Systèmes de transformation de ramifications paramétrées : définitions et applications

Julio Ernesto Lopez Medina

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
«Génie Informatique»

par

Julio Ernesto LOPEZ MEDINA

«SYSTEMES DE TRANSFORMATION DE RAMIFICATIONS PARAMETREES. DEFINITIONS ET APPLICATIONS».

Thèse soutenue le 25 Juin 1979 devant la commission d'examen.

Monsieur	L. BOLLIET	Président
Madame	A. RECOQUE	Examinateurs
Messieurs	E. ANDRE	
	C. BOITET	
	J. COURTIN	
	C. DELOBEL	
	G. VEILLON	

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1977-1978

Président : M. Philippe TRAYNARD

Vice-présidents : M. René PAUTHENET
M. Georges LESPINARD

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Electronique - automatique
	BESSON Jean	Chimie minérale
	BLOCH Daniel	Physique du solide - cristallographie
	BONNETAIN Lucien	Génie chimique
	BONNIER Etienne	Métallurgie
*	BOUDOURIS Georges	Electronique - automatique
	BRISSONNEAU Pierre	Physique du solide - cristallographie
	BUYLE-BODIN Maurice	Electronique - automatique
	COUMES André	Electronique - automatique
	DURAND Francis	Métallurgie
	FELICI Noël	Electronique - automatique
	FOULARD Claude	Electronique - automatique
	LANCIA Roland	Electronique - automatique
	LONGEQUEUE Jean-Pierre	Physique nucléaire corpusculaire
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean-Charles	Chimie - physique
	PAUTHENET René	Electronique - automatique
	PERRET René	Electronique - automatique
	POLOUJADOFF Michel	Electronique - automatique
	TRAYNARD Philippe	Chimie - physique
	VEILLON Gérard	Informatique fondamentale et appliquée

* en congé pour études

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuël	Electronique - automatique
	BOUVARD Maurice	Génie mécanique
	COHEN Joseph	Electronique - automatique
	GUYOT Pierre	Métallurgie physique
	LACOUME Jean-Louis	Electronique - automatique
	JOUBERT Jean-Claude	Physique du solide - cristallographie

.../...

MM.	ROBERT André	Chimie appliquée et des matériaux
	ROBERT François	Analyse numérique
	ZADWORNY François	Electronique - automatique

MAITRES DE CONFERENCES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	CHARTIER Germain	Electronique - automatique
	CHIAVERINA Jean	Biologie, biochimie, agronomie
	IVANES Marcel	Electronique - automatique
	LESIEUR Marcel	Mécanique
	MORET Roger	Physique nucléaire - corpusculaire
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	SABONNADIERE Jean-Claude	Informatique fondamentale et appliquée
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie Physique

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

M.	FRUCHART Robert	Directeur de Recherche
MM.	ANSARA Ibrahim	Maître de Recherche
	BRONOEL Guy	Maître de Recherche
	CARRE René	Maître de Recherche
	DAVID René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	Maître de Recherche
	MATHIEU Jean-Claude	Maître de Recherche
	MERMET Jean	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (décision du Conseil Scientifique)

E.N.S.E.E.G.

MM.	BISCONDI Michel	Ecole des Mines St. Etienne (dépt. Métallurgie)
	BOOS Jean-Yves	Ecole des Mines St. Etienne (Métallurgie)
	DRIVER Julian	Ecole des Mines St. Etienne (Métallurgie)

.../...

MM.	KOBYLANSKI André	Ecole des Mines St. Etienne (Métallurgie)
	LE COZE Jean	Ecole des Mines St. Etienne (Métallurgie)
	LESBATS Pierre	Ecole des Mines St. Etienne (Métallurgie)
	LEVY Jacques	Ecole des Mines St. Etienne (Métallurgie)
	RIEU Jean	Ecole des Mines St. Etienne (Métallurgie)
	SAINFORT	C.E.N. Grenoble (Métallurgie)
	SOUQUET	U.S.M.G.
	CAILLET Marcel	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	COULON Michel	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	GUILHOT Bernard	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	LALAUZE René	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	LANCELOT Francis	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	SARRAZIN Pierre	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	SOUSTELLE Michel	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	THEVENOT François	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	THOMAS Gérard	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	TOUZAIN Philippe	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	TRAN MINH Canh	Ecole des Mines St. Etienne (Chim. Min. Ph.)

E.N.S.E.R.G.

MM.	BOREL	Centre d'études nucléaires de Grenoble
	KAMARINOS	Centre national recherche scientifique

E.N.S.E.G.P.

M.	BORNARD	Centre national recherche scientifique
Mme	CHERUY	Centre national recherche scientifique
MM.	DAVID	Centre national recherche scientifique
	DESCHIZEAUX	Centre national recherche scientifique

REMERCIEMENTS

A l'occasion de cette thèse, je tiens à remercier :

Monsieur Louis BOLLIET, Professeur à l'Université des Sciences Sociales de Grenoble qui m'a fait l'honneur de présider le Jury de cette thèse.

Monsieur Gérard VEILLON, Professeur à l'Institut National Polytechnique de Grenoble de m'avoir accueilli dans son équipe et d'avoir dirigé ce travail. Je tiens à lui exprimer ma profonde gratitude pour la confiance qu'il m'a toujours témoignée et pour ses nombreux conseils et encouragements.

Madame Alice RECOQUE et Monsieur Edouard ANDRE, Ingénieurs de recherche à la Compagnie CII-HB qui ont accepté de juger cette thèse et de participer au jury.

Messieurs Claude DELOBEL, Professeurs à l'Université Scientifique et Médicale de Grenoble, et Jacques COURTIN, Maître de Conférences à l'Université des Sciences Sociales de Grenoble, qui ont accepté de participer au Jury.

Monsieur Christian BOITET, Maître de Conférences à l'Université Scientifique et Médicale de Grenoble, qui a accepté de faire partie du Jury, et Monsieur Pierre BERLIOUX, Maître Assistant à l'Institut National Polytechnique de Grenoble. Ils ont bien voulu lire et critiquer en détail la rédaction provisoire de cette thèse, et m'ont certainement aidé à préciser plusieurs points importants.

Tous ceux qui ont contribué aux travaux rapportés ici, directement, par leurs critiques et suggestions, ou indirectement par l'accueil qu'ils nous ont témoigné. Je me dois de mentionner : Augustin LUX, Jean-Claude LATOMBE, Françoise VEILLON, Ernest GRANDJEAN, Danielle DUJARDIN, Marc DYMETMAN.

Je remercie également toux ceux, nombreux, qui m'ont aidé à améliorer la rédaction. Bien entendu les erreurs qui restent sont à mettre entièrement à mon compte ainsi que les 'colombianismes' que le lecteur voudra bien excuser.

Je remercie Madame SOUILLARD qui a assuré la dactylographie de ce texte avec une grande compétence et avec beaucoup de rapidité, et Monsieur D. IGLESIAS et le personnel du service de reproduction qui ont terminé la réalisation matérielle de ce travail.

SOMMAIRE

INTRODUCTION 1

CHAPITRE 0 : RAPPELS - DEFINITIONS

0.1. Ramifications 7
0.1.1. Définition par la théorie des graphes 7
0.1.2. Définition algébrique 11
0.2. Langage algorithmique 14

CHAPITRE I : SYSTEMES GENERAUX DE TRANSFORMATIONS

I.1. SGT Church-Rosser 18
I.2. Systèmes de transformations paramétrées : STP 30
I.3. STP Church-Rosser 41
I.4. La puissance descriptive de STP T-fermés 49

CHAPITRE II : AUTOMATES ADAPTES AUX RAMIFICATIONS PARAMETREES

II.1. Les réseaux de transition 59
II.2. Les automates d'états finis adaptés aux ramifications paramétrées 63
II.2.1. Automate d'états finis adapté aux ramifications (AR) 63
II.2.2. AR paramétrées 72
II.2.3. Composition d'ARP 74
II.3. STP et ARPC 90

CHAPITRE III : APPLICATIONS

III.1. Le système PIAF : Programmes Interactifs d'Analyse du Français 103
III.2. Une application à l'indexation automatique 112
III.3. Une application à l'analyse de langages formels : le traitement des requêtes relationnelles algébriques dans une base de données répartie. 123
III.4. Une application au traitement automatique de la langue naturelle. 138

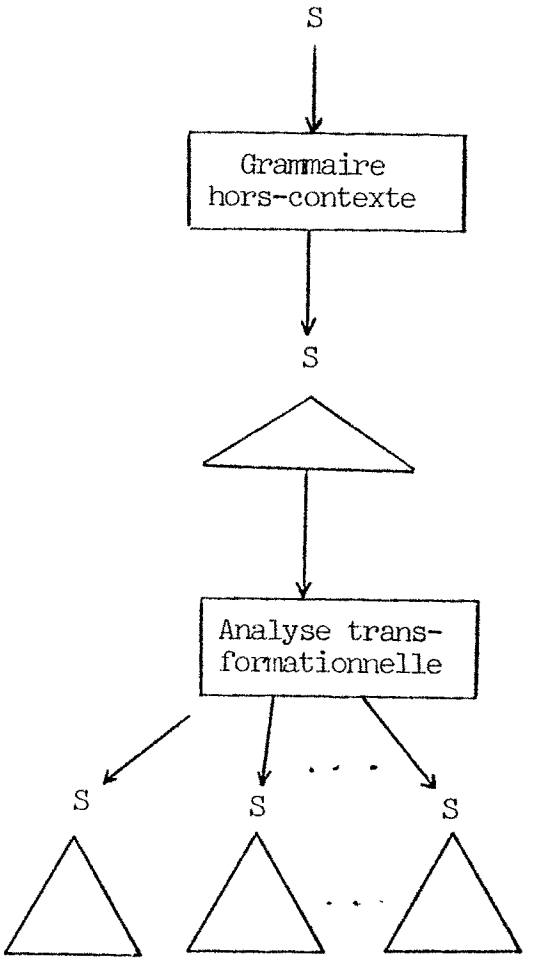
CONCLUSION 157

BIBLIOGRAPHIE 158

INTRODUCTION

Les transformations de ramifications jouent un rôle important dans l'analyse des langages formels et des langues naturelles. Bien que lorsque l'on a proposé l'analyse transformationnelle pour l'analyse des langues, les transformations de ramifications étaient liées à l'analyse syntaxique des langues naturelles [Chomsky,1955, 1957] , elles sont actuellement utilisées pour l'analyse post-syntaxique des langages formels [De Remer, 1974] ; et des langues naturelles [Gladky et Melt'chuk, 1969 ; Chauché, 1974].

La démarche proposée par N.Chomsky consistait à générer des phrases à partir d'une grammaire hors-contexte, puis sur chacune des structures syntaxiques produites (appelées structures profondes) à exécuter un ensemble de transformations qui exprimaient différents phénomènes linguistiques (négation, passé composé, forme passive, etc ...). Ainsi arrivait-on à produire un ensemble relativement grand de phrases à partir d'une grammaire hors-contexte simple.



On peut dire que les buts de l'analyse transformationnelle étaient de simplifier la grammaire et de mettre en évidence des phrases linguistiquement proches (la forme passive et la forme active d'une phrase avaient ainsi la même structure profonde).

Certains chercheurs (Friedman, Plath, Petrick, Chauché) ont tenté d'utiliser l'approche inverse pour l'analyse des phrases. Ils ont dû résoudre le problème lié à la nature de la chaîne d'entrée (la phrase) qui n'est pas une structure arborescente. Pour obtenir cette structure ils ont utilisé l'analyse transformationnelle ordonnée (transformations de ramifications) et bien que ses procédés donnent des résultats, l'analyse syntaxique qu'ils font est souvent considérée comme assez couteuse (Woods, 1970)

Dans le but de l'analyse des phrases, au sein de l'équipe d' Algorithmique et Intelligence Artificielle à l'IMAG nous avons conçu, pour le système PIAF, un modèle de transformations de ramifications fondé sur ces procédés et orienté vers l'analyse post-syntaxique des langages. Le système PIAF (qui comprend une analyse morphologique, et une analyse syntaxique en dépendances) est composé de plusieurs modules indépendants (mais qui peuvent être exécutés en coroutine, c'est-à-dire qu'un module peut être activé dès qu'un sous-résultat est déterminé par un module précédent) plutôt que d'un seul module complet et de ce fait, complexe. Ceci simplifie la mise au point et le contrôle des modèles composant le système.

Ce travail présente la description de ce système transformationnel, il est divisé en quatre chapitres. Le premier chapitre, le chapitre 0, introduit les concepts et la notation utilisés ; ceux-ci sont extraits des travaux de Pair et Quere (1968) et Berlioux (1972).

Au chapitre I nous faisons un rappel des systèmes de transformations non-ordonnées et nous étudions les propriétés de Church-Rosser (1936) et de fermeture ('closed systems', Rosen, 1973). Les systèmes transformationnels de Gingsburg et Partie (1969) et de Chauché (1974) définissent une transformation étiquetée comme composée de trois éléments : deux ramifications (celle à transformer, et celle qui exprime la transformation) et un prédicat logique sur les étiquettes de la première. Nous avons ajouté un quatrième élément : une expression algorithmique qui servira à modifier les étiquetttes de la ramification transformée. L'utilisation de ce nouvel élément sera illustré dans la suite. Nous avons défini aussi la propriété T-fermeture plus forte que les propriétés mentionnées ci-dessus. Le but de ces propriétés est une caractérisation des systèmes de transformations paramétrées,permettant d'utiliser des stratégies sur les règles appliquées et sur l'ordre d'application et ainsi de réduire le temps d'exécution et la place mémoire utilisée pour leur application.

L'implantation des systèmes de transformations paramétrées nous a conduit à l'analyse du problème de la manipulation et de la reconnaissance des ramifications paramétrées. De l'étude des automates pour la reconnaissance des ramifications [Pair et Quere, Berlioux, Brainerd, Thatcher] nous sommes arrivés à la définition d'un automate pour la reconnaissance d'une ramification paramétrée. Cet automate se présente comme un point de vue intermédiaire entre les automates d'états finis [Hopcroft et Ullman, 1969] et les automates des ramifications réguliers [Pair et Quere, 1968]. Une loi de composition de ces automates, fondée sur la notion de parallèlisme est introduite dans le but de reconnaître un ensemble donné de ramifications paramétrées. L'automate composé résultant (ARPC) est à rapprocher des réseaux de transition de Woods pour l'analyse sémantique, (appelés les "semantic ATN" [Woods, 1976]) la principale différence étant la construction automatique des ARPC à partir d'un ensemble de ramifications. A la fin du chapitre II on présente les algorithmes nécessaires pour l'utilisation des ARPC dans des systèmes de transformations T-fermés, Church-Rosser et non Church-Rosser.

Les théories exposées aux chapitres I et II ont donné lieu à l'élaboration d'un système informatique pour la reconnaissance et la transformation de ramifications. Ce système fait partie du logiciel PIAF [Courtin, Grandjean, Veillon] et est actuellement implanté à l'IMAG. Au chapitre III nous illustrons l'application de ce modèle dans les domaines de la documentation automatique, les bases de données et l'intelligence artificielle. Le problème abordé dans la première application est celui de l'indexation automatique et plus spécifiquement, la reconnaissance des groupes de mots non connexes, factorisés par la conjonction, et définis à l'aide de variables. Les deux autres applications concernent l'analyse post-syntaxique des langages formels (un langage d'interface avec une base de données relationnelle) et de langues naturelles (un interface en langue naturelle avec un système d'aide à la conception automatique). Nous développons les diverses techniques employées en faisant remarquer le caractère d'outil informatique des systèmes de transformations.

CHAPITRE 0

RAPPELS - DEFINITIONS

0.1. RAMIFICATIONS 7

0.1.1. Définition par la théorie des graphes 7

0.1.1.1. Graphe 7

0.1.1.2. Itinéraire 7

0.1.1.3. Cycle 7

0.1.1.4. Graphe connexe 7

0.1.1.5. Arborescence 7

0.1.1.6. Arborescence orientée 8

0.1.1.7. Etiquettes 8

0.1.1.8. Arborescence étiquetée 8

0.1.1.9. Pseudo-arborescences sur un vocabulaire V 8

0.1.1.9.1. Isomorphisme d'arborescences orientées 8

0.1.1.9.2. Equivalence d'arborescences orientées étiquetées 9

0.1.1.9.3. Définition 9

0.1.1.10. Ramifications sur un vocabulaire V 9

0.1.1.11. Sous-ramifications 9

0.1.1.11.1 Définition 9

0.1.1.11.2. Sous-ramification complète 10

0.1.1.11.3. Exemples 10

0.1.2. Définition algébrique 11

0.1.2.1. Binoïde 11

0.1.2.2. Structure de binoïde sur V 11

0.1.2.2.1. Concaténation 11

0.1.2.2.2. Enracinement. 11

0.1.2.3. Principe de récurrence 12
0.1.3. Définitions sur les ramifications 12
0.1.3.1. Mot de racines 12
0.1.3.2. Mot de feuilles 12
0.1.3.3. Sommets d'une ramification 13
0.1.3.4. Descendants d'un sommet 13
0.1.3.5. Sommets indépendants 13
0.1.3.6. Sous-ramification complète à partir d'un sommet 13
0.1.3.7. Longueur d'une ramification 13
0.1.3.8. Remplacement de ramifications 14
0.1.3.9. Ecriture préfixe des sommets d'une ramification 15

0.2. Langage algorithmique. 15

1. RAMIFICATIONS

On trouvera dans [Pair & Quere, 1968] et [Berlioux, 1972] les définitions et démonstrations suivantes.

1.1. Définition par la théorie de graphes.

1.1.1. Graphe

Un graphe est un doublet $\langle X,U \rangle$, où X est un ensemble de sommets et U une relation binaire sur X.

. $\forall x,y \in X$ on écrira $x \; U \; y$ si $(x,y) \in U$.

. On note $U(x)$ l'ensemble des $y \in X$ tels que $x \; U \; y$.

1.1.2. Itinéraire

C'est un ensemble ordonné de sommets $x_1,\ldots, x_n$ tel que

$\forall i \in N, 1 \leq i \leq n, x_i \; U \; x_{i+1}$ ou $x_{i+1} \; U \; x_i$

1.1.3. Cycle

C'est un itinéraire $x_1,\ldots, x_n$ tel que $x_1 = x_n$

1.1.4. Graphe connexe

C'est un graphe tel que pour tout couple de sommets (x_0, x_n) il existe au moins un itinéraire $x_0, x_1,\ldots, x_n$.

1.1.5. Arborescence

C'est un graphe connexe sans cycles tel que :

. $\forall x \in X$ il existe au plus un y tel que $y \; U \; x$.

. Il existe un sommet et un seul, appelé racine de l'arborescence pour lequel il n'existe aucun $y \in X$ tel que $y \; U \; X$.

On appelle feuille tout élément $x \in X$ pour lequel il n'existe aucun $y \in X$ tel que $x \; U \; y$.

1.1.6. Arborescence orientée

C'est un triplet <X, U, O> tel que :

- <X,U> est une arborescence
- <X,O> est un ordre partiel tel que :
  - . les restrictions de O à chacun des sous-ensembles U(x), x $\in$ X, sont des ordres totaux.
  - . Si x U y et x $\not U$ z, x, y, z $\in$ X alors y et z ne sont pas comparables par la relation O.

1.1.7. Etiquettes

Une étiquette est un élément de la famille des parties d'un ensemble V_E. L'ensemble d'étiquettes est noté V.

. $V \subset \mathcal{P}(V_E)$

V_E est appelé l'ensemble d'éléments d'étiquette.

1.1.8. Arborescence étiquetée

C'est un triplet <X, U, Δ> où Δ est une application de X dans V.

1.1.9. Pseudo-arborescences sur un vocabulaire V.

1.1.9.1. Isomorphisme d'arborescences orientées.

Deux arborescences orientées <X, U, O> et <X', U', O'> sont dites isomorphes s'il existe une application f de X dans X' tel que :

- f est une bijection
- $\forall$ x,y $\in$ X | x U y on a f(x) U' f(y)
- $\forall$ x,y $\in$ X | x O y on a f(x) O' f(y)

1.1.9.2. Equivalence d'arborescences orientées étiquetées

On définit $\sim$ une relation sur l'ensemble d'arborescences étiquetées sur un même vocabulaire V par :

$\langle X, U, O, \Delta \rangle \sim \langle X', U', O', \Delta' \rangle$ si et seulement si

- il existe un isomorphisme f entre $\langle X, U, O \rangle$ et $\langle X', U', O' \rangle$
- $\forall x \in X , \Delta(x) = \Delta'(f(x))$

Il est clair que $\sim$ est une relation d'équivalence.

1.1.9.3. Définition

On appellera pseudo-arborescence sur V tout classe de la relation $\sim$ précédemment définie. L'ensemble de pseudo-arborescences sur V est noté $V^!$

1.1.10. Ramifications sur un vocabulaire V

On appelle ainsi toute suite finie ordonnée de pseudo-arborescences sur V. On note $\hat{V}$ l'ensemble de ramifications sur V.

1.1.11. Sous-ramifications

1.1.11.1. Définition

Soit $r = \langle X, U, O, \Delta \rangle$ une ramification sur V. Notons O_s la relation successeur immédiat dans O:

$x \; O_s \; y$ ssi $x \; O \; y$ et $\nexists z \in X \mid x \; O \; z$ et $z \; O \; y$

On appelle sous-ramification de r toute ramification $r' = \langle X', U', O', \Delta' \rangle$ où :

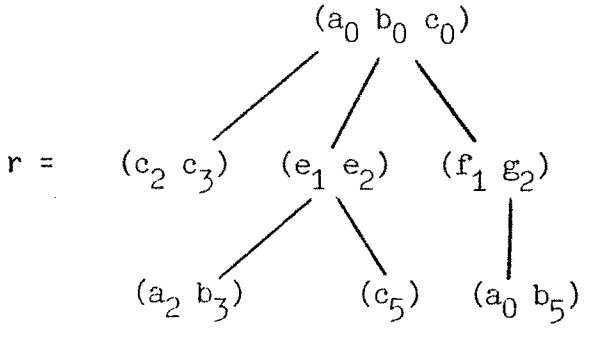
- $X' \subseteq X$
- Δ' est la restriction de Δ à X'
- $\forall\, x,y \in X'$
  - $x\, U'y \iff x\, U\, y$
  - $x\, O'_s y \iff x\, O_s\, y$

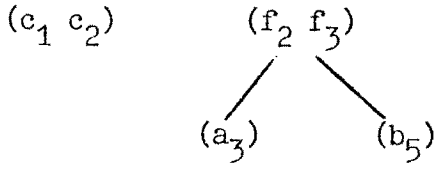
1.1.11.2. Sous-ramification complète

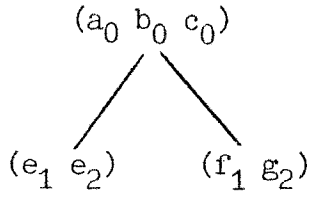
C'est une sous-ramification $r' = \langle X', U', O', \Delta\rangle$ de $r = \langle X, U, O, \Delta\rangle$ telle que $x \in X'$, $x\, U\, y$ impliquent $y \in X'$.

On écrit $r' \subset r$.

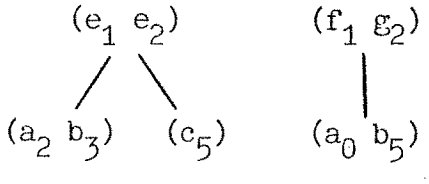
1.1.11.3. Exemples







est une sous-ramification.



est une sous-ramification complète

1.2. Définition algébrique

1.2.1. Binoïde

On appelle binoïde sur un ensemble V un ensemble muni :

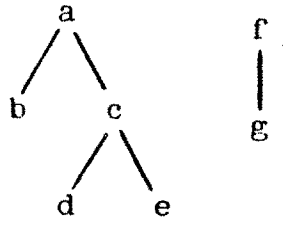
- d'une loi de composition interne associative admettant un élément neutre, noté ici +
- d'une loi de composition externe à opérateurs dans V, notée ×.

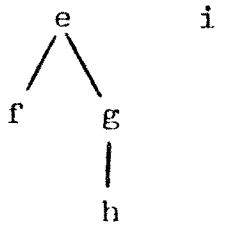
1.2.2. Structure de binoïde sur $\hat{V}$.

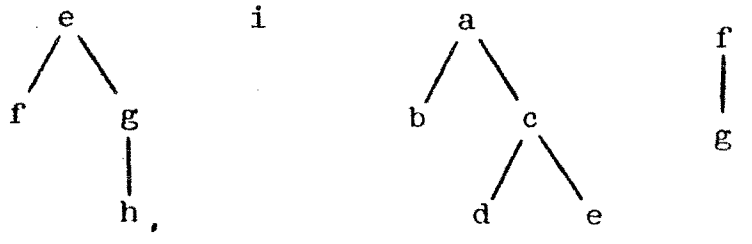
1.2.2.1. Concaténation

La loi + sur $\hat{V}$ est la concaténation de deux ramifications. Son élément neutre la ramification vide Λ.

Exemple :

r = 

, r' = 

r' + r = 

1.2.2.2. Enracinement

La loi × est appelée enracinement d'un élément de V sur une ramification.

- $\forall a \in V \quad a \times \Lambda = a$
- si r est une suite non vide de pseudo-arborescences $\langle X_i, U_i, 0_i, \Delta_i \rangle$ supposant les X_i disjoints deux à deux et désignant par e_i la racine de $\langle X_i, U_i \rangle$, a × r est la pseudo-arborescence $\langle X, U, 0, \Delta \rangle$.

. X est l'union des X_i et d'un élément e n'appartenant à aucun X_i.

. $x \; U \; y \iff (\exists i, x \; U_i \; y)$ ou $(x = e$ et $\exists i \mid y = e_i)$

. $x \; 0 \; y \iff (\exists i, x \; 0_i \; y)$ ou $(\exists i,j, i \leq j$ et $x = e_i$ et $y = e_j)$

. $\Delta(x) = \Delta_i(x)$ si $x \in X_i$; $\Delta(e) = a$

Réciproquement pour toute pseudo-arborescence s sur V il existe un élément a de V et un seul, et une ramification r sur V et une seule tels que $s = a \times r$. $\hat{V}$ muni de + et × est un binoïde sur V.

De ces définitions découle :
Pour toute $r \in \hat{V}$ non vide, il existe $a \in V$; r', $r'' \in \hat{V}$ tels que :

$$r = (a \times r') + r''$$

1.2.3. Principe de récurrence

Si P est un prédicat tel que :

- $P(\Lambda)$ est vrai
- $\forall\, r, s \in \hat{V}$

  $(P(r)$ et $P(s)) \Longrightarrow \forall\, a \in V\ ,\ P(a \times r + s)$

  alors $P(t)$ est vrai pour tout $t \in \hat{V}$.

1.3. Définitions sur les ramifications

1.3.1. Mot des racines

C'est le mot obtenu en concaténant les racines de la ramification de gauche à droite par l'application **rac** de $\hat{V}$ dans V^*.

$$rac(\Lambda) = \varepsilon$$

$$rac(a \times s + r) = a.rac(r)$$

1.3.2. Mot des feuilles

C'est le mot obtenu par l'application **feuille** de $\hat{V}$ dans V en concaténant les feuilles de la ramification de gauche à droite.

$$feuille(\Lambda) = \varepsilon$$

$$feuille(a \times s + r) = (\underline{si}\ s = \Lambda\ \underline{alors}\ a.feuille(r)\ \underline{sinon}\ feuille\ (s).\ feuille(r)).$$

1.3.3. Sommets d'une ramification

Si $r = \langle X, U, O, \Delta \rangle$ est une ramification, l'ensemble X des sommets de r sera parfois noté SOMMET(r).

1.3.4. Descendants d'un sommet

C'est la fermeture transitive de la relation U définie sur les sommets d'une ramification.

Soit r une ramification, $r = \langle X, U, O, \Delta \rangle$, $x, y \in X$

$x < y \iff y\ U^{+}\ x.$

1.3.5. Sommets indépendants

Soit $r = \langle X, U, O, \Delta \rangle$ une ramification, $x, y \in X$

$x \perp y \iff (x \not< y \text{ et } y \not< x)$

Par extension si $X_1 \subset X$, $Y_1 \subset X$

$X_1 \perp Y_1 \iff (\forall x \in X_1, y \in Y_1, x \perp y).$

1.3.6. Sous-ramification complète à partir d'un sommet

Soit r une ramification, la sous-ramification complète de r à partir de x, sommet de r, est la pseudo-arborescence notée $r/x = \langle X', U', O', \Delta'$ telle que :

- r/x est une sous-ramification complète de r.
- $\nexists\ y \in X' \mid y\ U'\ x$ ou $y\ O'\ x$ ou $x\ O'\ y$

1.3.7. Longueur d'une ramification

C'est le nombre d'éléments de la ramification.

longueur(Λ) = 0

longueur(a × s + r) = 1 + longueur(r)

1.3.8. Remplacement de ramifications

On appelle REPL l'ensemble de ramifications produites par le remplacement d'une occurrence et une seule d'une sous-ramification complète p d'une ramification r par une autre ramification t. REPL est donc une application de $\hat{V} \times \hat{V} \times \hat{V}$ dans la famille de parties de $\hat{V}$ définie comme suit :

$$\text{REPL}(\Lambda, p, t) = \Lambda$$

$$\text{REPL}(a \times u + v, p, t) = \underline{\text{si}}\ (1)\ \exists r', r'' \in \hat{V} \text{ tels que } a \times u + v = r' + p + r''$$

$$\underline{\text{alors}}\ \{r'+t+r''\} \cup \text{REPL}(r',p,t) \cup \text{REPL}(r'',p,t)\ \forall\ r', r''\ \text{satisfaisant (1).}$$

$$\underline{\text{sinon}}\ \{a \times \text{REPL}(u,p,t) + v\} \cup \{a \times u + \text{REPL}(v,p,t)$$

On appelle REPLN la ramification produite par le remplacement d'une pseudo-arborescence de racine n par une autre pseudo-arborescence ψ, dans la ramification r. REPLN est donc une application de $X \times \hat{V} \times \hat{V}$ dans $\hat{V}$ (où X est un ensemble de sommets) défini comme suit :

Si :

(i) r est une suite non vide de pseudo-arborescences

$$\langle X_i, U_i, O_i, \Delta_i \rangle\ ,\quad 1 \le i \le \ell;$$

(ii) n un élément de X_j, pour $1 \le j \le \ell$;

(iii) ψ une pseudo-arborescence $\langle X_\psi, U_\psi, O_\psi, \Delta_\psi \rangle$

alors REPLN (n, r, ψ) est la suite de pseudo-arborescences

$\langle X_i', U_i', O_i', \Delta_i' \rangle$, $1 \le i \le \ell$, telle que :

$$\langle X_i', U_i', O_i', \Delta_i' \rangle = \langle X_i, U_i, O_i, \Delta \rangle\ \forall\ 1 \le i \le \ell,\ i \neq j.$$

$$X_j' = (X_j - U(n)) \cup X_\psi$$

$$U_j' = U_j \cup U_\psi$$

$$O_j' = O_j \cup O_\psi$$

$$\Delta_j' = \Delta_j \cup \Delta_\psi$$

1.3.9. Ecriture préfixe des sommets d'une ramification

Toute ramification sur V peut être représentée par une chaîne de $(V \cup \{(,)\})^*$. Si on fait la convention d'écrire toute pseudo-arborescence dans une ramification comme sa racine suivi de ses fils entre parenthèses, la chaîne représentant la ramification sera appelée écriture préfixe des sommets de la ramification. On écrit PREORDRE l'application de $\hat{V}$ dans $(V \cup \{(,)\})^*$ qui donne l'ordre préfixé des sommets d'une ramification sur V.

