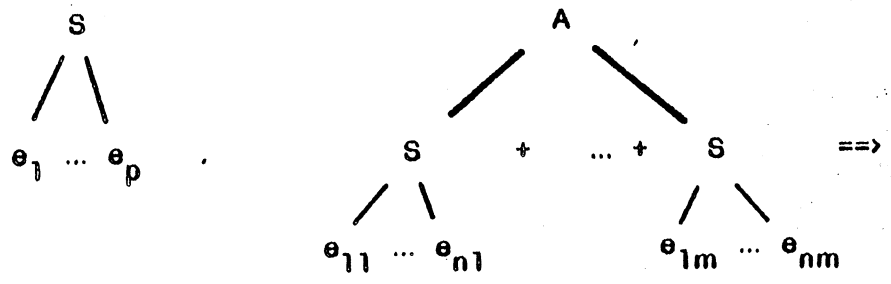
S'il existe un sous-arbre u_i ($1<=i<=k$) qui ne satisfait pas la condition, alors il y a un appel récursif (WORK) de NIVEL et u_i devient le nouveau q-arbre sur lequel on cherche un niveau j tel que tous ses sous-arbres satisfont la condition donnée plus haut. Si u_i n'existe pas, alors on passe à la

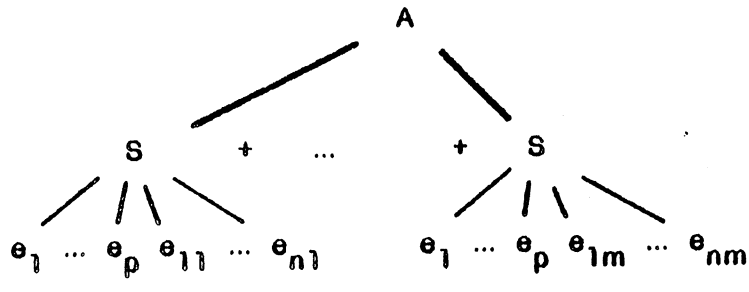
(2) Transformation d'un niveau.

On transforme deux à deux tous les sous-arbres du niveau. Il peut y avoir trois types de transformation selon la classe de sous-arbres. On les appellera

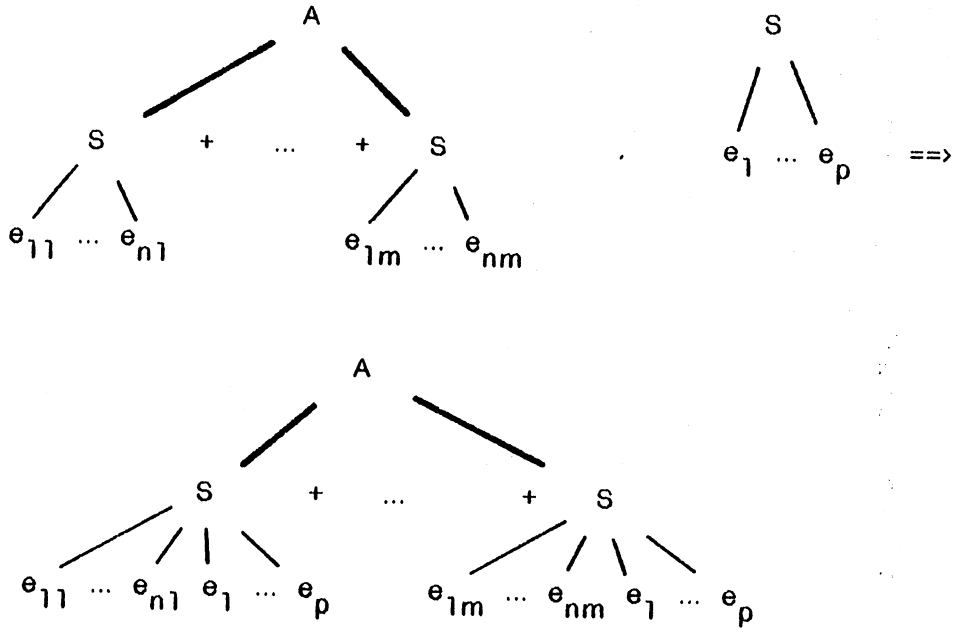
(I) SA réalisée par BELE
(II) AS réalisée par CELE
(III) AA réalisée par DELE

I. Transformation SA. Montrons-la graphiquement.

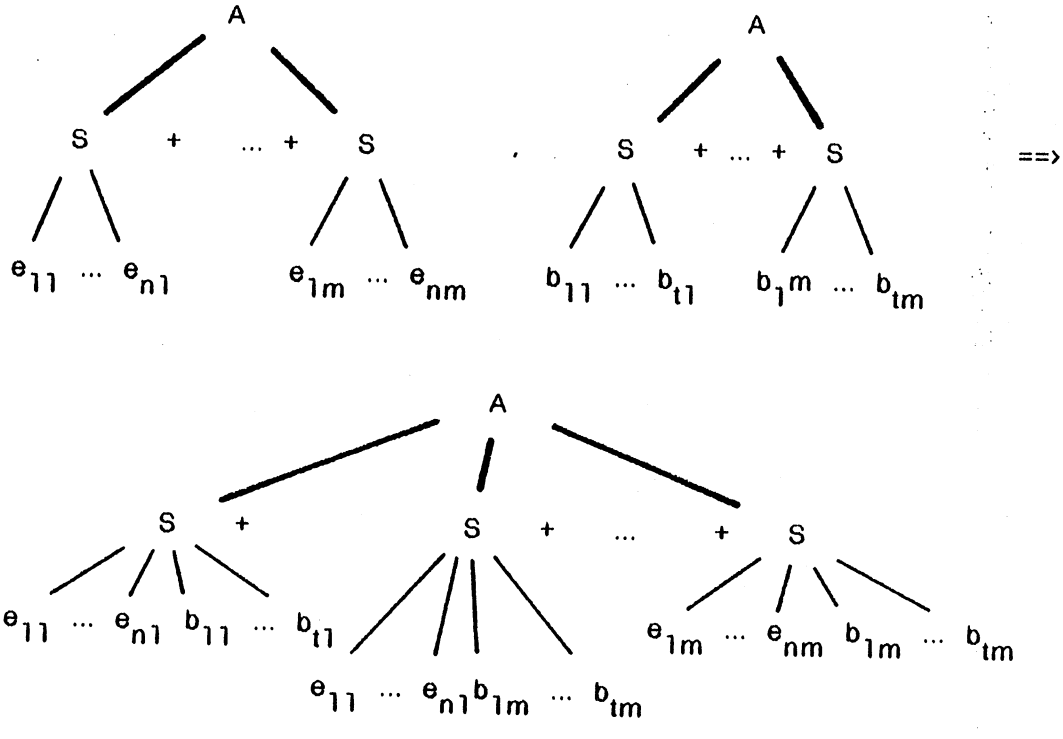




ii. Transformation AS.



iii. Transformation AA.



Ces transformations sur les sous-arbres $u_1,\ldots..u_k$ produisent un seul sous-arbre Aj avec la proprieté,

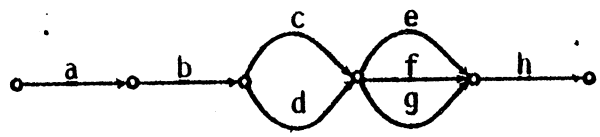
$$[racine(u_j)=A \;\&\; [(\forall y)[y=fils(u_j)] \Rightarrow y\in\{S,+\}] \;\&\; [(\forall x)[x=fils(fils(u_j))] \Rightarrow x\in E]]$$

On retourne au niveau $|-1\rangle=0$ et on remplace le noeud $père(u_j)$ par Aj. Lorsque l'on fait le remplacement au niveau 0, l'algorithme s'arrête.

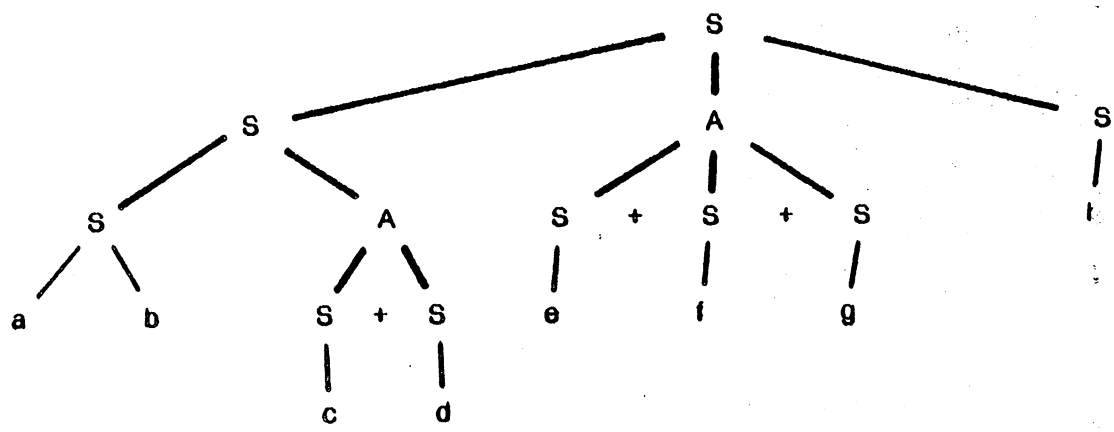
(3) Aplatissement du q-arbre.

L'énumération dans l'ordre (NETO) de toutes les feuilles de A1 produit le résultat recherché.

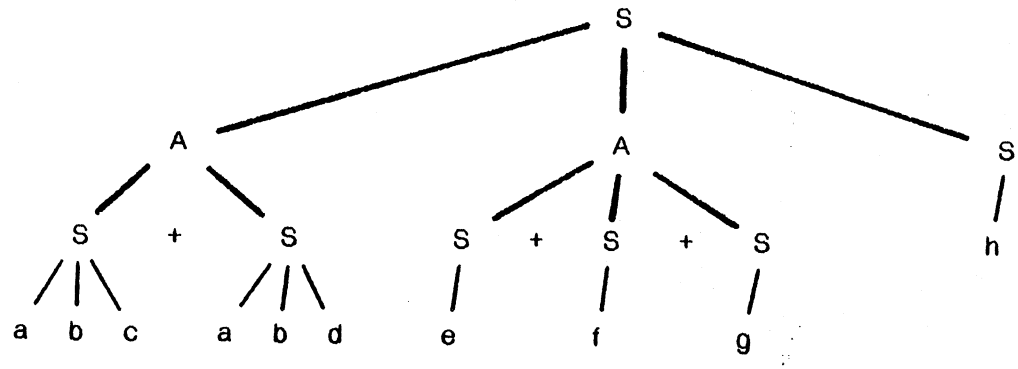
Exemple: Obtenons les trajectoires du E-graphe suivant: G= ab(c+d)(e+f+g)h



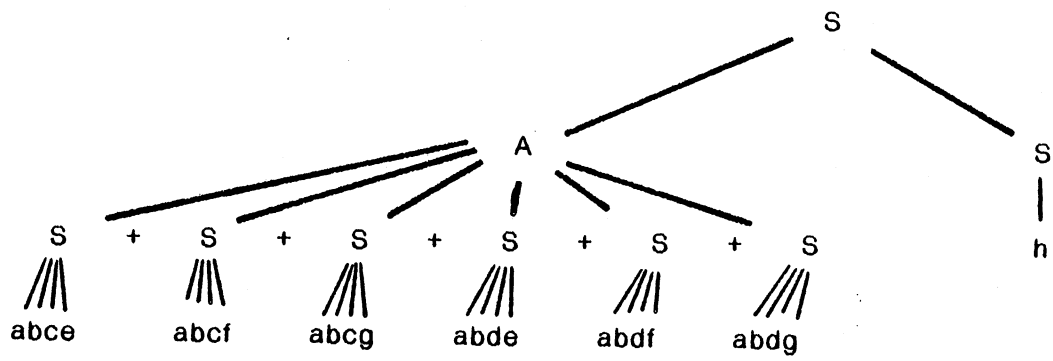
son q-arbre est



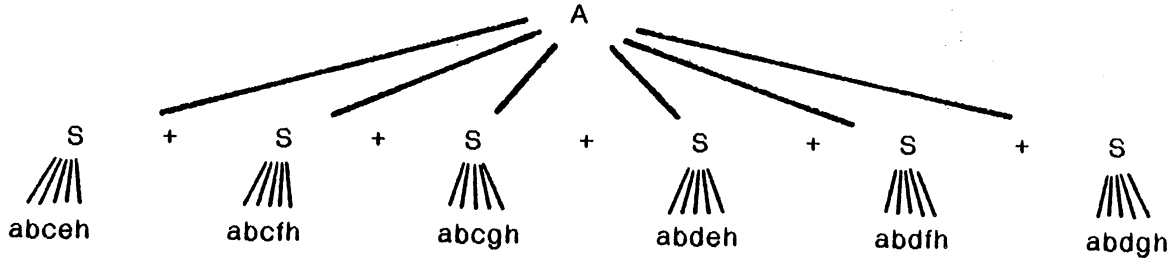
première transformation: SA



deuxième transformation: AA



troisième transformation: AS



et l'aplatissement produit finalement

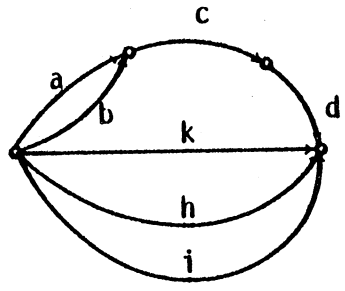
abceh + abcfh + abcgh + abdeh + abdfh + abdgh

3.3.5. ALT(*A,*B,*G) (alternation).

Etant donnés deux E-graphes *A et *B, ALT réalise l'alternation entre *A et *B. Le résultat l'unifie à *G.

L'algorithme consiste à concaténer les trois chaînes: *A, + et *B en les enfermant entre parenthèses. Si *A ou *B est une chaîne entre parenthèses, on les enlève pour réaliser la concaténation.

Exemple: si *A= (a+b)cd et *B= (k+h+l), ALT(*A,*B,*G) fournit comme résultat, *G= (a+b)cd+k+h+l.

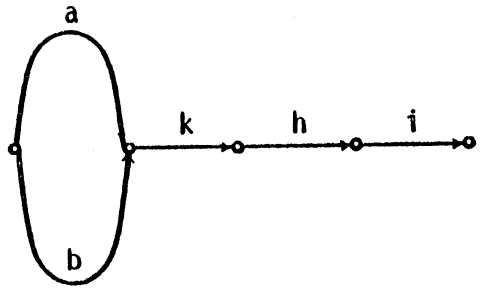


3.3.6. CONC(*X,*Y,*Z) (concaténation)

CONC réalise la concaténation entre deux chaînes *X et *Y. Il unifie le résultat à *Z.

Exemple: si *X= (a+b) et *Y= khl, CONC(*X,*Y,*Z) fournit comme résultat

*R= (a+b)khl.



3.3.7. EFFA(*E,*G,*R) (effacement).

Ce sous-programme efface un seg strict *E d'un E-graphe *G. Le résultat sera unifié à *R. Quatre parties composent l'algorithme: (1) la recherche de la classe du seg, (2) le test de forte isolabilité, (3) le test d'isolabilité moyenne et enfin (4) le test de faible isolabilité.

(1) Recherche de la classe du seg.

C'est le **moniteur** de l'algorithme. Il cherche à identifier la classe de *E d'après la caractérisation de chaque sous-structure.

L'ordre de recherche n'est pas arbitraire. On cherche d'abord *E dans la classe qui a les contraintes les plus fortes, celle des segfi. Si *E n'appartient pas à cette classe, on suppose alors qu'il est un segmi. Si *E non plus n'appartient pas à la classe des segmi, on teste s'il est un segif. Si on n'arrive pas à trouver la classe de *E, EFFA sort un message d'erreur.

(2) Test de forte isolabilité.

Deux cas peuvent se présenter: *E est un seg externe ou non-externe. Dans le premier cas, il doit être une sous-chaîne de *G et le nombre de parenthèses ouvrantes et fermantes dans la sous-chaîne tête, *AV, (de même que dans la sous-chaîne queue ,*AP) doit être égal.

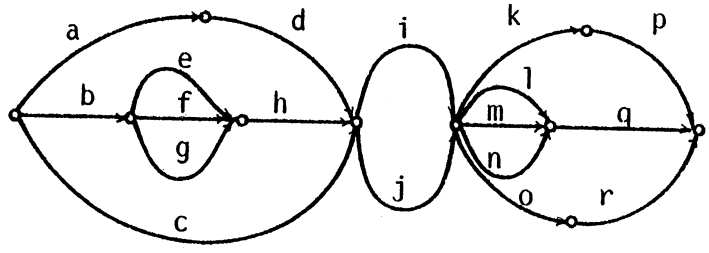
Si *E est non-externe, il doit être aussi une sous-chaîne de *G telle que tête(*AP)$\notin${+,)} et cola(*AV)$\notin${+,(}

Effacement.

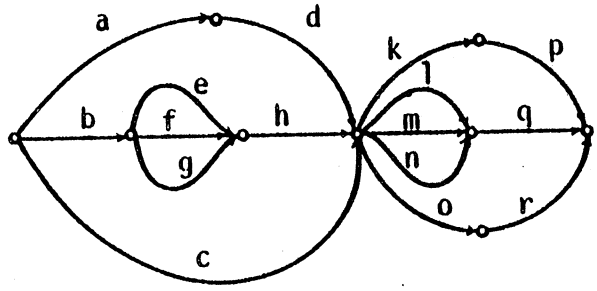
Si *E est un segfi, alors si l'entrée ou la sortie de *G est dans *E, on extrait *E de *G tout simplement. Sinon, on concatène *AV et *AP.

Exemple: soit le E-graphe *G= (ad+b(e+f+g)h+c)(i+j)(kp+(l+m+n)q+or)

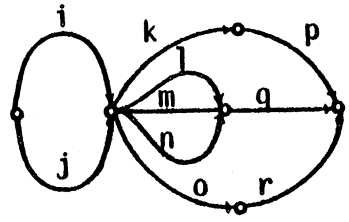
G =



- si *E= (i+j), alors
  *R= (ad+b(e+f+g)h+c)(kp+(l+m+n)q+or)



- si *E= (ad+b(e+f+g)h+c), alors
  *R= (i+j)(kp+(l+m+n)q+or)



(3) Test d'isolabilité moyenne.

Il y a également deux cas à traiter: les seg séquentiels et les faisceaux. Si *E est un seg séquentiel, il est forcément une sous-chaîne de *G et tête(*AP)$\in${(.),+} et cola(*AV)$\in${(.),+}.

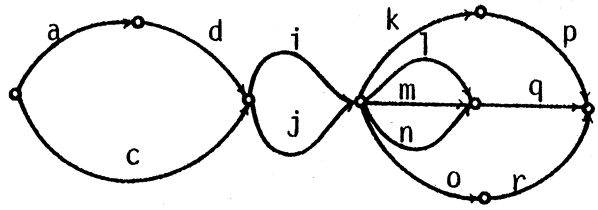
Si *E est un faisceau, alors chaque trajectoire t_i doit être un seg séquentiel de *G et tête(*AP)$\in${(.),+} et cola(*AV)$\in${(.),+} et tête(t_i)<v tête(t_{i+1}). (1<=i<nbre de trajectoires), c'est-à-dire pour tout t_i, t_{i+1} est dans la sous-chaîne queue.

Effacement.

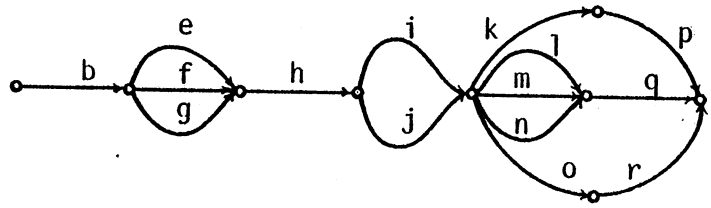
On extrait la sous-chaîne *E de *G et on concatène *AV et *AP.

Exemple: soit le E-graphe *G de (2),

- si *E= b(e+f+g)h
  *R= (ad+c)(i+j)(kp+(l+m+n)q+or)



- si *E= (ad+c)
  *R= b(e+f+g)h(i+j)(kp+(l+m+n)q+or)



(4) Test d'isolabilité faible.