PREORDRE(Λ) := ε

PREORDRE(a × r + t) := si r = Λ alors a.PREORDRE(t)
sinon a.(.PREORDRE(r).).PREORDRE(t) ;

où . est l'opération de concaténation des symboles.

2. LANGAGE ALGORITHMIQUE

Les algorithmes exprimés par la suite sont donnés sous forme de schémas de programme dans un langage de haut niveau dont les primitives sont :

- l'affectation notée :=
- l'instruction conditionnelle où snsi signifie sinon si
- l'itération indiquée par les instructions
  tant que expression booléenne faire et
  pour chaque élément dans un ensemble faire
- la définition de blocs début ... fin
- l'instruction :

  rendre expression ;

qui dans une fonction indique que l'algorithme s'arrête et que l'expression donnée est la valeur de la fonction.

CHAPITRE I

I.1. SGT CHURCH-ROSSER 18

I.1.1. Définitions 18

I.1.1.1. Dérivation 18

I.1.1.2. Ramification invariable 19

I.1.1.3. Boucle 19

I.1.1.4. Forme normale 19

I.1.1.5. SGT Précis 20

I.1.2. La propriété de Church-Rosser 20

I.1.3. SGT fermés 21

I.1.3.1. Fonction résidu 22

I.1.3.2. Définition 25

I.1.3.3. Théorème 25

I.1.3.4. Théorème 27

I.2. SYSTEMES DE TRANSFORMATIONS PARAMETREES : STP 30

I.2.1. Ramifications paramétrées 30

I.2.1.1. Paramètres 31

I.2.1.2. Ramifications paramétrées 32

I.2.1.3. Equivalence de ramifications 32

I.2.1.4. Exemple 34

I.2.2. STP 36

I.2.2.1. Règles de transformation 37

I.2.2.2. Application de transformations 40

I.3. STP CHURCH-ROSSER 41

I.3.1. STP T-fermés 42

I.3.2. STP fermés 45

I.3.3. Lemmes 46

I.4. LA PUISSANCE DESCRIPTIVE DE STP T-FERMES 49

Les ramifications et les structures arborescentes sont couramment utilisées dans l'étude des langues naturelles et des langages formels. Les systèmes de transformation sont un sous-ensemble important des systèmes de manipulation d'arborescences utilisés dans ces applications. Dans ce chapitre, on s'intéresse à l'étude de systèmes généraux de transformation sur des vocabulaires infinis ainsi qu'aux propriétés de Church-Rosser et de fermeture [B. Rosen, 1973] dans ces systèmes. Cette première propriété a ses origines dans les travaux de A. Church, et B. Rosser [1936] sur les systèmes de réduction de formules pour la logique formelle. Dans un système de transformation de Church-Rosser, le résultat de l'application exhaustive des transformations sur une ramification (c'est-à-dire jusqu'à ce qu'il n'y ait plus de règles applicables) est indépendant des règles de transformation appliquées et de l'ordre dans lequel elles sont appliquées. On remarque l'intérêt de travailler avec des systèmes qui possèdent cette propriété, car le fait d'avoir un résultat unique nous permet d'utiliser des stratégies sur les règles appliquées et sur l'ordre d'application dans le but de réduire le temps d'exécution et la place mémoire utilisée.

Dans le but de pouvoir décrire un nombre infini de règles de transformation par des moyens finis, nous allons étudier les systèmes de transformation paramétrées, et la propriété de T-fermeture (plus forte que celle de fermeture, qui est elle-même plus forte que celle de Church-Rosser). Dans un système T-fermé, le résultat du processus transformationnel sur une ramification est indépendant de l'ordre d'application des règles de transformation, mais une transformation reconnue comme applicable à un moment donné reste toujours applicable (jusqu'au moment de son application évidemment). Cette propriété permet l'exécution simultanée des processus de reconnaissance et d'application de transformations, d'où son intérêt.

Pour clore le chapitre, nous illustrons la puissance descriptive des systèmes paramétrés T-fermés avec un théorème sur sa capacité générative analogue à celui de A. Salomaa [1971] pour les grammaires transformationnelles de S. Ginsburg et B. Partee [1969].

I.1. Les Systèmes Généraux de Transformations et la propriété de Church-Rosser

Un système général de transformation (SGT) est un quadruplet : $G : \langle V, F, R, \Longrightarrow \rangle$ où

- V est un ensemble d'étiquettes ou vocabulaire ;
- F est un ensemble de ramifications sur V, $F \subseteq \hat{V}$, appelé l'ensemble de base ;
- R est l'ensemble des règles de transformation,

$$R \subseteq (\hat{V} - \Lambda) \times \hat{V}, \text{ et } \forall \langle \phi, \psi \rangle \in R \quad \phi \neq \psi .$$

si $r_i = \langle \phi, \psi \rangle \in R$ on écrira $r_i : \phi \rightarrow \psi$;

$\Longrightarrow$ est la relation de transformation de ramifications définie sur $\tilde{V} \times \hat{V}$:

$$(\Longrightarrow) = \{\langle p,q \rangle \mid$$

$$p,q \in \hat{V} \qquad (1)$$

$$\exists\, s \in \hat{V} \text{ et } r_i : \phi \rightarrow \psi$$

$$\text{tels que } s \subset p \text{ et } s = \phi \qquad (2)$$

$$q \in \text{REPL}(p, \phi, \psi) \qquad (3)$$

$$\}$$

où REPL est la fonction de remplacement de ramifications définie au paragraphe 0.3.7.

Si : $\langle p,q \rangle \in (\Longrightarrow)$ on écrit $p \overset{r_i}{\Longrightarrow} q$ où $r_i \in R$ est déterminé par (2).

I.1.1. Définitions

I.1.1.1. Dérivation

C'est une séquence finie de ramifications telle que chaque ramification est le résultat de l'application d'une règle de transformation r_i

sur la ramification qui la précède. Si la dérivation commence par p et finit par p', on dit alors que p' est dérivable de p.

$$p \xrightarrow{r_1} p_1 \xrightarrow{r_2} \ldots \xrightarrow{r_n} p'$$

ou

$$p \Longrightarrow^{+} p'$$

On écrit $p \Longrightarrow^{*} p'$ ssi $p = p'$ ou $p \Longrightarrow^{+} p'$

La dérivation composée de n applications de la même règle r_i sur une ramification p est écrite :

$$p \xrightarrow{r_i^n} p'$$

(i) $s \xrightarrow{r_i^+} t$ ssi $n \geq 1$ $s \xrightarrow{r_i^n} t$

(ii) $s \xrightarrow{r_i^*} t$ ssi $s = t$ ou $s \xrightarrow{r_i^+} t$

I.1.1.2. Ramification invariable

Une ramification est invariable relativement à un SGT G si aucune règle de transformation ne peut lui être appliquée.

I.1.1.3. Boucle

C'est une dérivation $p \Longrightarrow^{+} p_n$ telle que $p_n = p$.

I.1.1.4. Forme normale

Dans un SGT $G = V, F, R, \Longrightarrow >$ une ramification $s \in \hat{V}$ est une forme normale d'une ramification $p \in F$ ssi :

$p \Longrightarrow^{*} s$, et (1)

s est invariable relativement à G. (2).

I.1.1.5. SGT précis

Un SGT $G = \langle V, F, R, \Longrightarrow \rangle$ est précis ssi[1] R est une fonction, c'est-à-dire s'il n'y a pas deux règles de transformation $r_i : \phi_i \rightarrow \psi_i$, et $r_j : \phi_j \rightarrow \psi_j$ telles que $\phi_i = \phi_j$.

I.1.2. La propriété de Church-Rosser

Un système de transformations $G = (V, F, R, \Longrightarrow)$ possède la propriété de Church-Rosser ssi :

$(\forall\, p \in F;\ s, s' \in \hat{V})$

$p \Longrightarrow^* s$ et $p \Longrightarrow^* s'$ implique :

$\exists\, t \in \hat{V} \mid s \Longrightarrow^* t,\ s' \Longrightarrow^* t$, et t est invariable relativement à G.

Les corollaires suivants se déduisent facilement de cette propriété :

(1) Toute ramification dans F a une forme normale unique.

(2) Le résultat de l'application des règles de transformation sur une ramification donnée ne dépend pas de l'ordre dans lequel les règles de transformation ont été appliquées.

Si on veut exécuter un SGT par des moyens informatiques, ces corollaires simplifient le travail à faire. Il n'y aura pas besoin de suivre toutes les dérivations possibles à partir d'une ramification, car elles mènent toutes à la même ramification, sa forme normale. Il faut noter qu'il se peut que la dérivation d'une ramification vers sa forme normale comporte une boucle, et il faudra un traitement spécial pour que le programme informatique livre toujours un résultat.

---

[1] "si et seulement si"

Il reste tout de même le problème de vérifier la propriété de Church-Rosser pour un système donné. On ne connaît pas encore de solution à ce problème, aussi on définit des propriétés plus fortes et plus faciles à vérifier.

I.1.3. SGT Fermés

Dans ce paragraphe, nous allons caractériser certains systèmes généraux de transformations qui possèdent la propriété de Church-Rosser. Le but est de pouvoir reconnaître un SGT Church-Rosser par une étude de l'ensemble R des règles de transformation.

La notion de SGT fermé est illustrée par la figure suivante :

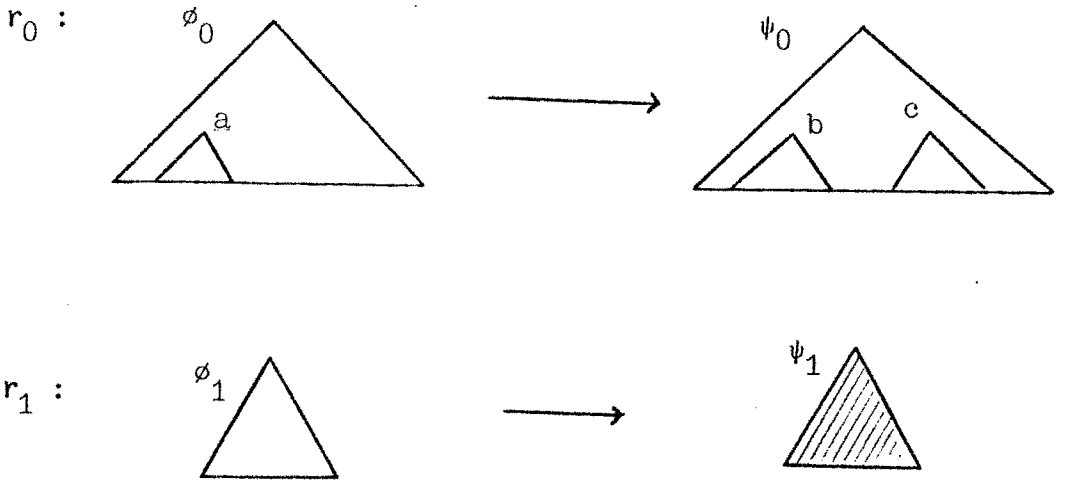


Fig. I.1.3.a.

Soient les règles de transformation $r_0 : \phi_0 \to \phi_0$, et $r_1 : \phi_1 \to \phi_1$ telles que $\phi_0/a = \psi_0/b = \psi_0/c = \phi_1$. Les sommets b et c sont appelés "résidus" de a, et on remarque que la règle r_1 est applicable sur les arborescences ϕ_0 (à partir du sommet a) et ψ_0 (à partir des sommets b

et c, les résidus de a).

L'application de la règle r_1 sur ϕ_0 et ψ_0, c'est-à-dire le remplacement de ϕ_0/a, ψ_0/b, et ψ_0/c par ψ_1 produit comme résultat les arborescences $\phi'_{0[a]}$ et $\psi'_{0[a]}$ (fig. I.3.b).

Fig. I.1.3.b.

La règle $r'_{0[a]} : \phi'_{0[a]} \rightarrow \psi'_{0[a]}$ est appelée une règle "de fermeture" de r_0. Un SGT est fermé si toutes les règles de fermeture des règles de transformation appartiennent au SGT.

I.1.3.1. Fonction résidu

Soit G = <V, F, R, ==> > un SGT, et $r_0 \in R$ une règle de transformation quelconque, une fonction :

$fr_0 : SOMMET(\phi_0) \rightarrow \mathcal{P}SOMMET(\psi_0)$ est une fonction résidu pour cette règle ssi :

$$\forall a \in SOMMET(\phi_0)$$

(1) $(\forall m \in fr_0(a))$ [$m \in SOMMET(\psi_0)$ et ($\exists r_i : \phi_i \rightarrow \psi_i$ |

$\phi_0/a = \psi_0/m = \phi_i$) et ($\nexists b \in SOMMET(\phi_0)$ | $m \in fr_0(b)$]

(2) $(\forall b \in SOMMET(\psi_0))$ [si $(\exists d \in SOMMET(\phi_0), r_i \in R \mid \psi_0/b = \phi_0/d = \phi_i)$

alors ($\exists x \in SOMMET(\phi_0)$ | $b \in fr_0(x)$)]

D'après (1) il peut y avoir dans ψ_0 des sommets b tels que $\psi_0/b = \phi_0/d$, pour un sommet d dans ϕ_0, et $b \notin fr_0(d)$ mais la condition (2) garantit que dans ce cas b est un résidu d'un autre sommet de ϕ_0. Donc, dans le cas où une arborescence ϕ_i se trouve incluse plusieurs fois dans ϕ_0 il y a plusieurs fonctions résidus pour certains sommets. Par exemple, si ϕ_0 est la règle :

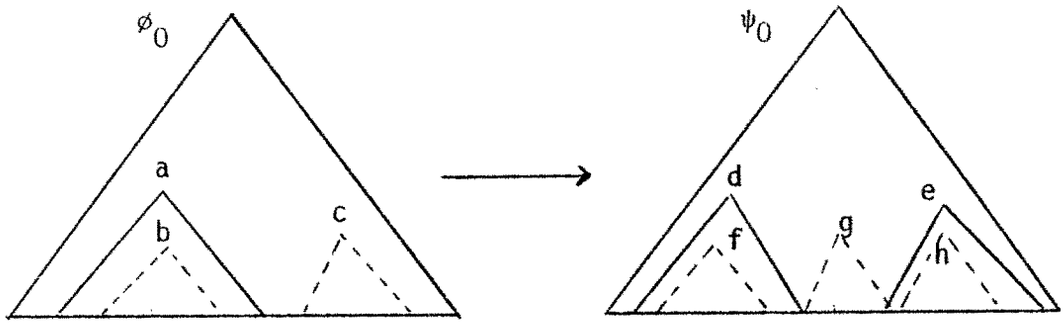


FIG. I.1.3.1.a.

Si $\phi_0/b = \phi_0/c = \psi_0/f = \psi_0/g = \psi_0/h = \phi_i$ pour un i quelconque, on a pour les sommets h et c les fonctions résidus suivantes :

(1) $fr_0(b) = \{f\} \Longrightarrow fr_0(c) = \{g,h\}$

(2) $fr_0(b) = \{f,h\} \Longrightarrow fr_0(c) = \{g\}$

(3) $fr_0(b) = \{f,g,h\} \Longrightarrow fr_0(c) = \phi$

etc ...

Chaque définition de $fr_0(b)$ donne suite à des règles de fermeture $r'_{0[b]}$ différentes. Si par exemple on prend la définition (1) on a :

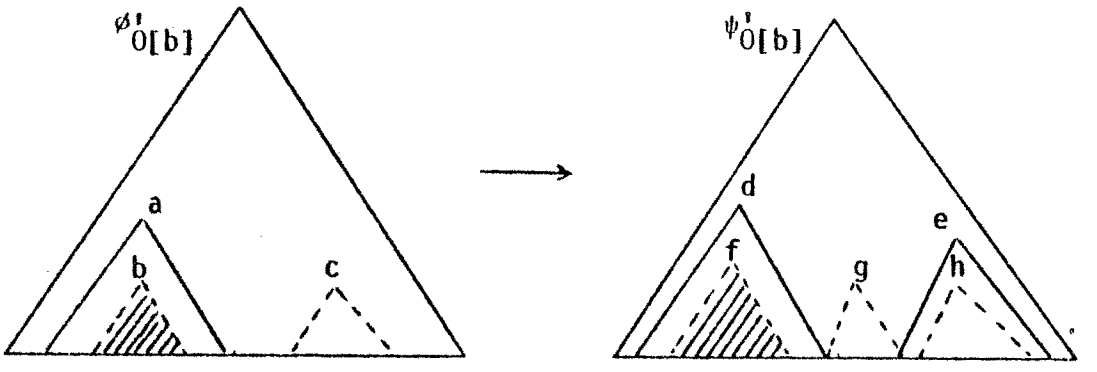


FIG. I.1.3.1.b.

où $\phi'_{0[b]}/b = \psi'_{0[b]}/d = \psi_i$.

Il faut remarquer que $fr_0(a) = \{d,e\}$, tandis que $fr'_{0[b]}(a) = \{d\}$. Cette modification de la fonction résidu du sommet a produit deux règles différentes avec la même partie gauche ; si $\phi_0/a = \phi_1$, ces deux règles sont $r'_{0[b][a]}$ et $r'_{0[a][fr_1(b)]}$. En conséquence, pour la construction d'une règle de fermeture la fonction résidu qui doit être prise est celle qui n'entraîne pas de modification des fonctions résidus des autres sommets de la règle.

I.1.3.2. Définition

Soit $G = \langle V, F, R, \Longrightarrow \rangle$ un SGT. G est un système fermé s'il est possible d'assigner une fonction résidu fr_0 à chaque règle $r_0 : \phi_0 \rightarrow \psi_0$ telle que :

Soit $M = \{m \mid m \in SOMMET(\phi_0)$ et $(\exists^1 r_m : \phi_m \rightarrow \psi_m \mid \phi_0/m = \phi_m)\}$

Pour tout m de M, il existe une règle dite "de fermeture"

(1) $r'_{0[m]} : \phi'_{0[m]} \rightarrow \psi'_{0[m]}$

où $\phi'_{0[m]} = REPLN(m, \phi_0, \psi_m^1)$

$\psi'_{0[m]} = REPLN(fr_0(m), \psi_0, \psi_m)$

(2) $(\forall a \in SOMMET(\phi_0), a \notin SOMMET(\phi_0/m)$

$fr_0(a) = fr'_{0[m]}(a)$

L'importance de la deuxième condition provient du fait qu'on évite ainsi la production de systèmes non-précis. Une autre conséquence est l'indépendance des ensemble fr_0 pour des sommets indépendants, comme le montre le théorème suivant :

I.1.3.3. Théorème

Dans un système fermé pour toute règle de transformation $r_0 : \phi_0 \rightarrow \psi_0$ deux sommets quelconques indépendants ont des résidus indépendants.

Soit $G = \langle V, F, R, \Longrightarrow \rangle$ un SGT fermé.

$r_0 = \phi_0 \rightarrow \psi_0 \in R$; $a,b \in SOMMET(\phi_0)$

$a \perp b \Longrightarrow fr_0(a) \perp fr_0(b)$

[1] La règle $r_m : \phi_m \rightarrow \psi_m$ est une règle quelconque de R appelée ainsi pour mieux visualiser sa relation avec ϕ_0/m.

Preuve (par l'absurde)

Supposons que $a,c \in$ SOMMET (ϕ_0)

$$a \perp c \text{ et } fr_0(a) \not\perp fr_0(c) \qquad (1)$$

(si on prend la figure I.1.3.1.a. par exemple, c'est le cas de la définition (1) de $fr_0(c)$).

Cela veut dire que $\exists\, h \in fr_0(c)$, $e \in fr_0(a)$ tels que $h < e$ (2)
et par définition d'une fonction résidu,

$$\psi_0/h = \phi_0/c = \phi_c \qquad (3)$$

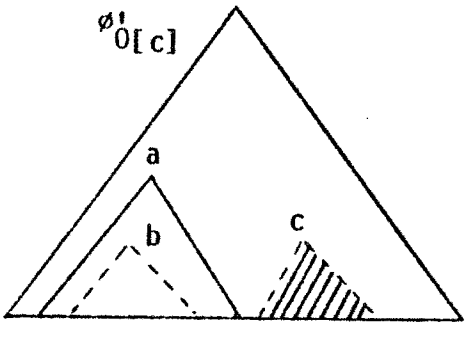
$$\psi_0/e = \phi_0/a = \phi_a \qquad (4)$$

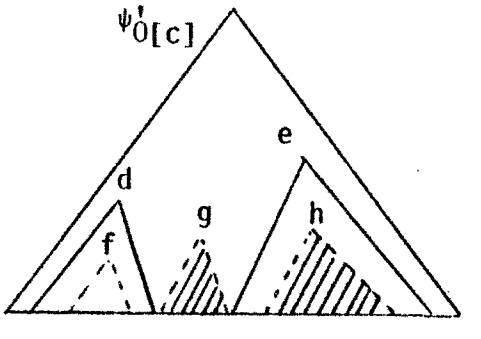
Par construction de la règle de fermeture r_0 au sommet c

$r'_{0[c]} : \phi'_{0[c]} \to \psi'_{0[c]}$ on a :

$\phi_0/a = \phi'_{0[c]}/a$ (et $a \perp c$)

et $\psi'_{0[c]}/h = \psi_c$ ($r_c : \phi_c \to \psi_c$)





puisque $\forall\, r_i : \phi_i \to \psi_i$, $\phi_i \neq \psi_i$ alors

$$\psi_0/e \neq \psi'_{0[c]}/e \quad \text{et} \quad \phi'_{0[c]}/a \neq \psi'_{0[c]}/e$$

ce qui veut dire que :

$$e \notin fr'_{0[c]}(a) \text{ et}$$

$$fr_0(a) \neq fr'_{0[c]}(a)$$

ce qui est en contradiction avec le fait que G est fermé.

Q.E.D.

Ce théorème s'avère très utile au moment de définir les fonctions résidus des sommets ; par exemple dans le cas de la figure I.1.3.1.a, les fonctions résidus (1) sont rejetées.

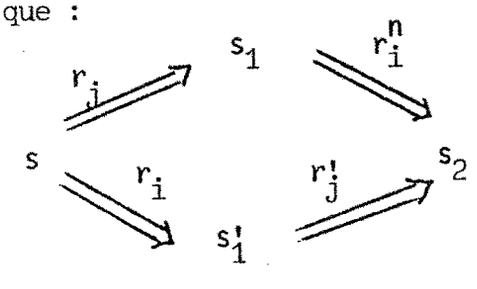
I.1.3.4. Théorème

Dans tout SGT fermé $G = \langle V, F, R, \Longrightarrow \rangle$, si sur une ramification s il y a deux règles de transformation r_i, r_j applicables :

$$s \xrightarrow{r_j} s_1$$

$$s \xrightarrow{r_i} s'_1$$

alors $\exists$ n, r'_j et s_2 tels que :



Preuve : Soit $r_i : \phi_i \rightarrow \psi_i$, $r_j : \phi_j \rightarrow \psi_j$, $a,b \in$ SOMMET(s) les sommets sur lesquels r_i et r_j sont applicables.

$$s/a = \phi_i, \quad s/b = \phi_j$$

Cas 1 : $a \perp b$, alors $n = 1$, $r'_j = r_j$.

Cas 2 : $a < b$

i) $s \overset{r_j}{\Longrightarrow} s_1$

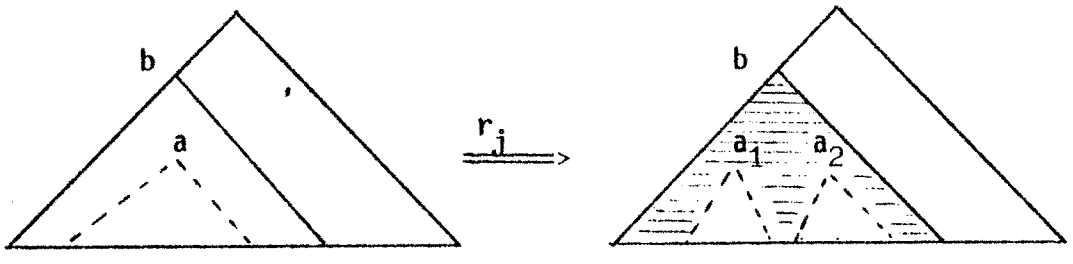


FIG. I.1.3.4.a.

On voit que $s_1/b = \psi_j$ et dans s_1 il y a zéro ou plusieurs sommets a_i pour lesquels $s_1/a_i = s/a = \phi_i$, c'est-à-dire $a_i \in fr_j(a)$. Soit n la cardinalité de cet ensemble :

$$n = |fr_j(a)|$$

r_i est donc applicable sur s_1 n fois :

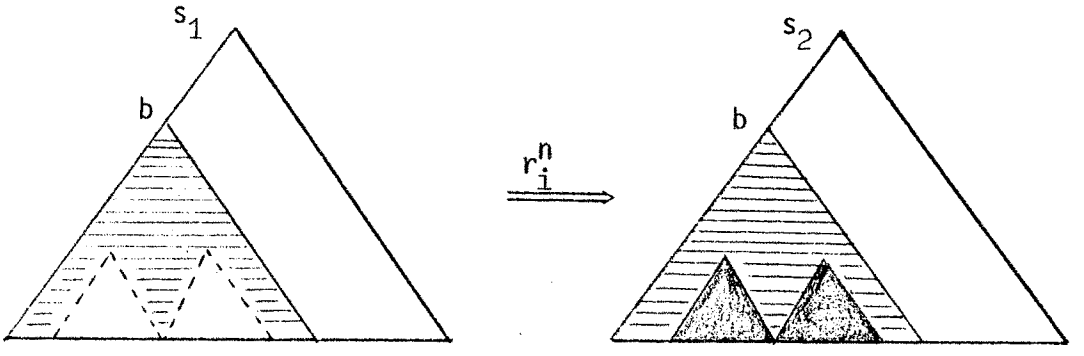


FIG. I.1.3.4.b.

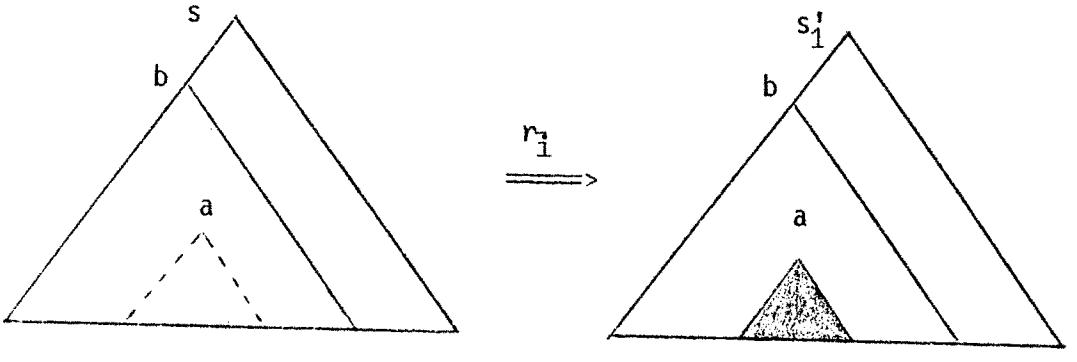
s_2 vérifie les énoncés suivants

(1) ($\forall$ m $\in$ SOMMET(s), m $\notin$ SOMMET(s/b))

alors m $\in$ SOMMET(s_2) et s/m = s_2/m

(2) s_2/b = REPLN ($fr_j(a)$, s_1/b, ψ_i)

ii) $s \xRightarrow{r_i} s_1'$



Alors $s'_1/a = \psi_i$, et la règle r_j n'est pas applicable. Mais, puisque G est fermé, il existe $r'_{j[a]} : \phi'_{j[a]} \rightarrow \psi'_{j[a]}$ tel que :

$$\phi'_{j[a]} = s'_1/b \quad \text{et} \quad \psi'_{j[a]} = \text{REPLN}(fr_j(a), \psi_j, \psi_i),$$

alors $s'_1 \overset{r'_{j[a]}}{\Longrightarrow} s_2$

Q.E.D.

Corollaire

Tout SGT fermé et précis possède la propriété de Church-Rosser

I.2. SYSTEMES DE TRANSFORMATIONS PARAMETREES

Dans le but de pouvoir décrire un nombre infini de règles de transformation par des moyens finis, nous allons étudier les systèmes de transformations paramétrées. Ceux-ci sont des STP complétés par un ensemble des "paramètres". Ces "paramètres" seront utilisés dans la description des règles de transformation (dénommées paramétrées) de façon à pouvoir représenter plusieurs règles (voire un nombre infini) d'un SGT avec une règle paramétrée.

I.2.1. Ramifications paramétrées

Une ramification paramétrée est une ramification au moyen de laquelle nous allons représenter un ensemble, peut être infini, de ramifications étiquetées. Chaque sommet d'une ramification paramétrée est appelé paramètre, et il peut représenter un ensemble d'étiquettes (sous-ensemble de V), ou un ensemble de ramifications (sous-ensemble de $\hat{V}$).

I.2.1.1. Paramètres

Un paramètre est un couple <nom,pred> , où nom est une chaîne de caractères servant à identifier le paramètre, et pred est un prédicat logique qui a comme domaine un ensemble d'éléments d'étiquettes V_E.

Par exemple si :

$\{$(CL SUBC), (GNR (MASC FEM)), (CL SUBP),(VARSEM (ANIME PHYSOBJ))$\} \subset V_E$

$<X_1$ (CL = SUBC et MASC $\in$ GNR)>

$<X_2$ (CL = (SUBC ou SUBP) ou ANIME $\notin$ VARSEM)>

sont des paramètres.

Si Z est un paramètre, on note Z_1 son nom , et Z_2 le prédicat associé. L'évaluation de Z_2 sur une étiquette a $\in$ V est noté $Z_2(a)$.

La notion habituelle de paramètre est un nom seulement. Cependant dans tout langage formel de programmation on a des paramètres d'un type déterminé, et on a des instructions spéciales pour déclarer le type de ces paramètres. Ici nous voulons étendre cette notion de type, et on associe à chaque paramètre une condition qui restreint l'ensemble d'étiquettes qu'il peut prendre comme valeur.

Bien que la contrainte du "type" d'un paramètre puisse être exprimable comme une condition du paramètre, nous avons préféré l'associer au nom du paramètre. Ainsi l'ensemble des noms de paramètres P est divisé en trois sous-ensembles disjoints : noms de paramètres représentant des

ramifications P_R, pseudo-arborescences P_S, ou étiquettes P_E.

$$P = P_R \cup P_S \cup P_E$$

Chacun de ces ensembles se divise en noms de paramètres obligatoires (P'_R , P'_S , P'_E), ne pouvant jamais représenter l'élément vide (Λ ou ϕ), et nom de paramètres optionnels (P''_R, P''_S, P''_E), qui peuvent représenter l'élément vide.

I.2.1.2. Ramification paramétrée

Une ramification paramétrée est un élément $p \in (P \times \mathcal{P}_{V_E})$ (où P est un ensemble de noms de paramètres, et $\mathcal{P}_{V_E}$ est un ensemble de prédicats sur V_E) tel que :

$$\forall\ Z \mid \exists\ x \in \text{SOMMET}(p) \text{ et } \Delta(x) = Z \text{ et } Z_1 \in (P_R \cup P_S)$$

$$\Longrightarrow Z \text{ est une feuille de } p.$$

C'est-à-dire tout paramètre ramification ou pseudo-arborescence est obligatoirement une feuille de p.

L'ensemble de ramifications paramétrées est noté ici $\hat{RP}$.

Un exemple de ramification paramétrée est donné à la figure I.2.1.4.a.

I.2.1.3. Equivalence des ramifications et valeur d'un paramètre

Au moyen des paramètres, toute ramification paramétrée x représente une famille de ramifications F_x. Cette famille est définie par la relation d'équivalence $\approx$ entre ramifications sur V et ramifications paramétrées.

En même temps qu'on définit l'équivalence on va définir la fonction VAL (valeur d'un paramètre). Etant données une ramification $v \in \hat{V}$ et une ramification paramétrée $y \in \hat{RP}$ équivalentes, cette fonction prend pour valeur pour chaque paramètre dans y l'élément de v qu'il est sensé représenter :

$$\begin{aligned} VAL_{[v \approx y]} \subset\ & [P'_R \times (\hat{V} - \Lambda) \\ & \cup\ P''_R \times \hat{V} \\ & \cup\ P'_S \times V^1 \\ & \cup\ P''_S \times (V^1 \cup \{\Lambda\}) \\ & \cup\ P'_E \times V \\ & \cup\ P''_E \times (V \cup \{\varepsilon\}) \quad] \end{aligned}$$

Rappel : V^1 est l'ensemble de pseudo-arborescences sur V, et ε représente la chaîne vide.

Equivalence de ramifications sur V et ramifications paramétrées.

Soit $a \times r + p \in \hat{V}$, $Z \times y + w \in \hat{RP}$ une ramification paramétrée, la relation d'équivalence $(\approx) \subset \hat{V} \times \hat{RP}$ est définie par les assertions suivantes :

(1) $\Lambda \approx \Lambda$

(2) $a \times r + p \not\approx \Lambda$

(3) $\Lambda \approx Z \times y + w \iff (Z \in (P''_S \cup P''_E)$ et $\Lambda \approx w)$

(4) $Z_1 \in (P_S \cup P_E)$

$$a \times r + p \approx Z \times y + w \iff$$

$$\begin{aligned} & [Z_2(a) \text{ et } (Z_1 \in P_S \text{ ou } r \approx y) \\ & \text{et } VAL(Z_1) := a \times r \text{ si } Z_1 \in P_S \\ & \qquad \text{ou } a \text{ si } Z_1 \in P_E] \\ & \text{ou} \\ & [Z_1 \in (P''_S \cup P''_E) \text{ et } a \times r + p \approx w] \end{aligned}$$

(5) $z_1 \in P_R$ (donc $y = \Lambda$)

$a \times r + p \simeq z + w \iff [z_2(a)$ et $p \simeq z + w$ et

$VAL(z_1) := VAL(z_1) + a \times r]$

ou $[(z_1 \in P''_R$ ou $VAL(z_1) \neq \Lambda)$

et $a \times r + p \simeq w]$

Si x est une ramification paramétrée, la famille de x, F_x est :

$F_x = \{r \mid r \in \hat{V}$ et $r \simeq x\}$

I.2.1.4. Un exemple

Pour la représentation graphique des ramifications paramétrées nous allons utiliser les conventions suivantes :

i) Les paramètres appartenant à P_R sont soulignés, et le trait qui les joint avec son père est un triangle pour mieux visualiser le fait qu'ils représentent des ramifications.

ii) Les noms des paramètres appartenant à P_E sont précédés du préfixe '&'.

iii) Les noms des paramètres appartenant à P_S sont précédés du préfixe '&&'.

iv) Les noms des paramètres optionnels ont le suffixe "'" et l'arc entre eux et leurs pères sera un trait en pointillé.

Avec ces conventions si :

$$\{<\&Y\ (a1\ et\ a2)>,\ <\&X2(b)>,\ <\underline{Z_1}(d1)>,\ <\&\&T'\ (c1\ ou\ c3)>,$$
$$<\&X3(e2\ et\ e3\ et\ \neg\ e4)>\} \subset (P \times \mathcal{P}_{V_E})$$

la ramification

$$x = \&Y \times (\&X2 \times (\underline{Z_1} + \&X3) + \&\&T')$$

(où on n'a écrit que les noms des paramètres pour ne pas rendre illisible la ramification) est dessiné comme le montre la figure :

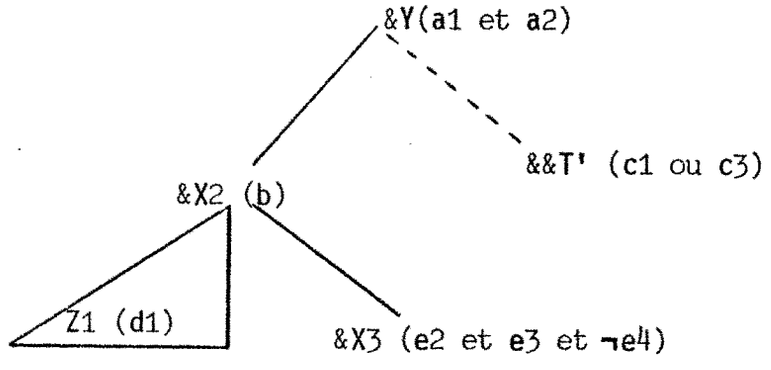


Figure I.2.1.4.a.

Si {<a a1 a2>, <b b1 b2>, <c c1>, <d d1 d2>, <e e2 e3>, <e d1>} $\subset$ alors la figure suivante représente une ramification s, équivalente à X.

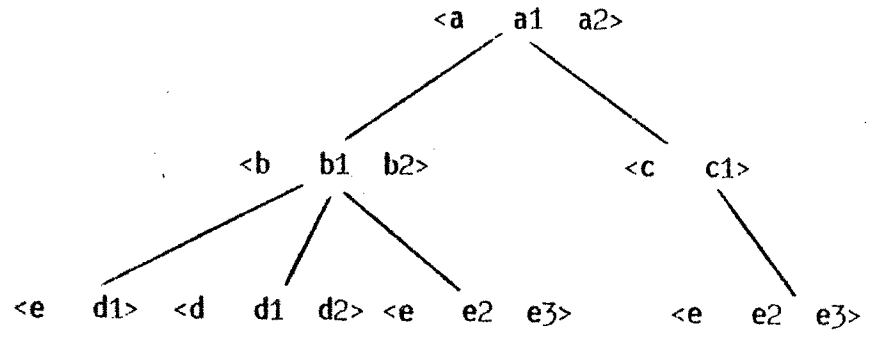


Figure I.2.1.4.b.

et la fonction VAL établie au moment de faire cette équivalence est :

$$VAL_{[s \approx X]}(\&Y) = \langle a \;\; a1 \;\; a2 \rangle$$

$$VAL_{[s \approx X]}(\&X2) = \langle b \;\; b1 \;\; b2 \rangle$$

$$VAL_{[s \approx X]}(\&\&T') = \langle c \;\; c1 \rangle \times \langle e \;\; e2 \;\; e3 \rangle$$

$$VAL_{[s \approx X]}(\underline{Z_1}) = \langle e \;\; d1 \rangle + \langle d \;\; d1 \;\; d2 \rangle$$

$$VAL_{[s \approx X]}(\&X3) = \langle e \;\; e2 \;\; e3 \rangle$$

I.2.2. STP

Un système de transformations paramétrées (STP) est un quintuplet $G = \langle V, F, \Longrightarrow P, \mathbb{R} \rangle$ où :

V est un ensemble d'étiquettes, ou vocabulaire ;

F est un ensemble de ramifications sur V, $F \subseteq \hat{V}$, appelé l'ensemble de ramifications de base ;

R est l'ensemble de règles de transformation paramétrées,

P est un ensemble de noms de paramètres, et $\Longrightarrow$ est la relation de transformation de ramifications étiquetées.

I.2.2.1. Règles de transformation paramétrées

Les transformations de ramifications étiquetées sont utilisées pour effectuer une ou plusieurs des opérations suivantes sur une ramification donnée : (1) modifier les relations entre les sommets, c'est-à-dire changer la structure de la ramification ; (2) ajouter des nouveaux sommets ; (3) éliminer des sommets ; (4) ajouter ou éliminer des éléments à certaines étiquettes.

Pour bien déterminer les structures qui seront transformées, une règle de transformation paramétrée est composée de : un schéma de transformation formé par deux ramifications paramétrées, un prédicat ou un ensemble de conditions, et un ensemble d'actions sur les étiquettes des sommets de la ramification transformée appelés effets secondaires. Les ramifications du schéma de transformation sont appelées aussi la partie gauche et la partie droite de la transformation. La partie gauche sert à déterminer une structure (dans une ramification, sur **V**, donnée) sur laquelle on pourrait appliquer la transformation. Les prédicats des paramètres de la partie gauche d'une règle expriment les conditions que doivent remplir chaque sommet d'une ramification étiquetée indépendamment des autres sommets de la structure. Dans certains cas, pour déterminer la ramification qui doit être transformée, il faut que des relations entre les étiquettes de plusieurs sommets de la ramification soient vérifiées (par exemple : les accords de variables). Ceci est le rôle du prédicat ou ensemble de conditions d'une règle de transformation paramétrée.

Une fois qu'on a déterminé la structure qui sera transformée tout nom d'un paramètre appartenant à la partie gauche de la règle aura une valeur. La ramification la plus petite, équivalente à la partie droite de la règle, construite avec ces valeurs pour les noms de paramètres est la ramification qui remplacera la ramification déterminée par la partie gauche et la transformation proprement dite sera ainsi effectuée. Notons que pour les noms de paramètres de la partie droite de la règle, n'appartenant pas à la partie gauche, ce sera la condition du paramètre qui détermine l'étiquette utilisée pour cette construction ; pour les autres paramètres, la condition exprimera une modification soit à l'étiquette qu'il a comme valeur (s'il appartient à P_E), soit la racine de l'arborescence qu'il a comme valeur (s'il appartient à P_S), ou soit à chaque racine de la ramification qu'il a comme valeur (s'il appartient à P_R).

Dans certains cas des modifications aux étiquettes doivent être exécutées sous des conditions qui portent sur les étiquettes des autres sommets. Afin de diminuer les nombres de règles, ces modifications conditionnelles, effets secondaires dans ce qui suit, sont le quatrième élément d'une règle de transformation paramétrée.

Définition :

Une règle de transformation paramétrée $t_i = \langle x_i, y_i, c_i, a_i \rangle$ est un 4-uplet où :

x_i et y_i sont des ramifications paramétrées

c_i est un prédicat qui a comme domaine l'ensemble SOMMET(x_i)

a_i est une expression dont le but est modifier les étiquettes associées aux sommets de y_i ; ce sont les effets secondaires.

tel que :

i) les noms de paramètres sont uniques dans x_i.

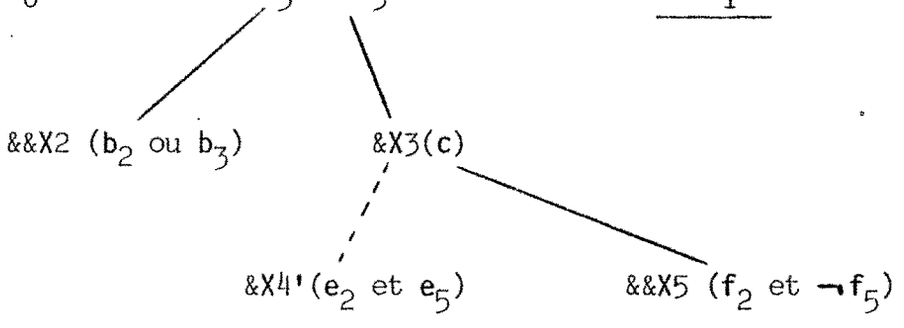
ii) c_i ne porte pas sur aucun paramètre optionnel dans x_i

Soit :

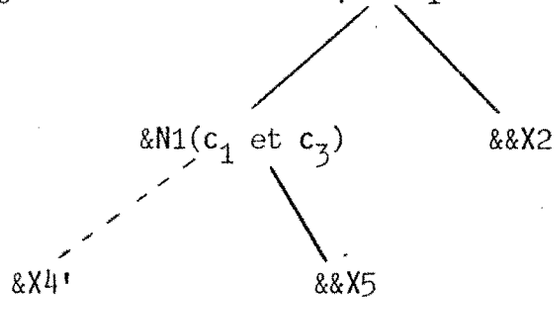
$\{a_i, b_i, c_i, d_i, e_i, f_i\} \subset V_E$, un ensemble d'étiquettes de V.

Un exemple de règle de transformation paramétrée $t_0 = \langle x_0, y_0, c_0, a_0 \rangle$ est :

x_0 : &X1 (a_3 et a_5) — Z1(d_1)



y_0 : &X3 (c_4 et f_1(A(&X1), A(&X2))) — Z1(a_7)



c_0 : $\mathcal{P}_1$(&&X2, &X3) ou $\mathcal{P}_2$(&&X2, &&X5)

a_0 : si $\mathcal{P}_3$(&X3) alors f_2(B(&&X2), B(&X3)) ;

où $\mathcal{P}_1$, $\mathcal{P}_2$ et $\mathcal{P}_3$ sont des prédicats, f_1 et f_2 sont des fonctions sur V_E^2 et A et B sont des fonctions sur P.

I.2.2.2. Application des transformations paramétrées

Une règle de transformation paramétrée $t_i = \langle x_i, y_i, c_i, a_i \rangle$ est applicable sur une ramification $r \in \hat{V}$ si et seulement si :

(1) Dans r il y a une sous-ramification complète x' tel que

$x' \simeq x_i$

(2) Le prédicat construit en remplaçant dans c_i tout paramètre Z de x_i par chaque étiquette de $rac(VAL_{[x' \simeq x_i]}(Z_1)$ est vérifié.

La sous-ramification x' est appelée domaine d'application de t_i ; et la plus petite sous-ramification de x' équivalente à x_i est appelée domaine restreint d'application de t_i. La ramification construite en remplaçant dans y_i tout paramètre Z par $VAL_{[x' \simeq x_i]}(Z_1)$ est notée y' et la construction de cette ramification est réalisée par la fonction CONST définie sur $\hat{V} \times \hat{RP}^2 \to \hat{V}$:

$CONST(x', x_i, \Lambda) = \Lambda$;

$CONST(x', x_i, Z \times u + v) = (\text{si } \exists Z' \in x_i \mid Z'_1 = Z_1 \text{ alors } VAL(Z_1) \cup Z_2$

sinon Z_2)

$\times CONST(t_i, u) + CONST(t_i, v)$;

et $y' = CONST(x', x_i, y_i)$

Une fois construite cette ramification il faut évaluer le terme a_i et exécuter les actions correspondantes. Ce processus d'évaluation de l'expression a_i sera appelé : EVAL: $\mathbb{R} \times \hat{V} \to \hat{V}$.

$y'' = EVAL(t_i, y')$.

LEMME : $y' \approx y_i$ et $\forall\, p \in P \mid \exists\, s \in SOMMET(x_i),\ t \in SOMMET(y_i)$

$$t_1 = s_1 = p \Longrightarrow VAL_{[x' \approx x_i]}(p) = VAL_{[y' \approx y_i]}(p).$$

Finalement, la relation ($\Longrightarrow$) pour un STP est définie comme suit :

$$(\Longrightarrow) = \{\langle p, q\rangle \mid p, q \in \hat{V} \tag{1}$$

$$\exists\, x' \in \hat{V} \text{ et } t_i = \langle x_i, y_i, c_i, a_i\rangle \in \mathbb{R}$$

tels que $x' \subset p$,

$x' \approx x_i$, et

a_i sur $rac(VAL_{[x' \approx x_i]})$ est vraie (2

$$q \in REPL(p, x', y'')$$

où $y'' = EVAL(t_i, CONST(x', x_i, y_i))$ (3)

}.

Si $\langle p, q\rangle \in (\Longrightarrow)$ on écrit $p \overset{t_i}{\Longrightarrow} q$, où $t_i \in \mathbb{R}$ est déterminé par (2).

I.3. STP CHURCH-ROSSER

Nous avons déjà vu l'importance de travailler avec des systèmes de transformations Church-Rosser. Ci-dessous nous allons indiquer les caractéristiques que doit avoir un système de transformations paramétrées pour posséder cette propriété. Tout d'abord on va introduire la notion de T-fermeture, plus forte que celle de fermeture pour les SGT.

I.3.1. STP T-fermés

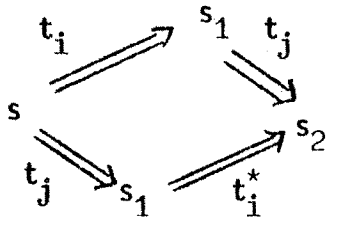
Un SGTP est T-fermé ssi

si sur une ramification s deux transformations T_i et T_j sont applicables telles que :

$$s \xrightarrow{t_i} s_1$$

$$s \xrightarrow{t_j} s'_1$$

alors



Ceci veut dire que dans un système T-fermé toute transformation reconnue comme applicable, reste toujours applicable (zéro ou plusieurs fois) sans que son moment d'application ait une influence sur le résultat de la transformation.

Le problème qui reste toujours posé est celui de savoir si un STP possède la propriété de T-fermeture. Le lemme suivant essaie d'éclaircir la question.

Lemme I.3.1.

Dans un système de transformations paramétrées $G = \langle V, F, \Longrightarrow, P, R \rangle$

$\forall\, t_i, t_j \in R$, $r \in \hat{V}$ et $\exists f \in F \mid f \Longrightarrow^* r$. Si t_i et t_j sont applicables sur r avec les domaines d'application d_i et d_j.

$(d_i \neq d_j)$ respectivement $(d_i \overset{t_i}{\Longrightarrow} d_i', d_j \overset{t_j}{\Longrightarrow} d_j')$ telles que :

(i) $d_i \not\subset d_j$ et $d_j \not\subset d_i$

ou (ii) $d_i \subset d_j$ et $\exists Z \in x_j \mid d_i \subset VAL_{[d_j \approx x_j]}(Z_1)$

ou (iii) $d_i \subset d_j$ et $\exists Z \in x_j, Z' \in x_i$

tels que $d_i = VAL_{[d_j \approx x_j]}(Z_1)$ (1)

et $rac(d_i) = rac(VAL_{[d_i \approx x_i]}(Z_1'))$ (2)

et $Z_2(rac(VAL_{[d_i' \approx y_i]}(rac(y_i))))$ (3)

et $Z_2'(rac(VAL_{[d_j' \approx y_j]}(Z_1)))$ (4)

alors G est T-fermé.

Cela veut dire que si dans un STP G on peut appliquer plusieurs transformations sur une ramification (par exemple voir figure I.3. a où d_1, d_2, d_3 et d_4 sont des domaines restreints d'application)

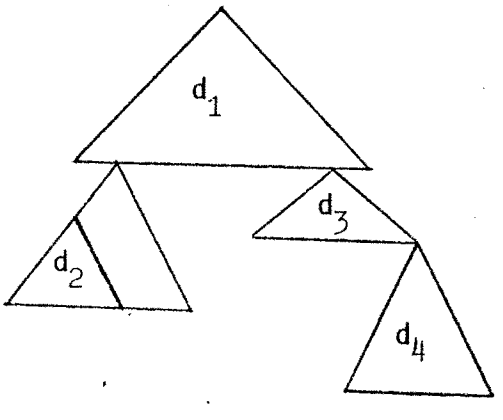


FIG. I.3.a.

de telle façon que les domaines restreints d'application soient disjoints (condition (i) : d_2 et d_3, ou condition (ii) d_1 et d_2), ou à la limite aient un sommet commun et les conditions (1) à (4) sont remplies alors G est fermé. Les conditions (1) et (2) servent à nommer Z et Z' le paramètre associé à ce sommet commun dans les domaines d'application contenant et contenu respectivement.

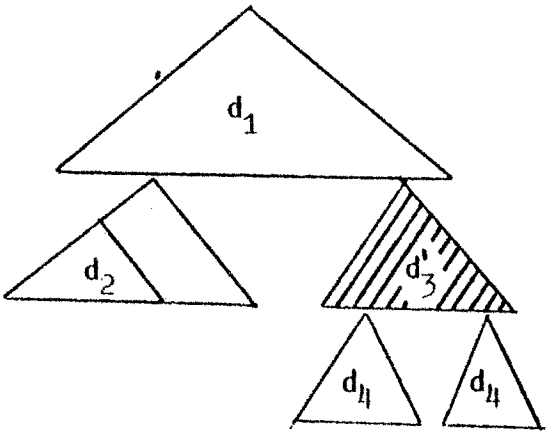


FIG. I.3.b.

La condition (3) spécifie que dans le cas où la règle plus petite est exécutée d'abord, (Figure I.3.b. d'_3 par rapport à d_1), la nouvelle racine satisfait le prédicat Z_2 et la règle grande est applicable. Si par contre la règle plus grande est exécutée avant (d'_3 par rapport à d_4) il faut que les modifications faites aux étiquettes racine de d_4 ne changent pas son caractère de domaine d'applications (c'est-à-dire le prédicat Z'_2 soit toujours vérifié).

Ce lemme peut être étendu aux cas où l'intersection des domaines restreints d'application de deux règles soient plusieurs sommets, ou même une sous-ramification.

Notons aussi que les règles t_i et t_j peuvent être deux applications différentes de la même règle. En conclusion, pour savoir si un STP est T-fermé, il faut vérifier que si deux règles risquent d'être applicables sur une ramification produite à partir de F, elles satisfont le lemme donné.

I.3.2. STP fermés

Il peut être convenable d'étendre l'intersection des domaines restreints d'application de plusieurs transformations sur une ramification, d'une sommet à une ramification.

Si l'application d'une transformation T_i modifie l'intersection de son domaine restreint avec celui d'une autre transformation T_j, de façon telle que T_j ne soit plus applicable le système ne sera pas T-fermé. Pour que le STP reste tout de même Church-Rosser il faut alors garantir que de toutes façons après l'application de certaines transformations on retrouve le même résultat. On définit alors les STP fermés analogues au SGT fermé (I.1.3.).

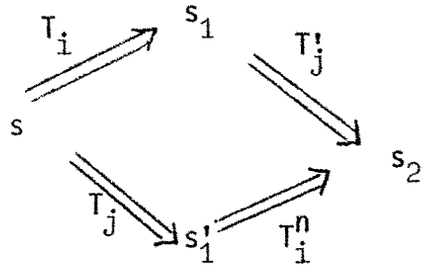
Définition :

Un STP est fermé ssi :

$$\forall\, T_i, T_j \in \mathbb{R}, \qquad s \xRightarrow{T_i} s_1$$

$$s \xRightarrow{T_j} s'_1$$

$\exists\, n, T'_j$ tels que



I.3.3. Quelques lemmes

Lemme I.3.3.1.

Tout SGT fermé peut être exprimé sous la forme d'un STP T-fermé.

Soit $G = \langle V, F, R, \Longrightarrow\rangle$ un SGT fermé. Nous allons construire le STP $G_p = \langle V, F, \Longrightarrow_p, P, R\rangle$.

Pour chaque règle $r_0 \in R$, tel que r_0 n'est pas une règle de fermeture : $r_0 : \phi_0 \rightarrow \psi_0$.

(1) $\forall a \in \text{SOMMET}(\phi_0)\ fr_0(a) = \phi$.

c'est-à-dire il n'y a aucune règle $r_i : \phi_i \rightarrow \psi_i$ tel que ϕ_i est une sous-ramification complète de ϕ_0. Alors on construit la règle $t_0 = \langle \phi_0^p, \psi_0^p, \varepsilon, \varepsilon\rangle$ où ϕ_0^p et ψ_0^p sont des ramifications paramétrées construites à partir de ϕ_0 et ψ_0. Pour la construction de ϕ_0^p on remplace chaque sommet a de ϕ_0 par un paramètre dont le prédicat est la conjonction des éléments d'étiquettes de a. Par exemple si a est l'étiquette $(a_1\ a_2\ a_3)$, il est remplacé dans ϕ_0^p par $(\&X1\ (a_1 \text{ et } a_2 \text{ et } a_3))$, où $\&X1 \in P$. Pour la construction de ϕ_0^p tout sommet de ϕ_0 contenu dans ϕ_0 est remplacé par un paramètre dont le nom est le même du paramètre correspondant dans ϕ_0^p, et le prédicat est toujours vrai. Pour les sommets non contenus dans ϕ_0 on suit le procédé utilisé pour les sommets de ϕ_0. Ainsi :

$$\forall s \in \hat{V} \quad s \xRightarrow{r_0} s' \quad \text{ssi} \quad s \xRightarrow[p]{t_0} s'$$

(2) $\exists a \in \text{SOMMET}(\phi_0) \mid fr_0(a) \neq \phi$

c'est-à-dire il y a au moins une règle $r_1 : \phi_1 \rightarrow \psi_1$ tel que ϕ_1 est une sous-ramification complète de ϕ_0 (Fig. I.1.3.a.). Alors on construit la règle $t_0 = \langle \phi_0^p, \psi_0^p, \varepsilon, \varepsilon\rangle$. Pour la construction de ϕ_0^p et ψ_0^p on suit le procédé

décrit avant ; ceci pour les sommets de ϕ_0 et ψ_0 qui n'appartiennent pas à ϕ_0/a ni ψ_0/m ($\forall$ m $\in$ $fr_0(a)$). S'il existe un prédicat p $\in$ (V × {'vrai' , 'faux'}) tel que p(a) soit 'vrai' seulement si a est la racine de ϕ_1 ou ψ_1, alors pour la construction de ϕ_0^p il faut remplacer a dans ϕ_0 par un paramètre dont le prédicat est p, et le nom appartient à P_S(ou P_R si ϕ_1 n'est pas une arborescence). Dans ϕ_0^p toutes les sous-ramifications de a sont éliminées. Pour la construction de ψ_0^p on remplace tous les sommets m ($\forall$ m $\in$ $fr_0(a)$) par un paramètre dont le nom est le même du paramètre correspondant à a et toutes les sous-ramifications de m sont éliminées.

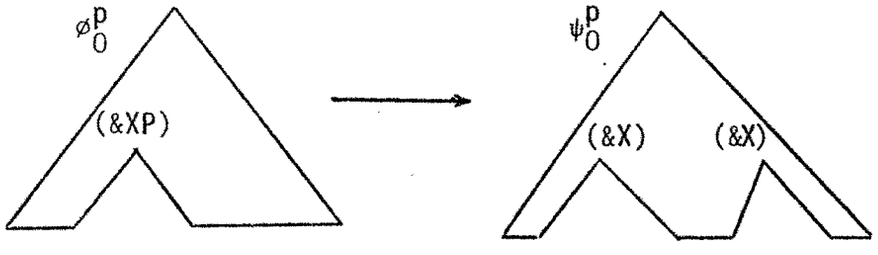
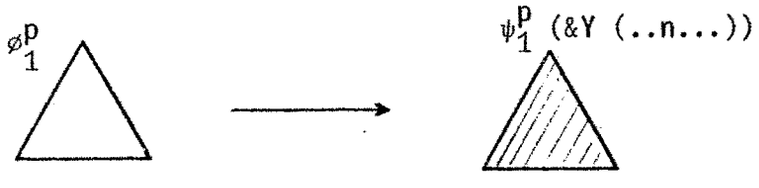


FIG. I.3.3.1.a.

Si le prédicat p mentionné n'existe pas alors dans t_1 =<ϕ_1^p, ψ_1^p, ε, ε> . on ajoute à la racine de ψ_1^p un nouvel élément d'étiquette n, qui sera ajouté à la racine de l'arborescence transformée par t_i. Pour la construction de ϕ_0^p et ψ_0^p, on peut maintenant utiliser le procédé décrit précédemment où le prédicat p est l'existence de l'élément d'étiquette n.



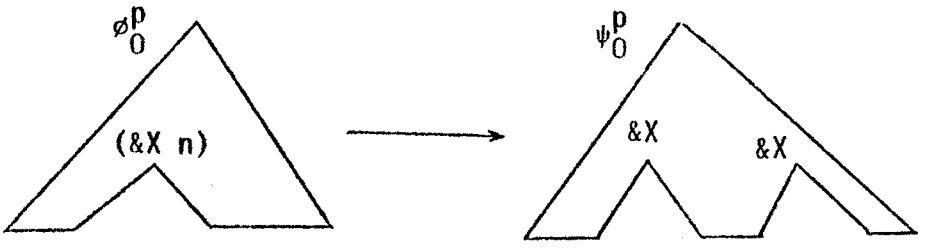


FIG. I.3.3.1.b.