On distingue encore deux cas: celui des seg séquentiels et celui des faisceaux. Dans le premier cas, pour que *E soit un segif, il faut et il suffit que dans *AV et dans *AP existent des sous-chaînes *AVs et *APs, les plus petites, telles que

*AVs.*E.*APs est un segmi.

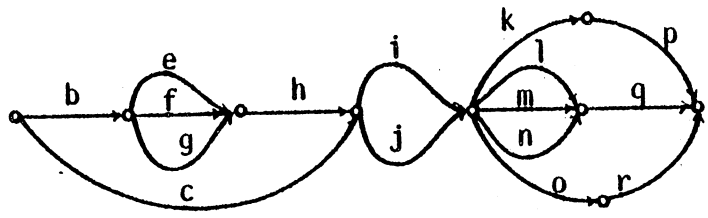
Si *E est un faisceau, chaque trajectoire t_i doit être un segmi et il existe une sous-chaîne *AVs dans *AV et une sous-chaîne *APs dans *AP qui satisfont la condition donnée plus haut. Dans ce cas-ci, *E doit être une sous-structure totale.

Effacement.

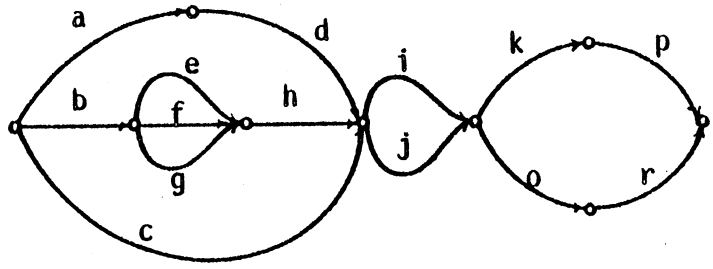
On efface le segmi: *AVs.*e.*APs

Exemple: soit toujours le E-graphe de (2)

- si *E= a, alors on efface *E'= ad
  *R= (b(e+f+g)h+c)(i+j)(kp+(l+m+n)q+or)



-si *E= (l+m+n), alors on efface *E'= (l+m+n)q
*R= (ad+b(e+f+g)h+c)(i+j)(kp+or)



3.3.8. AJOUT(*G,*SG,*GA,*R).

Etant donné un E-graphe *G et un seg *SG de *G, AJOUT met en parallèle, par le bas, sur *SG, un E-graphe *GA. Le résultat est unifié à *R.

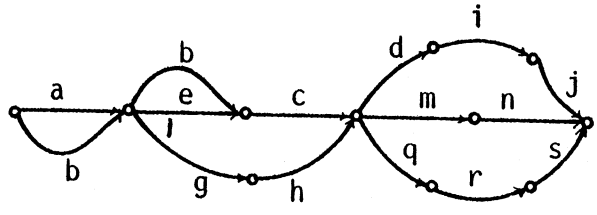
L'algorithme cherche l'occurrence la plus superficielle de *SG. On l'extrait, on ajoute par le bas *GA et enfin on la réinsère. Si *SG= *G, on effectue alors une alternation.

Exemple: soient

*G= (a+f)((b+e)c+gh)dij
*SG= dij
*GA= (mn+qrs)

AJOUT produira

*R= (a+f)((b+e)c+gh)(dij+mn+qrs)



3.3.9. SUBS(*G,*SG,*SR,*R) (substitution).

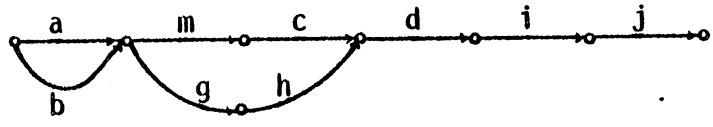
Soient *G et *SR deux E-graphes et *SG un seg de *G. SUBS réalise le remplacement de *SG par *SR. On trouvera le résultat unifié à *R. L'algorithme cherche l'occurrence la plus superficielle de *SG. On l'extrait et on insère *SR à sa place.

Exemple: soient

*G= (a+f)((b+e)c+gh)dij
*SG= (b+e)
*SR= m

SUBS fournira comme résultat

*R= (a+f)(mc+gh)dij



4. L I S T E S.

Voici la liste détaillée de chaque partie principale:

4.1. MONITEUR.

```
FILE: MONITY    PROLOG    A    VM/CMS/SP CHAMPOLLION/GRENOBLE 8101+PTF 20/8/8

-BOOLISTE!

** >>>>     PROGRAMME DE TRAITEMENT POUR LES E-GRAPHES     <<<< **.

-AJOP(".",1,"X¬(X¬X)")!

* ==========================================> MONITEUR *.


+MON -LG -SORM("+") -LG -LIRE(*U) -PTR(*U,*C.*LAR)
                                  -EXE(*C,*LAR)     -IMP.
+MON -MON.

+LIRE(*Q) -LOG(*K) -TRA(*K.NIL,*T) -SUITPRO(*T,*Q).

+TRA("?",SI) -/          (-(MON)) -T(FIN) -IMP.
+TRA("(",PG) -/.        +TRA(";",PV)  -/.
+TRA(")",PD) -/.        +TRA(",",VR)  -/.
+TRA(" ",BK) -/.        +TRA("+",PS)  -/.
+TRA(*K.NIL,*K).

+SUITPRO(PV,PV.NIL) -/.
+SUITPRO(BK,BK.*U)  -/ -LIRE(*U).
+SUITPRO(*T,*T.*Q)      -LO(*K) -TRA(*K.NIL,*A) -SUITPRO(*A,*Q).


* VERIFICATION DE SYNTAXE ET GENERATION DE LA FORME NORMALISEE *.

+PTR(*U,*V) -SYN(UNITES(*V).NIL,*U)       -IMPR(*V) -/.
+PTR(*U,*V) -SORM(" ERREUR DE SYNTAXE ") -IMP.

:UNITES(*Q) == :OPER(*U) ¬BK :LIARG(*U,*Q) ¬PV -/.
:UNITES(*Q) ==.

:OPER(*U) ==  ¬*L -LETTRE(*L) -/ :ROPER(*V) -UNIV(*U,(*L.*V).NIL).
:ROPER(*L.*U) == ¬*L -LETTRE(*L) -/ :ROPER(*U).
:ROPER(NIL)   == .

:SEP == ¬BK -/.                                  :SEP == ¬VR.

:LIARG(*U,*J.*Q)     == :EGRC(*A) :SEP -/ -KO(*A,NIL,*P)
                                          :LIARG(G(*P),*Q).
:LIARG(*U,*J.G(*Q)) == :EGRC(*A)          -KO(*A,NIL,*Q).
```

```
FILE: MONITY   PROLOG   A   V4/CMS/SP CHAMPOLLION/GRENOBLE 8101*PTF 20/8/81


:EGRC(*R) == :EGR(*A) :REGR(*A,*R).
:EGR(*R)  == ¬*A -ARBRE(*A) -/ :SEK(*A,*Q)     :ALTE(*Q,*W)
                                               :SEK(*W,*R) -/.
:EGR(*R)  == ¬PG :PARL(*A) ¬PD -KO(PG,*A,*RT) -KO(*RT,PD,*RS)
                                               :SEK(*RS,*R).

:REGR(*T,*R) == :EGR(*A) -/ -KO(*T,*A,*Z) :REGR(*Z,*R).
:REGR(*T,*T) ==.

:PARL(*R)      == :EGRC(*A) ¬PS     -KO(*A,PS,*B) :RPARL(*B,*R).
:RPARL(*T,*R) == :EGRC(*A) ¬PS -/ -KO(*A,PS,*B) :RPARL(*B,*S)
                                                -KO(*T,*S,*R).
:RPARL(*T,*R) == :EGRC(*S)         -KO(*T,*S,*R).

:SEK(*T,*R) == ¬*A -ARBRE(*A) :SEK(*A,*W) -KO(*T,*W,*R) -/.
:SEK(*T,*T) ==.

:ALTE(*T,*T.*R) == ¬PG :PARL(*A) ¬PD -/ -KO(PG,*A,*B)
                                        -KO(*B,PD,*R).
:ALTE(*T,*T)    ==.
```

```
FILE: EXCJT     PROLOG     A     VM/CMS/SP CHAMPOLLION/GRENOBLE 3101+PTF 20/8/

* ================================> EXECUTION DES COMMANDES *.

*    1 ARGUMENT    *.

+EXE(ARBE,G(*A1).*S)   -/ -ARBO(*A1,*R)       -SOR(*R)
                                              -EXE(ARBE,*S).
+EXE(ARBE,G(*A1))      -/ -ARBO(*A1,*R)       -SOR(*R).
+EXE(FACTO,G(*A1).*S)  -/ -FAKTO(*A1,*R,O)    -SOR(*R)
                                              -EXE(FACTO,*S).
+EXE(FACTO,G(*A1))     -/ -FAKTO(*A1,*R,O)    -SOR(*R).
+EXE(FACTG,G(*A1).*S)  -/ -FAKTO(*A1,*R,G)    -SOR(*R)
                                              -EXE(FACTG,*S).
+EXE(FACTG,G(*A1))     -/ -FAKTO(*A1,*R,G)    -SOR(*R).
+EXE(PARC,G(*A1).*S)   -/ -PARKY(*A1,*R)      -SOR(*R)
                                              -EXE(PARC,*S).
+EXE(PARC,G(*A1))      -/ -PARKY(*A1,*R)      -SOR(*R).
+EXE(TRAJ,G(*A1).*S)   -/ -TRAK(*A1,*R)       -SOR(*R)
                                              -EXE(TRAJ,*S).
+EXE(TRAJ,G(*A1))      -/ -TRAK(*A1,*R)       -SOR(*R).

*    2 ARGUMENTS    *.

+EXE(ALTE,G(*A1).G(*A2).*S) -/ -ALT(*A1,*A2,*R)
                                -EXE(ALTE,G(*R).*S).
+EXE(ALTE,G(*A1).G(*A2))    -/ -ALT(*A1,*A2,*R)      -SOR(*R).
+EXE(CONC,G(*A1).G(*A2).*S) -/ -CONC(*A1,*A2,*R)
                                -EXE(CONC,G(*R).*S).
+EXE(CONC,G(*A1).G(*A2))    -/ -CONC(*A1,*A2,*R)     -SOR(*R).
+EXE(EFAC,G(*A1).G(*A2).*S) -/ -EFFA(*A2,*A1,*R)
                                -EXE(EFAC,G(*R).*S).
+EXE(EFAC,G(*A1).G(*A2))    -/ -EFFA(*A2,*A1,*R) -SOR(*R).

*    3 ARGUMENTS    *.

+EXE(AJUT,G(*A1).G(*A2).G(*A3).*S) -/ -AJOUT(*A1,*A2,*A3,*R)
                                       -EXE(AJUT,G(*R).G(*A2).*S).
+EXE(AJUT,G(*A1).G(*A2).G(*A3))    -/ -AJOUT(*A1,*A2,*A3,*R)
                                       -SOR(*R).
+EXE(SUBS,G(*A1).G(*A2).G(*A3))    -/ -SUBS(*A1,*A2,*A3,*R)
                                       -SOR(*R).
+EXE(*C,*G) -SORM(" CMDE INEX OU NMBRE. PARM. ERRONE ") -IMP.
```

4.2. UTILITAIRES.

```
FILE: UTILY    PROLOG    A    VM/CMS/SP CHAMPOLLION/GRENOBLE 8101•PTF 20/8/81

* ==========================================> UTILITAIRES *.

*   RECHERCHE DE LA QUEUE   *.

•COLA(*T.*Q,*R,*GR)          -/ -KOLA(*T.*Q,*R,*GR).
•COLA(*T,*T,*T).
•KOLA(*T.*Q.*P,*CO,*T.*G) -/ -KOLA(*Q.*P,*CO,*G).
•KOLA(*T.*Q,*Q,*T).

*   INVERSION D'UNE LISTE   *.

•INVE(*G,*R) -VAR(*G)              -/.
•INVE(*G,*R) -COLA(*G,NIL,*GI) -/ -INVO(*GI,*N) -KO(*N,NIL,*R).
•INVE(*G,*R) -INVO(*G,*R).

•INVO(PG.*Q,*R) -INVO(*Q,*S) -/ -KO(*S,PO,*R).
•INVO(PO.*Q,*R) -INVO(*Q,*S) -/ -KO(*S,PG,*R).
•INVO(*T.*Q,*R) -INVO(*Q,*S) -/ -KO(*S,*T,*R).
•INVO(PO,PG)     -/.
•INVO(*T,*T).

*   NEGATION DU PREDICAT "VAR"   *.

•NVAR(*T)  -VAR(*T) -/ -IMP.
•NVAR(NIL)          -/ -IMP.
•NVAR(*T).

*   ELIMINATION D'UNE SOUS LISTE T DE G  (T¬=G)   *.

•ELIM(*T,*T,*R) -/ -IMP.
•ELIM(*T,*G,*R)    -APAR(*T,*G) -/ -ELOM(*T,*G,*R).
•ELIM(*T,*G,*G).

•ELOM(*T.*Q,*T.*S,*R)    -SEEK(*Q,*S)  -/ -ERAS(*T.*Q,*T.*S,*R).
•ELOM(*T,*T.*S,*S)    -/.
•ELOM(*T.*Q,*G,*R)    -/ -SUCH(*A,*T,*S,*G)
                         -ENCO(*A,*T.*Q,*S,*R).
•ELOM(*T,*G,*R)           -SUCH(*A,*T,*S,*G) -KO(*T,*S,*W)
                         -ERES(*A,*T,*W,*R).
```

```
FILE: UTILY    PROLOG    A    VM/CMS/SP CHAMPOLLION/GRENOBLE 81C1+PTF 23/3/

+ENCU(*A,*T.*Q,*S,*R) -SEEK(*Q,*S) -/ -ERES(*A,*T.*Q,*T.*S,*R).
+ENCU(*A,*T.*Q,*S,*R) -SUCH(*B,*T,*K,*S)
                      -KO(*T,NIL,*T1) -CONC(*A,*T1,*A1)
                      -CONC(*A1,*B,*A2)
                      -ENCU(*A2,*T.*Q,*K,*R).

+ERES(*A,*T,*T,*R)     -/ -EQUE(*A,*R).
+ERES(*A,*T,*G,*R)        -ERAS(*T,*G,*W) -CONC(*A,*W,*R).
+ERAS(*T.*Q,*T.*S,*R) -/ -ERAS(*Q,*S,*R).
+ERAS(*T,*T.*S,*S).
+EQUE(*T.NIL,*T)       -/.
+EQUE(*T.*Q,*T.*R)        -EQUE(*Q,*R).


*   LOCALISATION D'UNE SOUS CHAINE   *.

+SUCH(*A,*T.*K,*Q,*G)    -RUCH(*A,*T.*K,*Q,*G)    -/.
+SUCH(*AF,*T.*K,*QF,*G) -RUCH(*A,*T,*Q,*G)       -/ -NVAR(*Q)
                         -SUCH(*A1,*T.*K,*QF,*Q) -KO(*T,*A1,*AF).
+SUCH(*A,*T,*Q,*G)       -RUCH(*A,*T,*Q,*G).

+RUCH(NIL,*T.*Z,*B,*T.*G) -/ -TUCH(*Z,*B,*G).
+RUCH(NIL,*T,*B,*T.*B)    -/.
+RUCH(NIL,*T,*B,*T)       -/.
+RUCH(*K.*A,*Z,*B,*K.*G)     -RUCH(*A,*Z,*B,*G).

+TUCH(*T.*Z,*B,*T.*G) -/ -TUCH(*Z,*B,*G).
+TUCH(*T,*B,*T.*B)    -/.
+TUCH(*T,*B,*T).


*   OPERATEUR D'APPARTENENCE   *.

+APAR(*T,*T)     -/.
+APAR(*T,*G)        -APAS(*T,*G).
+APAS(*T.*Q,*G) -/ -SUCH(*A,*T,*W,*G) -ENKU(*T.*Q,*W).
+APAS(*T,*G)        -MBRE(*T,*G).

+ENKU(*T,*W)     -VAR(*W)      -/     -IMP.
+ENKU(*T.*Q,*W) -SEEK(*Q,*W) -/.
+ENKU(*T.*Q,*W) -SUCH(*A,*T,*K,*W) -ENKU(*T.*Q,*K).

+SEEK(*T,*T)        -/.
+SEEK(*T.*Q,*T.*W) -/ -SEEK(*Q,*W).
+SEEK(*T,*T.*W).

+NMBRE(*T,*G)    -MBRE(*T,*G) -/ -IMP.
+NMBRE(*T,*G).
+MBRE(*T,*T.*Q) -/.
+MBRE(*T,*T)    -/.
+MBRE(*T,*A.*Q) -INTE(*T,*Q).

+INTE(*T,*T.*Q) -/.
+INTE(*T,*A.*Q) -/ -INTE(*T,*Q).
+INTE(*T,*T).
```

```
ILE: UTILY    PROLOG    A   VM/CMS/SP CHAMPOLLION/GRENOBLE 81.01+PTF 20/8/81


   CONCATENATION SANS "NIL"   *.

KO(*Y,*X,*Z)              -VAR(*Y)     -VAR(*X) -/ -IMP.
KO(*Y,*X,*X).             -VAR(*Y) -/.
KO(*Y,*X,*Y)              -VAR(*X) -/.
KO(*X.*Y,*Z,*X.*T) -/ -KO(*Y,*Z,*T).
KO(*Y,*X,*Y.*X).


   UTILITAIRES D'ECRITURE   *.

SOR(*T.*Q) -COMPO(*T) -/ -SOR(*Q).
SOR(NIL)              -/.
SOR(*T)    -COMPO(*T).

COMPO(*T.*Q) -SORTIR(*T) -/ -COMPO(*Q).
COMPO(*T)    -SORTIR(*T).

SORTIR(PG) -SORM("(") -/.
SORTIR(PD) -SORM(")") -/.
SORTIR(BK) -SORM(" ") -/.
SORTIR(VR) -SORM(",") -/.
SORTIR(PS) -SORM("+") -/.
SORTIR(*T) -SORT(*T).


*   UTILITAIRES DE TRACE   *.

IMPR(*S) -PRET -/ -LG -SORT(*S) -LG.
IMPR(*S).

TRACE -PRET -/ -SUPP(+(PRET).NIL).
TRACE -AJOUT(+(PRET).NIL).

ST(*K)  -SORM("  ")   -SORT(*R).
LG      -LIGNE.
T(*R)   -LG -SORM(">>>>  ") -ST(*R) -LG.
ARBRE(*X) -LETTRE(*X).


*   QUASI-ARBORESCENCE   *.

QARB(*G,*R) -SYN(EGCO(*R).NIL,*G)       -IMPR(*R) -/.
QARB(*G,*R) -SORM(" ERREUR DANS QARB ") -IMP.

KR(*T.*Q,*T.*Q,S(*T.*Q)) -/.
KR(*T,*T,*T)             -/.
KR(*A,*W,S(*W)).

:EGCO(*R)    == :TEG(*A) :TREG(*A,*N) -KR(*A,*W,*R).

:TEG(*R) == ¬*A -ARBRE(*A) -/ :TSEK(*A,*Q) :TALTE(S(*Q),*R).
:TEG(*R) == ¬PG :TPAR(*A) ¬PD :TSEK(A(*A),*R).
```

FILE: UTILY PROLOG A VM/CMS/SP CHAMPOLLION/GRENOBLE 8101+PTF 20/8/

```
:TREG(*T,*R) == :TEG(*A) -KO(*T,*A,*Z) :TREG(*Z,*R) -/.
:TREG(*T,*T) ==.

:TPAR(*R)      == :EGCO(*A) ¬PS     -KO(*A,PS,*Z) :RTPAR(*Z,*R).
:RTPAR(*T,*W) == :EGCO(*A) ¬PS -/ -KO(*A,PS,*Z) :RTPAR(*Z,*R)
                                                -KO(*T,*R,*W).
:RTPAR(*T,*W) == :EGCO(*R)                      -KO(*T,*R,*W).

:TSEK(A(*T),A(*T).S(*W)) == ¬*A -ARBRE(*A) -/ :TSEK(*A,*W).
:TSEK(*T,*T.*W)          == ¬*A -ARBRE(*A) -/ :TSEK(*A,*W).
:TSEK(*T,*T)             ==.

:TALTE(*T,*T.A(*A)) == ¬PG :TPAR(*A) ¬PD -/.
:TALTE(*T,*T)       ==.
```

4.3. TRAITEMENT.

```
FILE: ARBY      PROLOG    A    V4/C*5/SP CHAMPOLLION/GRENOBLE 8101*PTF 20/8/81

% ================================> ARBORESCENCE D'UN E-GRAPHE *.

ARBO(*G,*R) -SYN(AGCO(*R).NIL,*G)          -IMPR(*R) -/.
ARBO(*G,*R) -SURM(" ERREUR DANS ARBO ") -IMP.

AGCO(*R)     == :AEG(*A) :AREG(*A,*W) -AR(*A,*W,*R).     * (1) *.

AR(*T.*Q,*T.*Q,ET(*T.*Q)) -/.
AR(*T,*T,*T)              -/.
AR(*A,*W,ET(*W)).

AEG(*R) == ¬*A -ARBRE(*A) -/ :ASEK(*A,*Q) :AALTE(ET(*Q),*R).
AEG(*R) == ¬PG :APAR(*A) ¬PD :ASEK(OU(*A),*R).
AREG(*T,*R) == :AEG(*A) -KO(*T,*A,*Z) :AREG(*Z,*R) -/.
AREG(*T,*T) ==.

                                                          * (2) *.
ASEK(OU(*T),OU(*T).ET(*W)) == ¬*A -ARBRE(*A) -/ :ASEK(*A,*W).
ASEK(*T,*T.*W)             == ¬*A -ARBRE(*A) -/ :ASEK(*A,*W).
ASEK(*T,*T)                ==.

AALTE(*T,*T.OU(*A)) == ¬PG :APAR(*A) ¬PD -/.          * (3) *.
AALTE(*T,*T)        ==.

APAR(*R)     == :AGCO(*A) ¬PS     :ATPAR(*A,*R).
ATPAR(*T,*W) == :AGCO(*A) ¬PS -/ :ATPAR(*A,*R) -KO(*T,*R,*W).
ATPAR(*T,*W) == :AGCO(*R)                      -KO(*T,*R,*W).
```

```
FILE: FAKTY     PROLOG     A   VM/CMS/SP CHAMPOLLION/GRENOBLE 8101+PTF 20/8/

* ==========================================> FACTORISATION *.

*   CONSTRUCTION D'UN Q-ARBRE   *.

+FAKTQ(*G,*R,D) -/ -INVE(*G,*P) -FAKTQ(*P,*Q,G)
                   -INVE(*Q,*R).
+FAKTQ(*G,*R,G)    -QARB(*G,*K)
                   -IMPR(*K)     -SYN(FACT(*K).NIL,*R) -/.
+FAKTQ(*G,*K,*T)   -SORM(" ERREUR DANS FAKTQ ")      -IMP.

:FACT(*K)              == :FAKT(*K,*W) -*K -APLA(*W,*R).
:FAKT(S(*T),S(*T))     == :ARBES(*T)    -/.
:FAKT(*T,*R)           == :NINIV(*T,*R).


*   (1) TRAVAIL A CHAQUE NIVEAU DU Q-ARBRE (FACT GENERALE)   *.

:NINIV(*T.*Q,*R.*W) == :DUTY(*T,*R) -/ :NINIV(*Q,*W).
:NINIV(PS.*Q,*W)    == :NINIV(*Q,*X)   -KQ(PS,*X,*W) -/.
:NINIV(*T,*R)       == :DUTY(*T,*R).

:DUTY(S(*T),S(*T)) == :ARBES(*T)     -/.
:DUTY(S(*T),S(*R)) == :NINIV(*T,*R) -/.
:DUTY(A(*T),*R)    == :LIST(*T)       -/ :FATG(*T,*W)
                                         -COCO(*T,*W,*R).
:DUTY(A(*T),*R)    == :NINIV(*T,*W) -/ :FATG(*T,*W)
                                         -COCO(*T,*W,*R).
:DUTY(*T,*T)       == -ARBRE(*T).

+COCO(*T,*T,A(*T)) -/.
+COCO(*T,*W,A(*W))    -MBRE(PS,*W) -/.      * CREATION DE LA *.
+COCO(*T,*W,S(*W)).                         * RACINE *.


*   (2) FACTORISATION ELEMENTAIRE   *.

*   FACTORISATION GAUCHE   *.

:FATG(*T.PS.*Q,*R) == :KLAS(*T,*Q,*V) -/:NOKL(*V,*W)
                      :COND(*V,*W,*Z)    :OUTS(*V,*T.PS.*Q,*N)
                      :GOON(*N,*S)       -DEKO(*Z,*S,*R).
:FATG(*T,*T)       ==.


*   RECHERCHE DES CLASSES   *.

:KLAS(S(*T.*U),S(*T.*V).PS.*Q,*R)                    ==
                          :KLAS(S(*T.*V),*Q,*W) -/
                          -KQ(S(*T.*U),PS,*X)
                          -KQ(*X,*W,*R).
:KLAS(S(*T.*U),S(*T.*V),S(*T.*U).PS.S(*T.*V)) == -/.
:KLAS(S(*T),S(*K).PS.*Q,*R)                          ==
                          :KLAS(S(*T),*Q,*R)     -/.
:KLAS(S(*T),S(*K),S(*T))                             ==.
```

```
ILE: FAKTY    PROLOG    A    VM/CMS/SP CHAMPOLLION/GRENOBLE 8101*PTF 20/3/81


   ELIMINATION DES ARCS DEJA FACTORISES   *.

OUTS(PS.*Q,*G,*R) == :OUTS(*Q,*G,*R) -/.
OUTS(*T,*T,NIL)   == -/.
OUTS(*T.*Q,*G,*R) == -ELIM(*T,*G,*W) -/ :OUTS(*Q,*G,*R).
OUTS(*T,*G,*R)    == -ELIM(*T,*G,*W)    -AFINA(*W,*R).

AFINA(NIL,NIL) -/.
AFINA(*W,*R)   -SANPS(*W,*V) -ELIM(NIL,*V,*X) -AVEPS(*X,*R).

SANPS(PS.*Q,*R)    -/ -SANPS(*Q,*R).
SANPS(PS,NIL)      -/.
SANPS(*T.*Q,*T.*R) -/ -SANPS(*Q,*R).
SANPS(*T,*T).

AVEPS(*T.*Q,*T.PS.*R) -/ -AVEPS(*Q,*R).
AVEPS(*T,*T).


   (3) ON APLATIT LE Q-ARBRE   *.

APLA(*G,*R) -SYN(APLO(*G).NIL,*W) -/    -ELIM(NIL,*W,*R).
APLA(*G,*R) -SORM(". ERREUR DANS APLA ") -IMP.

APLO(S(*T).*Q) == :ARBES(*T) ¬*T    -/ :APLO(*Q).
APLO(S(*T).*Q) == :APLO(*T)         -/ :APLO(*Q).
APLO(A(*T).*Q) == ¬PG :APLO(*T) ¬PD -/ :APLO(*Q).
APLO(A(*T))    == ¬PG :APLO(*T) ¬PD -/.
APLO(S(*T))    == :ARBES(*T) ¬*T    -/.
APLO(S(*T))    == :APLO(*T)         -/.
APLO(PS.*Q)    == ¬PS :APLO(*Q).


   UTILITAIRES   *.

:LIST(S(*T).*Q) == :ARBES(*T) -/ :LIST(*Q).
:LIST(S(*T))    == :ARBES(*T) -/.
:LIST(PS.*Q)    == :LIST(*Q).

:COND(*V,*V,*V)                == -/.
:COND(*V,S(*K).A(*T),S(*R)) == :FATG(*T,*W) -RARA(*T,*W,*Z)
                                -SIMPI(S(*K),*Z,*R).
▸RARA(*T,*T,A(*T)) -/.
▸RARA(*T,*W,*W).
▸SIMPI(S(*T),A(*Q),S(*T).A(*Q)) -/.
▸SIMPI(S(*T),S(*Q).*S,*R) -KO(*T,*Q,*W) -SIMPI(S(*W),*S,*R).

▸DEKO(*T,*Q,*R) -NVAR(*Q) -/ -KO(*T,PS,*Z) -KO(*Z,*Q,*R).
▸DEKO(*T,*Q,*T).

:GOON(NIL,*V) == -/.
:GOON(*T,*R)  == :FATG(*T,*R).
```

FILE: FAKTY PROLOG A VM/CMS/SP CHAMPOLLION/GRENOBLE 3101+PTF 20/8/

```
:WOKL(S(*T.*Q).*S,*R) == :ELTE(S(*T.*W).*S,*W) -/
                         -KO(S(*T),A(*W),*R).
:WOKL(S(*T),S(*T))    ==.

:ELTE(S(*T.*Q).PS.*S,S(*Q).PS.*W) == -/ :ELTE(*S,*W).
:ELTE(S(*T.*Q),S(*Q))             ==.
```

```
ILE: PARKY    PROLOG    A    VM/CMS/SP CHAMPOLLION/GRENOBLE 8101•PTF 20/8/81

==========================================>  PARCOURS D'UN E-GRAPHE *.

PARKY(*G,*R) -TRAK(*G,*T) -IMPR(*T) -PARK(*T,*R) -/.
PARKY(*G,*R) -SORM(" PARCOURS DIFFICILE ") -IMP.

   ENUMERATION DES TRAJECTOIRES POUR LA FUSION   *.

PARK(*P.PS.*S.*PS.*Q,*R) -/ -FUSI(*P,*S,*W) -PARK(*W.PS.*Q,*R).
PARK(*P.PS.*S.NIL,*R)    -/ -FUSI(*P,*S,*R).
PARK(*P,*P).

   FUSION DE DEUX TRAJECTOIRES   *.

FUSI(*P,*T.*Q,*R)     -MBRE(*T,*P) -/ -FUSI(*P,*Q,*R).
FUSI(*P,*T.*Q,*R) -/ -KO(*P,*T,*R)    -FUSI(*W,*Q,*R).
FUSI(*P,*T,*P)        -MBRE(*T,*P) -/.
FUSI(*P,*T,*R)        -KO(*P,*T,*R).
```

```
FILE: TRAKY     PROLOG     A     VM/CMS/SP CHAMPOLLION/GRENOBLE 81.01 PTF 20/8/
```

```
* ========================================> TRAJECTOIRES *.

*    CONSTRUCTION D'UN Q-ARBRE    *.

+TRAK(*G,*T) -QARB(*G,*R) -SYM(PATH(*R).NIL,*T) -IMPR(*R) -/.
+TRAK(*G,*T) -SORM(" ERREUR DANS TRAK ")          -IMP.

:PATH(*T) == :PATHS(*T,*R) :NETO(*R).


*    (1) RECHERCHE DE SOUS-ARBRES PAR NIVEAU    *

:PATHS(S(*T),*T)      == :ARBES(*T)      -/.
:PATHS(S(*T),*R)      == :NHAY(*T)       -/ :ECLA(*T,*R).
:PATHS(S(*T),*R)      == :NIVEL(*T,*U)   -/ :ECLA(*U,*R).
:PATHS(A(*T),A(*T))   == :LIS(*T)        -/.
:PATHS(A(*T),A(*R))   == :NIVEL(*T,*U)      :ECLA(*U,*R).

*    DESCENTE DANS L'ARBRE    *.

:NIVEL(*T.*Q,*R.*W) == :WORK(*T,*R)   -/ :NIVEL(*Q,*W).
:NIVEL(PS.*Q,*W)    == :NIVEL(*Q,*X)     -KO(PS,*X,*W) -/.
:NIVEL(*T,*R)       == :WORK(*T,*R).

:WORK(S(*T),S(*T)) == :NHAY(S(*T))   -/.
:WORK(A(*T),A(*T)) == :NHAY(A(*T))   -/.
:WORK(S(*T),*R)    == :NIVEL(*T,*U) -/ :ECLA(*U,*R).
:WORK(A(*T),*R)    == :NIVEL(*T,*U) -/ :ECLA(*U,*R).
:WORK(*T,*T)       == -ARBRE(*T).


*    SOUS-PROGRAMMES DE (1)    *.

:NHAY(S(*T).*Q) == :ARBES(*T) -/ :NHAY(*Q).
:NHAY(S(*T))    == :ARBES(*T) -/.
:NHAY(A(*T).*Q) == :LIS(*T)   -/ :NHAY(*Q).
:NHAY(A(*T))    == :LIS(*T)   -/.
:NHAY(PS.*Q)    == :NHAY(*Q).

:LIS(S(*T).*Q) == :ARBES(*T) -/ :LIS(*Q).
:LIS(S(*T))    == :ARBES(*T) -/.
:LIS(PS.*Q)    == :LIS(*Q).

:ARBES(*T.*Q) == -ARBRE(*T) -/ :ARBES(*Q).
:ARBES(*T)    == -ARBRE(*T).


*    (2) TRANSFORMATION D'UN NIVEAU DU SOUS-ARBRE    *.

:ECLA(S(*T).A(*P).*Q,*W) == -BELE(S(*T),*P,*R) -/
                                              :ECLA(A(*R).*Q,*W).
:ECLA(S(*T).A(*P),A(*R)) == -BELE(S(*T),*P,*R) -/.
:ECLA(A(*P).PS.*T.*Q,*W) == -ANYC(*P,*T,*Z)    -/
                                              :ECLA(A(*Z).*Q,*W).
```

```
FILE: TRACY    PROLOG    A    VM/CMS/SP CHAMPOLLION/GRENOBLE 8101*PTF 20/8/81


:ECLA(A(*P).PS.*T,A(*R)) == -ANYC(*P,*T,*R)     -/.

:ECLA(A(*P).S(*T).*Q,*W) == -CELE(*P,S(*T),*R) -/
                                                :ECLA(A(*R).*Q,*W).
:ECLA(A(*P).S(*T),A(*R)) == -CELE(*P,S(*T),*R) -/.
:ECLA(A(*P).A(*U).*Q,*W) == -DELE(*P,*U,*R)     -/
                                                :ECLA(A(*R).*Q,*W).
:ECLA(A(*P).A(*U),A(*R)) == -DELE(*P,*U,*R)     -/.
:ECLA(*T.*Q,*W)          == ¬*T                 :ECLA(*Q,*W) -/.
:ECLA(*T,*T)             == ¬*T.

*ANYC(*P,S(*T),*R) -KO(*P,PS,*Z) -/ -KO(*Z,S(*T),*R).
*ANYC(*P,A(*T),*R) -KO(*P,PS,*Z)    -KO(*Z,*T,*R).

*DELE(S(*T),S(*P).PS.*Q,S(*Z).PS.*W) -KO(*T,*P,*Z) -/
                                     -DELE(S(*T),*Q,*W).
*DELE(S(*T),S(*P),S(*Z))             -KO(*T,*P,*Z).

*CELE(S(*P).PS.*Q,S(*T),S(*Z).PS.*W) -KO(*P,*T,*Z) -/
                                     -CELE(*Q,S(*T),*W).
*CELE(S(*P),S(*T),S(*Z))             -KO(*P,*T,*Z).

*DELE(S(*T).PS.*U,*P,*R) -DELE(S(*T),*P,*X) -KO(*X,PS,*Z) -/
                         -DELE(*U,*P,*W)    -KO(*Z,*W,*R).
*DELE(S(*T),*P,*R)       -DELE(S(*T),*P,*X) -KO(*X,PS,*R).


*   (3) TRANSFORMATION ARBRE-CHAINE   *.

:NETO(A(*T)) == :SANS(*T) -/.
:NETO(*T)    == :SANS(*T).

:SANS(S(*T).*Q) == ¬*T -/ :SANS(*Q).
:SANS(PS.*Q)    == ¬PS -/ :SANS(*Q).
:SANS(S(*T))    == ¬*T -/.
:SANS(*T.*Q)    == ¬*T -/ :SANS(*Q).
:SANS(*T)       == ¬*T.
```

```
FILE: ALCUN    PROLOG    A    VM/CMS/SP CHAMPOLLION/GRENOBLE 8101+PTF 20/8/

* ================================================> ALTERNATION *.

+ALT(*A,*B,*G) -ELIM(NIL,*A,*M) -ELIM(NIL,*B,*N)
               -IMPR(*M)        -IMPR(*N)      -ALO(*M,*N,*G) -/.
+ALT(*A,*B,*G) -SORM(" ERREUR DANS ALT ")      -IMP.


+ALO(PG.*X,PG.*V,*R) -COLA(*X,PO,*G)  -/ -KO(PG,*G,*A)
                     -KO(*A,PS,*B)       -KO(*B,*V,*R).
+ALO(PG.*X,*V,*R)    -COLA(*X,PO,*G)  -/ -KO(PG,*G,*A)
                     -KO(*A,PS,*B) -KO(*B,*V,*C) -KO(*C,PO,*R).
+ALO(*X,PG.*V,*R)    -KO(PG,*X,*A)     -/ -KO(*A,PS,*B)
                                          -KO(*B,*V,*R).
+ALO(*X,*V,*R)       -KO(PG,*X,*A)        -KO(*A,PS,*B)
                     -KO(*B,*V,*C)        -KO(*C,PO,*R).


* ================================================> CONCATENATION *.

+CONC(NIL,*X,*X)       -/.
+CONC(*X.*Y,*Z,*X.*T) -CONC(*Y,*Z,*T).
```

```
FILE: EFFYD    PROLOG    A   VM/CMS/SP CHAMPOLLION/GRENOBLE 3101-PTF 20/8/31

-BOOLISTE!
-AJOP(".",1,"X¬(X¬X)")!
* ================================> EFFACEMENT D'UN E-GRAPHE *.

-EFFA(*E,*G,*R) -ELIM(NIL,*E,*E1) -ELIM(NIL,*G,*G1)
                                  -EFAK(*E1,*G1,*R) -/.
-EFFA(*E,*G,*R) -SORM(" ERREUR DANS EFFA ") -IMP.

-EFAK(*E,*E,*R) -/ -IMP.
-EFAK(*E,*G,*R)    -EFAZ(*E,*G,*R).


*   RECHERCHE DE LA CLASSE DE SOUS-E-GRAPHE    *.

-EFAZ(*E,*G,*R)        -ISOFO(*E,*G,*R) -/.
-EFAZ(*E,*G,*R)        -ISOMO(*E,*G,*R) -/.
-EFAZ(*E,*G,*R)        -ISOFA(*E,*G,*R).


*   TEST DE FORTE ISOLABILITE    *.

-ISOFO(*E,*G,*R) -SUCH(*A,*E,*B,*G) -PEGL(*A) -PEGL(*B) -/
                                    -EFAFO(*A,*E,*B,*R).
-ISOFO(*E,*G,*R) -SUCH(*A,*E,*B,*G) -TAVA(*A) -TAPR(*D)
                                    -CONC(*A,*B,*R).

-TAVA(*A) -COLA(*A,PS,*T) -/ -IMP.
-TAVA(*A) -COLA(*A,PG,*T)    -IMP.
-TAPR(PS.*Q) -/ -IMP.
-TAPR(PD.*Q)    -IMP.

-EFAFO(NIL,*E,*B,*B) -/.                    * EFFACEMENT *.
-EFAFO(*A,*E,*B,*A)     -VAR(*B) -/.
-EFAFO(*A,*E.*B,*R)     -CONC(*A,*B,*R).


*   TEST D'ISOLABILITE MOYENNE  (PREMIER CAS)    *.

-ISOMO(*E,*G,*R) -NMBRE(PS,*E) -SUCH(*A,*E,*B,*G) -CAVA(*A,*X)
                    -CAPR(*B,*Y) -EFAMO(*A,*E,*B,*X,*Y,*R).

-CAVA(*A,PS) -COLA(*A,PS,*T) -/.
-CAVA(*A,PG) -COLA(*A,PG,*T) -/.
-CAVA(*A,PD) -COLA(*A,PD,*T).

-CAPR(PS.*Q,PS) -/.
-CAPR(PG.*Q,PG) -/.
-CAPR(PD.*Q,PD).

-EFAMO(*A,*E,PS.*D,PS,PS,*R) -CONC(*A,*B,*R) -/.    * EFFACEMENT *.
-EFAMO(*A,*E,*B,PS,PD,*R)    -COLA(*A,PS,*T)
                             -INVSUCH(*X,PG,*Y,*T)
                             -ANALY(*X,*Y,*T,*B,*R) -/.
-EFAMO(*A,*E,PS.*D,PG,PS,*R) -SUCH(*X,PD,*Y,*B)
                             -ANDLY(*A,*B,*X,*Y,*R).
```

```
FILE: EFFYD     PROLOG    A    VM/CMS/SP CHAMPOLLION/GRENOBLE 8101•PTF 20/8/
```

```
•ANALY(*X,*Y,*T,*B,*R)     -MBRE(PS,*Y)      -/ -CONC(*T,*Y,*R).
•ANALY(*X,*Y,*T,PD.*B,*R) -CONC(*X,*Y,*N)      -CONC(*N,*B,*R).

•ANOLY(*A,*B,*X,*Y,*R) -MBRE(PS,*X)      -CONC(*A,*B,*R) -/.
•ANOLY(*A,*B,*X,*Y,*R) -COLA(*A,PG,*T) -CONC(*T,*X,*N)
                                          -CONC(*N,*Y,*R).


*   TEST D'ISOLABILITE FAIBLE !  *.

•ISOFA(*E,*G,*R) -SUCH(*A1,*E,*K,*G) -ELIM(NIL,*A1,*A)
                 -TROG(*A,*T,*Q)       -TROD(*K,*TE,*QU)
                 -KO(*G,*E,*W)         -KO(*W,*TE,*R).

•TROG(PG.*Q,*T,*K) -MBRE(PS,*A)       -TROG(PG.*Q,*T,*K) -/.
•TROG(PG.*Q,*X,*Q).
•TROG(*A,*A,*K)     -COLA(*A,PS,*B)   -/.
•TROG(*A,*B,*K)     -COLA(*A,*A,*A)   -/ -IMP.
•TROG(*A,*T,*K)     -COLA(*A,*QU,*B)      -TROG(*B,*T,*Q)
                                          -KO(*Q,*QU,*K).

•TROD(*K,*T,*Q) -VAR(*K)          -/.
•TROD(*K,*T,*Q) -MBRE(PS,*K)     -TROD(*K,*T,*Q) -/.
•TROD(*K,*T,*Q) -COLA(*K,PD,*T).

•TROD(PS.*Q,*R,PS.*Q) -/.
•TROD(*T.*Q,*W,*K)      -/ -TROD(*Q,*R,*K) -KO(*T,*R,*W).
•TROD(*T,*T,*K).


*   UTILITAIRES   *.

•COCA(*G,*CH,*N,*R) -SUCH(*A,*CH,*B,*G) -PLUS(1,*N,*K)
                    -COCA(*B,*CH,*K,*R) -/.
•COCA(*G,*CH,*R,*R).

•PEGL(NIL) -/.
•PEGL(*IL)      -VAR(*IL) -/.
•PEGL(*G)       -COCA(*G,PD,0,*N) -COCA(*G,PG,0,*M) -EGALF(*N,*M).

-TTY!
```

```
FILE: AJSUB     PROLOG    A    VM/CMS/SP CHAMPOLLION/GRENOBLE 3101+PTF 20/3/81

 ==========================================================> AJOUT  *.

AJOUT(*G,*G,*GA,*R) -/ -ALT(*G,*GA,*R).
AJOUT(*G,*SG,*GA,*R) -ELIM(NIL,*G,*G1)      -ELIM(NIL,*SG,*SG1)
                     -ELIM(NIL,*GA,*GA1)    -SUCH(*A,*SG1,*Q,*G1)
                     -ALT(*SG1,*GA1,*Z)     -ELPA(*G,*Z,*X)
                     -CONC(*A,*X,*Y)        -KO(*Y,*Q,*R).

ELPA(PS.*Q,PG.*X,*R) -COLA(*X,PO,*R) -/.
ELPA(*Q,*X,*X).


 ==========================================================> REMPLACEMENT  *.

SUBS(*G,*G,*SR,*SR) -/.
SUBS(*G,*SG,*SR,*R) -ELIM(NIL,*G,*G1)       -ELIM(NIL,*SG,*SG1)
                                            -ELIM(NIL,*SR,*SR1)
                    -SUCH(*A,*SG1,*Q,*G1)   -/
                    -CONC(*A,*SR1,*X)       -KO(*X,*Q,*R).
SUBS(*G,PG.*SG,*SR,*R) -ELIM(NIL,*G,*G1)    -ELIM(NIL,*SG,*SG1)
                                            -ELIM(NIL,*SR,*SR1)
                    -COLA(*SG1,PO,*SG2)     -SUCH(*A,*SG2,*Q,*G1)
                    -CONC(*A,*SR1,*X)       -KO(*X,*Q,*R).
```