Ainsi le prédicat p = existence de n est vérifié uniquement par la racine de ψ_1^p après l'application de t_1, donc la règle t_0 ne sera appliquée qu'après la règle t_1.

Il faut noter que cette démarche introduit un ordre d'application des règles paramétrées ainsi construites, et que $\forall\ r_0 \in R$, si dans la dérivation d'une ramification vers sa forme normale on appliquait r_0, dans le STP construit on appliquera toujours t_0 aussi, et le résultat est le même. Donc :

$$s \Longrightarrow^* s' \quad \text{et } s' \text{ ets la forme normale de } s_1$$

$$\text{alors : } \quad s \Longrightarrow_p^* s'$$

Lemme I.3.3.2.

Pour tout STP fermé existe un STP T-fermé équivalent, et la construction du système T-fermé est analogue à celle du lemme précédent.

I.4. LA PUISSANCE DESCRIPTIVE DES STP T-FERMES

Une des caractéristiques importantes de tout système de transformations est sa puissance.

Pour illustrer cet aspect des STP T-fermés nous allons nous mettre dans le cadre d'une application précise : la génération des langages formels. Dans cette application on engendre toutes les chaînes qui composent un langage formel à partir d'une grammaire appelée grammaire de base. Cette grammaire de base fournit un ensemble d'arborescences de base sur lesquelles on appliquera un SGT, ou un STP dans ce cas précis. L'ensemble des mots des racines de certaines arborescences résultantes formera le langage qu'on veut engendrer. Le théorème et sa preuve sont analogues au théorème de Salomaa (Salomaa A, 1971) sur les grammaires transformationnelles de Ginsburg et Partee (1969).

THEOREME

Etant donné un vocabulaire I, il y a un ensemble R de transformations paramétrées tel que tout langage L_0 récursivement énumérable et ne contenant pas le mot vide soit généré à partir d'une grammaire régulière et le STP T-fermé : $G = \langle I, F, \Longrightarrow, P, R \rangle$.

La preuve du théorème est fondée sur le fait que L_0 peut être exprimé en termes de langages réguliers, langages Dyck, intersection et homomorphismes. L'ensemble F d'arborescences de base, est déterminé par la grammaire régulière ; l'appartenance aux langages de Dyck ainsi que les opérations d'intersection et les homomorphismes seront vérifiés (ou exécutés) à l'aide de règles de transformation.

Un langage de Dyck est un langage hors-contexte engendré par une grammaire composée de : {S} l'ensemble des symboles non terminaux : $\{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_k\}$ ensemble de symboles terminaux, et les productions $S \rightarrow SS$, $S \rightarrow \epsilon$, et $S \rightarrow a_i S b_i$ pour $1 \leq i \leq k$. Un langage de Dyck peut être considéré comme un ensemble de chaînes composées de k types de parenthèses.

NOTE : On fait la convention que chaque fois qu'on introduit un vocabulaire, il est disjoint des vocabulaires existants.

Soit I un vocabulaire de r lettres, et $r \geq 1$. Gingsburg, Greibach et Harrison [1967] ont démontré que pour tout langage récursivement énumérable L_0 sur I il y a un homomorphisme h et deux langages hors-contexte L_1, et L_2 (sur un vocabulaire I_1 de r + 2 lettres) tels que :

$$L_0 = h(L_1 \cap L_2) \qquad (1)$$

Chomsky [1962], et plus tard Stanley [1965] et Gingsburg [1966] ont prouvé que tout langage hors-contexte peut être exprimé comme un homomorphisme de l'intersection d'un langage de Dyck et d'un langage régulier. Formellement il y a un langage de Dyck D(sur un vocabulaire I_2 de 2r + 8 symboles) et un homomorphisme $h_1 : I_2^* \to I_1^*$ tel que pour tout langage hors-contexte L_i (sur I_1), il y a un langage régulier K_i (sur I_2) tel que :

$$L_i = h_1(D \cap K_i) \qquad (2)$$

En utilisant cette expression dans la formule (1).

$$L_0 = h(h_1(D \cap K_1) \cap h_1(D \cap K_2)) \qquad (3)$$

Il faut noter que le langage de Dyck D,et les homomorphismes h_1 et h sont déterminés par le vocabulaire I, indépendamment de L_0. En outre, si L_0 ne contient pas le mot vide les langages K_1 et K_2 non plus.

Le STP G = <V, F, ⟹ , P, R> qui engendre L_0 à partir d'une grammaire régulière G_R est défini comme suit.

Soit G_R la grammaire qui produit le langage $K_1 . a_0 . K_2$ où a_0 est un symbole additionnel et le point représente l'opération de concaténation des chaînes. G_R est régulier puisque la concaténation des langages réguliers est un langage régulier. Les arbres produits par G_R seront du type :

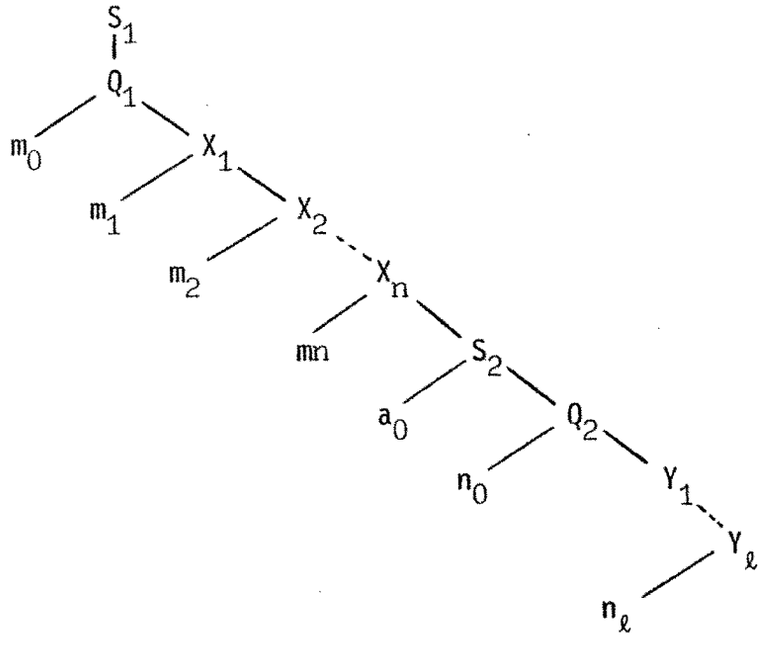


Figure 4.1.a.

où $\{m_0, \ldots, m_n, n_0, \ldots, n_\ell\} \subset I_2$; les X_i et Y_i sont des éléments de N_1 et N_2, les ensembles de symboles non terminaux des grammaires K_1 et K_2 (respectivement) ; Q_i est le symbole initial de K_i et, S_1 et S_2 des symboles auxiliaires de G_R. S_1 est le symbole initial de G_R.

$$V = \{(x_i\ y_i) \mid X_i \in (I \cup I_1 \cup I_2 \cup N_1 \cup N_2 \cup \{a_0, S_1, Q_1, S_2, Q_2\})$$

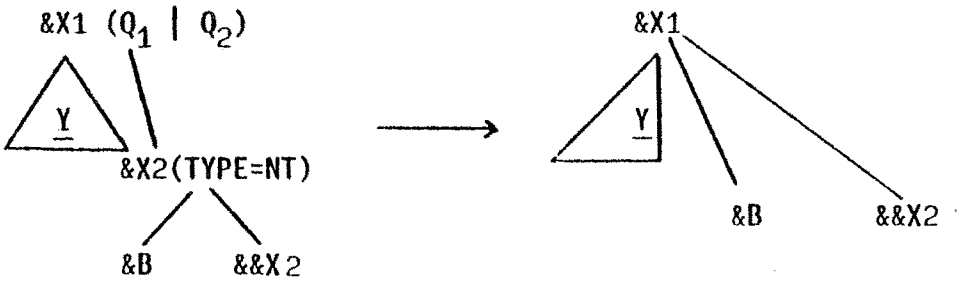
et y_i est une expression TYPE = a_i, $a_i \in \{NT, A_1, A_2, A_3, A_4, A_5\}$

Initialement la valeur NT sera associée aux éléments de N_1 et N_2 (symboles non-terminaux). Les indicateurs A_1 à A_4 seront assignés au fur et à mesure des transformations :

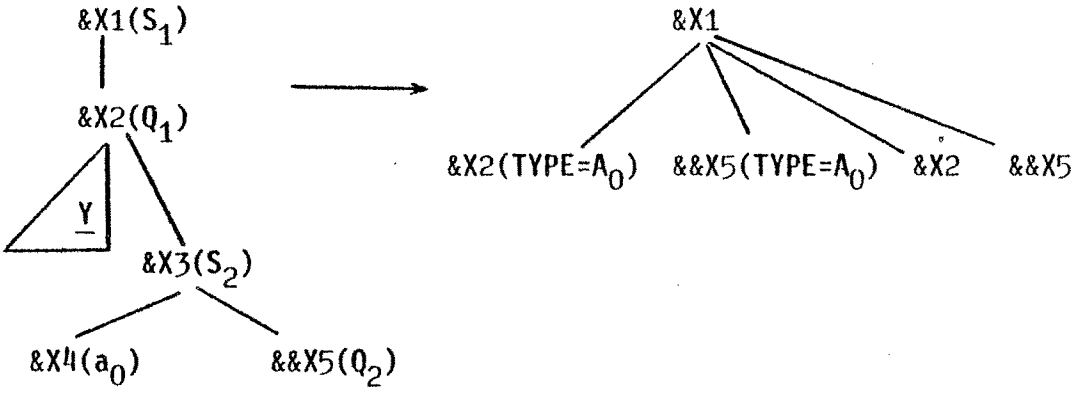
$$F = \{x \mid x \text{ est un arbre produit par } G_R\}$$

L'ensemble des règles paramétrées R est le suivant :

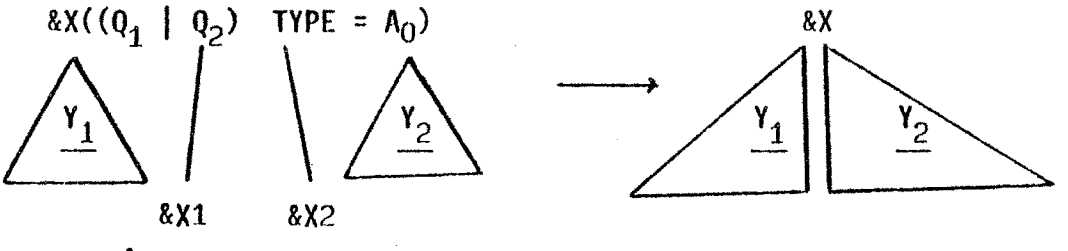
T1 :



T2 :



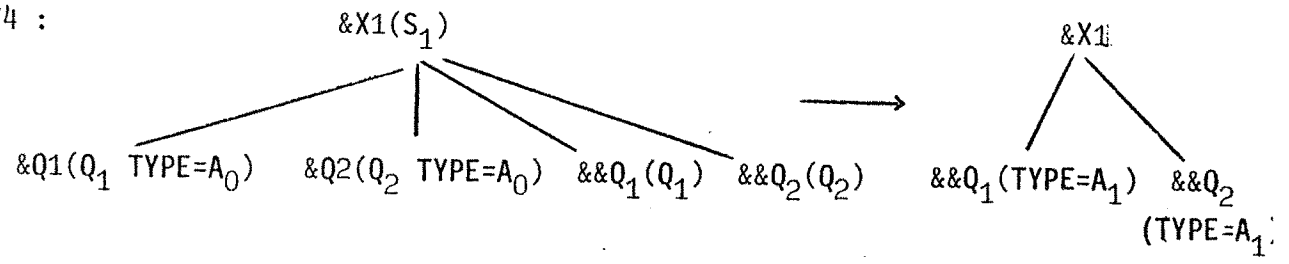
T3 :



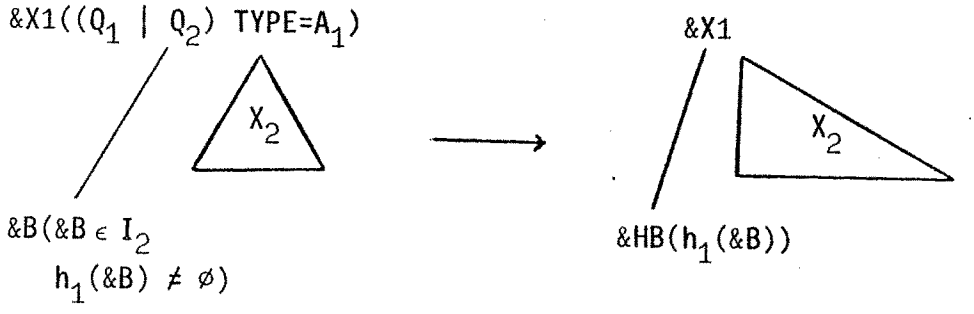
C_3 : &X2 = CORR(&X1)

CORR est une fonction de I_2 sur I_2 et exprime le fait que deux symboles sont correspondants au sens de la définition du langage de Dyck D. Par exemple CORR ("[") = "]".

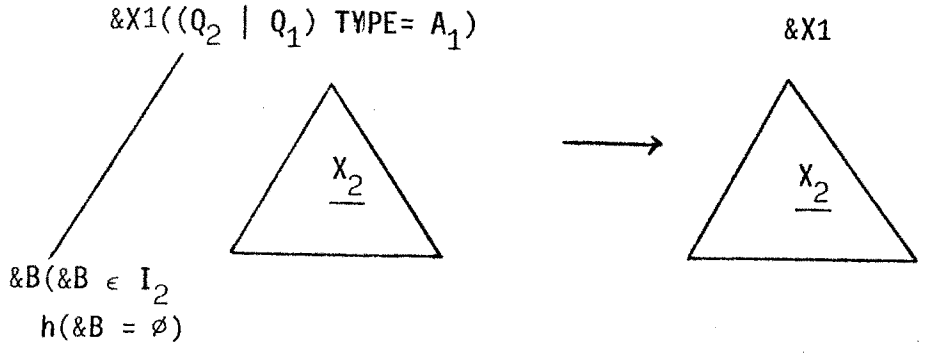
T4 :



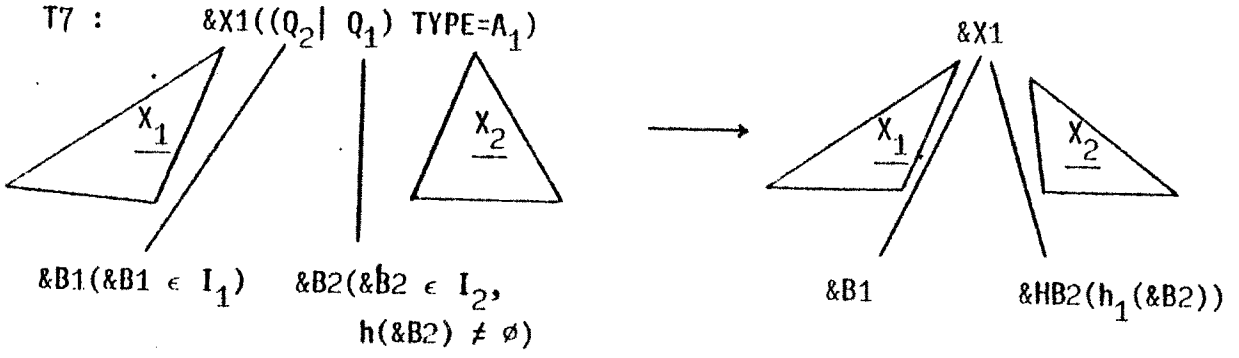
T5 :



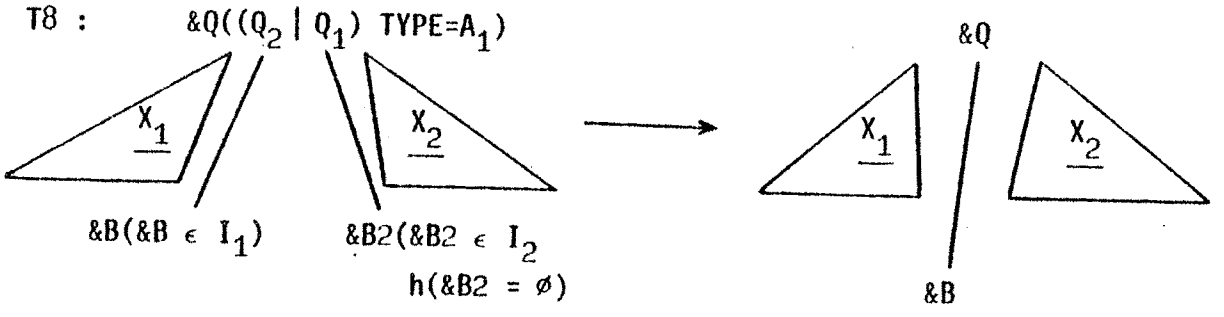
T6 :



T7 :

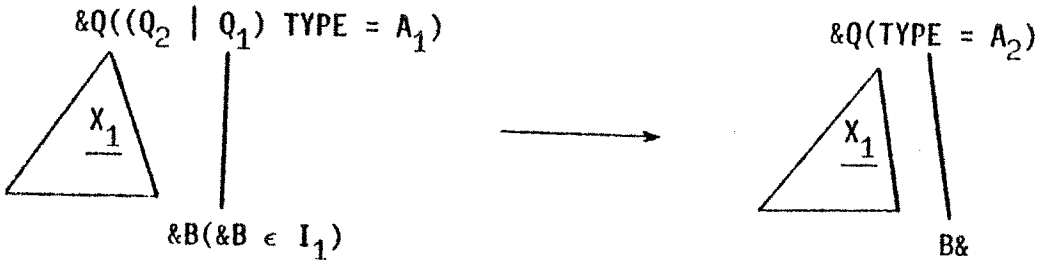


T8 :

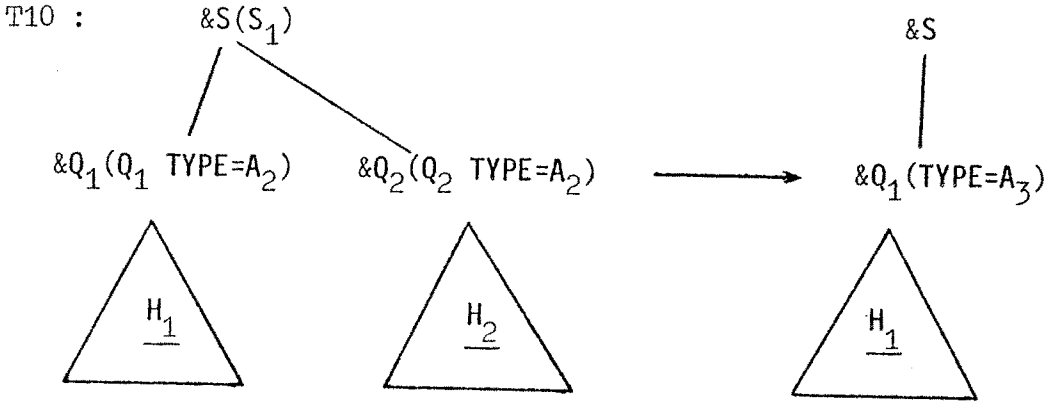


Dans T5, T6, T7, et T8 &B par sa position représente un symbole de I_2 ; h_1(&B) est l'application de l'homomorphisme h_1 sur ce symbole.

T9 :

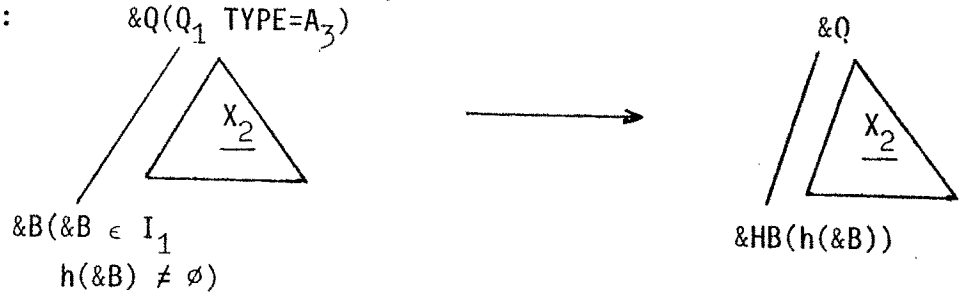


T10 :

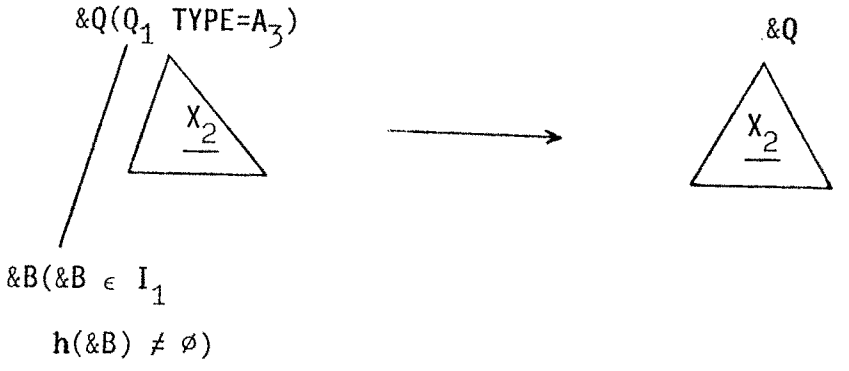


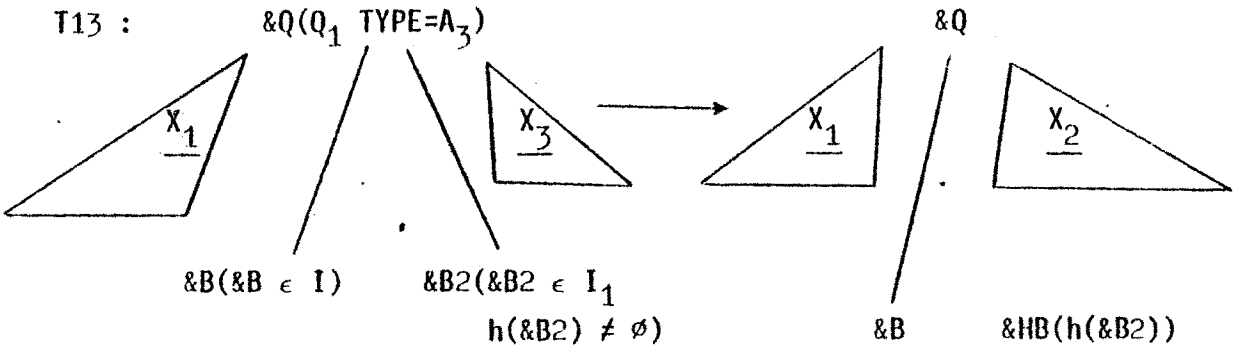
C10 = H1 = H2

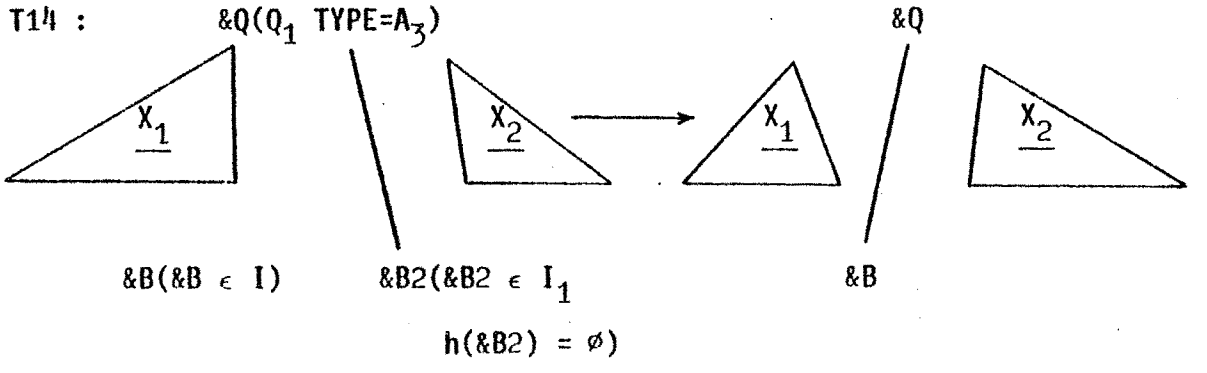
T11 :

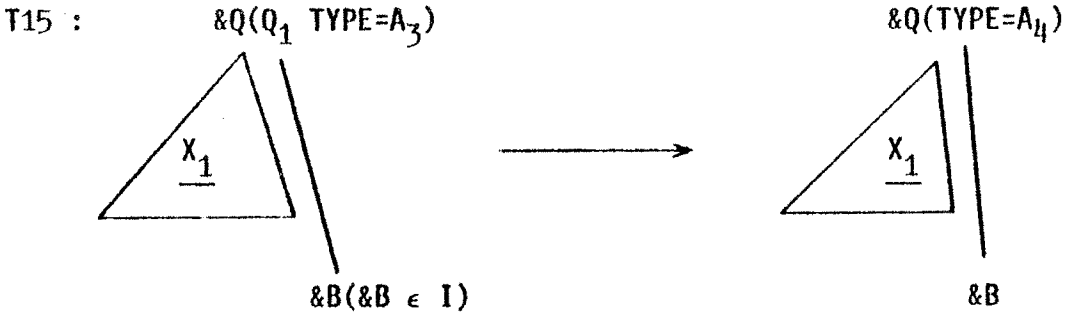


T12 :









L'ensemble P de paramètres est évidemment formé pour tous les paramètres utilisés dans ces quinze règles. La propriété de T-fermeture de R est vérifiable par le fait que pour tout couple de règle, soit elles ne peuvent pas être appliquées sur la même arborescence, soit elles ont au maximum un noeud en commun.

L'application du STP G ainsi défini sur une arborescence du type de la figure 4.1.a. se déroule de la façon suivante : les règles T1 et T2 font disparaître tous les symboles non-terminaux de G_R et dupliquent le mot de feuilles donné, c'est-à-dire si $k_1\ a_0\ k_2$ était le mot de feuilles de l'arborescence d'entrée, après l'application de T1 et T2 le mot de feuilles sera $k_1\ k_2\ k_1\ k_2$ seulement après avoir appliqué la règle T2, T3 sera applicable, et si k_1 et k_2 appartiennent à D on pourra appliquer T4 (notez que $\&Q_1$ est un paramètre de type feuille).

T5 à T8 appliquent l'homomorphisme h_1 à $k_1\ k_2$. T9 indique la fin de l'application de h_1 et T10 vérifie que $h_1(k_1)$ soit égale à $h_1(k_2)$ et élimine une des deux copies. T11 à T14 appliquent l'homomorphisme h. T15 indique la fin de cette application et donne à Q_1 le type A_4. Si le résultat de l'application de G à une arborescence de F donne une arborescence dont la racine est l'étiquette (Q_1 TYPE = A_4) alors le mot de feuilles de cet arborescence est un mot de L_0.

Q.E.D.

Il faut remarquer dans la construction du système G précédent l'utilisation de la variable TYPE et ses valeurs A_1 à A_4. On peut résumer en disant que malgré le fait que les règles sont appliquées sans un ordre défini à l'avance, le constructeur du STP peut imposer un ordre d'application en utilisant les variables qui font partie des étiquettes des noeuds.

CHAPITRE II

II.1. LES RESEAUX DE TRANSITIONS 59
II.1.1. Description générale 59
II.1.2. Réseaux de transitions pour les ramifications paramétrées. 61

II.2. LES AUTOMATES D'ETATS FINIS ADAPTES AUX RAMIFICATIONS PARAMETREES 63
II.2.1. Automate d'états finis adapté aux ramifications (AR) 63
II.2.1.1. Définitions 64
II.2.1.2. Exemple de AR 65
II.2.1.3. Fonction de transition généralisée 66
II.2.1.4. Bilangage reconnu par un AR 66
II.2.1.5. Proposition 67
II.2.1.6. AR Non-déterministe 69
II.2.1.7. Lemmes 70
II.2.2. AR paramétrées 72
II.2.2.1. Définitions 73
II.2.2.2. Lemme 74
II.2.3. Composition d'ARP. 74
II.2.3.1. Définitions 75
II.2.3.2. Exemple 79
II.2.3.3. Construction d'un ARPC à partir d'un ensemble de ramifications paramétrées. 82

II.3. STP et ARPC 90
II.3.1. Systèmes de transformations non Church-Rosser 90
II.3.2. STP Church-Rosser 94
II.3.3. STP T-fermés. 98

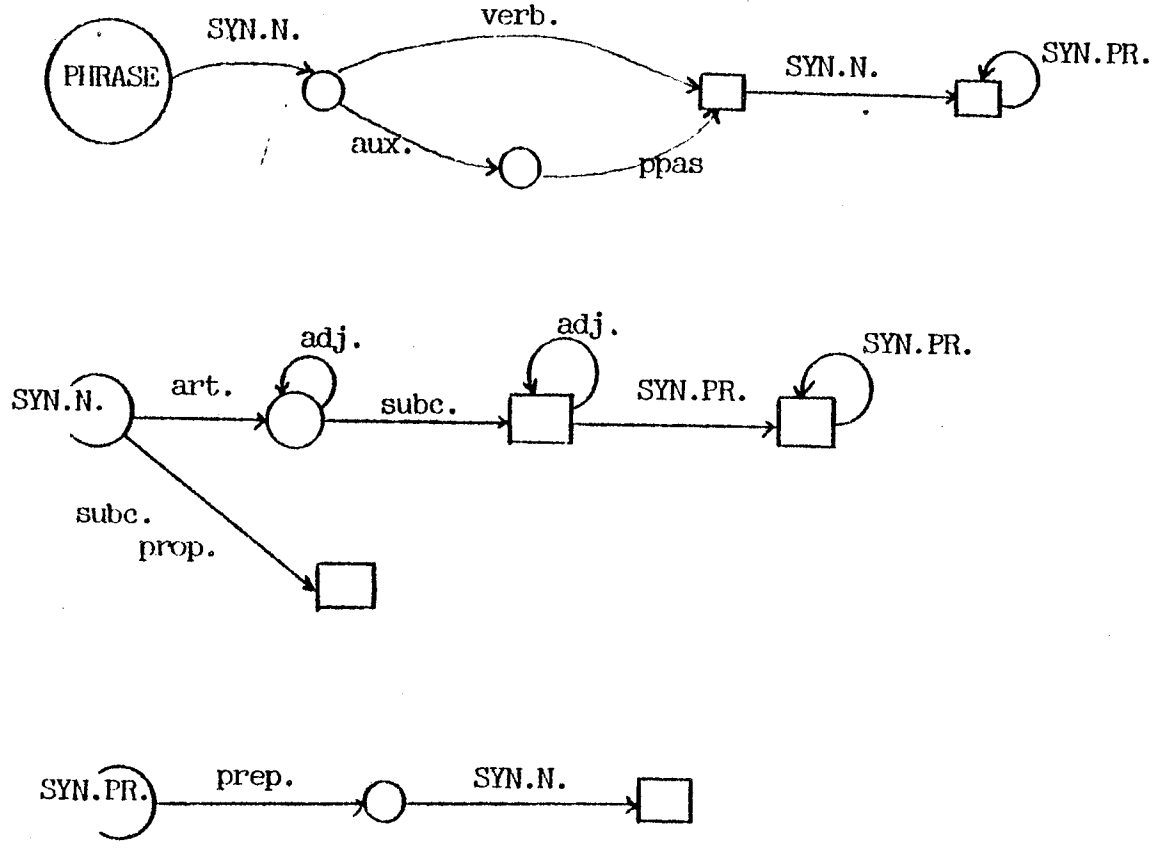
Un processus transformationnel peut être décomposé en deux pas essentiellement : d'abord une phase de reconnaissance dans la ramification donnée avec le but de savoir quelles sont les règles de transformation applicables, et postérieurement la phase de transformation proprement dite. Le but de ce chapitre est l'exposition d'un algorithme grâce auquel, en un seul parcours de la ramification donnée, on réussit à déterminer toutes les règles immédiatement applicables. Cet algorithme a ses origines dans la notion de réseaux de transitions [Conway 63, Bobrow and Fraser 69, Woods 70], aussi ce chapitre débute-t-il avec un court rappel de cette notion. Les réseaux de transitions ont été formalisés comme des automates à pile [Lomet, 1973], ou comme des transducteurs composés [Chauché, 1974]. Les modifications que nous avons introduites ne nous permettent pas d'utiliser ces descriptions et en conséquence nous avons défini les automates d'états finis adaptés aux ramifications paramétrées (ARP) et une règle de composition de ces automates, basée principalement sur la notion de parallèlisme.