4.4. EXEMPLES D'EXECUTION.

```
** >>>>    PROGRAMME DE TRAITEMENT POUR LES E-GRAPHES    <<<< **.
-AJOP(".",1,"X¬(X¬X)")!


*
       EXEMPLES D'EXECUTION    <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<.
-MON!

+

ARBE (AB+CD+F)(GH+IJ);
ET(OU(ET(A.B).ET(C.D).ET(F)).OU(ET(G.H).ET(I.J)))
+
FACTG (GB+GC+D)(EF+IJ);
(G(B+C)+D)(E.F+I.J)
+
PARC (AB+CD+F)(GH+IJ);
ABGHIJCDF+
+
TRAJ (AB+CD+F)(GH+IJ);
ABGH+ABIJ+CDGH+CDIJ+FGH+FIJ+
+
ALTE A(GB+GC+D)(EM+FM),(K+L+M);
(A(GB+GC+D)(EM+FM)+K+L+M)
+
CONC A(GB+GC+D)(EM+FM),(K+L+M);
A(GB+GC+D)(EM+FM)(K+L+M)
+
EFAC A(GB+GC+D)(EM+FM),(EM+FM);
A(GB+GC+D)
+
EFAC A(GB+GC+D)(EM+FM),GC;
A(GB+D)(EM+FM)
+
AJUT (AB+BC+F)(GH+IJ)K,(GH+IJ),PQR;
(AB+BC+F)(GH+IJ+PQR)K
+
SUBS (AB+BC+F)(GH+IJ)K,(GH+IJ),A(R+S);
(AB+BC+F)A(R+S)K
+

?

>>>>    FIN
-TTY!

SPOOL CONS STOP CLOSE
```

III. C O N C L U S I O N.

On a programmé en PROLOG un petit système. Il est évident qu'il y a un certain nombre de "coquilles". D'abord, on ne manquera pas de remarquer l'existence du q-arbre et de l'arborescence associée, notions qui font presque "double emploi". Cela est dû au fait que l'arborescence associée est née comme une amélioration du q-arbre (par exemple, pour le q-arbre, on a voulu garder les séparateurs + comme sentinelles mais leur interêt est moindre et ils alourdissent la programmation).

D'autre part, on a été obligé d'utiliser un parcours autre que le parcours canonique, étant donné que l'on travaillait sur des chaînes et qu'un parcours séquentiel était plus aisé. Le même cas s'est présenté pour la factorisation gauche, où un appel récursif sur un sous-arbre est plus pratique qu'un appel sur tout l'arbre. Ces deux altérations, commodes pour le traitement des chaînes, peuvent ne pas l'être pour le cas général.

Quant aux chaînes, il faut dire qu'il y a une certaine incohérence dans leur définition. Au début, on voulait ‹vide› comme chaîne vide et on a programmé en conséquence. Plus tard, on s'est aperçu que les grammaires de métamorphose utilisent nil comme chaîne vide, ce qui a produit parfois une dualité gênante dont KO et CONC sont la preuve.

Enfin, on pourrait faire le reproche que l'on a fait une étude sur les E-graphes alors que dans le prototype présenté il y a un travail très important sur les arbres. En fait, cela est vrai, mais il ne faut pas perdre de vue que l'on s'est imposé un modèle réduit et que la codification exigée par PROLOG est un arbre. D'autre part, cela montre jusqu'à quel point les transformations arbres ‹--› E-graphes réguliers sont aisées. Cette dernière particularité n'est pas sans importance, puisqu'on veut une nouvelle structure de données, mais on voudrait également continuer à utiliser des arbres pour certaines parties du traitement.

Comme dernier point: il faudrait encore réaliser un système de réécriture et compléter le prototype pour avoir un exemple fonctionnel de chaque étape du processus de traduction.

TABLE BIBLIOGRAPHIQUE.

[1] Bolt (76a)

[2] Bolt*(78)

[3] Chau (74)

[4] Chau (75)

[5] Clem (79)

[6] Coel (81)

[7] Coel*(80)

[8] Colm (70)

[9] Colm (75)

[10] Colm*(72)

[11] Colm*(79)

[12] Hirs (75)

[13] Kowa (79)

[14] Knut (73)

[15] Melo (76)

[16] Nils (67)

[17] Pere*(78)

[18] Rich (79)

[19] Robl (63)

[20] Robl (65)

[21] Robl (71)

[22] Rous (75)

[23] VEmd (78)

CONCLUSION GENERALE.

On a fait une étude de cette nouvelle structure de données que sont les E-graphes. On les a défini, on a analysé quelques transformations, on a montré des algorithmes de traitement et on a programmé en PROLOG un prototype réduit.

En général, les notions, formalisations et définitions données ont été satisfaisantes, exception faite du parcours (la factorisation a aussi été légèrement modifiée, mais elle aurait pu fonctionner telle qu'elle a été définie sans complication majeure). Il me semble que pour le cas général, le parcours canonique a le mérite de la simplicité, mais pour les E-graphes réguliers en écriture polynômiale, il n'est pas aussi adéquat qu'un parcours séquentiel.

Les E-graphes réguliers se sont révélés être une classe très intéressante. On a déjà énuméré leurs avantages et inconvenients, cependant on voudrait insister sur le fait de la nécessité de mieux étudier leurs possibilités. Ce serait intéressant, par exemple, de définir de sous-systèmes de puissance adaptée à chaque étape du processus de traduction, les E-graphes réguliers constituant la représentation pour les étapes moins complexes. Ils auraient cependant la possibilité d'être traités par un sous-système plus puissant sans problèmes de cohérence de représentation.

Bien qu'on n'ait pas analysé le problème des opérations sur les étiquettes, on a ressenti le besoin d'E-graphes récursifs (e.g. lors de la factorisation). Il serait peut-être intéressant d'étudier les E-graphes récursifs étiquetés par des E-graphes réguliers (étant donnée la facilité de manipulation de ces derniers) en vue de ces opérations.

Il me semble que la prochaîne étape du travail serait la formalisation et la programmation d'un petit transducteur de E-graphes pour passer ensuite à la définition d'un prototype de toute une chaîne de traduction.

B I B L I O G R A P H I E.

Cette bibliographie comporte les références citées à la fin de chaque chapitre et aussi des références non citées dans le texte.

Andr (67) Andreyev, N.D. THE INTERMEDIARY LANGUAGE AS THE FOCAL POINT OF MACHINE TRANSLATION. In Booth, A.D., MACHINE TRANSLATION. North-Holand. 1967.

Berg (73) Berge, C. GRAPHES ET HYPERGRAPHES. Dunod. Paris. 1973.

Bolt (74) Boitet, Ch. SEMANTIQUE ET TRADUCTION AUTOMATIQUE. QUELQUES REMARQUES. Document GETA No.G-3000-A. Décembre 1974.

Bolt (76a) Boitet, Ch. UN ESSAI DE REPONSE A QUELQUES QUESTIONS THEORIQUES ET PRATIQUES LIEES A LA TRADUCTION AUTOMATIQUE. DEFINITION D'UN SYSTEME PROTOTYPE. Thèse d'Etat. Grenoble. Avril 1976.

Bolt (76b) Boitet, Ch. PROBLEMES ACTUELS EN TRADUCTION AUTOMATIQUE: UN ESSAI DE REPONSE. Communication présentée à COLING-76. Ottawa. 1976.

Bolt (77a) Boitet, Ch. OU EN EST LE GETA DEBUT 1977 ?. T.A. Informations, No.1. 1977 & Communication au Colloque de la CEE "Franchir la barrière linguistique". Luxembourg. Mai 1977.

Bolt (77b) Boitet, Ch. MECHANICAL TRANSLATION AND THE PROBLEM OF UNDERSTANDING NATURAL LANGUAGES. Document GETA (2nd. draft) & Communication Colloque Moscou. Septembre 1977.

Bolt (79a) Boitet, Ch. AUTOMATIC PRODUCTION OF CF AND CS ANALYSERS USING A GENERAL TREE-TRANSDUCER. Rapport de Recherche IMAG, No.218 & Communication presentée a Saarbruecken. Novembre 1979.

Bolt (81) Boitet, Ch. TENDANCES FUTURES EN TRADUCTION AUTOMATISEE. Le langage et l'homme No.45. Janvier 1981. pp 16-28.

Bolt*(74) Boitet, Ch et Chauché, J. APPROCHES SEMANTIQUES POUR LES MODELES D'ANALYSE AUTOMATIQUE DE LANGUES NATURELLES. Colloque "Modèles logiques et niveaux d'analyse linguistique". Metz. Novembre 1974.

Bolt*(77a) Boitet, Ch., Vauquois, B., Quézel-Ambrunaz, M., Guillaume, P., Jaeger, D., Nedobejkine, N. et Daun Fraga, P. NOUVEAU SYSTEME, ETUDE DE DEFINITION-1. Document GETA*. Février 1977.

Bolt*(77b) Boitet, Ch., Vauquois, B., Quézel-Ambrunaz, M., Guillaume, P., Jaeger, D., Nedobejkine, N. et Daun Fraga, P. NOUVEAU SYSTEME, ETUDE DE DEFINITION-2. Document GETA*. Mars 1977.

Bolt*(78) Boitet, Ch., Guillaume, P. et Quézel-Ambrunaz, M. MANIPULATION D'ARBORESCENCES ET PARALLELISME: SYSTEME "ROBRA". Communication presentée à COLING-78. Bergen. Août 1978.

Bolt*(80) Boitet, Ch., Chatelin, Ph. et Daun Fraga, P. PRESENT AND FUTURE PARADIGMS IN THE AUTOMATIZED TRANSLATION OF NATURAL LANGUAGES. Communication presentée à COLING-80. Tokyo 1980.

Boot (67) Booth, A.D. MACHINE TRANSLATION. North-Holand. 1967.

Brai (68) Brainerd, W.S. THE MINIMALIZATION OF TREE AUTOMATA. I&C 13. 1968. pp 484-491.

Chat (79) Chatelin, P. SPECIFICATIONS ET MANIPULATIONS DE PROGRAMMES: CAS D'UN ENSEMBLE DE FORMES BILINEAIRES. Thèse d'Etat. Grenoble. Juin 1979.

Chau (74) Chauché, J. TRANSDUCTEURS ET ARBORESCENCES. ETUDES ET REALISATIONS DE SYSTEMES APPLIQUEES AUX GRAMMAIRES TRANSFORMATIONNELLES. Thèse d'Etat. Grenoble. Décembre 1974.

Chau (75) Chauché, J. PRESENTATION DU SYSTEME C.E.T.A. Document GETA No.G-3100-A. Janvier 1975 & AJCL. Microfiche 17, 21-39.

Chau*(72) Chauché, J., Guillaume, P. et Quézel-Ambrunaz, M. LE SYSTEME A.T.E.F (Analyse de Textes en Etats Finis). Document GETA No.G-2600-A. Octobre 1972 et AJCL. Microfiche 17, 21-39.

Clem (79) Clemente-Salazar, M. DEFINITION D'UN LANGAGE ET IMPLANTATION D'UN COMPILATEUR DECOMPILATEUR. Rapport DEA Informatique (IMAG). Septembre 1979.

Coel (81) Coelho, H. LOGIC PROGRAMMING BIBLIOGRAPHY (2nd. Edition). Laboratorio Nacional de Engenharia Civil. Centro de Informatica. Lisboa. October 1981.

Coel*(80) Coelho, H., Carlos Cotta, J. et Moniz Pereira, C. HOW TO SOLVE IT WITH PROLOG (2nd Edition). Laboratorio Nacional de Engenharia Civil. Centro de Informatica. Lisboa. 1980.

Colm (70) Colmerauer, A. LES SYSTEMES-Q OU UN FORMALISME POUR ANALYSER ET SYNTHETISER DES PHRASES SUR ORDINATEUR. Département d'Informatique. Publication Interne No.43. Université de Montréal. Septembre 1970.
Aussi dans:
TAUM71. Projet de Traduction Automatique de l'Université de Montréal. TAUM. Montréal. 1971. pp 1-45.

Colm (75) Colmerauer, A. LES GRAMMAIRES DE METAMORPHOSE. Groupe d'Intelligence Artificielle. UER Scientifique de Luminy. Université d'Aix-Marseille II. Novembre 1975.
Aussi dans:
Natural language communication with computer. Lecture notes in Computer Science. Springer Verlag. 1978, 63, pp 133-189.

Colm*(72) Colmerauer, A., Kanoui, H., Roussel, Ph. et Pasero, R. UN SYSTEME DE COMMUNICATION HOMME-MACHINE EN FRANCAIS. Groupe d'Intelligence Artificielle. UER Scientifique de Luminy. Université d'Aix-Marseille II. Février 72 à Juin 73.

Colm*(79) Colmerauer, A., Kanoui, H. et Van Caneghem, M. ETUDE ET REALISATION D'UN SYSTEME PROLOG. Groupe d'Intelligence Artificielle. UER Scientifique de Luminy. Université d'Aix-Marseille II. Mai 1979.

Daun (77a) Daun Fraga, P. ANALISES MORFOLOGICA E SINTATICA DA LINGUA PORTUGUESA. NUM PROJETO DE TRADUCAO AUTOMATICA. Relatorio Interno No.35. Dpto. de Ciencia da Computaçao. Universidad Estadual de Campinas. Campinas, S.P. Brasil. Março 1977.

Daun (77b) Daun Fraga, P. ANALISE MORFOLOGICA DA LINGUA PORTUGUESA USANDO UM AUTOMATO DE ESTADOS FINITOS. Relatorio Interno No.78. Dpto. de Ciencia da Computaçao. Universidad Estadual de Campinas. Campinas, S.P. Brasil. Novembro 1977.

Feld (68) Feldman, J and Gries, D. TRANSLATOR WRITING SYSTEMS. CACM. V11. N2. February 1968. pp 77-113.

Floy (67) Floyd, R.W. NONDETERMINISTIC ALGORITHMS. JACM. V14. N4. October 1967. pp 636-644.

Glad (69) Gladky, A.V. and Mel'tchuk, I.A. TREE GRAMMARS (Δ-GRAMMARS). International Conference on Computational Linguistics. 1-4 September 1969. Preprint No.1. Classification: AL3.2. Sanga-Saby, Sweden.

Gins (59a) Ginsgurg, S. ON THE REDUCTION OF SUPERFLUOUS STATES IN A SEQUENTIAL MACHINE. JACM. V6. April 1959. pp 259-282.

Gins (59b) Ginsgurg, S. A TECHNIQUE FOR THE REDUCTION OF A GIVEN MACHINE TO A MINIMAL STATE MACHINE. IRE Transactions on Electronic Computers. Vol.EC-8. September 1959. pp 346-355.

Gins (59c) Ginsgurg, S. SYNTHESIS OF MINIMAL-STATE MACHINES. IRE Transactions on Electronic Computers. Vol.EC-8. December 1959. pp 441-449.

Golo (65) Golomb, S.W. and Baumert, L.D. BACKTRACK PROGRAMMING. JACM. V12. N4. October 1965. pp 516-524.

Grel (79) Greif, I. and Meyer, A. SPECIFYING PROGRAMMING LANGUAGE SEMANTICS. Conference record of the sixth annual ACM symposium on Principles of Programming Languages. January 29-31, 1979. pp 180-189.

Guil (80) Guilbaud, J.Ph. ANALYSE MORPHOLOGIQUE DE L'ALLEMAND EN VUE DE LA TRADUCTION PAR ORDINATEUR DE TEXTES TECHNIQUES SPECIALISES. Thèse 3e-Cycle. Université de la Sorbonne Nouvelle. Paris III. Paris. Juin 1980.

Haue (79) Hauenschild, C. ESQUISSE D'UN SYSTEME DE TRADUCTION AUTOMATIQUE: SALAT. Institut fuer Angewandte Sprachwissenschaft. Universitaet Heidelberg. Juin 1979.

Haue*(78) Hauenschild, C., Huckert, E. and Maier, R. SALAT: MACHINE TRANSLATION VIA SEMANTIC REPRESENTATION. Konstanz Colloquium on Semantics. 18-22 September 1978.

Hirs (75) Hirsberg, D.S. A LINEAR SPACE ALGORITHM FOR COMPUTING MAXIMAL COMMON SUBSEQUENCES. CACM. V18. N6. June 1975. pp 341-343.

Hofm (78) Hofmann, T. DESCRIPTION SEMANTIQUE ET DYNAMIQUE DU DISCOURS. Thèse 3e-Cycle. Linguistique. Université de PARIS-SORBONNE. Paris IV. Mai 1978.