Finalement, on verra l'application de ces automates dans le cadre de systèmes de transformations paramétrées, et spécialement l'influence du type du STP utilisé sur le processus général.

II.1. LES RESEAUX DE TRANSITIONS

II.1.1. Description générale

Un réseau de transition est un ensemble de diagrammes d'états finis dans lesquels les noms des arcs peuvent être soit des symboles terminaux, soit des noms d'états initiaux d'un diagramme du réseau. Ils sont utilisés principalement pour l'analyse syntaxique des langages hors-contexte. Par exemple, le réseau suivant peut être utilisé pour l'analyse des phrases simples en français :



où SYN.N. et SYN.P. signifient respectivement "syntagme nominal" et "syntagme prépositionnel".

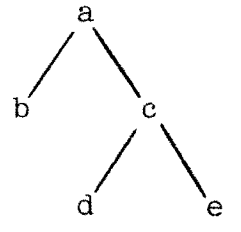
Woods (1970) a ajouté à ces réseaux des registres et des tests et des actions sur ces registres. Les tests et les actions sont associés aux transitions et permettent ainsi d'ajouter des restrictions à la grammaire (par exemple : limiter le nombre d'adjectifs qui modifient un nom, ou vérifier certains accords entre les variables morphologiques de deux mots), et de construire une structure syntaxique (ou une autre forme quelconque) associé à la phrase, au fur et à mesure de l'analyse. Les réseaux de transitions ainsi modifiés sont appelés "Augmented Transition Networks" ou ATN.

L'analyse effectuée par les ATN de Woods (1970) est une analyse descendante, aussi sa principale restriction est de ne pas pouvoir manipuler des grammaires avec récursivité à gauche. Tout de même Lomet (1973) a démontré qu'ils peuvent analyser tous les langages hors-contexte déterministes.

II.1.2. Réseaux de Transitions pour les Ramifications Paramétrées

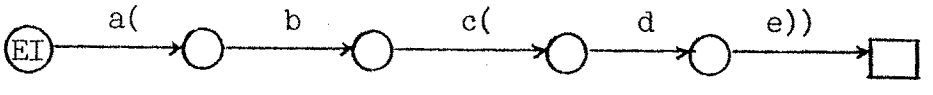
En nous fondant sur la notion de réseaux de transitions, nous avons développé un automate pour la reconnaissance de sous-ramifications dans une ramification donnée. Intuitivement, si on veut reconnaître la ramification :

r =

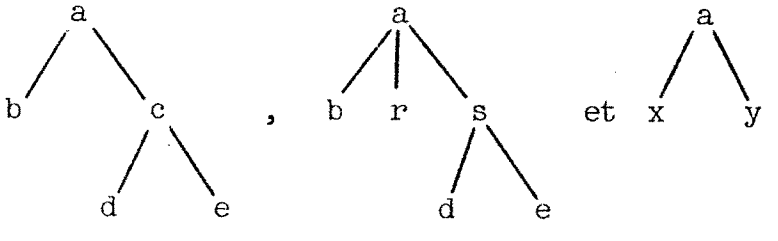


notre démarche sera d'associer un diagramme de transitions au parcours en préordre de la ramification ; par exemple pour la ramification r
PREORDRE(r) = a(b c(d e))

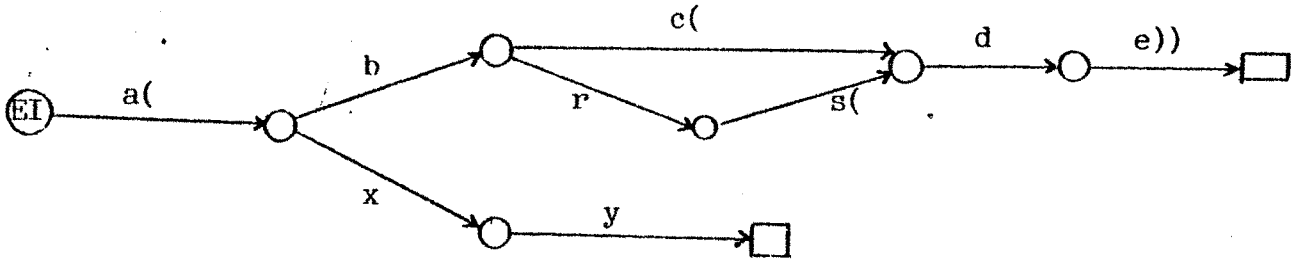
et le diagramme associé est :



Si nous voulons reconnaître les ramifications :



le réseau formé par l'ensemble des diagrammes est :



Au moyen du réseau on obtient, en fait, une mise en facteur des éléments communs de ramifications.

Si on utilise des ramifications paramétrées il y aura des paramètres sur les arcs correspondants aux transitions. Un paramètre (§ I.2.1.1.) est un couple <nom, pred> . La partie nom sera considérée comme le nom d'un registre qui aura comme valeur une ramification, une arborescence ou un noeud selon que le paramètre appartient respectivement aux ensembles P_R, P_S ou P_N.

Par rapport aux réseaux de transitions augmentés de Woods on remarque les principales différences :

i) les transitions sont occasionnées par des symboles terminaux de la grammaire (paramètres), ce qui entraîne la non-existence d'appels d'un diagramme à un autre.

ii) En conséquence, le réseau comporte un seul état initial.

iii) l'algorithme d'analyse se développe maintenant en ascendant, c'est-à-dire à partir de la chaîne d'entrée.

II.2. LES AUTOMATES D'ETATS FINIS ADAPTES AUX RAMIFICATIONS PARAMETREES

Pour bien comprendre le fonctionnement et l'utilisation que nous faisons des réseaux de transitions introduits au paragraphe précédent, nous allons les formaliser en termes d'automates. Cette formalisation procède par étapes successives à partir des automates destinés à reconnaître une ramification sur V, jusqu'à la composition d'automates pour les ramifications paramétrées.

II.2.1. Automate d'états finis adapté aux ramifications

Un automate déterministe d'états finis adapté aux ramifications (AR) est un 5-uplet $M = \langle Q, V, q_0, F, \delta \rangle$ où :

Q est un ensemble fini d'états ;
V est un vocabulaire ;
$q_0 \in Q$ est l'état initial ;
$F \subseteq Q$ est l'ensemble des états finaux ;
δ une application de $Q \times V$ sur $Q \times \mathbb{N}$, où $\mathbb{N}$ est l'ensemble d'entiers ; δ est appelée la fonction de transition de l'automate.

Un automate d'états finis possède un état de contrôle interne. Le parcours d'une chaîne donnée, appartenant à V^* autorise des transitions de cet état. Dans un AR, on a ajouté un numéro à chaque état. Ce numéro est interprété comme le niveau de l'arborescence dans lequel doit se trouver un symbole de V pour réaliser une transition. Pour déterminer le comportement de l'automate pendant l'analyse d'une chaîne nous devons connaître :

i) l'état courant interne ($q \in Q$),

ii) le niveau auquel cet état attend un symbole de V ($m \in \mathbb{N}$);

iii) Une chaîne de caractères à analyser, dont on considère le caractère le plus à gauche comme celui en cours d'analyse ($w \in (V \cup \{"(", ")"\})^*$), et

iv) le niveau de l'arborescence auquel se trouve ce caractère ($n \in \mathbb{N}$).

Ces éléments forment la configuration de l'automate.

II.2.1.1. Définitions

Configurations

Si $M = \langle Q, V, \delta, q_0, F\rangle$ est un AR, alors un quadruplet $\langle q, m, n, w\rangle$ de $Q \times \mathbb{N}^2 \times (V \cup \{"(", ")"\})^*$ est une configuration de M. Une configuration de la forme $\langle q_0, 0, 0, w\rangle$ est appelée configuration initiale, et une de la forme $\langle q_f, m, 0, \varepsilon\rangle$ pour n'importe quels $m \in \mathbb{N}$, $q_f \in F$, une configuration finale.

Opérations sur les chaînes

On appelle **prem(a)** la fonction qui rend le premier élément d'une chaîne a, et **reste(a)** la fonction telle que :

$$a = \text{prem}(a).\ \text{reste}(a)$$

où le . est l'opération de concaténation des chaînes.

La longueur d'une chaîne, c'est-à-dire le nombre d'éléments qui la composent est donné par la fonction ℓ_g.

Configuration successeur

$\langle q', m', n', a'\rangle$ est une configuration successeur de $\langle q, m, n, a\rangle$ ssi :

si prem(a) = "(" alors $n' = n + 1$; $q' = q$; $m' = m$; $a' = \text{reste}(a)$.

si prem(a) = ")" alors $n' = n - 1$; $q' = q$; $m' = m$; $a' = \text{reste}(a)$

si $\text{prem}(a) \in V$ alors si $m = n$

$n' = n$; $a' = \text{reste}(a)$; $q' = q_1$ et $m' = m_1 + 1$

où $\delta(q,a) = \langle q_1, m_1\rangle$;

on écrit $\langle q, m, n, a\rangle \vdash_M \langle q', m', n', a'\rangle$; $\vdash_M^+$ et $\vdash_M^*$ sont les fermetures transitive et transitive-réflexive (respectivement) de $\vdash_M$. S'il n'y a pas d'ambiguité sur la machine M, on n'écrit pas l'indice M($\vdash$ tout simplement).

II.2.1.2. Exemple de AR

Soit l'AR M = <Q, V, q_0, F, δ> avec

Q = {q_0, q_1, q_2, q_3, q_4}

V = {S, a, b}

F = {q_3}

$\delta(q_0, S) = <q_1, 1>$

$\delta(q_1, a) = <q_2, 0>$

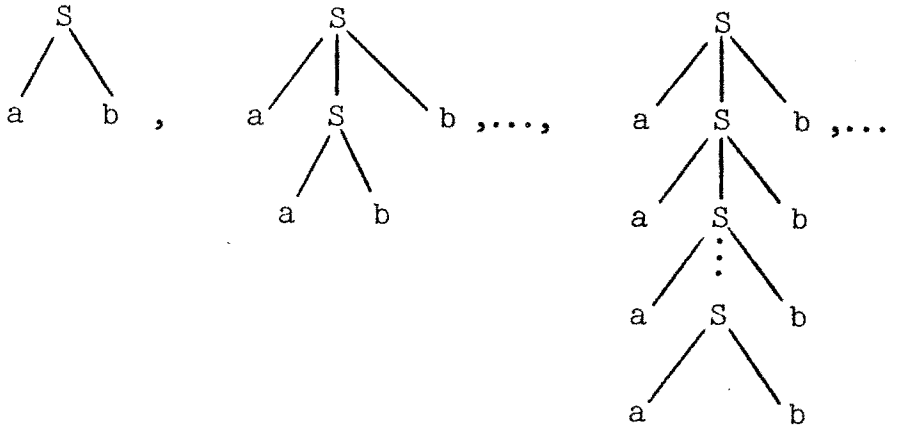
$\delta(q_2, S) = <q_1, 1>$

$\delta(q_2, b) = <q_3, -1>$

$\delta(q_3, b) = <q_3, -1>$

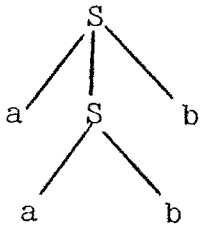
δ prend la valeur q_4 partout ailleurs.

L'automate reconnaît le bilangage :



Reconnaissance de :

r =



PREORDRE(r) = S(a S(a b)b)

$\langle q_0,0,0,S(a\ S(a\ b)b)\rangle \vdash \langle q_1,1,0,\ (a\ S(a\ b)\ b)\rangle$

$\vdash \langle q_1,1,1,\ a\ S(a\ b)\ b)\rangle$

$\vdash \langle q_2,1,1,\ S(a\ b)\ b)\rangle$

$\vdash \langle q_1,2,1,\ (a\ b)\ b)\rangle$

$\vdash \langle q_1,2,2,\ a\ b)\ b)\rangle$

$\vdash \langle q_2,2,2,\ \ b)\ b)\rangle$

$\vdash \langle q_3,1,2,\ )\ b)\rangle$

$\vdash \langle q_3,1,1,\ b)\rangle$

$\vdash \langle q_3,0,1,\ )\rangle$

$\vdash \langle q_3,0,0,\ \varepsilon\rangle$

II.2.1.3. Fonction de transition généralisée d'un AR

C'est une application $\hat{f}$ de $(Q \times \mathbb{N}) \times \mathbb{N} \times \hat{V}$ dans $(Q \times \mathbb{N})$ telle que pour tout $a \in V$; $r, s \in \hat{V}$

$\hat{f}(\langle q, m\rangle, n, \Lambda) := \langle q, m\rangle$

$\hat{f}(\langle q,m\rangle , n, a \times r + s) := \hat{f}(\hat{f}(\delta'(\langle q,m\rangle , n, a), n + 1, r), n, s)$

où $\delta'(\langle q,m\rangle , n, a) :=$ (si $m = n$ alors $\langle q', m' + n\rangle$

tel que $\delta(q, a) = \langle q', m'\rangle$

sinon indéfini).

II.2.1.4. Bilangage reconnu par un AR.

Une ramification est reconnue par un AR $M = \langle Q, V, q_0, \delta, F\rangle$ si et seulement si $\hat{f}(\langle q_0, 0\rangle, 0, r) = \langle q_f, 0\rangle$ avec $q_f \in F$.

Le bilangage reconnu par M est l'ensemble des ramifications reconnues par M.

$$L(M) = \{r \in \hat{V} \mid \exists\, q_f \in F,\ \hat{f}(<q_0,0>,\ 0,\ r) = <q_f,\ 0>\}$$

II.2.1.5.

Proposition :

Soit $M = <Q, V, q_0, F, \delta>$ un AR, alors

$$\forall (r \in \hat{V},\ q \in Q,\ n \in \mathbb{N}),\ \exists (q' \in Q,\ n' \in \mathbb{N})$$

$$\hat{f}(<q,n>,\ n,\ r) = <q',\ n'> \iff <q,\ n,\ n,\ \text{PREORDRE}(r)> \overset{*}{\vdash} <q',\ n',\ n,\ \varepsilon>$$

Démonstration :

Par récurrence sur r (§ 0.1.2.3).

. La proposition est vraie pour $r = \Lambda$

$$\hat{f}(<q,\ n>,\ n,\ \Lambda) = <q,\ n>$$

et $<q, n, n, \Lambda>$ n'a pas de successeur.

. Si la proposition est vraie pour $r = t$ et $r = u$, on démontre qu'elle est vraie pour tout $r = a \times t + u$, $a \in V$.

Par hypothèse de récurrence : $\forall (q_t, q_u \in Q, n_t, n_u \in \mathbb{N}),\ \exists (q'_t, q'_u \in Q, n'_t, n'_u \in \mathbb{N})$

(1) $\hat{f}(<q_t, n_t>, n_t, t) = <q'_t, n'_t> \iff <q_t, n_t, n_t, \text{PREORDRE}(t)> \overset{*}{\vdash} <q'_t, n'_t, n_t, \varepsilon>$

(2) $\hat{f}(<q_u, n_u>, n_u, u) = <q'_u, n'_u> \iff <q_u, n_u, n_u, \text{PREORDRE}(u)> \overset{*}{\vdash} <q'_u, n'_u, n_u, \varepsilon>$

Par définition de $\hat{f}$:

$$\hat{f}(<q, n>, n, a \times t + u) = \hat{f}(\hat{f}(\delta'(<q,n>, n, a), n + 1, t), n, u) \qquad (3)$$

$$\delta'(<q, n>, n, a) = <q_1, m' + n> \text{ où } <q_1, m'> = \delta(q, a)$$

$$\hat{f}(<q, n>, n, a \times t + u) = \hat{f}(\hat{f}(<q_1, m' + n>, n + 1, t), n, u) \qquad (4)$$

et par définition de préordre et de configuration successeur :
PREORDRE(a × t + u) = a (.PREORDRE(t).).PREORDRE(u)

$<q, n, n, a(.PREORDRE(t).).PREORDRE(u)>$ ⟵

$<q_1, m'+ n , n, (.PREORDRE(t).).PREORDRE(u)>$ ⟵

$<q_1, m'+n, n +1, (.PREORDRE(t).).PREORDRE(u)>$
où $<q_1, m'> = \delta(q, a)$

Dans (4) et (5)

. Si $m' \neq 1$ alors

$\hat{f}(<q, n> , n, a \times t + u)$ est indéfini et (6)
$<q, n, n, PREORDRE(a \times t + u)>$ est indéfini.

. Si $m' = 1$, par (1) $\exists q_1' \in Q, n_1' \in \mathbb{N}$

$$\hat{f}(<q, n>, n, a \times t + u) = \hat{f}(\hat{f}(<q_1, n + 1> , n + 1, t) n, u)$$
$$= \hat{f}(<q_1', n_1'> , n, u)$$

<⟹>

$$<q, n, n, PREORDRE(a \times t + u)> \vdash^{*} <q_1, n + 1, n + 1, PREORDRE(t).).PREORDRE(u)>$$
$$\vdash^{*} <q_1', n_1', n + 1,).PREORDRE(u)>$$
$$\vdash <q_1', n_1', n, PREORDRE(u)>$$

Si $n_1' \neq n$ alors on revient sur (6), autrement par hypothèse de récurrence (2), $\exists q' \in Q, n' \in \mathbb{N}$.

$$\hat{f}(q, n, n, a \times t + u) = \hat{f}(\langle q_1', n_1' \rangle, n, u) = \hat{f}(\langle q_1', n \rangle, n, u)$$

$$= \langle q', n' \rangle$$

$$\Longleftrightarrow$$

$$\langle q, n, n, \text{PREORDRE}(a \times t + u) \rangle \overset{*}{\longmapsto} \langle q_1', n_1', n, \text{PREORDRE}(u) \rangle$$

$$\overset{*}{\longmapsto} \langle q_1', n, n, \text{PREORDRE}(u) \rangle$$

$$\overset{*}{\longmapsto} \langle q', n', n, \varepsilon \rangle$$

Q.E.D.

Corollaire :

$$L(M) = \{ r \in \hat{V} \mid \exists\, q_f \in F, \langle q_0, 0, 0, \text{PREORDRE}(r) \rangle \overset{*}{\longmapsto} \langle q_f, 0, 0, \varepsilon \rangle \}$$

II.2.1.6. AR Non-déterministe

Un AR non-déterministe est un quintuplet $M = \langle Q, V, q_0, F, \delta \rangle$ où δ est une application de $Q \times V$ sur l'ensemble de parties de $Q \times \mathbb{N}$. Les autres éléments sont les mêmes que dans la définition.

Fonction de transition généralisée

$\hat{f}$, la fonction de transition généralisée est une application de $(Q \times \mathbb{N}) \times \mathbb{N} \times \hat{V}$ dans l'ensemble de parties de $Q \times \mathbb{N}$, définie par récurrence sur $\hat{V}$:

$$\forall\, a \times r + s,\ a \in V,\ r, s \in \hat{V}$$

$$\hat{f}(\langle q, m \rangle, n, \Lambda) := \langle q, m \rangle$$

$$\hat{f}(\langle q, m \rangle, n, a \times r + s) := \bigcup_{i \in \delta'(\langle q, m \rangle, n, a)} \hat{f}(\hat{f}(i, n+1, r), n, s)$$

$$\text{et } \delta'(\langle q, m \rangle, n, a) := (\text{si } m = n \text{ alors } \{ \langle q', m' + n \rangle \mid \langle q', m' \rangle \in \delta(q, a) \} \text{ sinon } \phi)$$

II.2.1.7.

Lemme 1 :

Pour toute ramification $r \in \hat{V}$ il y a un AR M tel que L(M) = {r} .

$M = \langle Q, V, q_0, \{q_F\}, \delta \rangle$ est construit de la façon suivante :

Soit . PREORDRE(r) = $a_0\ p_0\ a_1\ p_1 \cdots a_n\ p_n$

où $a_i \in$ SOMMET(r)

$p_i \in \{(\} \cup \{)\}^*$ $\quad 0 \leq i \leq n$

alors $Q = \{q_E, q_i \quad 0 \leq i \leq n\}$.

δ est défini par l'expression :

$(\forall\ a_i\ p_i \subset$ PREORDRE(r))

$p_i \in \{(,\varepsilon\} \Longrightarrow \delta(q_i, a_i) = \langle q_{i+1}, \ell_g(p_i) \rangle$ (1)

$p_i \in \{)\}^+ \Longrightarrow \delta(q_i, a_i) = \langle q_{i+1}, -\ell_g(p_i) \rangle$

δ prend la valeur q_E partout ailleurs.

On trouvera ci-dessous un algorithme qui, pour une ramification donnée r, construit l'AR $M = \langle Q, V, q_0, F, \delta \rangle$ associé. Il utilise la fonction GEN-EDO, sans argument, qui engendre des états.

---

(1) RAPPEL : ℓ_g (chaîne de symboles) est la longueur de la chaîne (§ II.2.1.1.)

```
Fonction CONSTAR (r) ;
        q := q0 ;
        Q := (q0) ;
        ℓg := 0 ;
        r := r. # ;
        a := prem(r) ;
        r := reste(r) ;
        tant que a ≠ # faire
            début
              si prem(r) = "(" alors ℓg := 1
             snsi prem(r) = ")" alors ℓg := ℓg - 1
              sinon
                  début
                      q' := GEN-EDO ( ) ;
                      Q := Q. q' ;
                      δ(q,a) = <q', ℓg> ;
                      q := q' ;
                      a := prem(r) ;
                      ℓg := 0 ;
                  fin ;
              r := reste(r) ;
           fin ;
        F := (q) ;
fin fonction ;
```

Lemme 2 :

Pour tout ensemble de ramifications R produit par une grammaire d'états finis G il y a un AR M tel que L(M) = R.

Si $G = (V_N, V_T, P, S_0)$ alors $M = \langle Q, (V_N \cup V_T), q_0, \{q_F\}, \delta \rangle$ où δ et Q sont construits comme suit :

Soit P l'ensemble de productions

(1) $S_i \rightarrow a_i\ S_{i+1}$ et/ou

(2) $S_i \rightarrow a_{2i}$

où $S_i \in V_N$; a_i, $a_{2i} \in V_T$; $0 \leq i \leq |V_N|$ (la cardinalité de l'ensemble V_N).

alors $Q = \{q_F, q_E, q_i \mid 0 \leq i \leq |V_N|\}$

La fonction δ est construite à partir de l'ensemble de productions P. Chaque production de P du type (1) provoque les définitions de δ:

$$\delta(q_{2i}, a_i) = \langle q_{2i+1}, 0\rangle$$

$$\delta(q_{2i+1}, S_{i+1}) = \langle q_{2i+2}, 1\rangle$$

et chaque production du type (2) :

$$\delta(q_{2i}, a_{2i}) = \langle q_F, -1\rangle$$

$\forall\ 1 \leq i \leq |V_N|$.

En outre :

$$\delta(q_F, \varepsilon) = \langle q_F, -1\rangle\ ,$$

et δ prend la valeur q_E partout ailleurs.

Il faut remarquer que s'il y a deux productions (1) et (2) telles que les symboles a_i et a_{2i} sont égaux, l'AR construit est non-déterministe.

II.2.2. AR paramétrés

Notre but étant la reconnaissance de ramifications paramétrées, on va ajouter aux AR la manipulation de paramètres (§ I.2.1.1.). L'automate résultant est appelé un AR paramétré (ARP).

II.2.2.1. Définitions

Un automate d'états finis adapté aux ramifications paramétrées (ARP) est une 6-uple $M = <Q, V, P, q_0, F, S_p>$ où Q, V, q_0, et F ont déjà été définis (§ II.2.1.),

P est un ensemble de paramètres (§ I.2.1.1.),

δ_p fonction de transition qui est une application de $Q \times P$ sur $Q \times \mathbb{N}$

Fonction de transition généralisée d'un ARP

C'est une application de $(Q \times \mathbb{N}) \times \mathbb{N} \times \hat{V}$ dans $Q \times \mathbb{N}$ telle que $\forall A \in V$; $r, s \in \hat{V}$:

$$\hat{f}_p(<q, m>, n, \Lambda) := <q, m>,$$

$$\hat{f}_p(<q,m>, n, A \times r + s) := \hat{f}_p(\hat{f}_p(\delta'_p(<q,m>, n, A), n + 1, r), n, s) ;$$

où $\delta'_p(<q, m>, n, A)$:= (si $n = m$ et ($\exists p \mid \delta_p(q, p) = <q', m'>$ et $p_2(a)$)[1]

alors $<q', m' + n>$

sinon indéfini)

De façon similaire aux AR une configuration d'un ARP est un 4-uplet $<q, m, n, w>$ de $Q \times \mathbb{N}^2 \times (V \cup \{"(", ")"\})^*$, et les configurations

$$<q_0, 0, 0, w> \text{ et } <q_f, 0, 0, \epsilon> \text{ (où } q_f \in F)$$

sont appelées initiale et finale respectivement.

$<q', m', n', a'>$ est une configuration successeur de $<q, m, n, a>$ dans un ARP M_p si et seulement si :

---

(1) RAPPEL : $p_2(A)$ est l'évaluation du prédicat de p sur l'étiquette A.

si prem(a) = "(" alors n' := n + 1 ; q' := q ; m' := m ; a' := reste(a).

si prem(a) = ")" alors n' := n - 1 ; q' := q ; m' := m ; a' := reste(a).

si prem(a) $\in$ V alors n' := n ; a' := reste(a) ; $<q', m'> = \delta'_p(<q, m>, n, prem(a$

on écrit $<q, m, n, a> \vdash_{M_p} <q', m', n', a'>$

Bilangage reconnu par un ARP

Une ramification r de $\hat{V}$ est reconnue par un ARP $M_p = <Q, V, P, F, q_0, \delta_p>$ ssi $\hat{f}_p(<q_0, 0> , 0, r) = <q_f, 0>$ avec $q_f \in F$.

Le bilangage reconnu par M_p est l'ensemble des ramifications reconnues par M_p.

$$L(M_p) = \{r \in \hat{V} \mid \hat{f}_p(<q_0, 0> , 0, r) = <q_f, 0> \text{ et } q_f \in F\}$$

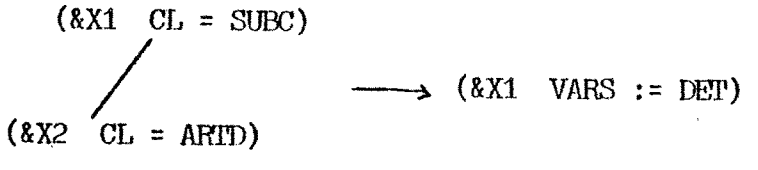
II.2.2.2. Lemme

Pour toute ramification paramétrée $r \in \hat{P}_E$ il y a un ARP tel que le bilangage reconnu soit la famille de r (§ I.2.1.3).

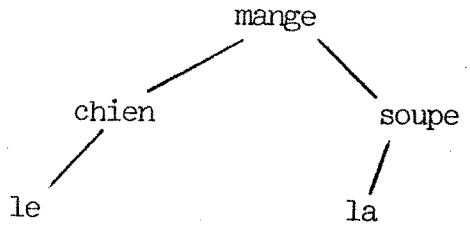
La construction de cet ARP est similaire à celle du lemme 1, § II.1.6.

II.2.3. Composition d'ARP

Etant donné un système général de transformations paramétrées et une ramification à transformer, s'il y a une règle de transformation qui peut être appliquée elle peut l'être plusieurs fois, et pas nécessairement au niveau 0 de la ramification. Par exemple si nous avons la règle :



dans la phrase :



cette règle est applicable aux groupes nominaux ; 'le chien', et 'la soupe'.

Il se peut aussi que sur une ramification il y ait plusieurs règles applicables ce qui implique la reconnaissance de plusieurs ramifications paramétrées dans une ramification étiquetée.

Nous avons développé une structure de contrôle pour la composition des ARP afin de reconnaître toutes les sous-ramifications équivalentes à la partie gauche d'une règle de transformation, en un seul parcours d'une ramification donnée. Cette structure de contrôle servira aussi à la manipulation des paramètres de type sous-arborescence (P_S) et de ceux de type ramification (P_R).