Hopc (69) Hopcroft, J.E. and ULLman, J.D. FORMAL LANGUAGES AND THEIR RELATION TO AUTOMATA. Addison-Wesley. Massachusetts. 1969.

Huff (54) Huffman, D.A. THE SYNTHESIS OF SEQUENTIAL SWITCHING CIRCUITS. Journal of the Franklin Institute. Part I: Vol.257, No.3. March 1954, pp 161-190. Part II: Vol.257, No.4. April 1954, pp 257-303.

Kowa (74) Kowalski, R. PREDICATE LOGIC AS A PROGRAMMING LANGUAGE. Information Processing 74. North-Holand. 1974.

Kowa (79) Kowalski, R. ALGORITHM = LOGIC + CONTROL. CACM. V22. N7. July 1979. pp 424-436.

Knut (73) Knuth, D.E. FUNDAMENTAL ALGORITHMS (The Art of Computer Programming. Vol.1). Addison-Wesley. 2nd Edition. Massachusetts. 1973.

Kula*(69) Kulagina, O.S. and Mel'cuk, I.A. AUTOMATIC TRANSLATION: SOME THEORETICAL ASPECTS AND THE DESIGN OF A TRANSLATION SYSTEM. In Booth, A.D., MACHINE TRANSLATION. North-Holand. 1969.

Kunt (72) Kuntzmann, J. THEORIE DES RESEAUX (GRAPHES). Dunod. Paris. 1972.

Ljud (73) Ljudskanov, A. MACHINE ET SIGNIFICATION. T.A. Informations. N1. 1973. pp 2-23.

McCa (60) Mc Carthy, J. RECURSIVE FUNCTIONS OF SYMBOLIC EXPRESSIONS AND THEIR COMPUTATION BY MACHINE, PART I. CACM. V3. N4. April 1960. pp 184-195.

Melo (76) Meloni, H. PROLOG - MISE EN ROUTE DE L'INTERPRETEUR ET EXERCICES. Groupe d'Intelligence Artificielle. UER Scientifique de Luminy. Université d'Aix-Marseille II. 1976.

Nedo (76) Nedobejkine, N. NIVEAU D'INTERPRETATION DANS UNE TRADUCTION MULTILINGUE: APPLICATION A L'ANALYSE DU RUSSE. Communication présentée à COLING-76. Ottawa 1976.

Neth (59) Netherwood, D.B. MINIMAL SEQUENTIAL MACHINES. IRE Transactions on Electronic Computers. Vol.EC-8. September 1959. pp 339-345.

Nils (67) Nilsson, N. PROBLEM SOLVING METHODS IN ARTIFICIAL INTELLIGENCE. Mc Graw-Hill. New York. 1967.

Pere*(78) Pereira, L.M., Pereira, F.C.N. and Warren, D.H.D. USER'S GUIDE TO DECsystem-10 PROLOG. Provisional Version (Relatorio). Ministerio da Habitaçao e Obras Publicas. Laboratorio Nacional de Engenharia Civil. Lisboa. Outubro de 1978.

Rich (79) Richards, M. A COMPACT FUNCTION FOR REGULAR EXPRESSION PATTERN MATCHING. Sofware - Practice and Experience. V9. 1979. pp 527-534.

Robi (63) Robinson, J.A. THEOREM-PROVING ON THE COMPUTER. JACM. 10. April 1963. pp 163-174.

Robi (65) Robinson, J.A. A MACHINE-ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE. JACM. V12. N1. January 1965. pp 23-41.

Robi (71) Robinson, J.A. BUILDING DEDUCTION MACHINES. Artificial Intelligence and Heuristic Programming. Edited by N.V. Findler and B. Meltzer. American Elsevier Inc. New York. 1971.

Rous (75) Roussel, Ph. PROLOG. MANUEL DE REFERENCE ET D'UTILISATION. Groupe d'Intelligence Artificielle. UER Scientifique de Luminy. Université d'Aix-Marseille II. Septembre 1975.

Scot (77) Scott, D.S. LOGIC AND PROGRAMMING LANGUAGES. CACM. V20. N9. Septembre 1977. pp 634-641.

Stew (75) Stewart, G. LE LANGAGE DE PROGRAMMATION REZO. Rapport de stage en vue de l'obtention du grade de maîtrise ès Sciences. Département d'Informatique. Université de Montréal. Septembre 1975.

Thou (76) Thouin, B. SYSTEME INFORMATIQUE POUR LA GENERATION MORPHOLOGIQUE DE LANGUES NATURELLES EN ETATS FINIS. Communication présentée à COLING-76. Ottawa 1976.

VEmd (78) Van Emden, M.H. RELATIONAL PROGRAMMING. Illustrated by a program for the game of Mastermind. Research Report CS-78-48. Department of Computer Science. University of Waterloo. December 1978.

Vauq (68) Vauquois, B. A SURVEY OF FORMAL GRAMMARS AND ALGORITHMS FOR RECOGNITION AND TRANSFORMATION IN MECHANICAL TRANSLATION. Reprint from: IFIP CONGRESS 68 preprints. North-Holland. 1968.

Vauq (75) Vauquois, B. LA TRADUCTION AUTOMATIQUE A GRENOBLE. Document de Linguistique Quantitative No.24. Dunod. 1975.

Vauq (76) Vauquois, B. AUTOMATIC TRANSLATION - A SURVEY OF DIFFERENT APPROACHES. Communication présentée à COLING-76. Ottawa 1976.

Veil*(67) Veillon, G., Veyrunes, J. et Vauquois, B. UN METALANGAGE DE GRAMMAIRES TRANSFORMATIONNELLES. Document GETA G2300-A. Janvier 1967 & Communication présentée pour la 2e Conférence Internationale sur le Traitement Automatique des Langues. Grenoble. Août 1967.

Walk (60) Walker, R.L. AN ENUMERATIVE TECHNIQUE FOR A CLASS OF COMBINATORIAL PROBLEMS. Amer. Math. Soc. Proc. Symp. Appl. Math. 10.1960. pp 91-94.

Warr (80) Warren, D.H. LOGIC PROGRAMMING AND COMPILER WRITING. Software - Practice and Experience. V10. 1980. pp 97-125.

Warr*(77) Warren, D. Pereira, L.M. and Pereira, F. PROLOG - THE LANGUAGE AND ITS IMPLEMENTATION COMPARED WITH LISP. Proc. Symp. on Artif. Intell. and Programming Languages. SIGPLAN Notices (ACM). V12. N8. SIGART Newsletters (ACM) N64. August 1977. pp 109-115.

Wate (70) Waterman, D.A. GENERALIZATION LEARNING TECHNIQUES FOR AUTOMATING THE LEARNING OF HEURISTICS. Artificial Intelligence 1. 1970. pp 121-170.

Welz (66) Weizenbaum, J. ELIZA - A COMPUTER PROGRAM FOR THE STUDY OF NATURAL LANGUAGE COMMUNICATION BETWEEN MAN AND MACHINE. CACM. V9. N1. January 1966. pp 36-45.

Wirt (76) Wirth, N. ALGORITHMS + DATA STRUCTURES = PROGRAMS. Prentice-Hall. 1976.

Wood (70) Woods, W.A. TRANSITION NETWORK GRAMMARS FOR NATURAL LANGUAGE ANALYSIS. CACM. V13. N10. October 1970. pp 591-606.

*EURO(80) * (EUROTRA). STUDY OF EUROTRA'S DATA STRUCTURES AND C-LEVEL LANGUAGE. FINAL REPORT. GETA/ EUROTRA/ April 1980.

*GLEI(75) * (GROUPE LEIBNIZ). PROJET DE REPRESENTATION DES STRUCTURES DE PHRASE AU NIVEAU DU TRANSFERT. Réunion Groupe Leibniz. Lugano. Mars 1975.

*TAUM(71) * (TAUM). TAUM-71. Projet de Traduction Automatique de l'Université de Montréal. Montréal. Janvier 1971.

*TAUM(77) * (TAUM). PROJET AVIATION. Rapport d'étape. TAUM (Groupe de Recherches pour la Traduction Automatique). Université de Montréal. Montréal. Mai 1977.

*TAUM(79) * (TAUM). GROUPE DE RECHERCHE EN TRADUCTION AUTOMATIQUE DE L'UNIVERSITE DE MONTREAL (TAUM). Montréal. Juin 1979.

****

ANNEXE

BASES POUR UNE PROGRAMMATION GENERALE ET EFFICACE DU PROTOTYPE

I N T R O D U C T I O N.

Dans cet annexe, on présente les algorithmes qui réalisent les fonctions déjà vues, mais en considérant cette fois les E-graphes en général et un "langage" de haut niveau: le pseudo-PASCAL. On s'interessera aux fonctions qui ont un certain interêt algorithmique. Bien qu'il ne soit pas encore totalement formalisé, on montre un système de réécriture qui dans son état actuel est une simulation des systèmes-Q. Celui-ci nous servira de point de départ pour une transduction des E-graphes. Enfin, on trouvera un algorithme de localisation de segfl, algorithme de grande utilité dans le traitement.

I. STRUCTURE: REPRESENTATION ET NOTATIONS.

Afin de décrire plus rigoureusement les algorithmes qui vont suivre, on doit fixer une structure de données et une représentation des E-graphes et de leurs rélations d'ordre. Le choix de cette représentation sera évidemment déterminé par les opérations à réaliser, en particulier par les opérations d'adjonction et d'effacement des arcs, et par celles où intervient un parcours de la structure. Puisque, d'autre part, le nombre n d'éléments n'est pas donné a priori, une organisation convenable est la liste linéaire.

Comme [8] le fait remarquer, il est difficile d'avoir une méthode de représentation unique pour les listes linéaires dans laquelle toutes les opérations possibles seront efficaces. Dans cette première approche, on ne cherche pas l'efficacité à outrance mais plutôt une représentation qui permettra de fixer et de tester les idées.

On voudrait coder certaines informations. D'une part, celles concernant un élément de réseau: les arcs, les sommets, les E-arbres. D'autre part deux ordres: le vertical et l'horizontal. Enfin, on voudrait effectuer éventuellement des marquages. Pour les informations sur un élément de réseau, on confondera toujours les arcs et les E-arbres. On ne détaillera pas le codage des sommets.

On utilise quatre fonctions: sm, alt, su et ant. Les deux premières codent les deux ordres, la troisième le parcours et la dernière fournit l'information sur un arc qui "précède" un arc donné. Ce sont toutes des applications de l'ensemble d'arcs sur lui-même. Voici leur définition, soient α, β et λ des arcs et A l'ensemble d'arcs.

- suivant-minimal (sm)

$$sm(\alpha) = \begin{cases} \min \underline{\sigma}(\tau(\alpha)) & \text{si } \tau(\alpha) \neq 0 \\ 0 & \text{sinon} \end{cases}$$

On favorise l'arc minimal de l'ensemble d'arcs issus de $\tau(\alpha)$. On note,

$$sm^j(\alpha) = sm(sm^{j-1}(\alpha)) \qquad (j = 2,\ldots) \text{ et } sm^1(\alpha) = sm(\alpha)$$

le j-ième élément sm.

- alternant (alt)

$$alt(\alpha)= \begin{cases} \beta & \text{si } \alpha<_v\beta \ \&\ (\nexists\lambda)[\alpha<\lambda<\beta] \\ 0 & \text{sinon} \end{cases}$$

alt désigne donc l'arc suivant dans l'ordre vertical. On note le j-ième élément.

$$alt^j(\alpha)= alt(alt^{j-1}(\alpha)) \qquad (j= 2,\ldots) \text{ et } alt^1(\alpha)= alt(\alpha)$$

- suivant (su)

Etant donnée l'énumération canonique

$$\alpha_1, \ldots, \alpha_n \qquad n= |A|$$

du E-graphe, on a

$$su(\alpha_i)= \begin{cases} \alpha_{i+1} & \text{si } 1<=i<n \\ 0 & \text{si } i=n \end{cases}$$

- antécédent (ant)

$$ant(\alpha)= \begin{cases} \beta & \text{si } (sm(\beta)=\alpha \ \&\ (\exists\lambda)[sm(\lambda)=\alpha \ \&\ \lambda\neq\beta]) \ \vee \\ & (alt^i(sm(\beta))=\alpha \ \&\ (\exists\lambda)[alt^i(sm(\lambda))=\alpha \ \&\ \lambda\neq\beta]) \\ 0 & \text{sinon} \end{cases}$$

On appelle un arc α premier arc d'un E-graphe si

$$\alpha= \min \underline{\sigma}(i)$$

Une sous-trajectoire sera une sous-séquence d'une trajectoire donnée. Finalement, l'arc 0 est un "pseudo-arc" avec la propriété suivante.

$$sm(\alpha)= \min \underline{\sigma}(i)$$

II. A L G O R I T H M E S.

1. PARCOURS CANONIQUE PARC(α)

C'est un algorithme d'énumération des arcs compris entre le sommet $\sigma(\alpha)$ (considéré comme entrée) et la sortie du E-graphe. La méthode consiste tout simplement à parcourir chaque sous-trajectoire en sortant dans l'ordre horizontal tous les arcs qui n'ont pas encore été marqués (i.e. parcourus). Chaque arc est marqué immédiatement après sa sortie.

L'algorithme a été originalement conçu pour atteindre la sortie, mais par un changement trivial on peut le transformer en

PARC(I,O,α)

c'est-à-dire que l'on peut parcourir des sous-structures à entrée I et sortie O en commençant par un arc α de l'ensemble $\underline{\sigma}(I)$ d'arcs issus de I.

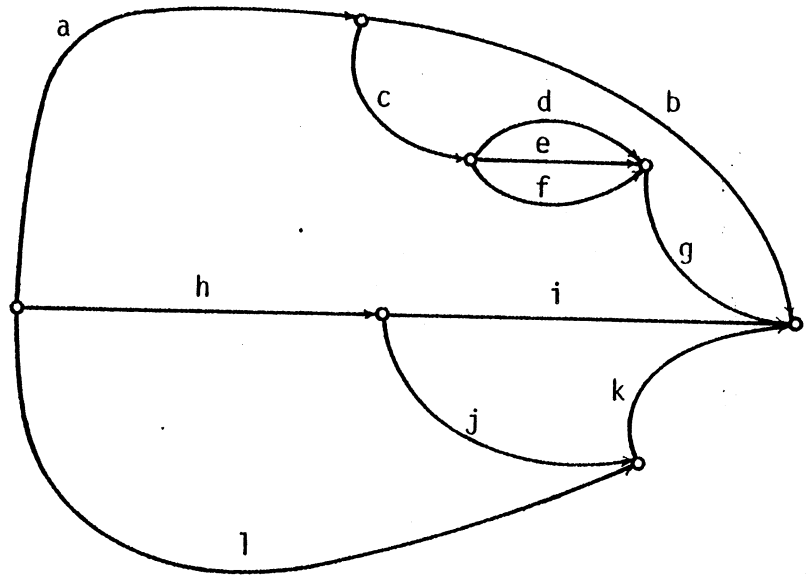
1.1. A L G O R I T H M E.

```
PARC(α)
debut  * ALGORITHME DE PARCOURS CANONIQUE *

α : arc;    nonmarqué:booléen;

    si  α=0  alors  α:= sm(α)  fsi

    si  nonmarqué(α)  alors

        sortir α
        marquer α

        * parcours des sm *
        si  sm(α)≠0  alors  PARC(sm(α))  fsi

        *  parcours des alternants  *
        si  alt(α)≠0  alors  PARC(alt(α))  fsi

    fsi
fin
```

1.2. E X E M P L E.

Soit le E-graphe suivant.

G =



avec l'appel: PARC(0), on obtient le parcours

abcdgefhijkl

2. TRAJECTOIRES. TRAJ(α,pile)

C'est un algorithme d'énumération des trajectoires comprises entre le sommet $\sigma(\alpha)$, considéré comme entrée, et la sortie du E-graphe en commençant par l'arc α. On stocke dans la pile, les arcs qui constitueront une trajectoire.

La méthode consiste à parcourir canoniquement le E-graphe en empilant chaque arc renconté et à énumérer les éléments de la pile au moment où on arrive à la sortie. Pour chercher la trajectoire suivante, on réalise un backtrack, en dépilant les arcs qui n'ont pas d'alternant, afin d'atteindre le premier arc β (dans le sens du backtrack) à alternant non-vide. On empile β et on refait les mêmes opérations jusqu'à épuiser tous les arcs du E-graphe.

De même que pour l'algorithme de parcours, TRAJ a été conçu pour atteindre la sortie, mais aussi par un changement trivial, on peut le transformer en

TRAJ(I,O,α,pile)

c'est-à-dire qu'on a le même algorithme pour énumérer les trajectoires des sous-structures à entrée I et sortie O.

N.B. L'instruction SORTIR pile énumère les éléments de la pile sans les dépiler.

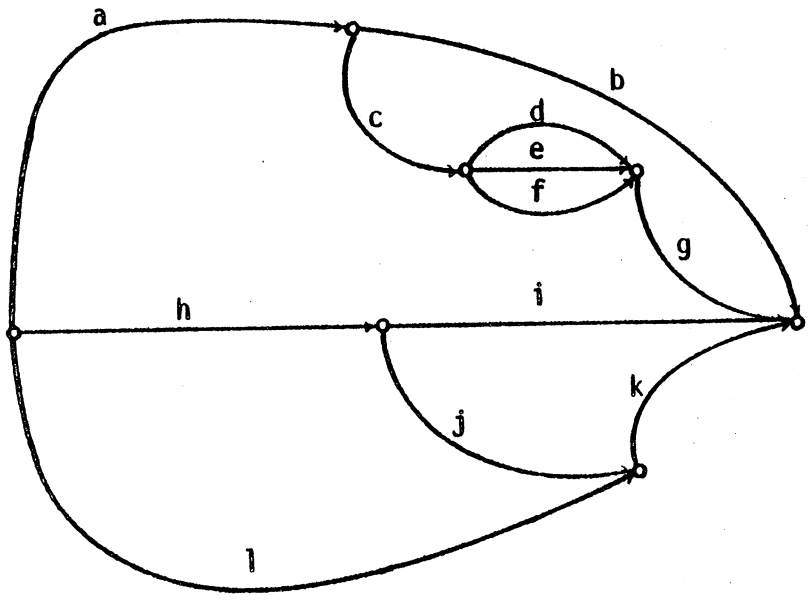
2.1. A L G O R I T H M E.

```
TRAJ(α, pile)
debut  * GENERATION DE TOUTES LES TRAJECTOIRES  *

α:arc;    pile:pile d'arcs;

     *  (1) parcours et sortie d'une trajectoire  *

     si  α=0  alors  α:= sm(α)  fsi
     empiler α
     si  sm(α)≠0  alors  TRAJ(sm(α),pile)
     sinon  sortir pile  fsi

     *  (2) recherche de la trajectoire suivante  *

     dépiler
     si  alt(α)≠0  alors  TRAJ(alt(α),pile)  fsi

fin
```

2.2. E X E M P L E.

Soit le E-graphe suivant.

G =



avec l'appel: TRAJ(0, pile), on obtient les trajectoires suivantes.

ad
acdg
aceg
acfg
hi
hjk
lk

3. F A C T O R I S A T I O N.

3.1. FACTORISATION GENERALE. FACTG(I,O,α)

FACTG cherche, en parcourant canoniquement, un E-graphe G à entrée I, sortie O et premier arc α, tous les segfi-élémentaires contenus dans G. Ceux-ci sont aussitôt factorisés récursivement. Une fois terminée cette opération sur tous les segfi-élémentaires, on passe à la factorisation élémentaire de G qui produira un et un seul arc, étiqueté par le E-graphe factorisé. Enfin, on "décursivise" (récursivement) l'arc en question pour obtenir le résultat final.

Le problème trivial concerne les cas des E-graphes à un seul arc ou des E-graphes séquentiels où il n'y a aucun travail à réaliser.