II.2.3.1. Définitions

Un ARP composé $\mathcal{M}$ est un ensemble d'ARP M_i, ayant tous le même état initial :

$$\mathcal{M} = \{M_i \mid M_i = \langle Q_i, V, P_i, F_i, q_0, \delta_i \rangle\} \text{ est un ARP}$$

Formellement, étant donné un ensemble d'ARP $M_i = \langle Q_i, V, P_i, F_i, q_0, \delta_i \rangle$ un ARP composé $\mathcal{M}$ est un 6-uple $\mathcal{M} = \langle Q, V, P, F, q_0, \delta_c \rangle$ où :

$Q = \bigcup_i Q_i$

q_0 est l'état initial commun aux M_i,

$F = \bigcup_i F_i$.

V est le vocabulaire étiqueté

$P = \bigcup_i P_i$ l'ensemble de paramètres

et δ_c est une application de $Q \times P$ sur $\mathcal{P}(Q \times \mathbb{N})$

$$\delta_c(q,p) = \{<q',m'> \mid \exists\, i \quad \delta_i(q,p) = <q',m'>\}$$

Fonction de Transition Généralisée

C'est une application $\hat{f}_c$ de $(Q \times \mathbb{N}) \times \mathbb{N} \times \hat{V}$ dans l'ensemble des parties de $Q \times \mathbb{N}$ définie par récurrence sur $\hat{V}$ comme suit :

$$\hat{f}_c(<q,m>, n, \Lambda) = <q, m>$$

$$\hat{f}_c(<q,m>, n, a \times r + s) = \bigcup_{i \in \delta'_c(<q,m>, n, a)} \hat{f}_c(\hat{f}_c(i, n+1, r), n, s)$$
$$\cup \{<q',m' + n> \mid \exists p, \ <q',m'> \in \delta_i(q_0,p) \text{ et } p_2(a)\} \qquad (1)$$

où $\delta'_c(<q,m>, n, a)$:= si $m < n$ alors $\{<q,m>\}$ sinon
si $m = n$ et $(\exists\, p \mid \delta(q,p) \neq \phi$ et $p_2(a))$
alors $\{<q',m'+ n> \mid \exists\, p, <q',m'> \in \delta_c(q,p)$ et $p_2(a)$

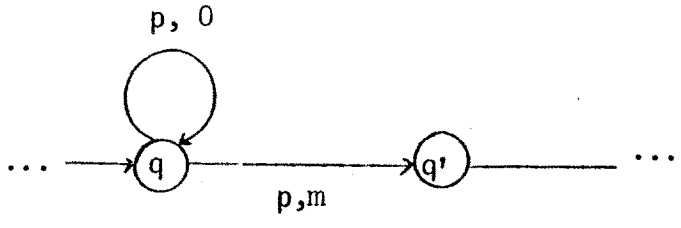
sinon si $q \in F$ alors $\{<q,-m>\}$
sinon ϕ ;

La fonction δ'_c et l'expression (1) expriment en fait la structure de contrôle utilisée. Au moyen de l'expression (1) tout sommet d'une ramification peut déclencher l'activation de n'importe quelle ARP M_i par une transition de l'état initial vers un état dans Q_i. Dans la fonction δ'_c tout couple <q,m> (représentant un état actif dans l'automate) : (i) est éliminé si m est plus grand que le niveau de l'étiquette qu'on analyse, (ii) reste invariable si m lui est inférieur, ou (iii) si m lui est égale et s'il existe une transition on l'effectue.

Le traitement de paramètres de type sous-arborescence est implicite dans δ'_c : si une étiquette b occasionne une transition d'un couple $<q_1, n_1>$ vers un couple $<q_2, n_2>$ et si le paramètre associé à la transition est de type sous-arborescence, alors n_2 est obligatoirement égal ou inférieur à n_1. En conséquence l'analyse de la sous-arborescence en dessous de b ne modifie pas le couple $<q_2, n_2>$.

Ceci implique que dans une transition associée à un paramètre de type étiquette qui est une feuille dans la ramification paramétrée, on doit ajouter au prédicat cette contrainte (ne pas avoir des sommets dépendants) autrement il sera traité comme un paramètre sous-arborescence.

Le traitement de paramètres de type ramifications est fait dans l'automate moyennant une transition d'un état vers lui-même. Ainsi si <z,p> est un paramètre ramification, dans l'automate on aura quelque part une transition :



correspondant à : $\delta_c(q,p) = <q, 0>$, et $\delta_c(q,p) = <q',m>$. Ainsi à l'état q on accepte un nombre indéfini d'arborescences, c'est-à-dire une ramification.

En ce qui concerne la structure de contrôle, elle est équivalente à exécuter en parallèle tous les automates correspondants à l'ensemble des ramifications paramétrées que l'on veut reconnaître. Un automate composé possède donc une liste d'états de contrôle, et pour chaque état, le niveau de la ramification dans lequel doit se trouver un symbole de V pour réaliser une transition. Pour déterminer le comportement de l'automate pendant l'analyse d'une chaîne nous devons connaître :

i) une liste de couples : <état, niveau>;
ii) une chaîne de symboles à analyser représentant tout ou partie de l'écriture en préordre d'une ramification étiquetée ;

iii) le **niveau** de la ramification auquel se trouve le premier symbole de la chaîne analysée.

Configurations

Si $\mathcal{M}$ est un ARP composé alors une 3-uple $\langle C_i, n, w\rangle$ de $\mathcal{P}(Q \times \mathbb{N}) \times \mathbb{N} \times (V \cup \{"(",")"\})^*$ est une configuration de $\mathcal{M}$. Une configuration de la forme :

(1) $\{\langle q_0, 0\rangle\}, 0, w\rangle$ est appelée initiale

(2) $\langle C_j, 0, \varepsilon\rangle$ où $(\forall \langle q, m\rangle \in C_j, q \in F)$ est appelée finale.

Configuration successeur

$\langle C_i, n, a\rangle \vdash_{\mathcal{M}} \langle C_j, n', a'\rangle$ ssi :

si prem(a) = "(" alors n' = n + 1, a' = reste(a) ; et $C_j = C_i$

si prem(a) = ")" alors n' = n - 1, a' = reste(a) ; et $C_j = C_i$

si prem(a) $\in$ V alors n' = n ; a' = reste(a) ; et

$$C_j = \bigcup_{i \in C_i \cup C_0} \delta'_c (i, n, \text{prem}(a))$$

où $C_0 = \{\langle q', m' + n\rangle \mid \exists\, p, \delta_c(q_0, p) = \langle q', m'\rangle$ et $p_2(\text{prem}(a)$

On note que par construction pour toute ARPC $\mathcal{M}$, et pour toute ramification $r \in \hat{V}$.

$$\langle\{\langle q_0, 0\rangle\}, 0, \text{PREORDRE}(r)\rangle \vdash^*_{\mathcal{M}} \langle C_j, 0, \varepsilon\rangle$$

où $\langle C_j, 0, \varepsilon\rangle$ est une configuration finale.

I.2.3.2.

Exemple

Soit l'ARP composé $\mathcal{M}$ <Q, V, P, q_0, F, δ_c> formé par les ARP M_1, M_2, M_3 avec :

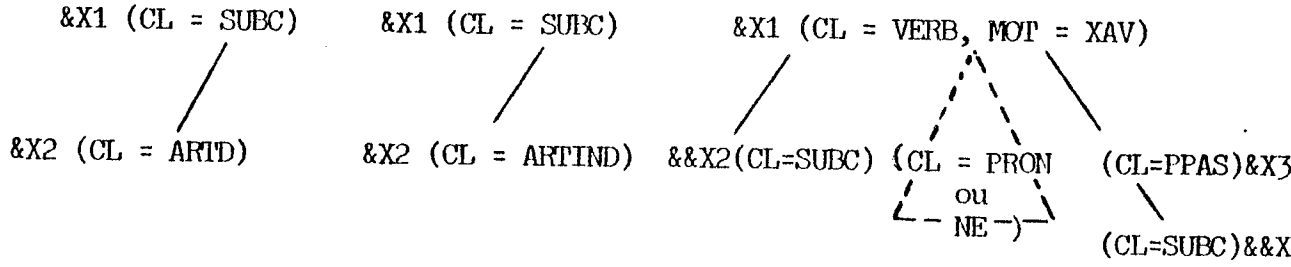
$Q_1 = \{q_0, q_1, q_2\}$
$P_1 = \{(CL = SUBC), (CL = ARTD)\}$
$F_1 = \{q_2\}$
$\delta_1(q_0, (CL = SUBC)) = <q_1, 1>$
$\delta_1(q_1, (CL = ARTD)) = <q_2, -1>$
} M_1

$Q_2 = \{q_0, q_1, q_3\}$
$P_2 = \{(CL = SUBC), (CL = ARTIND)\}$
$F_2 = \{q_3\}$
$\delta_2(q_0, (CL = SUBC)) = <q_1, 1>$
$\delta_2(q_1, (CL = ARTIND)) = <q_3, -1>$
} M_2

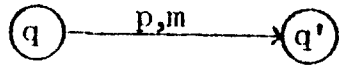
⋮

$Q_3 = \{q_0, q_4, q_5, q_6, q_7, q_8\}$
$P_3 = \{(CL = VERB, MOT = XAV), (CL = PPAS)\}$
$F_3 = \{q_7\}$
$\delta_3(q_0, (CL = VERB, MOT = XAV)) = <q_4, 1>$
$\delta_3(q_4, (CL = SUBC)) = <q_5, 0>$
$\delta_3(q_4, (CL = PRON ou NE)) = <q_5, 0>$
$\delta_3(q_5, (CL = PRON ou NE)) = <q_5, 0>$
$\delta_3(q_5, (CL = PRON ou NE)) = <q_6, 0>$
$\delta_3(q_6, (CL = PPAS)) = <q_7, 1>$
$\delta_3(q_7, (CL = SUBC)) = <q_8 - 2>$
} M_3

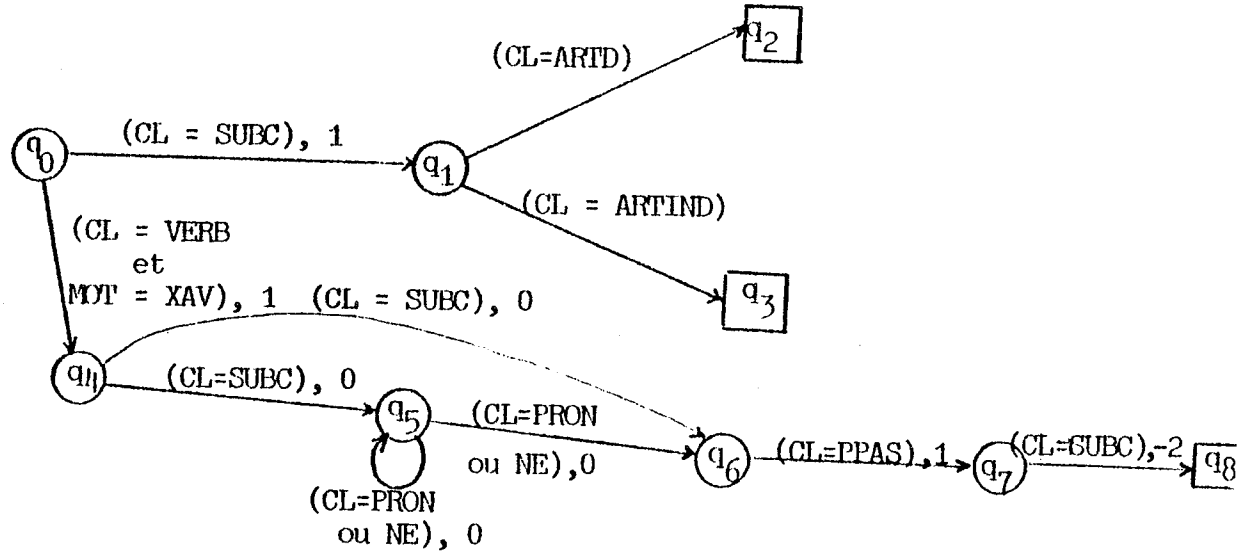
Les automates M_1, M_2 et M_3 reconnaissent les ramifications paramétrées suivantes (respectivement) :



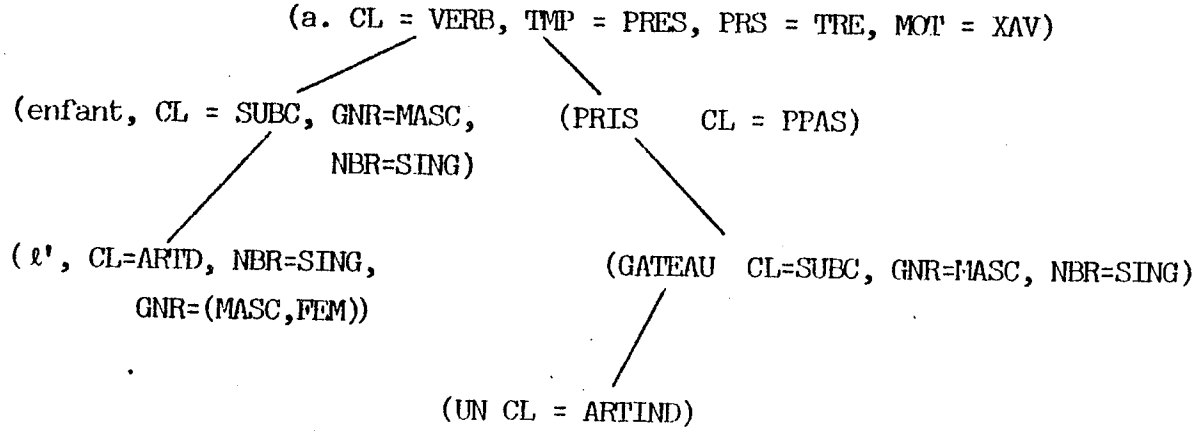
Si on représente graphiquement $\delta_c(q,p) = \langle q', m \rangle$ par :



l'ARP composé $\mathcal{M}$ peut être représenté par le diagramme suivant :



Voyons l'analyse de la ramification :



Pour simplifier la représentation linéaire, on écrira seulement le premier élément de chaque étiquette, les autres éléments n'étant pas explicites.

Nous avons donc :

$<\{<q_0, 0>\}, 0, A(ENFANT(L') PRIS (GATEAU (UN))) >$

$\vdash_{\mathcal{M}}$ $<\{<q_4, 1>\}, 0, (ENFANT (L') PRIS (GATEAU (UN))) >$

$<\{<q_4, 1>\}, 1, ENFANT (L') PRIS (GATEAU (UN))) >$

$<\{<q_6, 1>, <q_5, 1>, <q_1, 2>\}, 1, (L') PRIS (GATEAU (UN))) >$

$<\{<q_6, 1>, <q_5, 1>, <q_1, 2>\}, 2, (L')PRIS (GATEAU (UN))) >$

$<\{<q_6, 1>, <q_5, 1>, <q_2, -1>\}, 2,) PRIS (GATEAU (UN))) >$

$<\{<q_6, 1>, <q_5, 1>, <q_2, -1>\}, 1, PRIS (GATEAU (UN))) >$

$<\{<q_7, 2>, <q_2, -1>\}, 1, (GATEAU (UN))) >$

$<\{<q_7, 2>, <q_2, -1>\}, 2, GATEAU (UN))) >$

$<\{<q_8, 0>, <q_2, -1>, <q_1, 3>\}, 2, (UN))) >$

$<\{<q_8, 0>, <q_2, -1>, <q_1, 3>\}, 3, UN))) >$

$<\{<q_8, 0>, <q_2, -1>, <q_3, -2>\}, 3,))) >$

$<\{<q_8, 0>, <q_2, -1>, <q_3, -2>\}, 2,)) >$

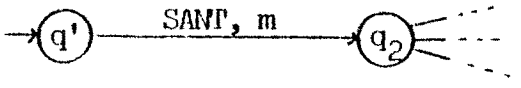
$<\{<q_8, 0>, <q_2, -1>, <q_3, -2>\}, 1,) >$

$<\{<q_8, 0>, <q_2, -1>, <q_3, -2>\}, 0, \varepsilon >$

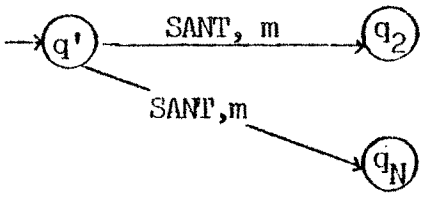
Le fait que dans la configuration finale on trouve les états finaux q_2, q_3 et q_8 signifie que les ramifications paramétrées associées aux ARP M_1, M_2 et M_3 se trouvent dans la ramification donnée au niveau indiqué par la valeur absolue du nombre qui les accompagne.

II.2.3.3. Construction d'un ARPC à partir d'un ensemble de ramifications paramétrées

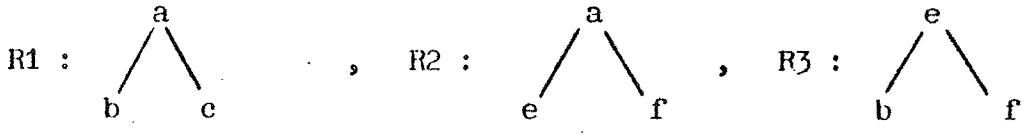
Etant donné un ensemble de ramifications paramétrées on cherche à construire l'ARP composé associé. On suit le schéma du programme itératif pour la construction d'un AR à partir d'une ramification (§ II.2.1.7), mais avant de créer un état et une transition, on vérifie la non-existence de cette transition. Ceci dans le but de réduire le nombre total d'états et de transitions, et on aura :



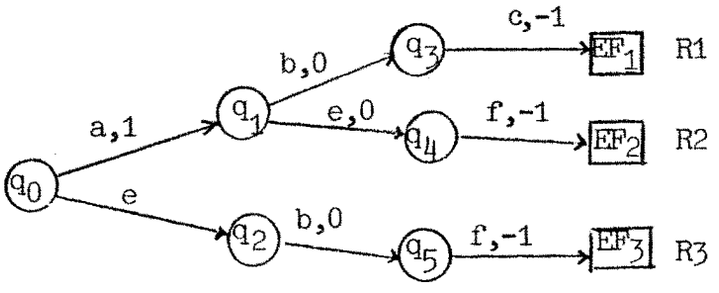
plutôt que :



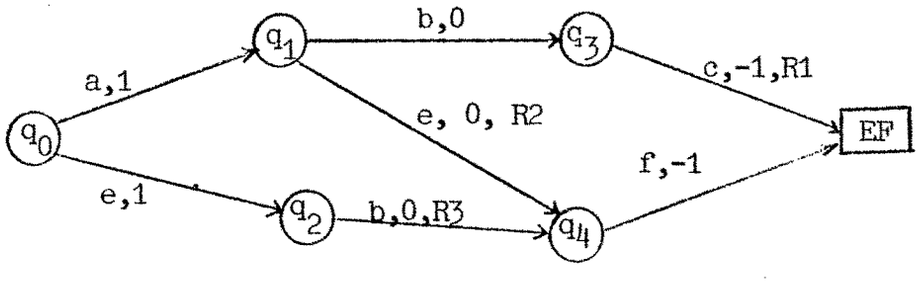
De cette façon on obtient une mise en facteur vers l'état initial de l'ensemble d'automates composant l'ARPC. On pourrait essayer aussi une mise en facteur vers l'état final mais ceci impliquerai : i) l'unicité de l'état final, ii) la nécessité d'ajouter un troisième élément à la fonction de transition pour identifier l'automate au moyen duquel on arrive à l'état final. Cet identification est importante, surtout dans le cadre de systèmes de Transformations puisqu'elle nous indique la règle de transformation qu'il faut appliquer. Par exemple : si on a les ramifications paramétrées :



L'ARPC avec mise en facteur vers l'état initial seulement serait :



où on a mis explicitement la relation entre les états finaux et les ramifications p. Si on considère un seul état final, les états q_4 et q_5 peuvent être réduits en un seul état, mais il faut indiquer sur les transitions vers q_4 et q_5 quelle serait la ramification reconnue dans le cas où on arrive à l'état final :



Cette mise en facteur est effectuée par la procédure REDUCTION, dont on trouvera le schéma de programme ci-après :

```
fonction CONST-ARPC (e) ;
    co e est un ensemble de ramifications paramétrées,
       ram est une ramification paramétrée,
       Q_i est l'ensemble d'états de l'automate associé à ram,
       lopt est un ensemble de triplets dans Q × N² utilisé pour le traitement
       de paramètres optionnels
       ℓ est un ensemble d'états fco
pour chaque ram dans e faire
début
    ram := ram . #;
    Q_i := φ ;
    ℓ  := (q_0) ;
    ℓ_g := 0 ;
     a := prem(ram) ;
    ram := reste(ram) ;
    tant que a ≠ # faire
    début
        si a = "(" alors ℓ_g := 1
        snsi a = ")" alors ℓ_g := ℓ_g -1
        sinon
        début  co si pour un état dans ℓ il existe la transition <a, ℓ_g>
               alors on ne la duplique pas ; ℓ est ainsi divisé en ℓ'
               (ces états) et ℓ" (les états pour lesquels il faut créer
               la nouvelle transition) fco
               pour chaque q dans ℓ faire
               si ∃ q_2 | (q_2, ℓ_g) ∈ δ(q,a) et q_2 ∉ F et (nombre de transi-
                                  tions vers q_2 = 1) et (a ∉ P_R ou δ(q, 0) ∈ δ(q,a))
               alors ℓ' := q_2. ℓ'
               sinon ℓ" := q_2. ℓ" ;
               pour chaque <q,m,n> dans lopt faire
               co le troisième élément du triplet sert à traiter les sous-
               arborescences dont la racine est un paramètre (de type
               étiquette) optionnel fco
```

```
début
     n := n + ℓ_g ;
     si n ≤ 0 alors lopt ":= <q,m>. lopt"
              sinon lopt' := <q,m,n>. lopt' ;
fin ;
ℓ_n := lopt" ∪ {<q, g> | q ∈ ℓ"} ;
si ℓ_n ≠ φ alors
début
     q_N := GEN-EDO ;
     q'_N := GEN-EDO ;
     Q_i := q_N . Q_i ;
     pour chaque <q,m> dans ℓ_n faire
     si a ∉ P_R co les paramètres de type ramification fco
     alors δ(q,a) := (q_N,m).δ(q,a)
     snsi q ∈ Q_i alors
     début
          δ(q,a) := (q,0).δ(q,a) ;
          δ(q,a) := (q,m).δ(q,a) ;
     fin ;
     sinon co on ne peut pas modifier q puisque c'est un
           état commun avec un autre automate fco
     début
          δ(q,a) := (q'_N, 0).δ(q,a) ;
          δ(q'_N,a):= (q'_N, 0).δ(q'_N,a);
          δ(q'_N,a):= (q_N, m).δ(q'_N, a) ;
          Q_i := q'_N. Q_i ;
     fin ;
     ℓ' := q_N.ℓ' ;
fin ;
si a ∈ P" co les paramètres optionnels fco
alors début
     pour chaque <q,m> dans ℓ_n faire
     lopt' := <q,m,0>. lopt' ;
     pour chaque q dans ℓ' faire
     lopt' := <q, ℓ_g, 0 >. lopt' ;
     fin ;
```

```
                lopt := lopt' ;
                ℓ := ℓ' ;
                ℓ_g := 0 ;
            fin ;
            a := prem(ram) ;
            ram := reste(ram) ;
        fin ;
        F := ℓ ∪ F ;
fin ;
REDUCTION ;
fin fonction ;
```

La procédure de REDUCTION

Dans la procédure de réduction on suppose que la fonction de transition δ c'est une application de $Q \times P$ dans $Q \times \mathbb{N} \times \mathcal{P}(R)$, où R est l'ensemble des "noms" associés à chaque ramification de E (l'ensemble de ramifications p donné). Les fonctions δ_1, δ_2 et δ_3 sont respectivement le premier, deuxième et troisième élément de δ.

```
fonction REDUCTION ;
    CHANGE := "vraie" ;
    tant que CHANGE faire
    début
        CHANGE :="faux" ;
        VALQ1 := NIL ;
        VALQ2 := NIL ;
        pour chaque q1 dans Q faire
        si ∃ q2 ∈ Q | ∀ pi pour lequel δ(q1, pi) est définie on a
                δ(q1,pi) = <qi,mi,val_i1> <==> δ(q2,pi) = <qi,mi,val_i2>
        alors co on peut faire de q1 et q2 un seul état fco :
```

début

i) CHANGE := "vraie" ; q_N := GEN-EDO() ;

ii) pour chaque p_i pour lequel $\delta(q_1,p_i)$ est définie faire

$\delta(q_N,p_i) := \langle q_i, m_i, \delta_3(q_1,p_i) \cup \delta_3(q_2,p_i)\rangle$;

$VALQ_1 := VALQ_1 \cup \delta_3(q_1,p_i)$;

$VALQ_2 := VALQ_2 \cup \delta_3(q_2,p_i)$;

iii) pour chaque q dans Q faire

pour chaque p pour lequel $\delta(q,p)$ est définie faire

si $\delta_1(q,p) = q_1$ alors

$\delta(q,p) := \langle q_N, \delta_2(q,p),$ si $\delta_3(q,p) = \phi$ alors VALQ

sinon $\delta_3(q,p)\rangle$

sinon si $\delta_1(q,p) = q_2$ alors

$\delta(q,p) = \langle q_N, \delta_2(q,p),$ si $\delta_3(q,p) = \phi$ alors $VALQ_2$

sinon $\delta_3(q,p)\rangle$;

iv) ajouter q_N dans Q ;

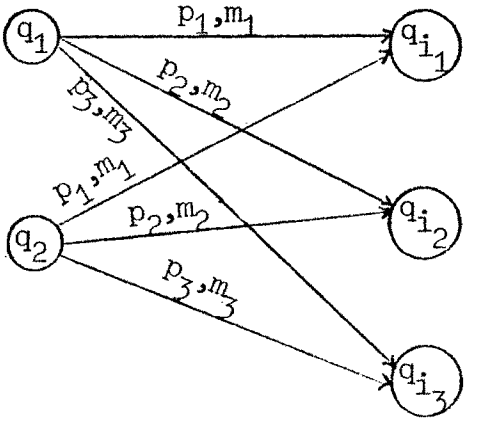
retirer q_1 et q_2 de Q ;

fin ;

fin ;

fin fonction ;

Dans l'algorithme de REDUCTION la condition que doivent remplir deux noeuds q_1 et q_2, pour être "réduits" à un seul est l'égalité absolue de la fonction de transition δ. Elle doit être définie pour les mêmes paramètres, et doit occasionner des transitions vers les mêmes états. Graphiquement on a :



Aussi une condition nécessaire (mais non suffisante) pour "réduire" deux états est qu'ils aient des transitions vers le même état. Or, par la construction de l'ARPC avant l'exécution de REDUCTION ceci n'est vrai que pour les états connectés aux états finaux. Donc ce n'est pas la peine de tester sur tout l'ensemble d'états mais seulement sur ceux-ci. Une fois fait un premier passage de réduction sur ces états, les seuls états susceptibles d'être réduits (toujours à cause de la condition ci-dessus) sont les états connectés aux états réduits dans le pas antérieur.

On définit l'application δ_1^{-1} de Q dans $\mathcal{P}(Q)$ comme

$$\delta_1^{-1}(q_1) = \{q \mid \exists\, p,m \text{ pour lesquelles } \delta(q,p) = \langle q_1, m\rangle\}$$

δ_1^{-1} est la fonction de transition inverse. Elle sera généralisée aux ensembles de la façon suivante :

$$\text{si } Q_1 \subset Q \text{ alors } \delta_1^{-1}(Q_1) = \bigcup_{q_i \in Q_1} \delta_1^{-1}(q_i)$$

Si on tient compte des remarques précédentes, on a une deuxième version de l'algorithme de REDUCTION :

```
fonction REDUCTION
    CHANGE := "vraie" ;
    Q' := (EF) ;
    tant que CHANGE faire
    début
        CHANGE := "faux"
        VALQ1 := nil ;
        VALQ2 := nil ;
        pour chaque q1 dans δ1^-1 (Q') faire
```

si $\exists\, q_2 \in \delta_1^{-1}(Q') \mid \forall\, p_i$ pour lequel $\delta(q_1,p_i)$ est définie on a

$\delta(q_1,p_i) = \langle q_i, m_i, vali_1\rangle \Longleftrightarrow \delta(q_2,p_i) = \langle q_i, m_i, vali_2\rangle$

alors co on peut faire de q_1 et q_2 un seul état fco

début

i) CHANGE := "vraie" ;

q_N := GEN-EDO() ;

ii) pour chaque p_i pour lequel $\delta(q_1,p_i)$ est définie faire

$\delta(q_N,p_i) := \langle q_i, m_i, \delta_3(q_1,p_i) \cup \delta_3(q_2,p_i)\rangle$;

$VALQ_1 := VALQ_1 \cup \delta_3(q_1,p_i)$;

$VALQ_2 := VALQ_2 \cup \delta_3(q_2,p_i)$;

iii) pour chaque q dans $\delta_1^{-1}(q_1)$ faire

pour chaque p pour lequel $\delta_1(q,p) = q_1$ faire

$\delta(q,p) := \langle q_N, \delta_2(q,p),$ si $\delta_3(q,p) = \phi$ alors $VALQ_1$ sinon $\delta_3(q,p)\rangle$;

iv) pour chaque q dans $\delta_1^{-1}(q_2)$ faire

pour chaque p pour lequel $\delta_1(q,p) = q_2$ faire

$\delta(q,p) := \langle q_N, \delta_2(q,p),$ si $\delta_2(q,p) = \phi$ alors $VALQ_2$ sinon $\delta_3(q,p)\rangle$;

v) ajouter q_N dans Q et Q' ;

retirer q_1 et q_2 de Q ;

fin ;

fin ;

fin fonction ;

II.3. SYSTEMES DE TRANSFORMATIONS PARAMETREES ET ARPC

Le processus de transformation d'une ramification dans un STP peut être ramené aux étapes suivantes :

i) reconnaissance d'une sous-ramification équivalente (§ I.2.1.3.) à la partie gauche d'une règle de transformation $T_i: < X_i, Y_i, C_i, A_i >$. Simultanément on associe à chaque paramètre de X_i un élément de la sous-ramification déterminée, c'est ce qu'on a appelé la valeur du paramètre.

ii) vérification de la condition C_i

iii) l'application de la transformation proprement dite c'est-à-dire la construction d'une ramification (d'après Y_i et les valeurs des paramètres) qui remplacera la sous-ramification reconnue au premier pas.

iv) Finalement, l'application des effets secondaires A_i sur la ramification construite ci-dessus.

C'est à l'exécution du premier pas que nous allons nous servir des ARPC. Si on construit un ARPC avec les parties gauche des règles de transformation, en un seul parcours de l'arborescence nous pourrons reconnaître toutes les règles de transformation applicables. Il faut certainement assigner à chaque paramètre sa valeur au moment d'effectuer les transitions, et quand on arrive à l'état final on peut exécuter le pas (ii). En ce qui concerne l'application de la transformation, il faut tenir compte des caractéristiques du système de transformations.

II.3.1. Systèmes de Transformations non Church-Rosser.

Si le système de transformations ne possède pas la propriété de Church-Rosser, la transformation d'une ramification peut donner comme résultat plusieurs ramifications différentes, suivant l'ordre d'application des règles. Il faut donc suivre toutes les dérivations possibles à partir d'une ramification donnée pour obtenir toutes ses formes normales. Ceci est le but des algorithmes donnés ci-après. Ils parcourent la ramification

en préordreet chaque fois qu'on découvre une règle applicable, la ramification produite par l'application de cette règle est ajoutée à l'ensemble résultat.

Ces algorithmes seront utilisés postérieurement dans la fonction de contrôle pour les systèmes de transformations qui ne sont pas Church-Rosser.

```
fonction trans (ramification) ;
     résultat := nil ;
     niveau := 0 ;
     liste d'états := (q0) ;
     pour chaque arbre dans ramification faire
          transarb(arbre) ;
     rendre résultat ;
fin fonction ;

fonction transarb(arbre) ;
     règle applicable := nil ;
     niveau := niveau + 1 ;
     si arbre est un noeud alors père := arbre
          sinon père := racine (arbre) ;
                fils := reste (arbre) ;
          fsi ;
     liste d'états := ajouter (<q0, niveau>, liste d'états) ;
     liste d'états := transitions (liste d'états, père) ;
     si règle applicable et cond (règle applicable) alors résultat := exécuter
                                   (règle applicable). résultat ;
                        fsi ;
     si fils alors pour chaque arb dans fils faire transarb (arb) ;
            fsi ;
     niveau:= niveau - 1 ;
fin fonction ;
```

```
fonction transitions (liste d'états, noeud) ;
     états result := nil ;
     pour chaque état dans liste d'états faire
          co état = <q, m, règles> fco
          si m(état) < niveau alors états result := état. états result
           snsi m(état) = niveau alors pour chaque état nouv dans
                δ(q(état), noeud) faire
          si q(état nouv) = état final alors
             règle applicable := règles (état nouv) ∩ règles(état)
           sinon états result := <q(état nouv), m(état nouv) + niveau,
                 règles (état nouv) ∩ règles(état)> . états result
           fsi
          fsi ; fsi ;
     rendre états result ;
fin fonction ;
```

Les algorithmes utilisent les fonctions auxiliaires :

cond (règle) qui teste si la condition associée à une règle donnée est vérifiée par la sous-ramification équivalente à la partie gauche de la transformation.

exécuter (règle) qui exécute une règle de transformation donnée sur la variable globale "ramification". Elle délivre comme résultat la ramification transformée et la variable "ramification" n'est pas modifiée.

δ(q, noeud), la fonction de transition de l'ARPC. Etant donnés un état et un noeud, cette fonction délivre la liste d'états vers lesquels on peut faire une transition. Elle vérifie que le noeud remplit la condition exigée pour chaque paramètre d'une transition et dans ce cas, assigne la valeur correspondante au paramètre.

Etant donnée la fonction TRANS qui délivre toutes les ramifications qui peuvent être obtenues en appliquant une seule transformation (n'importe laquelle, mais une seule fois) sur une ramification donnée, nous allons

étudier l'algorithme de contrôle pour les STP non Church-Rosser. L'idée générale de l'algorithme est de : (i) stocker toutes les ramifications obtenues par des dérivations de même longueur (n) dans la liste RAMIFICATIONS ; (ii) appliquer la fonction TRANS à chaque ramification de cette liste (ce qui donne les ramifications obtenues par dérivations de longueur n + 1) et éliminer de la liste donnée par la fonction TRANS les ramifications qui ont déjà été obtenues au cours du traitement (les ramifications "intermédiaires"). Dans le cas où le résultat de TRANS est la liste vide, alors la ramification donnée comme argument est irremplaçable.

Afin de garantir l'arrêt de l'algorithme dans le cas de dérivations de longueur infini, il faut fixer une limite maximum pour la longueur des dérivations ; il faut, bien sûr, choisir une limite assez grande pour avoir la quasi-certitude d'arrêter l'algorithme seulement dans ce cas. L'algorithme ainsi construit s'arrête puisque de plus le cas de boucles a été résolu en stockant toutes les ramifications intermédiaires.

```
fonction transf-no-Ch.R (ramification) ;
     compteur := 0 ;
     ramifications := (ramification) ;
     ram intermédiaires := nil ;
     ram irremplaçables := nil ;
étiq : ram nouv := nil ;
     pour tout ram dans ramifications faire
          début
               ram intermédiaires := ram. ram intermédiaires ;
               temp := trans (ram) ;
               si temp alors
                 pour tout ramtemp dans temp faire
                      si ramtemp ∉ ramintermédiaires et
                         ramtemp ∉ ramifications
                         alors ram nouv := ramtemp. ram nouv
                         fsi
```

```
                sinon ram irremplaçables := ram.ram irremplaçables
                fsi ;
            fin ;
        compteur := compteur + 1 ;
        si compteur = limit max alors rendre"chemin infini";
        ramifications := ram nouv ;
        si ramifications alors allera etiq fsi ;
        rendre ram  irremplaçables ;
fin fonction ;
```

II.3.2. STP Church-Rosser

Si on travaille avec un système de transformations Church-Rosser il est évident qu'on n'a pas besoin d'utiliser l'algorithme précédent (bien qu'il soit toujours valable) car on sait que toutes les dérivations possibles mènent à la même ramification : la forme normale. Il suffit donc d'en suivre une pour y arriver.

La stratégie d'analyse que nous avons choisie consiste à exécuter les transformations en même temps que l'ARPC effectue le parcours de la ramification ; ainsi dès qu'on reconnaît une ramification équivalente à la partie gauche d'une règle de transformation, on l'applique si aucune sous-ramification de cette dernière n'a été transformée. Cette restriction vient du fait que dans les Systèmes Church-Rosser si les domaines restreints d'application de deux (ou plusieurs) règles de transformations ne sont pas disjoints (figure 2.3.2.a.)

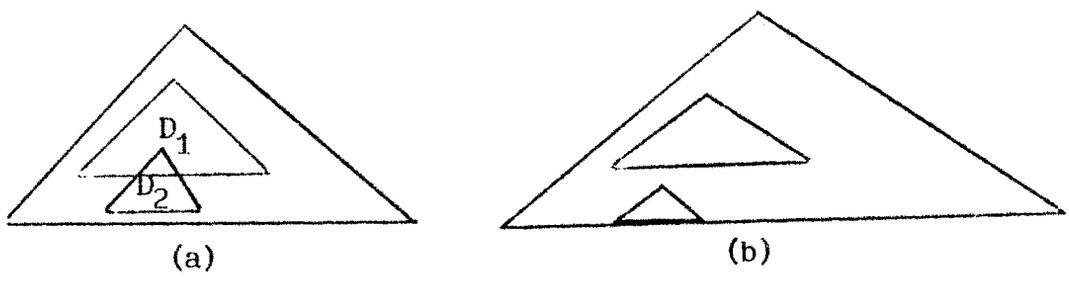


Fig. 2.3.2.

(et alors l'un des domaines d'application est forcément une sous-ramification de l'autre), il est fort probable que l'application d'une des deux règles modifie la partie commune interdisant l'application de l'autre. Dans ce cas, on exécute la règle dont la racine (ou les racines) du domaine d'application est au niveau le plus grand (c'est-à-dire le plus bas) D2 dans ce cas, et après on refait le parcours de la ramification pour vérifier, soit que l'autre règle (associée à D1) reste applicable, soit qu'il faut en appliquer une différente. Il faut souligner que cette technique est utilisée aussi dans le cas de la figure 2.3.2.b. puisqu'on refait le parcours systématiquement après l'exécution de toute transformation.

Dans les pages suivantes on trouvera les différents algorithmes de contrôle pour un SGTP Church-Rosser. L'algorithme consiste à transformer la ramification donnée jusqu'à ce qu'il n'y ait plus de règles applicables : pour cela on utilise une variable logique "ram-transf" qui prend la valeur "vrai" dès qu'on a exécuté une règle sur la ramification donnée.

Le traitement des dérivations infinies et des boucles dans la dérivation se fait de façon similaire au paragraphe précédent au moyen d'un compteur du nombre de pas et d'un stockage des ramifications intermédiaires.

Dans ce cas la fonction "exécuter" qui exécutait une règle de transformation reconnue applicable a été modifiée, et elle est supposée exécuter la transformation directement sur la variable "ramification".

```
fonction transf-Ch.R (ramification) ;
     compteur := 0 ;
     boucle := "faux" ;
     ram-intermédiaires := nil ;
     tant que (trans(ramification) et compteur < limit-max et
           non boucle faire nil ;
     si compteur = limit-max alors traitement-dérivation-infinie
          sinon si boucle alors traitement boucle
          sinon rendre ramification fsi, fsi ;
fin fonction ;
```

```
fonction trans(ramification) ;
     ram-transf := "faux" ;
     niveau := 0 ; l-états-finals := nil ;
     liste-d'états := nil ;
     pour chaque arbre dans ramification faire
          si non boucle alors
             ram-transf := ram-transf ou transarb (arbre) ;
             fsi ;
     rendre ram-transf ;
fin fonction ;

fonction transarb (arbre) ;
     règle-applicable := vartemp1 := vartemp2 := temp := nil ;
     niveau := niveau + 1 ;
     si arbre est un noeud
          alors père := arbre
          sinon père := racine(arbre) ;
                fils := reste(arbre) ;
          fsi ;
     liste-d'états := <q0, niveau> . liste-d'états ;
     vartemp1 := liste-d'états ;
     vartemp2 := l-états-finals ;
etiq : sous-ram-transformée := "faux" ;
     liste-d'états := transitions (liste-d'états, père) ;
     si fils
          alors  pour chaque arb dans fils faire
                 si non boucle alors
                    sous-ram-transformée := sous-ram-transformée ou transarb(arb)
          fsi ; fsi ;
     si sous-ram-transformée
          alors liste-d'états := vartemp1 ;
                l-états-finals := vartemp2 ;
                allera etiq ; fsi ;
```

```
        pour chaque état final dans l-états-finals faire
            co état-final = <m, règle>  ;
            si m = niveau
                    alors règle-applicable := règle.règle-applicable ;
                    sinon temp := état-final.temp ;
                    fsi ;
        l-états-finals := temp ;
        tant que règle-applicable et not cond(prem(règle-applicable))
            faire règle-applicable := reste(règle-applicable) ;
        si règle-applicable
            alors exécuter (règle-applicable) ;
            sous-ram-transformée := "vraie" ;
            compteur := compteur + 1 ;
            si ramification ∈ ram-intermédiaires ou compteur ≥ limit max
               alors boucle := "vraie"
               sinon ram-intermédiaires := ramification. ram-intermédiaires
               fsi ;
            fsi ;
        niveau := niveau - 1 ;
        rendre sous-ram-transformée ;
fin fonction ;


fonction transitions (liste-d'états, noeud) ;
        états-result := nil ;
        pour chaque état dans liste-d'états faire
        co état :=  <q, m, règles> fco
        si m(état) < niveau alors états-result := état. états-résult
            sinon si m(état) = niveau alors
                    pour chaque état nouv dans δ(q(état), noeud) faire
                    si q(état nouv) = état-final alors
                        l-états-finals :=  <m(état-nouv) + niveau; règles (état)
                         ∩ règles (état-nouv)> . l-états-finals ;
```

```
                    sinon
                    états-résult :=  <q(état-nouv), m(état-nouv) + niveau,

                    règles (état-nouv) ∩ règles (état)> . états-résult
                fsi ; fsi ;
            fsi ;
        rendre  états-result ;
fin fonction ;
```

Par rapport aux algorithmes du paragraphe précédent, il est à noter qu'on exécutait une transformation immédiatement après avoir analysé le dernier noeud en préordre du domaine d'application ; dans celui-ci on a modifié les fonctions transition et transarb, et on a ajouté une liste d'états finaux pour lancer cette exécution au niveau de la racine (ou des racines) du domaine d'application. Comme il peut y avoir plusieurs règles applicables on exécute la première règle qui remplit la condition et un parcours de la ramification (occasionné par la variable logique "sous-ram-transformée") détermine s'il y a d'autres règles applicables.

II.3.3. SIP T-fermés

Les algorithmes nécessaires pour la manipulation de systèmes T-fermés sont de même nature que les algorithmes précédents. La seule différence majeure est que dans ce cas on n'a pas besoin de refaire le parcours d'une ramification car une fois qu'une règle est reconnue comme applicable elle le sera toujours malgré l'application des autres règles.

Quant à l'exécution d'une règle de transformation on peut, grâce au parcours en préordre, l'effectuer avant ou après l'analyse (et la transformation) des sous-ramifications dépendantes. Pour illustrer ce choix supposons que sur une ramification donnée A (fig. 2.3.3.a.) nous puissions exécuter les règles de transformation T1 et T2 sur les sous-ramifications D1 et D2 respectivement.

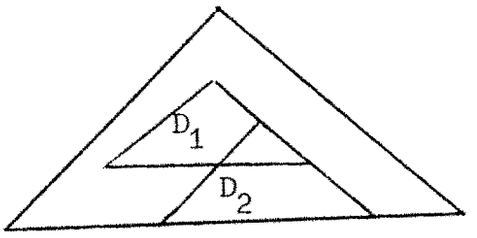


Fig. 2.3.3.a.

Si on exécute d'abord la règle T1, on risque de créer plusieurs copies de la sous-ramification D2, et le nombre total de transformations exécutées sera ainsi augmenté. En conséquence on analyse tout d'abord les fils d'un noeud, pour ensuite exécuter les transformations reconnues.

Les algorithmes de contrôle du paragraphe précédent peuvent être tous utilisés à l'exception de "transarb" que l'on doit modifier ainsi :

```
fonction transarb (arbre) ;
    règle-applicable := temp := nil ;
    niveau := niveau + 1 ;
    si arbre est un noeud
        alors père := arbre
        sinon père := racine (arbre) ;
              fils := reste (arbre) ;
        fsi ;
    liste-d'états := <q0, niveau> . liste-d'états ;
    liste-d'états := transitions (liste-d'états, père) ;
    si fils
        alors pour tout arb dans fils faire
              si non boucle
                    alors transarb(arb)
                    fsi ;
        fsi ;
```

CHAPITRE III

APPLICATIONS

III.1. LE SYSTEME PIAF : PROGRAMMES INTERACTIFS D'ANALYSE DU FRANCAIS 103

III.1.1. Le modèle de traitement de chaînes 104
III.1.2. Le modèle de dépendances 105
III.1.3. Le modèle de contrôle de variables 107
III.1.4. Le modèle de reconnaissance et transformation de ramifications 109
III.1.5. Le modèle d'évaluation sémantique. 111

III.2. UNE APPLICATION A L'INDEXATION AUTOMATIQUE 112

III.2.1. L'indexation automatique et le système PIAFDOC 112
III.2.2. Le traitement de groupes de mots 113
III.2.2.1. Mots-clés composés non-connexes. 113
III.2.2.2. La coordination 113
III.2.2.3. Relations avec un thésaurus 114
III.2.3. Utilisation des ARPC 114
III.2.3.1. Reconnaissance de sous-ramifications non-complètes 115
III.3.2.3.2. Traitement de la coordination 116
III.2.4. Exemples 117

III.3. UNE APPLICATION A L'ANALYSE DE LANGAGES FORMELS : LE TRAITEMENT DES REQUETES RELATIONNELLES ALGEBRIQUES DANS UNE BASE DE DONNEES REPARTIE 123

III.3.1. Le traitement de requêtes relationnelles 123
III.3.2. Définitions 124
III.3.3. La vérification de requêtes relationnelles 125

III.3.4. Optimisation statique des requêtes relationnelles 127
III.3.4.1. La décomposition des relations globales 127
III.3.4.2. Transformations de nature algébrique 128
III.3.4.3. La localisation des données 131
III.3.5. Exemples. 132

III.4. UNE APPLICATION AU TRAITEMENT AUTOMATIQUE DE LA LANGUE NATURELLE 138

III.4.1. Analyse sémantique 139
III.4.1.1. Modèle linguistique et modèle algorithmique 139
III.4.1.2. Signification d'une phrase 140
III.4.1.3. Ambiguïté syntaxique et ambiguïté sémantique 140
III.4.2. Utilisation des ARPC et des STP 141
III.4.2.1. Vérification et évaluation 142
III.4.2.2. Traitement de paraphrases 145
III.4.2.3. Résolution d'ambiguïtés sémantiques. 154

Les théories exposées dans les chapitres antérieurs ont donné lieu à l'élaboration d'un système de reconnaissance et transformation de ramifications paramétrées. Ce système fait partie du logiciel PIAF (Programmes Interactifs d'Analyse du Français)[1] développé au sein de l'équipe Algorithmique et Intelligence Artificielle à l'IMAG. Dans ce chapitre, nous faisons d'abord une description sommaire des différents modules qui composent le système PIAF et puis nous développons les diverses applications dans lesquelles le module de reconnaissance et transformation de ramifications intervient. Ces applications intéressent les domaines de la documentation automatique, des bases de données et de l'intelligence artificielle, et se rapportent au traitement des langages formels et des langues naturelles.

III.1. LE SYSTEME PIAF : Programmes Interactifs pour l'Analyse du Français

Le système PIAF, comme son nom l'indique, est un ensemble de programmes qui a pour but l'analyse du français à tous les niveaux : phonétique, morphologique, syntaxique et sémantique. Ce système a été conçu et élaboré par l'équipe "Algorithmique et Intelligence Artificielle de l'IMAG", dans le cadre de recherches orientées vers la définition d'outils logiciels généraux pour le traitement automatique assisté des langues. Les algorithmes qui sont réalisés dans le système sont entièrement paramétrés par les diverses informations linguistiques nécessaires à une application. Le fonctionnement en mode interactif permet la mise à jour et la modification en cours d'exécution de tous les paramètres linguistiques (dictionnaires, grammaires, chaînes d'entrée), ceci au moyen de compilateurs incrémentiels qui vérifient la cohérence des modifications faites.

---

[1] voir principalement : Courtin (1977) et les publications des membres de l'équipe : Chassagne, Chiaramella, Courtin, Dujardin, Grandjean, Joloboff, Mathieu et Veillon.

PIAF est composé de cinq modèles indépendants : i) un modèle de traitement de chaînes, (ii) un modèle d'analyse en dépendance, (iii) un modèle de contrôle des variables, (iv) un modèle de reconnaissance et de transformation de ramifications, et (v) un modèle d'évaluation sémantique.

III.1.1. Le modèle de traitement de chaînes

Le modèle de traitement de chaînes est un transducteur général d'états finis. Ses objectifs sont : (i) segmenter une chaîne et (ii) déterminer un certain nombre de renseignements sur les chaînes fournies par la segmentation, c'est-à-dire effectuer une transduction.

Dans le cas de l'utilisation de ce modèle pour l'analyse morphologique des phrases en français (ou une autre langue) la segmentation d'une phrase donne les mots ou groupes de mots qui la composent. Chaque mot pouvant être découpé en une base précédée ou non de préfixes, suivie ou non de suffices, et se terminant par une désinence ; la détermination d'un segment consiste donc,en sa décomposition en préfixe, base, suffixe et désinence. L'analyse des chaînes fournies par la segmentation détermine la catégorie du mot (verbe, subc, etc ...) et ses variables morphologiques (genre, nombre, etc ...).

Pour effectuer ces tâches le modèle dispose :

- d'un dictionnaire contenant des bases, des préfixes, des suffixes, des désinences et des formes composées. Chaque élément du dictionnaire fait référence à un autre élément du dictionnaire appelé modèle (le paradigme).

- d'une liste de modèles et de variables associées. Chaque modèle caractérise un sous-ensemble de mots du dictionnaire.

- d'une grammaire à validations et saturations [Courtin, 1977] (équivalente à une grammaire d'états finis) formée d'un ensemble de règles. Son rôle est infirmer ou confirmer la concaténation des constituants d'un segment et effectuer la transduction des variables fournies par les modèles des constituants.

Exemple :

Dans le dictionnaire on trouve :

/COUVENT/HOMME/
/COUV/AIM/
/ENT␣/ENT␣/

où "HOMME" est le modèle associé au substantif commun masculin, "AIM" représente la base des verbes du premier groupe, et "ENT" est un suffixe.

L'analyse de la chaîne "COUVENT␣" produit un segment avec les deux découpages possibles :

COUVENT - ␣ : SUBC, MAS, SIN
COUV - ENT ␣ : VERB, 3ème, PLU, (IND, SUB).

où des règles de la grammaire ont vérifié la concaténation des constituants, et ont produit les variables morphologiques associées.

Ce modèle étant très général permet non seulement d'effectuer l'analyse morphologique des langues mais également des tâches telles que : la transduction phonétique [Y. Chiaramella, 1976], l'indexation automatique [E. Grandjean, 1978], la traduction du français en Braille abrégé [B. Mathieu, 1978], ou la traduction de programmes [C. Chassagne, 1979].

III.1.2. L'analyseur de dépendances

En fonction des catégories syntaxiques obtenues par le modèle morphologique après l'analyse d'une phrase donnée, le modèle de dépendances a pour but la construction d'une ou de plusieurs structures de dépendance [D. Hays, 1964 ; Veillon, 1970 ; Courtin, 1977] de la phrase. Ces structures de dépendance sont construites à l'aide d'une grammaire de règles de dépendances établie par les linguistes . Chaque règle de la grammaire exprime une relation de dépendance entre deux catégories syntaxiques et la position relative de la catégorie dépendante par rapport aux autres catégories subordonnées à la catégorie donnée comme gouverneur.

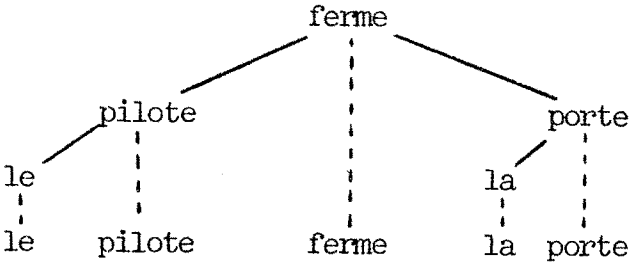
Chaque règle est écrite sous la forme :

gouverneur * dépendant := poids

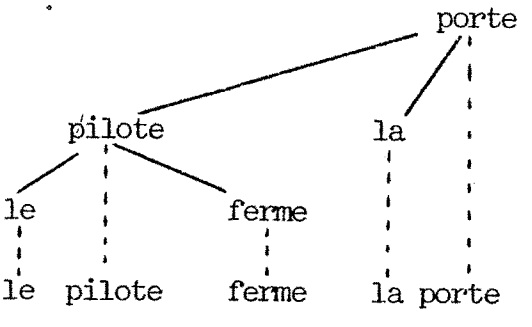
où "poids" est un ensemble d'entiers entre - 99 et 99 qui expriment la (ou les) position(s) relative(s) de la catégorie dépendante. Par exemple si nous avons l'ensemble de règles :

verb * subc := -32, 32 ;
verb * pron := -32, -16, -8, 1 ;
subc * art := - 32 ;
subc * adjq := -25, -24, -16, 16 ;

alors il y aura pour la phrase "le pilote ferme la porte" les structures de dépendances possibles suivantes :



et



La principale propriété des structures construites est leur pro - jectivité (voir figure). Formellement ceci veut dire que toute phrase est équivalente à l'écriture de sa (ou ses) structure(s) de dépendance dans l'ordre suivant : pour tout sommet, on écrit (i) d'abord ses fils

de droite, (ii) le sommet, et (iii) finalement ses fils de gauche. Ceci est considéré comme une restriction parce que pour certaines phrases comme "nous l'avons tous vu", on ne peut pas construire une structure dans laquelle "tous" soit dépendant de "nous" sans changer les relations de dépendance couramment établies (tout verbe gouverne son sujet). Malgré cela, la projectivité a pour avantage de simplifier l'algorithme qui construit les structures de dépendance.

Le modèle étant très général teste seulement les relations topologiques entre les mots d'une phrase, ce qui rend nécessaire l'utilisation du modèle de contrôle de variables (cf. ci-dessous) afin d'éliminer (entre autres problèmes) les phrases mal formées à cause de l'absence de mots, ou d'erreurs d'accord de variables grammaticales (genre, nombre, etc ...).

III.1.3. Le modèle de contrôle de variables

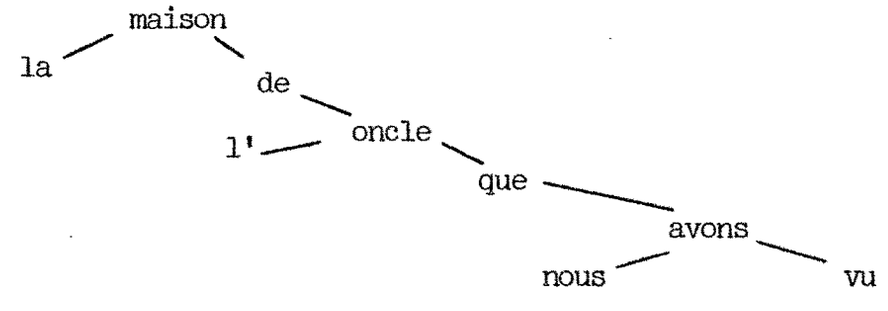
Etant donnée une sous-structure de dépendances, ce modèle a pour but de vérifier qu'elle correspond à une phrase bien formée. Ceci est fait grâce à un ensemble de règles qui permettent la manipulation de variables syntaxiques pour imposer la présence de certains mots (un sujet pour les verbes, un article pour les noms communs, etc ...) et de variables morphologiques pour vérifier les accords grammaticaux (accord genre-nombre entre un substantif et ses dépendants, accord de personne entre un verbe et un pronom, etc...). Le modèle ressemble aux grammaires d'attributs de Knuth [1968] où chaque règle est associée à une relation syntaxique et peut modifier les valeurs des variables des mots de la phrase. Le modèle de PIAF se distingue de celui de Knuth principalement par : (i) l'existence d'une condition associée à chaque règle ; si cette condition n'est pas satisfaite la relation est considérée non-correcte, et (ii) étant donnée une relation gouverneur-dépendant, la règle ne peut évaluer d'attributs que pour le mot gouverneur. Cette dernière condition impose un ordre d'application depuis les feuilles jusqu'à la racine, ce qui réduit la complexité générale de ce modèle, sans diminuer sa puissance (comme Knuth [1968] l'a démontré).

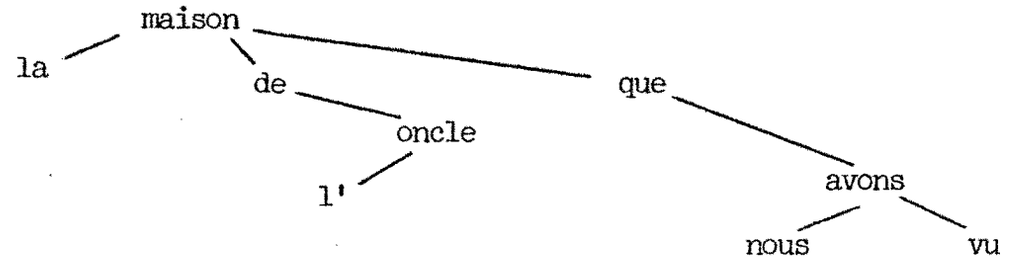
Exemple de règle :

```
VOO1 : GNOMO * VERB ==>  VERB
     si ¬SET(G) & ¬IMP(G) & ART(D) & WNBRR & WPRSS
        ALORS
         NBR := NBR(G).NBR(D) ; PRS := PRS(G).PRS(D) ; VBO := SET $
```

Cette règle est appliquée sur la relation groupe nominal-verbe lorsque le verbe est le gouverneur et que le groupe nominal dépend de lui à gauche. La règle peut être paraphrasée de la façon suivante : si le verbe n'a pas encore pris son sujet (variable SET) et n'est pas à la forme impérative, si le groupe nominal possède la variable ART, et s'il y a accord de variables en nombre (WNBRR) et personne (WPRSS), alors le nombre et la personne du verbe seront égaux à l'intersection des valeurs des variables respectives pour le verbe et le groupe nominal, et on indique que le verbe a un mot à la place de sujet.

Etant données les structures de dépendance d'une phrase quelconque, l'application de ces règles permet de rejeter certaines de ces structures. Par exemple, pour la phrase "la maison de l'oncle que nous avons vu" nous avons les structures suivantes :





La deuxième structure sera rejetée par ce modèle puisque l'accord de genre entre "maison" et "vu" n'est pas vérifié. Si par contre on a la phrase "la maison de l'oncle que nous avons vue", ce sera la structure semblable à la première qui sera rejetée.

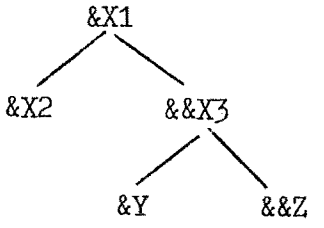
En ce qui concerne son exécution, le modèle de contrôle de variables peut être exécuté dès que l'analyse de dépendances a construit une sous-structure. Ceci permet de déceler les mauvaises structures dès qu'elles sont construites. Enfin, notons que, outre les contrôles grammaticaux, ce modèle peut aussi être utilisé pour l'évaluation d'attributs des sommets d'une structure arborescent quelconque (cf. III.4.2.1.).