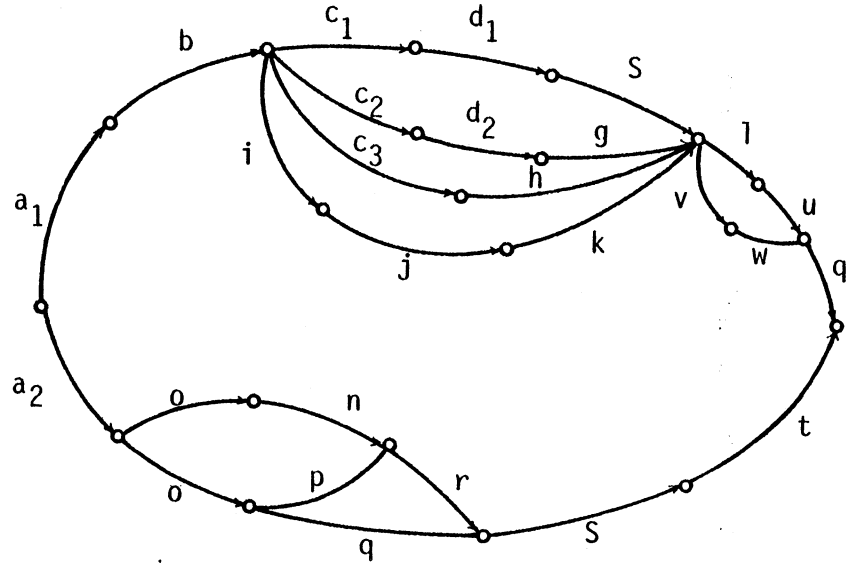
3.1.1. A L G O R I T H M E.

```
FACTG(I, O, α)
debut  *  PROGRAMME DE FACTORISATION GENERALE  *

α : arc;    x:sommet;

    si  problème trivial  alors  le résoudre
    sinon

        α:= min σσ(α)

        *  parcours de tout le E-graphe  *

        tantque  α≠0  faire

            SEGFI(σ(α),x)                  * existe-t-il un segfi? *
            si x≠0 & x≠O  alors            * si oui, on le factorise *
                FACTG(σ(α), x, min σσ(α))
                α:= min σ(x)
            sinon
                si  x=O & σ(α)≠I  alors    * le segfi satisfait
                                                        OєS'? *
                    FACTG(σ(α), x, min σσ(α))
                    α:=0
                sinon  α:=su(α)  fsi
            fsi
        ftq

        * factorisation élémentaire du (sous) E-graphe G=(I,O,α)  *

        FACTEL(I, O, min σ(I))
        si  I & O ne sont pas l'entrée et la sortie d'un seg alors
            DEFACT(I, O, min σ(I))
        fsi
    fsi
fin
```

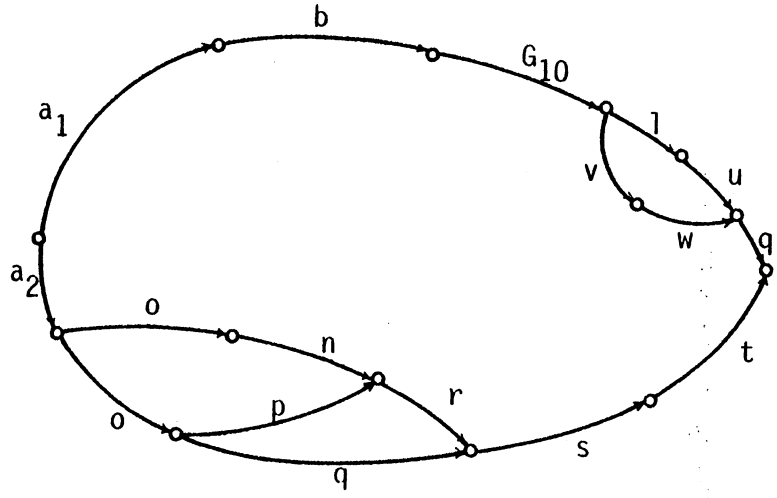
3.1.2. E X E M P L E.

On montre les transformations successives de G1 pour arriver au résultat G6.

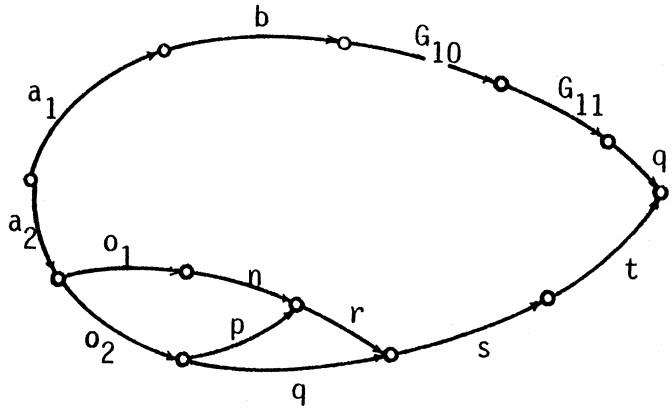
G1 =



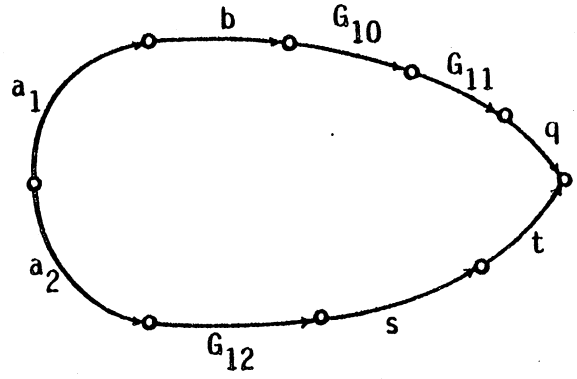
G2 =



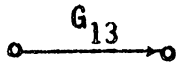
G3 =



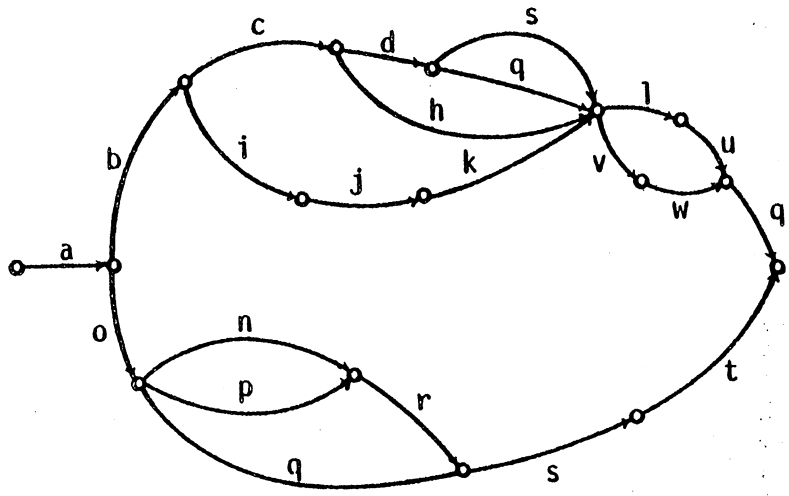
G4 =



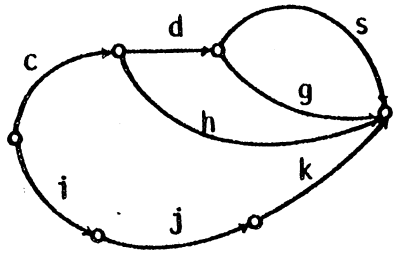
G5 =



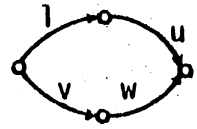
G6 =



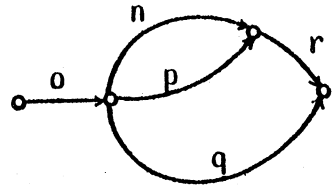
G10 =



G11 =



G12 =



3.2. FACTORISATION ELEMENTAIRE. FACTEL(I,O,α)

FACTEL transforme un E-graphe à entée I, sortie O et premier arc α. Il se compose de deux parties: une factorisation gauche et une transformation en un E-graphe récursif.

(1) Factorisation gauche.

Dans cette partie, on cherche (en parcourant canoniquement le E-graphe) un arc à alternant non-nul. Si on appelle α cet arc, on réalise alors les opérations suivantes sur l'ensemble $\underline{\sigma}\sigma(\alpha)$ d'arcs issus de $\sigma(\alpha)$:

On saute d'abord les arcs dont le but a d'autre arcs entrants. Le premier arc (dans l'ordre vertical) ayant un but à un prédécesseur devient le représentant d'une classe pour la relation μ ("etre étiqueté par le même arbre"). Le pas suivant consiste à chercher les arcs qui appartiennent à la même classe. Pour chaque arc d'une classe donnée, on fait un transfert (par le bas et dans l'ordre) de l'ensemble d'arcs sortant du but de l'arc considéré, vers le but du représentant. Une fois finiès les opérations sur une classe, on les refait pour la classe suivante, si elle existe.

(2) Transformation en un E-graphe récursif.

Ce qui est propre à la factorisation élémentaire est réalisé ici, à savoir: création d'un arc, affectations du contenu et chaînage dans le E-graphe.

3.2.1. A L G O R I T H M E.

```
FACTEL(I, O, α)
debut  * PROGRAMME DE FACTORISATION ELEMENTAIRE *

α, α1, α2, β, k:arcs

    * (1) factorisation gauche *

    α:= min σ(I);    α2:=α
    tantque  su(α)≠0  faire     * parcours de tout le E-graphe *

        α1:=α
        tantque  alt(α)≠0  faire    * travail sur un sommet: recherche
                                      et transformation des classes *
         *sauter les arcs qui ont plus d'un arc entrant en τ(α)*

          tantque  |τ⁻τ(α)|≠1 & alt(α)≠0  faire α:=alt(α)  ftq

          si  alt(α)≠0  alors        * α, 1er arc de la classe *

            k:= β:= α
            α:=alt(k)
            repeter              * parcours de tous les alternants
                                   d'une classe, suceptibles de transf. *
                si k=α & τ(α)≠O & |τ⁻τ(α)|=1  alors

                    transférer les arcs de l'ensemble στ(α) comme
                    alternants de alt^n(sm(k)), où n= |στ(k)|-1

                    alt(β):= alt(α)

                sinon  β:=α  fsi

                si  alt(α)≠0  alors  α:=alt(α)  fsi

            jusqu'à  alt(α)=0  frep

            α:=alt(k)
          fsi
        ftq

        si  su(α1)≠0  alors  α:=su(α1)  fsi

    ftq
```

* (2) transformation en un E-graphe récursif *

créer arc λ

E-graphe(λ):= (I, O, α2)

si ant(α2)≠0 alors (∀β)[β∈$\underline{\Gamma}$σ(α2) => sm(β):=λ] fsi

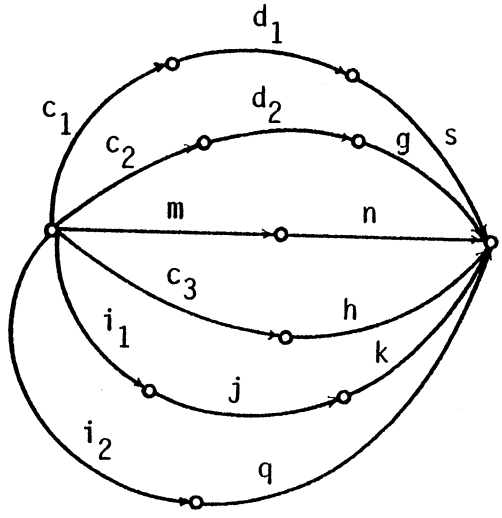
si sm(α)≠0 alors (∀β)[β∈$\underline{\Gamma}$(O) => sm(β):= min $\underline{\sigma}$(O)] fsi

fin

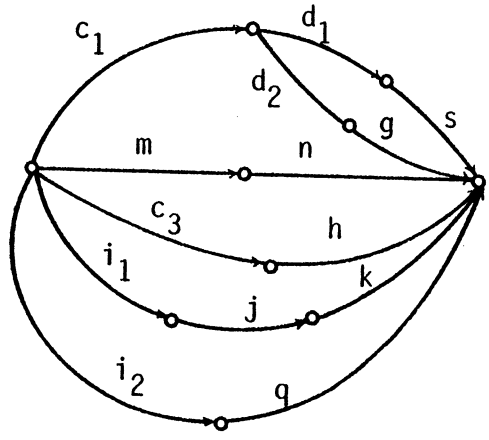
3.2.2. E X E M P L E.

On montre les transformations successives de G1 pour arriver au résultat G6.

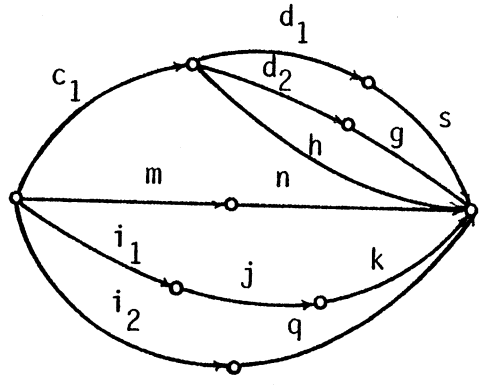
G1 =



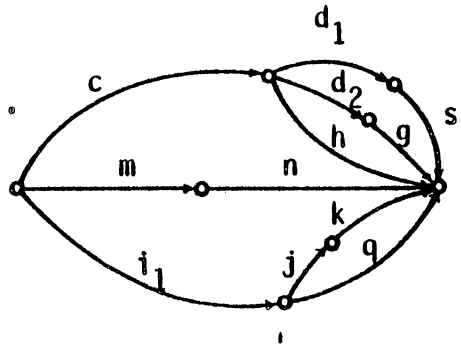
G2 =



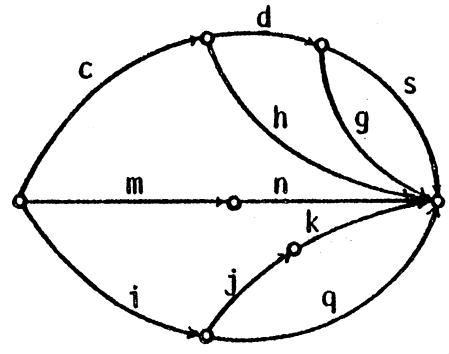
G3 =



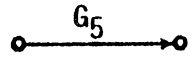
G4 =



G5 =



G6 =



4. ARBRE ASSOCIE A UN E-GRAPHE.

4.1. MONITEUR DE L'ALGORITHME. ARBO(I,O,α)

ARBO construit un arbre associé à un E-graphe G déterminé par l'entrée I, la sortie S et son premier arc α. Cette construction est d'abord réalisée récursivement sur tous les segfi-élémentaires au moyen du sous-programme SEGFI. Une fois transformés tous les segfi-élémentaires, on teste si G est non-régulier ce qui, dans l'affirmative, nous conduit à produire les trajectoires de G. Chaque trajectoire est transformée en un arbre par SEKEN (opération θ1) et le faisceau résultant est transformé en un arc par PARA (opération θ2). On change enfin la racine de cet arbre par alt (opération θ3).

Si G est régulier, alors on transforme tous les seg séquentiels (SEKEN), contenus dans G, en arbres. Puis, tous les faisceaux (PARA) aussi. On repète le processus jusqu'à avoir un seul arc étiqueté par l'arbre associé à G.

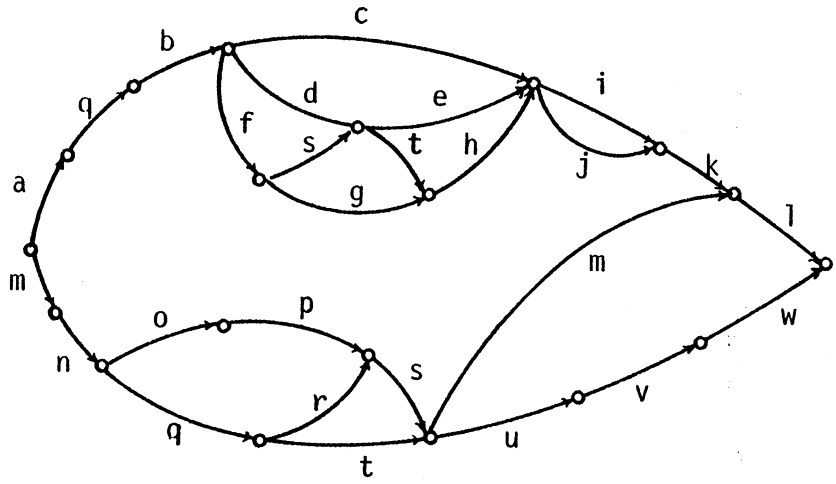
4.1.1. A L G O R I T H M E.

```
ARBO(I, O, α)
debut  * CONSTRUCTION D'UN ARBRE ASSOCIE A UN E-GRAPHE *

α:arc;   x:sommet;   Δ:arbre;   g:E-graphe

    * recherche de tous les segfl contenus dans le (sous) E-gr. *

    α:= min σ(I)
    repeter

        SEGFI(σ(α), x)
        si  x=O & σ(α)=I  alors  α:=su(α)
        sinon

            si  x≠0  alors
                Δ:= ARBO(σ(α), x, su(α))
                si  x≠O  alors  α:= min σ(x)
                sinon  α:=su(Δ)  fsi
            sinon  α:=su(α)  fsi
        fsi
    jusqu'a  α=0  frep

    * transformation du (sous) E-graphe *

    si  NONREGULIER(I, O, min σ(I))  alors

        g:= TRAJ(I, O, α)
        SEKEN(I', O', min σ(I))          * I'= I(g), O'= O(g) *
        PARA(I', O', min σ(I))
        racine(α):= alt

    sinon

        tantque  su(α)≠0  faire
            SEKEN(I, O, min σ(I))
            PARA(I, O, min σ(I))
        ftq

    fsi
fin
```

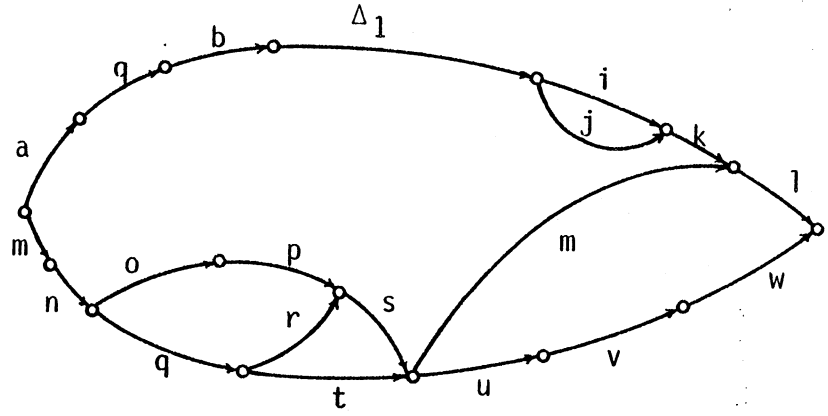
4.1.2. E X E M P L E.

Obtenons l'arbre associé à G1.

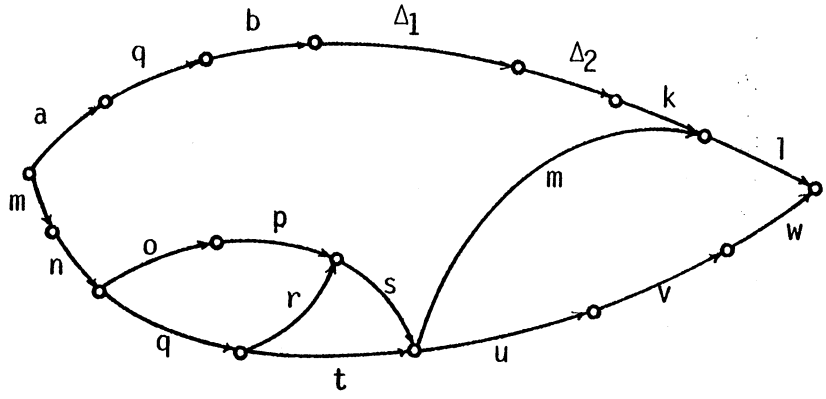
G1 =



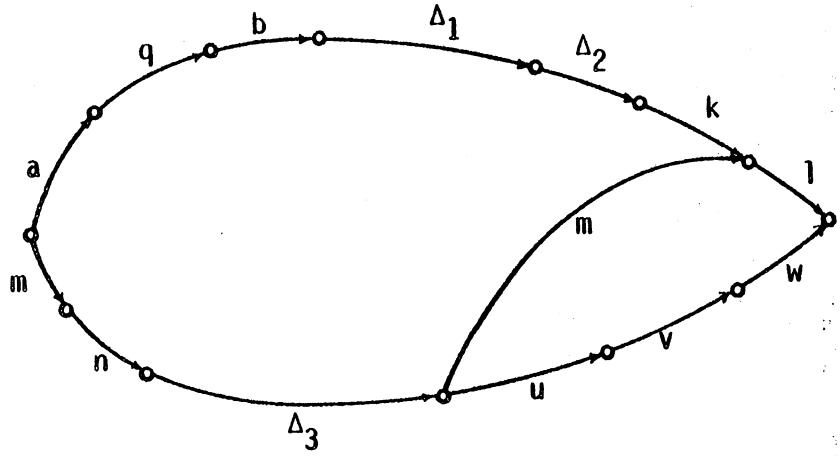
G2 =



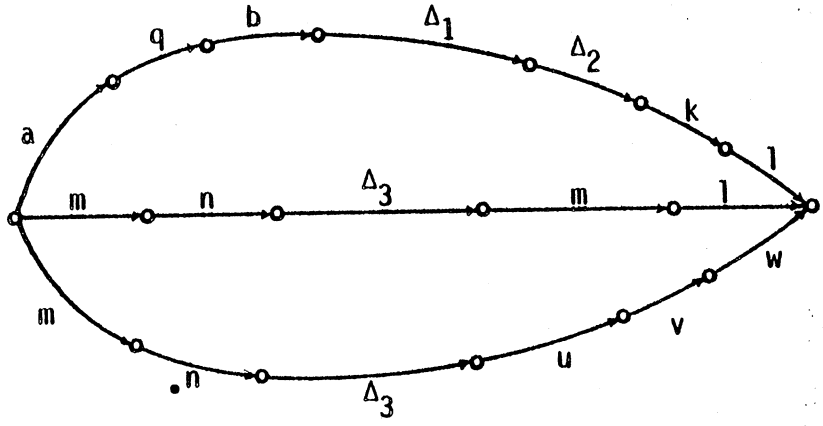
G3 =



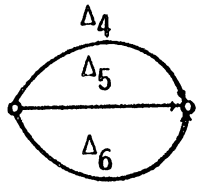
G4 =



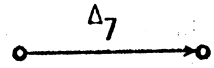
G5 =



G6 =



G7 =



avec Δ1= alt(e, et(d,e), et(d,t,h), et(f,s,e), et(f,s,t,h), et(f,g,h));
Δ2= ou(i,j) ; Δ3= ou(et(o,p,s), et(q,r,s), et(q,t));
Δ4= et(a,q,b,Δ1,Δ2,k,l); Δ5= et(m,n,Δ3,m,l);
Δ6= et(m,n,Δ3,u,v,w); Δ7= ou(Δ4,Δ5,Δ6).

4.2. OPERATION θ1 SEKEN(I,O,α)

SEKEN réalise la transformation de chaque seg séquentiel contenu dans un E-graphe G en un arbre. Cet E-graphe est déterminé par son entrée I, sa sortie S et son premier arc α.

L'algorithme parcourt canoniquement le E-graphe. Toute séquence d'arcs sans alternant et sans antécédent est stockée dans l'ordre dans une variable (Δ) de type liste. Ayant délimité la séquence, on passe à la transformation en un arbre. Celle-ci comporte la création d'un arc (λ), les affectations du contenu et son chaînage dans le E-graphe.

N.B. On adopte la notation et(Δ) pour désigner un arbre ayant comme racine et et comme fils la suite des valeurs de Δ.

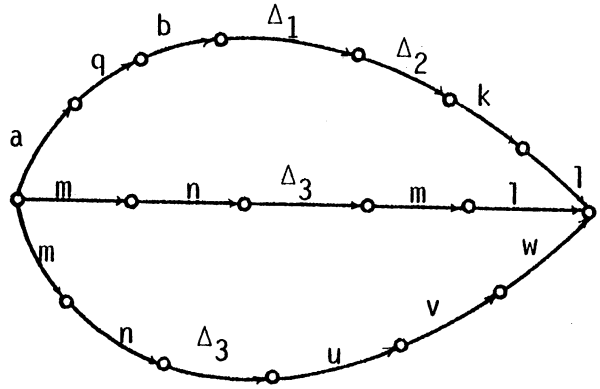
4.2.1. A L G O R I T H M E.

```
SEKEN(I, O, α)
debut  * TRANSFORMATION DES SEG-SEQUENTIELS EN ARBRES *

α, β, α1:arcs;    Δ:liste

    si  α=0  alors  α:= sm(α)  fsi
    β:= α

    tantque  su(α)≠0  faire      * parcours de tout le E-graphe *

        Δ:= 0

        * recherche d'un seg-séquentiel *
        repeter
            •
            Δ:= Δ.α
            α1:= α
            α:= sm(α)

        jusqu'à  α=0 V alt(α)≠0 V ant(α)≠0  frep


        * transformation en arbre *

        créer arc λ                          * création et chaînage *
        σ(λ):= σ(β)
        τ(λ):= τ(α1)
        alt(λ):= alt(β)
        sm(λ):= sm(α1)

        si  ant(α1)≠0  alors  (∀α)[α∈τ(ant(α1)) => sm(α):=λ]
        sinon  sm(0):=λ  fsi

        si  α=0 & su(α1)≠0  alors  α:= su(α1)  fsi
        β:= α

    ftq
fin
```

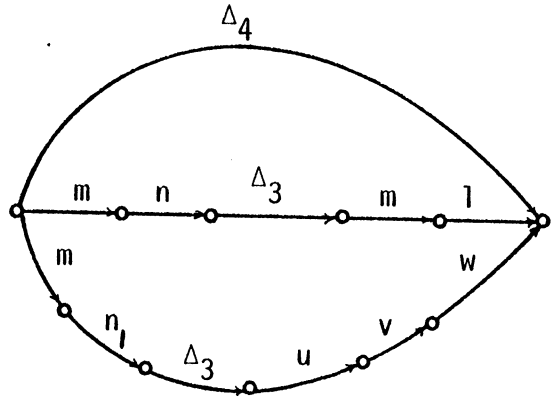
4.2.2. E X E M P L E.

Soit G5 de l'exemple précédent.

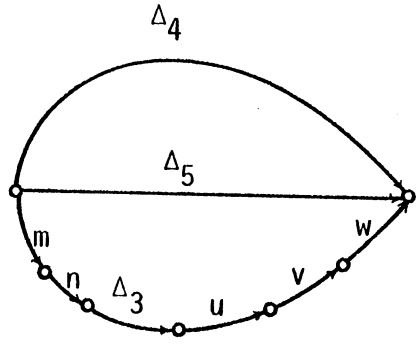
G5 =



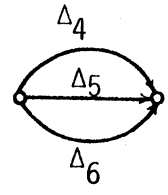
G6 =



G7 =



G8 =



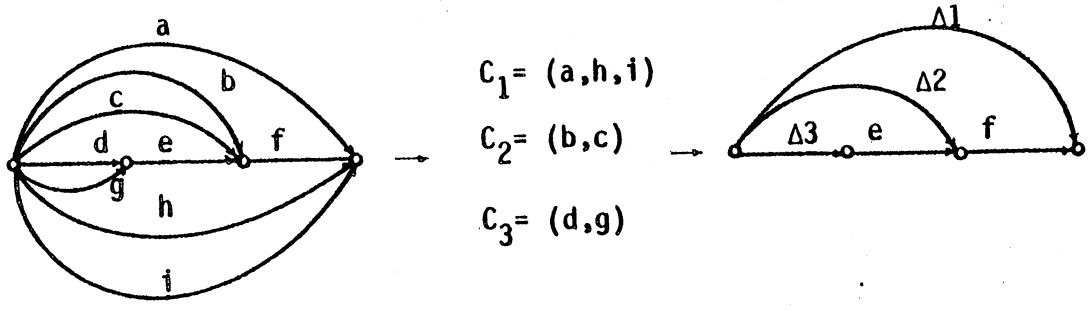
4.3. OPERATION θ2 PARA(I,O,α)

PARA réalise une recherche des faisceaux contenus dans un E-graphe G à entrée I, sortie O et premier arc α. Chaque faisceau rencontré est sitôt transformé en un arc étiqueté par son arbre associé.

L'algorithme parcourt G canoniquement à la recherche d'arcs à alternant non-nul. En ayant trouvé un, on effectue les opérations suivantes (admettons que c'est α l'arc rencontré):

On cherche les classes d'équivalence pour la rélation "avoir le même sm" sur l'ensemble $\overline{\sigma\sigma}(\alpha)$ d'arcs issus de $\sigma(\alpha)$. Puis, chaque classe est transformée en un arc étiqueté par son arbre associé, avec les opérations de création, d'affectation et de chaînage pour le nouvel arc.

Exemple des opérations réalisées.



avec Δ1= ou(a,h,i), Δ2= ou(b,c) et Δ3= ou(d,g).

4.3.1. A L G O R I T H M E.

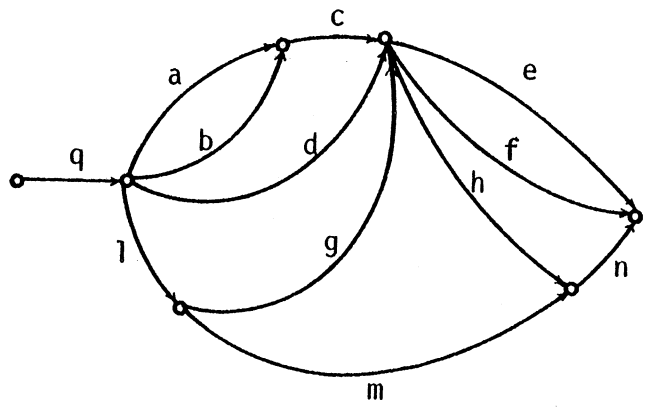
```
PARA(I, O, α)
debut  * TRANSFORMATION DES FAISCEAUX EN ARCS *

α, α1, α2, β, λ:arcs;   Δ:liste;   fini:booléen

    α:= min σ(I)

    * parcours de tout le E-graphe *

    repeter

        α2:= α

        * transformation du faisceau *
        tantque  alt(α)≠0  faire

            α1:= α;   Δ:=0;   β:=0

            * parcours de tous les alternants de la même classe *
            tantque  alt(α)≠0  faire

                fini:= faux

                * recherche d'arcs ayant "même sm" *
                tantque  sm(α)=sm(alt(α)) & -fini  faire

                    Δ:= Δ.α
                    si  alt(alt(α))≠0  alors  α:= alt(α)
                    sinon
                        Δ:= Δ.alt(α)
                        β:= alt(α)
                        fini:= vrai

                    fsi
                ftq

                * on saute les arcs n'ayant pas même sm *
                si  sm(α)≠sm(alt(α))  alors

                    Δ:= Δ.α
                    β:= α
                    tantque  sm(α)≠sm(alt(β)) & alt(β)≠0  faire
                        β:= alt(β)
                    ftq
                    si alt(β)≠0 & sm(α)=sm(alt(β)) alors α:=β fsi
                sinon  α:= alt(α)  fsi
            ftq
```

```
        * création et chaînage d'un nouvel arc *

          si  card Δ≠1  alors

              créer arc λ
              arbre(λ):= ou(Δ)
              σ(λ):= σ(α1)
              τ(λ):= τ(α1)
              sm(λ):= sm(α1)
              alt(λ):= alt(β)

              si  α2∉Δ  alors  alt(α1):=λ
              sinon

                  α2:=λ
                  si  ant(α1)=0  alors
                        (∀α)[α∈τ(ant(α1)) => sm(α):=λ]
                  sinon  sm(0):=λ  fsi
              fsi

              α:=λ
          fsi

          si  alt(α)≠0  alors  α:=alt(α)  fsi
        ftq

        si  su(α2)≠0  alors  α:=su(α2)  fsi

    jusqu'à  su(α)=0  frep
fin
```

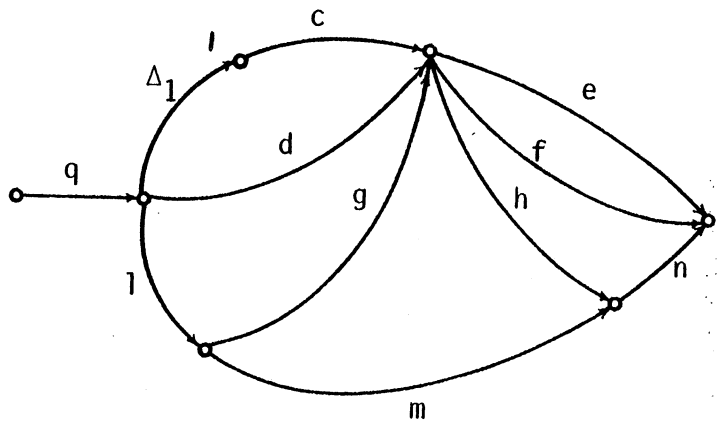
4.3.2. E X E M P L E.

Considérons maintenant le E-graphe suivant.

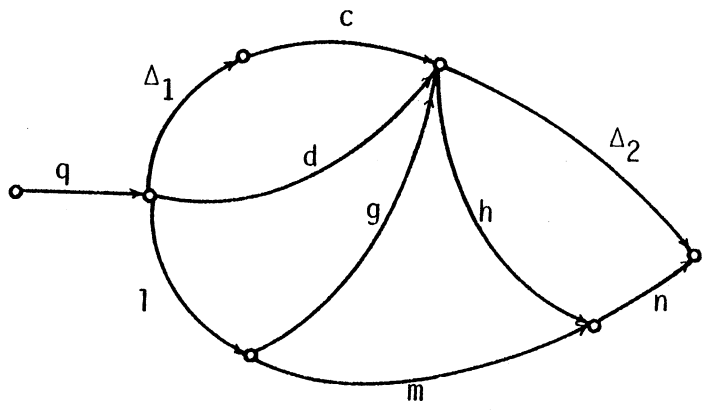
G1 =



G2 =



G3 =



5. EFFACEMENT D'UN SEG EFFAC((I',O',α'),(I,O,α))

EFFAC est l'algorithme qui élimine un seg G' à entrée I', sortie O' et premier arc α', d'un E-graphe G à entrée I, sortie O et premier arc α.

La méthode consiste à parcourir G en cherchant un arc égal à α'. S'il existe, on teste s'il y a coincidence entre G' et un seg de G au moyen du prédicat .COINCID. Celui-ci est défini par,

$$\text{COINCID}((I',O',\alpha'),\sigma(\beta)) \Leftrightarrow (\exists G'')[G'' \text{ seg de } G \;\&\; \beta \text{ premier arc de } G'' \;\; G''=G' \;\&\; \beta=\alpha']$$

Ayant trouvé G"=G' dans G, on regarde si G" est un segfi de G. Ceci est fait d'après la définition de segfi par le prédicat TESTSEGFI. Si G" est un segfi, on l'efface. On n'entre pas dans les détails puisque cette opération est essentiellement une manipulation de pointeurs.

Si G" n'est pas un segfi, on regarde s'il est un segmi au moyen du prédicat TESTSEGMI.

Si G" n'est pas un segmi, alors on analyse G afin de déterminer le segmi à effacer. On produit ainsi les arbres $\Delta\alpha'$ et $\Delta\alpha$ de G' et G respectivement, puis on cherche le sous-arbre $\Delta\alpha'$ dans $\Delta\alpha$. Deux cas peuvent se présenter,

(1) si $\rho(\Delta\alpha')$=et, alors on transforme le sous-arbre $\underline{\Delta\alpha'}$ en E-graphe.
(2) sinon, on transforme le sous-arbre $\underline{\text{père}(\rho(\Delta\alpha'))}$ en E-graphe.

Le E-graphe produit par la transformation de (1) ou de (2) deviendra le nouveau seg à effacer par l'appel récursif d'EFFAC.

5.1. A L G O R I T H M E.

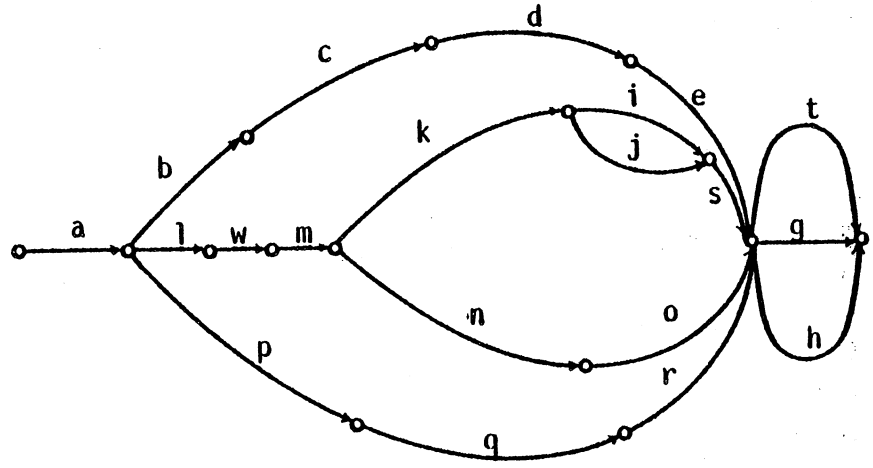
```
EFFAC((I', O', α),(I, O, β))
debut *ALGORITHME D'EFFACEMENT D'UN SEG G' D'UN E-GRAPHE G*

α, β, β1:arcs;    fini:booléean

    β1:=β
    fini:= faux

    tantque  -fini  faire

        * recherche dans G du premier arc de G' *
        tantque  β≠α & su(β)≠0  faire  β:=su(β)  ftq

        * existe-t-il G" seg de G, tel que G"=G'? *
        si  β=α & COINCID((I',O',α), σ(β))  alors

            * G" est-il un segfi? *
            si  TESTSEGFI((I",O",α),(I,O,β))  alors
                effacer segfi G" de G
                fini:= vrai
            sinon

                * G" est-il un segmi? *
                si  TESTSEGMI((I",O",α),(I,O,β))  alors
                    effacer segmi G" de G
                    fini:= vrai
                sinon

                    * Il faut chercher le segmi à effacer *
                    Δα:= ARBO(I',O',α)
                    Δβ:= ARBO(I,O,β1)

                    chercher Δα comme sous-arbre de Δβ

                    si  ρ(Δα)=et  alors
                        transformer le sous-arbre
                        Δα en E-graphe
                    sinon
                        transformer le sous-arbre
                        père(ρ(Δα)) en E-graphe
                    fsi
                fsi
            fsi
        fsi
    ftq
fin
```

5.2. E X E M P L E S.

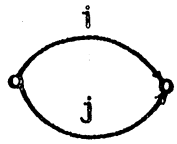
5.2.1. Effacement d'un segfl

On efface le segfl sg de G1 et G2 montre le résultat.

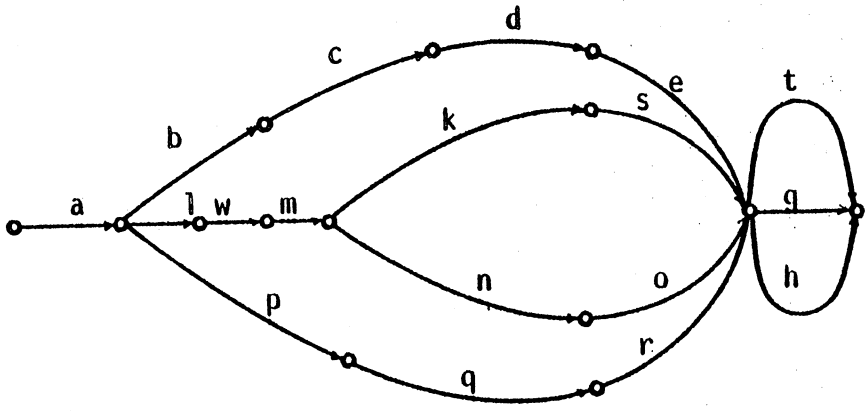
G1 =



sg =



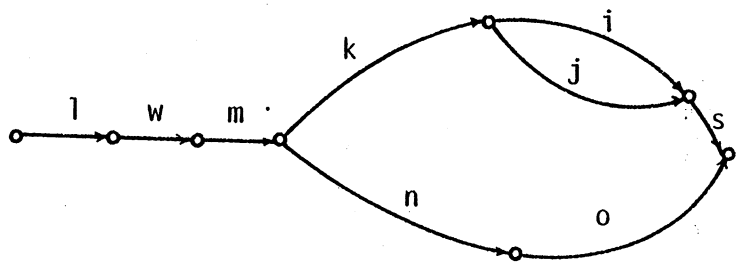
G2 =



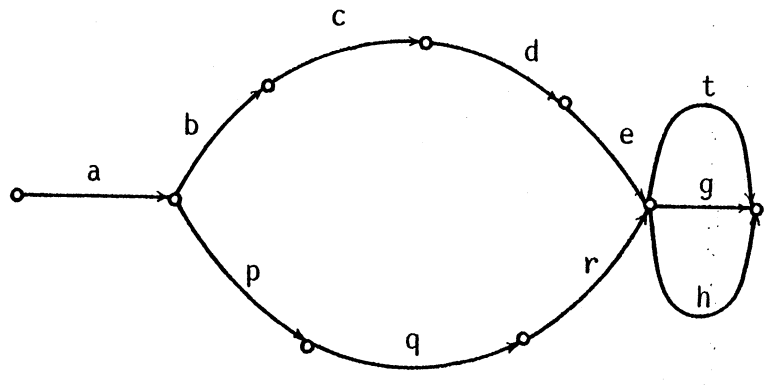
5.2.2. Effacement d'un segmi.

On efface le segmi sg_2 de G1. G3 montre le résultat.

sg_2 =



G3 =

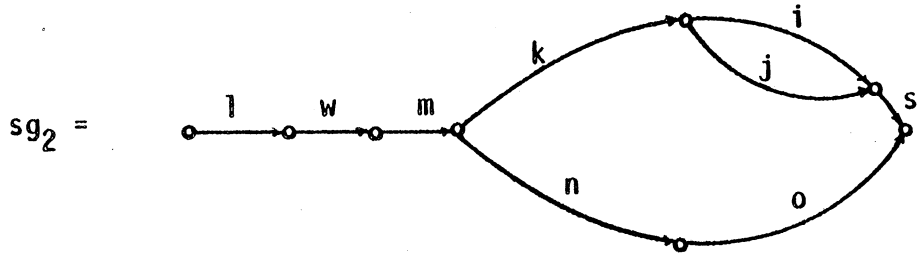


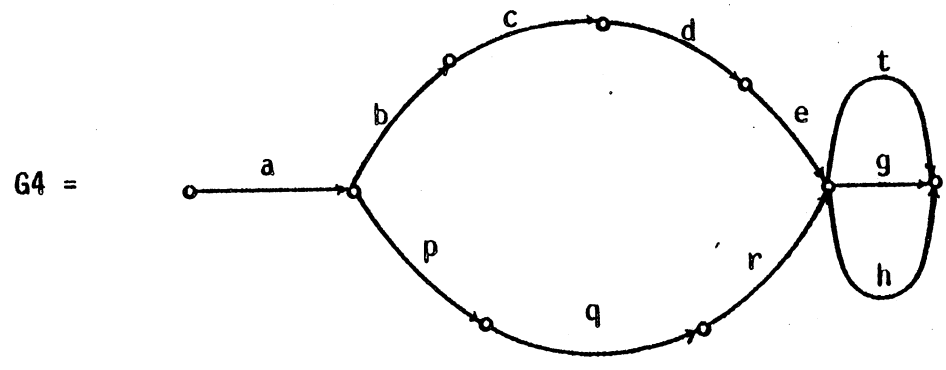
5.2.3. Effacement d'un segif.

On efface le segif sg_1=lw de G1. On montre les arbres:

Δ =et(l,w) et
Δ_{β}^{α}= et(a,ou(et(b,c,d,e),et(l,w,m,ou(et(k,ou(i,j),s),et(n,o))),et(p,q,r)),ou(t,g,h))

associés au segif et à G1 respectivement, ainsi que le "nouveau" seg à effacer qui n'est autre que sg_2 de l'exemple précédent. On obtient comme résultat G4.





6. RECHERCHE D'UN SEGFI

SEGFI(x,y)

Cet algorithme admet comme paramètre d'entrée un noeud $\underline{x}$ à partir duquel est commencée la recherche de l'entrée du segfi. Si $\underline{x}$ n'est pas l'entrée, il prend pour valeur celle de l'entrée. Comme paramètre de sortie, on a le noeud $\underline{y}$ qui aura pour valeur la sortie du segfi rencontré. S'il n'existe pas de segfi, $\underline{y}$ prendra pour valeur 0.

L'algorithme se divise en trois parties. La première recherche le premier noeud $\underline{z}$ tel que $|\underline{\sigma}(z)|>1$ et qui deviendra l'entrée si $|\underline{\sigma}(x)|=1$. On cherche aussi une sortie tentative en parcourant deux trajectoires distinctes jusqu'à ce que leur intersection soit non-vide. Si l'intersection entre trajectoires est toujours vide, on prend alors la sortie du E-graphe comme sortie tentative.

La deuxième partie vérifie que toute trajectoire sortant de $\underline{x}$ arrive à $\underline{y}$. S'il y en a une qui ne remplit pas cette condition, alors on la prend comme trajectoire courante et on revient à la première partie pour chercher une autre sortie tentative.

Enfin, la troisième partie teste la non-existence d'une trajectoire entrante en effectuant l'intersection entre trajectoires. Si $\underline{y}$ n'est pas la sortie du E-graphe, toutes les intersections doivent etre vides. Si $\underline{y}$ est la sortie du E-graphe, alors si $\underline{x}$ n'est pas l'entrée, toute trajectoire doit arriver à $\underline{x}$.

Voici quelques remarques concernant la notation. On désigne par

$$T(x,t,i)$$

la t-ième trajectoire considérée uniquement entre le noeud $\underline{x}$ et le but du i-ème arc (i.e. c'est un chemin particulier).

INTERSEC est un prédicat à trois arguments défini par,

$$\text{INTERSEC}(\gamma,\delta,\alpha) \iff (\gamma \text{ et } \delta \text{ sont des sous-trajectoires}) \;\&\; (\alpha = \text{le premier arc de l'intersection})$$

Finalement, en changeant le première instruction par,

$\underline{\text{si}}$ $|\underline{\sigma}(x)|>1$ $\underline{\text{alors}}$ tout le programme $\underline{\text{fin}}$

on aura un algorithme qui se prête mieux aux exigences d'autres programmes où on parcourt le E-graphe et dans lesqueles il serait gênant que $\underline{x}$ change de valeur. C'est-à-dire, on aura la formalisation,

$$y = \begin{cases} O' & \text{si } (\exists G' \text{ segfi de } G) \;\&\; x=I' \\ 0 & \text{sinon} \end{cases}$$

tandis que maintenant on a,

$$\exists G' \text{ segfi de } G \iff (\exists x,y)[x=I' \;\&\; y=O']$$

6.1. A L G O R I T H M E.

```
SEGFI(x,y)
debut  * LOCALISATION D'UN SEGFI DIFFERENT D'UN ARC *

α,β:arcs;    x,y:sommets;    t,tr,l,j:entiers
intervide,trajectoires,arcs,fini:booléen

    si  x≠O  alors

        * (1) recherche d'une sortie tentative y *

        intervide:= vrai;    t:= l:= j:= 1;    tr:=2

        tantque  intervide  faire

            * recherche d'un arc ou d'un sommet commun *
            tantque  intervide  faire
                si  INTERSEC(T(x,t,l),T(x,tr,j),α)  alors
                    y:=σ(α);    intervide:= faux
                sinon
                    si  T(x,t,l+1)=0 & T(x,tr,j+1)=0  alors
                        y:=O;    intervide:= faux
                    sinon
                        si T(x,t,l+1)≠0 alors l:=l+1 fsi
                        si T(x,tr,j+1)≠0 alors j:=j+1 fsi
                    fsi
                fsi
            ftq
```

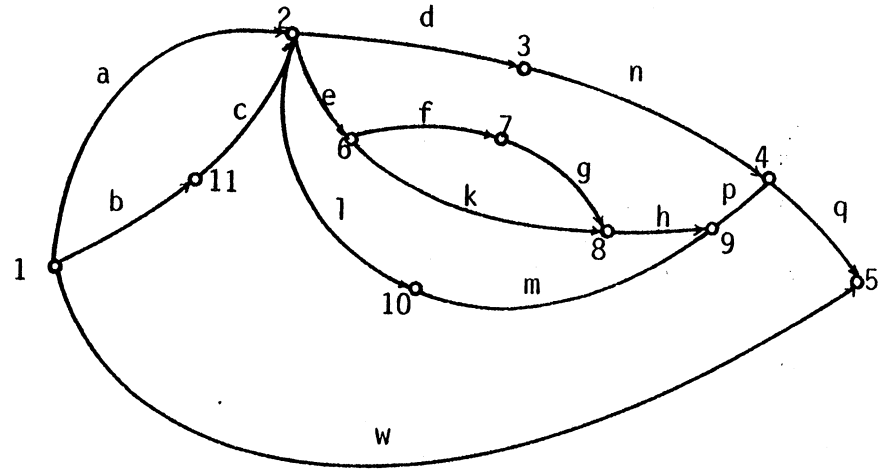
```
* (2) vérification que le reste des trajectoires
       de x passe par y *

si  y≠0  alors

    tr:=3;    j:= i:= 1
    si T(x,tr,j)≠0 alors
         trajectoires:= vrai
         arcs:= vrai
    fsi

    encore  trajectoires  faire

         encore  arcs  faire
              si INTERSEC(T(x,tr,j),α,α) alors
              ________ exit
              sinon
                   si T(x,tr,j+1)≠0 alors j:=j+1
                   sinon arcs:= faux fsi
              fsi
         fen

         si  -arcs  alors
              j:= t:= 1
              intervide:= vrai
         _________ exit
         sinon
              si T(x,tr+1,j)≠0 alors
                   tr:=tr+1
                   j:=1
              sinon
                   trajectoires:= faux
                   y:=σ(α)
              fsi
         fsi
    fen
fsi
ftq
```

```
*(3) test sur la non existence d'une trajectoire entrante*

t:= l:= 1;    intervide:= vrai;    fini:= faux

si  y≠0  alors

     encore  -fini  faire

          si  -INTERSEC(T(l,t,l),α,β)  alors
               si τ(T(l,t,l))≠x & T(l,t,l+1)≠0 alors l:=l+1
               sinon
                    si T(l,t+1,l)≠0 alors
                         t:=t+1
                         l:=1
                    sinon fini:= vrai fsi
               fsi
          sinon
               y:=0
          ________ exit
          fsi
     fen

sinon

     si  x≠l  alors

          encore  -fini  faire

               si -INTERSEC(T(l,t,l),min σ(x),β) alors
                    si T(l,t,l+1)≠0 alors l:=l+1
                    sinon
                         y:=0
               _____________ exit
                    fsi
               sinon
                    si T(l,t+1,l)≠0 alors
                         t:=t+1
                         l:=1
                    sinon fini:= vrai fsi
               fsi
          fen
     fsi
fsi
sinon  y:=0  fsi
fin
```

6.2. E X E M P L E.

Soit le E-graphe G suivant. On cherche un segfi de G à partir du sommet 2 avec l'appel: SEGFI(2 ,y)

G =



on obtient comme résultat: y=4.

III. SYSTEME DE REECRITURE.

Cet algorithme réalise un système de réécriture fondé sur les systèmes-Q. Comme eux, il se compose de deux parties: la phase d'"adjonction" et la phase de "nettoyage". De part son interêt du point de vue algorithmique, c'est la première qui est décrite ici.

1. PREMIERE PHASE. PHASAD(α,g)

L'idée de l'algorithme est d'appliquer toutes les règles d'une grammaire g sur tous les chemins d'un E-graphe, dont le premier arc est α, une fois et une fois seulement. Ce faisant, des arcs nouveaux sont introduits et l'algorithme s'applique aussi aux nouveaux chemins ainsi crées.

La méthode consiste à stocker une trajectoire (au début, la première), puis à effectuer deux parcours sur celle-ci. Le premier (po) est un "parcours ordinaire" et le second (pe) un parcours depuis l'entrée jusqu'au dernier arc de l'énumération de po sur la trajectoire. Les deux parcours sont réliés de la façon suivante: chaque arc produit par po provoque un parcours pe. C'est pendant ce dernier que l'on cherche à appliquer les règles de g sur les sous-chemins de la trajectoire. Le po ayant atteint la sortie, on stocke la trajectoire suivante et on refait les mêmes opérations.

Avec un changement trivial, on peut tenir compte d'une entrée I et d'une sortie O, transformant PHASAD en,

PHASAD(I,O,α,g)

on peut alors appliquer l'algorithme à des sous-structures.

1.1. A L G O R I T H M E.

```
PHASAD(α, g)
debut  * PHASE PREMIERE DE L'ALGORITHME DES SYSTEMES-Q *

α, β:arcs;    γ:liste d'arcs

    si  α=0  alors  α:=sm(α)  fsi
    γ:= première trajectoire

    * parcours de tout le E-graphe *
    tantque  α≠0  faire

        * parcours d'une trajectoire *
        tantque  α≠0  faire

            β:=0
            repeter        * avancement depuis I jusqu'à α *
                β:= su(β) dans γ
                APLREGLE(β, α, γ, g)
            jusqu'à  β≠α  frep
            α:=sm(α)
        ftq

        α:=β

        * trajectoire suivante *

        si  sm(α)=0  alors
            si  su(α)≠0  alors
                α:=su(α)
                γ:= trajectoire suivante
            sinon  α:=0 fsi
        fsi
    ftq
fin
```

1.2. APPLICATION DES REGLES. APLREGLE(β,α,γ,g)

APLREGLE cherche à appliquer toutes les règles d'une grammaire g sur une trajectoire γ, avec la restriction que la règle doit avoir β comme premier arc et α comme dernier pour les deux ordres, aussi bien horizontal que vertical.

La méthode consiste à parcourir une à une toutes les règles de g et à tester si la restriction est satisfaite. Ayant une règle susceptible d'être applicable, on demande que son membre droit (md) ne soit pas déjà un sous-chemin marqué du E-graphe pour procéder à l'ajout et au marquage du sous-chemin "délimité" par $\beta-\alpha$ dans γ.

Remarquez que tout chemin, produit par l'application d'une règle, est ajouté par le bas afin qu'il puisse être parcouru ultérieurement sans altérer le sens du parcours.

1.3. ALGORITHME D'APPLICATION DES REGLES.

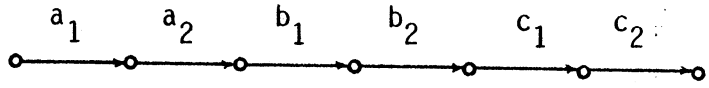
```
APLREGLE(β, α, γ, g)
debut  * APPLICATION D'UNE REGLE DANS UN E-GRAPHE *

r:règle

    r:= première règle de g

    * parcours de la grammaire *
    repeter

        si  tête(mg(r))=β & cola(mg(r))=α &
            tout arc de mg(r)∈γ  alors

            AJOUTER(md(r))
            marquer le chemin β-α
        fsi

        r:= règle suivante

    jusqu'à  r=0  frep

fin
```

2. E X E M P L E.

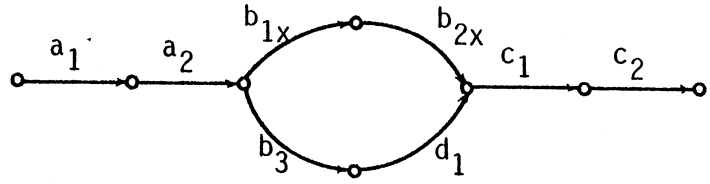
Soit G1 le E-graphe où on va appliquer la grammaire.

g = (1) abc == s
(2) asdc == s
(3) dc == cd
(4) bb == bd

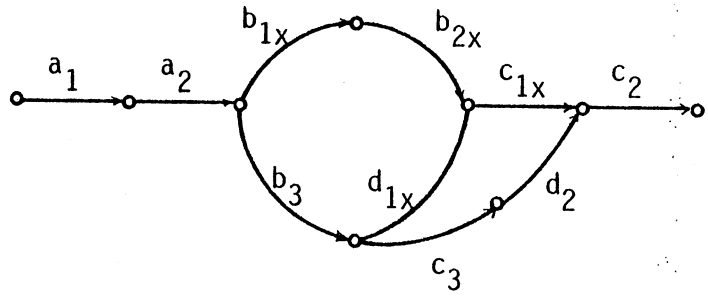
G1 =



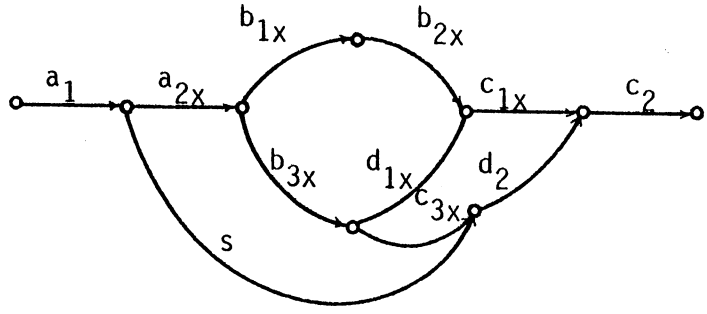
G2 =



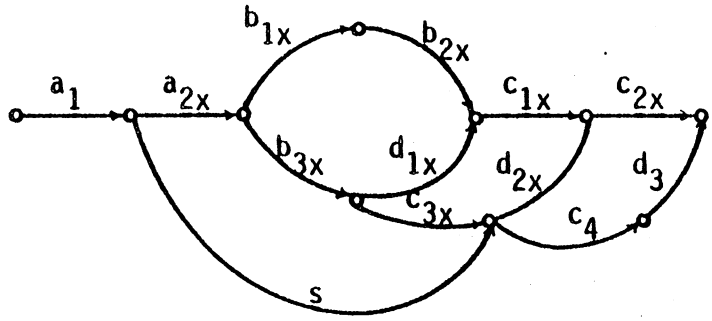
G3 =



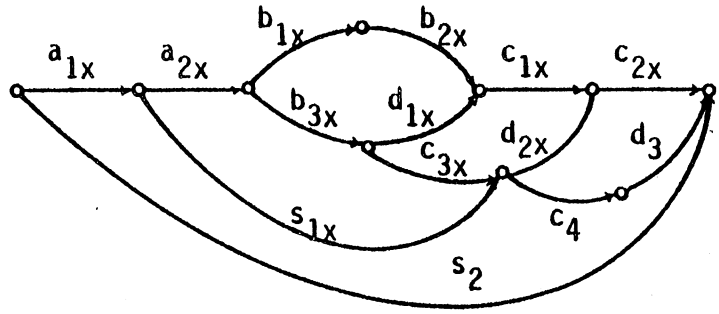
G4 =



G5 =


b1x
b2x
a1
a2x
c1x
c2x
b3x
d1x
d2x
d3
c3x
c4
s

G6 =


b1x
b2x
a1x
a2x
c1x
c2x
b3x
d1x
d2x
d3
c3x
c4
s1x
s2

G7 = G6

IV. C O N C L U S I O N.

Il y a quelques points sur lesquels je voudrais attirer l'attention. D'abord sur les algorithmes (et sauf pour les systèmes-Q), on a choisi de considérer toute sous-structure comme un E-graphe et non comme un schéma de E-graphe. Un autre choix est celui de prendre la première sous-structure rencontrée dans le parcours, lorsqu'il s'agit d'en localiser une. Il me semble cependant que les schémas d'algorithmes sont encore assez souples pour pouvoir considérer des schémas de E-graphe au lieu de E-graphes.

D'autre part, bien que je n'aie pas fait une comparaison systématique entre E-graphes non-réguliers et E-graphes réguliers, il apparaît que ces derniers ont quelques avantages sur les premiers qu'il faudrait analyser avec soin, notamment:

- simplification non négligeable des algorithmes pour le cas régulier (exception faite des systèmes-Q).
- concision de l'écriture polynômiale.

Ainsi, pour l'implementation d'un système important, il serait souhaitable de considérer les possibilités de remédier à leur défaut d'être une classe propre, sans évidemment diminuer les avantages offerts par la structure. On pourrait aussi analyser, pour les systèmes-Q, le cas où la partie gauche est un E-graphe au lieu d'un chemin et où on pourrait demander, par exemple, que les schémas soient réguliers.

TABLE BIBLIOGRAPHIQUE.

[1] Bolt*(77a)

[2] Bolt*(77b)

[3] Colm (70)

[4] Colm (75)

[5] Floy (67)

[6] Golo (65)

[7] Hirs (75)

[8] Knut (73)

[9] Wirt (76)

Dernière page d'une thèse

VU

Grenoble, le 6 mai 1982

Le Président de la thèse

F. [illegible]

Vu, et permis d'imprimer,

Grenoble, le 10.5.82

Le Président de l'Université Scientifique et Médicale

M. T ANC[illegible]