III.1.4. Le modèle de reconnaissance et de transformation de ramifications

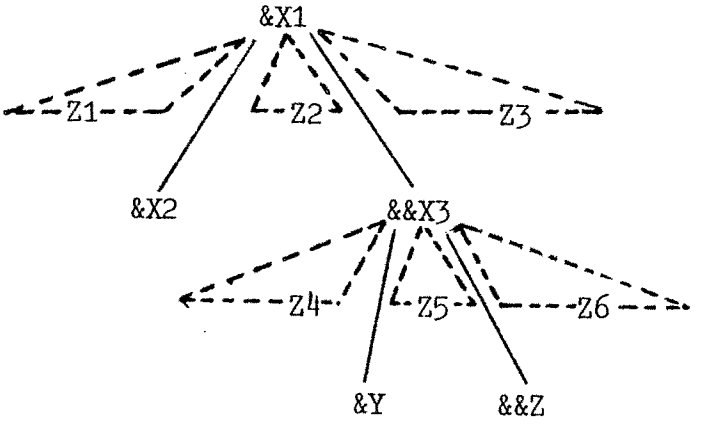
Etant donnée une structure de dépendances d'une phrase, le but de ce modèle est, soit de reconnaître une sous-ramification dans celle-ci, soit de lui appliquer un ensemble de règle de transformation. La reconnaissance de ramifications dans ce modèle est faite à l'aide des ARPC (§ II.2.3.).

Le système permet de définir plusieurs ensembles de règles de transformation indépendants (de façon interactive), puis son application séquentielle, dans un ordre donné par l'utilisateur, sur une structure de dépendances.

La description des applications de ce modèle, et quelques exemples d'application sont présentés dans ce chapitre. L'implémentation actuelle de ce modèle, qui a été réalisé avant le développement des théories exposées dans cette étude, diffère légèrement du modèle théorique exposé aux premiers chapitres. Aussi dans les ramifications paramétrées des sorties de l'ordinateur, on ne trouvera pas de paramètres de type ramification, mais on aura des paramètres de type sous-arborescence aux sommets non-feuilles. Ces paramètres permettent qu'au moment de tester l'équivalence avec une ramification étiquetée ses fils soient entourés de ramifications. La racine de toute arborescence paramétrée est considérée aussi comme appartenant à ce type de paramètres. Ainsi la ramification :



représente, en fait, la ramification paramétrée :



Ceci a été fait ainsi, principalement parce que nous voulions écrire les règles de transformation dans un langage simple, de plus, on se fondait sur le fait que les structures à transformer sont la sortie d'un modèle de dépendances et donc qu'elles sont structuralement correctes. Le problème qui nous a poussé à créer les paramètres de type ramification est le suivant : si on veut transformer la ramification donnée ci-dessus, on ne peut pas exprimer les manipulations à effectuer sur les ramifications Z1, ..., Z6.

On peut aussi signaler que le système a été conçu pour être exécuté en coroutine avec le modèle de dépendances. C'est pourquoi les ARPC parcourent les ramifications en préordre.

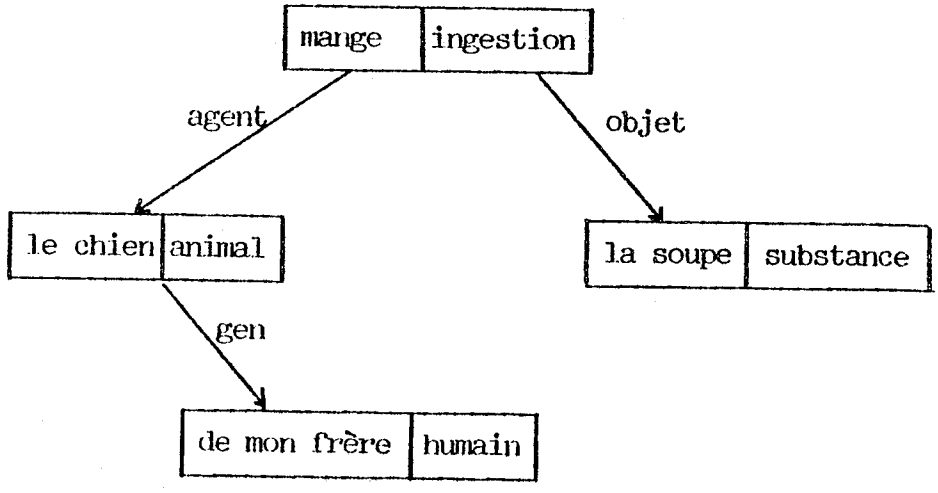
III.1.5. Le modèle d'évaluation sémantique (Joloboff, 1978)

Le but de ce modèle est d'établir une structure fonctionnelle d'une phrase donnée, c'est-à-dire de reconnaître dans la structure syntaxique la fonction des éléments de la phrase par rapport à un contexte défini. Ainsi on associe à une phrase une structure, correspondant à une grammaire de cas, (Tesnière, 1959 ; Fillmore, 1968) assez proche du texte original (ce qui s'avère assez utile dans le cas d'une application documentaire).

Ce modèle travaille sur des structures syntaxiques déjà normalisées : regroupement des groupes nominaux, détection de locutions composées (temps composés, verbes composés), Tous les mots du dictionnaire possèdent un ou plusieurs couples <modèle casuel, fonction sémantique> . Le modèle casuel est une structure arborescente qui exprime tous les actants possibles du mot, l'ordre dans lequel ils doivent se trouver dans la phrase, et certaines conditions (sous la forme de traits sémantiques) sur les actants. On voit là une généralisation de la notion de cas à toutes les classes syntaxiques.

Un algorithme d'unification trouve pour une sous-structure donnée le modèle casuel associé à sa racine. Une fois celui-ci trouvé, on analyse les sous-structures de la structure analysée, puis on procède à l'évaluation de la fonction sémantique. Le but de cette fonction est d'évaluer les attributs de la racine de la structure à partir des traits sémantiques des actants déterminés par le modèle casuel ; ainsi, on peut déterminer la signification d'un mot.

Par exemple, la phrase "le chien de mon frère mange la soupe" sera représentée par :



III.2. UNE APPLICATION A L'INDEXATION AUTOMATIQUE

Les recherches exposées dans ce paragraphe ont été effectuées autour du projet PIAFDOC [E. Grandjean 1977, 1978], [H. Pauchard, 1978]. Les expériences ont été réalisées sur une base documentaire du Centre de Documentation Française contenant des dépêches d'événements politiques ayant eu lieu en France. Ces dépêches sont évidemment rédigées en français.

III.2.1. L'indexation automatique et le système PIAFDOC

Le but principal des systèmes de recherche d'informations est la manipulation par l'ordinateur de grandes quantités de documents (en langue naturelle dans certains cas) de façon à pouvoir satisfaire des demandes concernant le contenu de ces documents. La démarche habituelle utilisée consiste à assigner à chaque document un ensemble de descripteurs de son contenu. Toute requête de l'utilisateur est analysée de façon analogue, et la comparaison entre les descripteurs de chaque document et ceux de la requête montrera si le document en question est concerné ou non par la requête. La construction automatique des descripteurs, que ce soit ceux du document ou ceux de la requête, est ce qu'on appelle l'indexation automatique.

A l'origine, dans les systèmes qui manipulaient des documents en langue naturelle, le choix des descripteurs d'un document était fait en extrayant les mots du document qui étaient supposés le mieux exprimer son contenu ; ces mots sont appelés les mots-clés du document. Nous utiliserons indistinctement les termes mot-clé et descripteur d'un texte. Les méthodes de sélection automatique des mots-clés d'un document sont principalement : (i) les méthodes statistiques [G. Salton, 1971 ; Andreewsky, Fluhr, 1975], et (ii) les méthodes lexicographiques. Dans le cadre du système PIAF, un logiciel d'aide à l'indexation automatique par des méthodes lexicographiques a été développé : le système PIAFDOC. Il utilise le transducteur général d'états finis (§ III.1.1.) et effectue en parallèle les opérations de contrôle de saisie et de sélection des descripteurs d'un texte qui peut être soit un document, soit une requête.

Les principales caractéristiques du traitement des mots-clés dans le système PIAFDOC sont : (i) la normalisation des formes morphologiques (les formes conjuguées d'un verbe, les variations en genre et nombre des substantifs, etc ...) ; (ii) la définition de mots-clés composés (par exemple : "Conférence de presse") ; (iii) le traitement des formes équivalentes ou synonymes (par exemple : "réunion de presse" et "conférence de presse") par la définition d'un synonyme préférentiel, et (iv) le traitement assisté de polysémies et homographies (mots qui ont plusieurs significations comme "président", qui peut être un verbe ou un substantif). Dans ce cas, le système propose les différentes possibilités et le documentaliste choisit la solution d'après le contexte.

III.2.2. Le traitement des groupes de mots

Le traitement des mots-clés composés dans le système PIAFDOC s'est avéré insuffisant et a permis de mettre en évidence certains problèmes relatifs à l'analyse des groupes de mots. La résolution de ces problèmes fait appel à des modèles plus complexes que ceux du traitement de chaînes, tels que l'analyse syntaxique et les algorithmes de reconnaissance de structures arborescentes.

III.2.2.1. Mots-clés composés non-connexes

Un groupe de mots définis dans le dictionnaire ne peut pas être reconnu tel quel s'il existe des mots qui peuvent s'y insérer. Par exemple, la notion documentaire "victime de guerre" doit être décelée dans le groupe nominal "les victimes de la première guerre mondiale".

III.2.2.2. La coordination

Un autre problème est la mise en facteur des mots-clés composés au moyen de la coordination ; par exemple dans l'expression "l'enseignement primaire et secondaire" où l'on souhaite retrouver les mots-clés "enseignement primaire" et "enseignement secondaire". Cette mise en facteur peut

avoir lieu au début des groupes de mots (comme dans l'exemple précédent), à la fin (par exemple "pluralité de l'information", et "indépendance de l'information" dans "pluralité et indépendance de l'information") ou être un peu plus complexe comme dans "pays producteurs de pétrole" et "pays consommateurs de pétrole" dans "pays procuteurs et consommateurs de pétrole".

III.2.2.3. Relations avec un thésaurus

Il peut s'avérer utile de permettre qu'un ou plusieurs membres d'un mot-clé composé soit une classe de mots. Ceci permettrait la reconnaissance et l'indexation d'une notion générique à partir d'une notion spécifique. Par exemple "aide au tiers monde" à partir de "aide financière au Congo", ou "travailleurs étrangers" à partir de "ouvriers portugais, espagnols ou maghrébin". Bien sûr, ceci implique l'existence d'un réseau, appelé thésaurus, dans lequel on spécifie l'appartenance d'un mot à une ou plusieurs classes conceptuelles (par exemple "Congo" doit être déclaré comme un "pays du tiers monde") et l'existence de relations entre ces classes, telles les notions de spécifique, générique ou synonyme.

III.2.3. Utilisation des ARPC

Les ARPC ont été conçus pour la reconnaissance de sous-ramifications paramétrées dans des ramification étiquetées.

La démarche à suivre est : (i) Définition de mots-clés composés au moyen de ramifications paramétrées et construction d'un ARPC avec ces ramifications. (ii) A chaque phrase proposée à l'indexation, le modèle de dépendances du système PIAF assignera une (ou plusieurs) structures de dépendances. (iii) Activation de l'ARPC construit au premier pas sur la (ou les) arborescence(s) donnée(s) par le pas précédent. Ainsi chaque état final atteint au cours de cette analyse est un mot-clé décelé dans la phrase proposée.

Il faut tout de même faire quelques modifications aux ARPC définis dans le chapitre antérieur. D'une part, ils ont été conçus pour la reconnaissance de sous-ramifications complètes et d'autre part la mise en facteur des ramifications par la coordination n'a pas été prévue.

En ce qui concerne la relation avec un thésaurus, l'utilisation de ramifications paramétrées dans lesquelles les noeuds peuvent être déterminés par une valeur quelconque faisant partie d'une étiquette est tout à fait adaptée à cette application.

III.2.3.1. Reconnaissance de sous-ramifications non complètes

Pour la reconnaissance de sous-ramifications non complètes, il suffit de modifier la fonction de transition généralisée définie au paragraphe II.2.3.1. La nouvelle fonction $\hat{f}$ est définie comme suit :

$\forall A \in V ; r,s \in \hat{V}$

$$\hat{f}(\langle q,m\rangle, n, A) = \langle q,m\rangle$$

$$\hat{f}(\langle q,m\rangle, n, A \times r + s) = \bigcup_{i \in \delta'_c(\langle q,m\rangle, n, A)} \hat{f}(\hat{f}(i, n+1, r), n+1, s) \cup \{\langle q', m'+n\rangle \mid \exists p, \langle q',m'\rangle \in \delta_c(q_0,p) \text{ et } p_2(a)\}$$

$$\text{et } \delta'_c(\langle q,m\rangle, n, A) = \underline{\text{si}}\ m < n\ \underline{\text{alors}}\ \{\langle q,m\rangle\}\ \underline{\text{sinon}}$$
$$\underline{\text{si}}\ m = n\ \text{alors}\ \{\langle q,m\rangle\} \cup \{\langle q', m'+n\rangle \mid \exists p, \langle q',m'\rangle \in \delta_c(q,p) \text{ et } p_2(a)\}$$
$$\underline{\text{sinon si}}\ q \in F\ \underline{\text{alors}}\ \{\langle q, -m\rangle\}$$
$$\underline{\text{sinon}}\ \phi$$

où la différence avec la définition du paragraphe II.2.3.1. a été soulignée.

III.2.3.2. Traitement de la coordination

Dans la grammaire de dépendances du système PIAF la représentation des groupes de mots dans lesquels on trouve une coordination est :

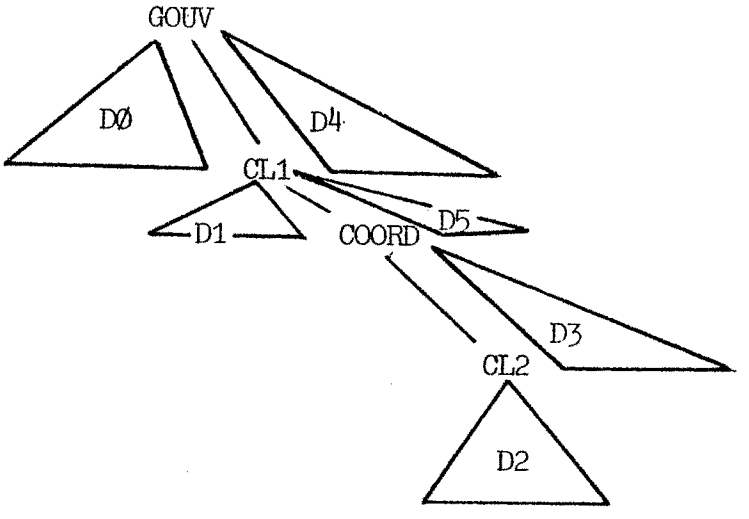


Figure 2.3.2.a.

où CL1 et CL2 sont les deux catégories lexicales connectées par la coordination (COORD) ; GOUV est le mot gouverneur du groupe et DØ, D1, D2, D3, D4 et D5 sont des ramifications quelconques (peut être vides). Il faut noter que GOUV, ainsi que DØ, et D4 sont des éléments optionnels et que dans ce cas le mot CL1 serait le gouverneur de la phrase.

Le parcours de l'arborescence de la figure 2.3.2.a. par un ARPC doit être équivalent (c'est-à-dire produire le même résultat, ou encore arriver aux mêmes états finals) au parcours de la ramification suivante :

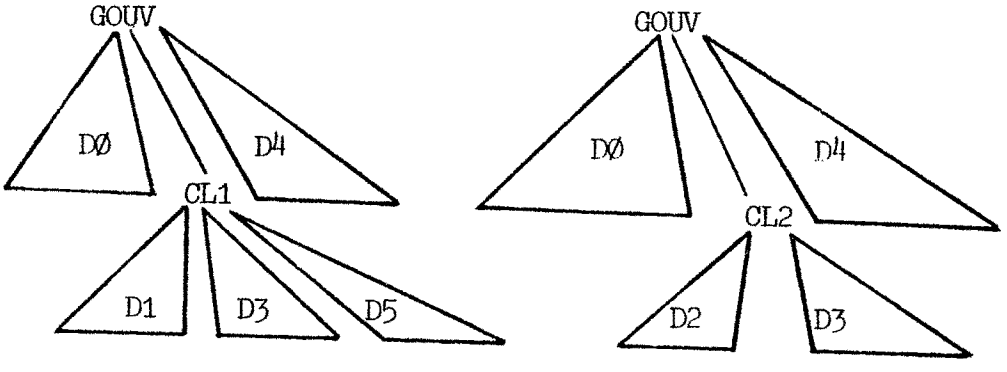


Figure 2.3.2.b.

Nous avons donc fait les modifications suivantes à l'ARPC :

- le mot CL2 doit être validé ou analysé avec la liste d'états utilisés pour l'analyse de CL1. Il faut donc stocker quelque part les listes d'états utilisées pour valider tous les ancêtres d'un noeud.

- D'après l'analyse du mot CL2 et de ses dépendants :

i) tous les fils du mot CL1 (la ramification D3) qu'on attendait au niveau de la coordination sont dans la phrase au niveau de CL2. Il faut donc générer à partir des états concernés (<q,m> où m = niveau de CL1) de nouveaux états.

ii) la ramification D3, attendue aussi au niveau des fils de CL2 se trouve au niveau de CL2.

et iii) la ramification D4 attendue au niveau de CL2 se trouve au niveau de CL1.

Il faut donc générer à partir des états concernés des nouveaux états qui expriment ces faits :

i) <q,m>, m = niveau de la coordination produit <q, m+1>

ii) <q,m>, m = niveau en dessous de CL2 devient <q, m-1>

iii) <q,m>, m = niveau de CL2 devient <q, m-2>

III.2.4. Exemples

Dans la suite on trouvera des exemples du fonctionnement du système interactif de reconnaissance de groupes de mots. Dans les sorties de l'ordinateur les textes en minuscules sont l'entrée donnée par l'utilisateur, et les textes en majuscules sont le résultat de l'exécution du système.

Tout d'abord on voit l'exécution de la fonction AJOUTER. Cette fonction permet de définir un groupe de mots et de lui associer un ou plusieurs mots clés comme synonymes préférentiels. Ce groupe de mots peut être donné au système soit sous la forme d'une ramification paramétrée (ARBRE), soit sous la forme d'un texte en français (PIAF). Dans ce dernier

cas les modules d'analyse morphologique et analyse de dépendances du système PIAF produisent une arborescence correspondant à la structure de dépendances du groupe de mots donné. Chaque arborescence ainsi fournie au système sera ajoutée à un ARPC construit au fur et à mesure que l'on déclare des groupes de mots.

```
 ajouter()
FORME DE LA PHRASE ? (ARBRE OU PIAF OU FIN)
piaf
?
ministre des affaires etrangeres.
ECRIVEZ LES MOTS CLES ASSOCIES (EN FORME DE LISTE):
==>
($$/ministre des affaires etrangeres/)
(MINISTRE DES AFFAIRES ETRANGERES)
D'ACCORD ? (OUI OU NON)
oui
FORME DE LA PHRASE ? (ARBRE OU PIAF OU FIN)
piaf
?
enseignement du premier degre.
ECRIVEZ LES MOTS CLES ASSOCIES (EN FORME DE LISTE):
==>
(44E
($$/enseignement primaire/)
(ENSEIGNEMENT PRIMAIRE)
D'ACCORD ? (OUI OU NON)
oui
FORME DE LA PHRASE ? (ARBRE OU PIAF OU FIN)
piaf
?
enseignement du second degre.
ECRIVEZ LES MOTS CLES ASSOCIES (EN FORME DE LISTE):
==>
($$/enseignement secondaire/)
(ENSEIGNEMENT SECONDAIRE)
D'ACCORD ? (OUI OU NON)
oui
```

Quand on veut définir un mot clé à l'aide d'une arborescence (ou une ramification), celle-ci est écrite linéairement en préordre par l'utilisateur . Le symbole "*" est utilisé pour séparer les dépendants à gauche d'un sommet de leurs dépendants à droite. Le système "dessine" la structure donnée par l'utilisateur avec les conventions suivantes : (i) les dépendants d'un sommet sont décalés à droite par rapport à lui, (ii) les dépendants à gauche sont écrits dans les lignes au-dessus de celle où est écrit son gouverneur et (iii) les dépendants à droite sont situés au-dessous.

L'utilisation de la forme arborescente permet de définir un groupe de mots à l'aide de variables morphologiques ou sémantiques (c'est la relation avec un thésaurus), et d'exprimer des variants linguistiques de la même tournure. Dans l'exemple qui suit la variable PTM est supposée distinguer les "pays du tiers monde".

```
FORME DE LA PHRASE ? (ARBRE OU PIAF OU FIN)
arbre
==>
(aide * ((&x1 a\ ou au) * (&x2 ptm)))
! AIDE-
        !
        ! &X1- (A\ OU AU)
                ! &X2- (PTM)

D'ACCORD ? (OUI OU NON)
oui
ECRIVEZ LES MOTS CLES ASSOCIES (EN FORME DE LISTE):
==>
($$/aide au tiers monde/)
(AIDE AU TIERS MONDE)
D'ACCORD ? (OUI OU NON)
oui
$FORME DE LA PHRASE ? (ARBRE OU PIAF OU FIN)
piaf
?
pays producteur de petrole.
ECRIVEZ LES MOTS CLES ASSOCIES (EN FORME DE LISTE):
==>
($$/pays producteur de petrole/)
(PAYS PRODUCTEUR DE PETROLE)
D'ACCORD ? (OUI OU NON)
oui
```

Le module d'analyse morphologique du système PIAF permet interactivement de corriger le texte d'entrée ou d'introduire des nouveaux mots dans le dictionnaire. Quand il rencontre une chaîne de caractères inconnue, elle est déclarée comme incorrecte à l'utilisateur. Ce dernier peut, soit l'ajouter au dictionnaire et déclarer son modèle morphologique, soit la corriger.

```
FORME DE LA PHRASE ? (ARBRE OU PIAF OU FIN)
piaf
?
pays conconmateur de petrole.
CHAINE INCORRECTE : CONCONMATEUR DE PETR
-
ortho
>
/conco/conso/
ECRIVEZ LES MOTS CLES ASSOCIES (EN FORME DE LISTE):
==>
($$/pays consonmateur de petrole/)
(PAYS CONSONNATEUR DE PETROLE)
D'ACCORD ? (OUI OU NON)
oui
FORME DE LA PHRASE ? (ARBRE OU PIAF OU FIN)
fin.@
FIN
```

L'analyse d'un texte pour la recherche de mots clés non connexes est effectuée par la fonction ANALYSEZ. Le texte donné subi des analyses morphologiques et syntaxiques pour obtenir une ou plusieurs structures de dépendances. Chaque structure est analysée par le système de reconnaissance (à l'aide de l'ARPC construit au moment de la déclaration de groupes de mots) qui délivre tous les mots clés reconnus.

Dans les exemples qui suivent on voit les problèmes évoqués précédemment : groupes de mots non-connexes, factorisation dûe à la conjonction, et utilisation des variables de type sémantique (remarquez que dans les variables du mot "CONGO" on trouve 'PTM').

```
>analysez()
PAS A PAS OU CONT ? (PAS OU CONT)
pas
?
Une aide militaire et financiere au Congo.
.        ! UNE- ((NUM) IDE FEM SIN)
! AIDE- ((SUBC) FEM SIN)
         !
         ! MILITAIR- ((ADJQ) MAS FEM SIN)
         !          !
         !          ! ET- ((COCO))
         !                    ! FINANCIER- ((ADJQ) MAS FEM SIN)
         !
         ! AU- ((PECN) MAS SIN)
                  ! CONGO- ((SUBP) PTM MAS SIN)

$AIDE AU TIERS MONDE
?
l'enseignement du premier et second degre.
.        ! L'- ((ARTD) MAS FEM SIN)
! ENSEIGNEMENT- ((SUBC) MAS SIN)
         !
         ! DU- ((PECN) PAT MAS SIN)
                  !
                  !          !
                  !          ! PREMIER- ((ADJQ) MAS SIN)
                  !          !          !
                  !          !          ! ET- ((COCO))
                  !          !                    ! SECOND- ((ADJQ) MAS S^
IN)
                  ! DEGRE- ((SUBC) MAS SIN)

ENSEIGNEMENT SECONDAIRE
ENSEIGNEMENT PRIMAIRE
?
```

```
?
Les pays producteurs et consommateurs de petrole.
$.        ! LES- ((ARTD) MAS FEM PLU)
! PAYS- ((SUBC) MAS SIN PLU)
          !
          ! PRODUCTEUR- ((ADJQ) MAS PLU)
                    !
                    ! ET- ((COCO))
                              ! CONSOMMATEUR- ((ADJQ) MAS PLU)
                              !
                              ! DE- ((PEPC))
                                        ! PETROLE- ((SUBC) MAS SIN)

.         ! LES- ((ARTD) MAS FEM PLU)
! PAYS- ((SUBC) MAS SIN PLU)
          !
          ! PRODUCTEUR- ((ADJQ) MAS PLU)
          !          !
          !          ! ET- ((COCO))
          !                    ! CONSOMMATEUR- ((ADJQ) MAS PLU)
          !
          ! DE- ((PEPC))
                    ! PETROLE- ((SUBC) MAS SIN)

PAYS CONSOMMATEUR DE PETROLE
PAYS PRODUCTEUR DE PETROLE
?
le ministre britannique des affaires etrangeres.
.         ! LE- ((ARTD) MAS SIN)
! MINISTRE- ((SUBC) MAS SIN)
          ! BRITANNIQUE- ((ADJQ) MAS SIN)
          !
          ! DES- ((PECN) IDE PAT MAS FEM PLU)
                    ! AFFAIRES ETRANGERES- ((SUBC) FEM PLU)

MINISTRE DES AFFAIRES ETRANGERES
?
fin.
FIN
```

III.3. UNE APPLICATION A L'ANALYSE DE LANGAGES FORMELS : Traitement des requêtes relationnelles algébriques dans une base de données répartie

Les idées exposées dans ce paragraphe ont donné lieu à l'élaboration d'un outil informatique utilisé actuellement dans le système de bases de données réparties POLYPHEME.

Henry Gally est chargé de cette application; avec le support théorique des travaux de J.Y. Caléca [1978] et G.T. Nguyen [1978].

Nous attirons l'attention du lecteur sur le fait que le but de cette section n'est pas d'établir un algorithme d'optimisation de requêtes relationnelles mais, à partir d'un ensemble de propositions, de donner aux spécialistes de bases de données une liste non limitative des utilisations possibles des ARPC dans le traitement des requêtes, et d'un point de vue plus général de présenter une application à l'analyse de langages formels.

III.3.1. Le traitement de requêtes relationnelles

Pour la manipulation de systèmes informatiques conçus pour être utilisés par des non-spécialistes il est toujours possible d'utiliser des langages artificiels. Certains de ces langages essaient de se rapprocher de la langue naturelle, d'autres restent à un stade formel et concis, mais malgré tout, dans la plupart des cas il existe des paraphrases syntaxiques (un énoncé peut être exprimé de plusieurs façons ou encore plusieurs énoncés correspondent au même ensemble d'opérations internes) et de paraphrases sémantiques (des énoncés différents produisent le même résultat, ou pour chaque énoncé, la machine exécute un ensemble d'opérations différentes convergeant finalement vers le même résultat).

Dans le cas des bases de données, des utilisateurs qui ne sont spécialisés ni dans la manipulation de bases relationnelles ni dans le langage d'expression des requêtes relationnelles (ou simplement qui ne connaissent pas l'implémentation du système) peuvent commettre des erreurs

de logique ou tout simplement avoir un temps de réponse plus long parce qu'ils ont exprimé leur requête différemment. D'autres part, si on se place dans le cadre d'une base de données répartie sur plusieurs machines il faut tenir compte de faits tels que : le coût de transport des données dans le réseau, la différence de rapidité entre certaines machines, ou la différence de coût entre différentes opérations équivalentes sur une même machine, etc .. Ce sont ces raisons entre autres qui exigent une analyse des requêtes exprimées par les utilisateurs. Cette analyse peut être schématisée en deux phases venant compléter les analyses lexicographiques et syntaxiques traditionnelles : d'abord une vérification dite "sémantique" pour déceler les erreurs logiques dans la requête, et ensuite une phase d'optimisation destinée à réduire le temps de réponse et les coûts d'exécution. Ce processus d'optimisation est composé d'une optimisation statique de la requête et d'une optimisation dynamique qui est faite en même temps que l'exécution de la requête. Dans les paragraphes qui suivent, nous étudierons les processus de vérification et d'optimisation statiques de la requête.

III.3.2. Définitions

On appelle relation locale un ensemble de n-uplets, où chaque valeur d'un n-uplet appartient à un autre ensemble déterminé par une chaîne de caractères appelée attribut.

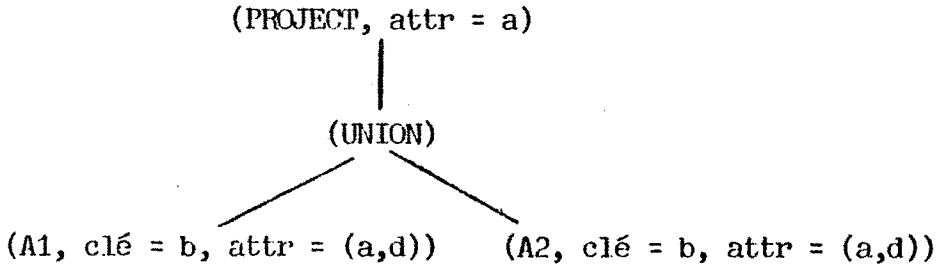
Par exemple, l'ensemble de 4-uplets (#emp, NOM, DEPARTEMENT, SALAIRE) forment la relation EMPLOYE. L'attribut #emp pour lequel il n'y a pas deux 4-uplets avec la même valeur est appelé clé de la relation.

Une requête relationnelle algébrique [Codd, 1970] est une expression qui comporte des opérateurs relationnels (unaires ou binaires en général) et des relations locales ou globales (cf. ci-dessous).

Le formalisme relationnel permet la définition de nouvelles relations, appelées relations globales, par application d'opérateurs relationnels sur des relations locales et/ou globales. Cette définition reste au niveau lexical ; toute relation globale a donc d'un point de vue physique un cardinal NUL.

Toute requête relationnelle peut être exprimée comme une pseudo-arborescence sur un vocabulaire étiqueté V, tel que dans toute étiquette de V il existe un élément particulier qui est, soit un opérateur relationnel, soit le nom d'une relation.

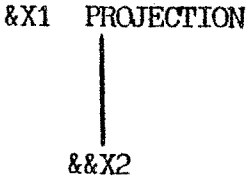
Par exemple :



où PROJECT est l'opérateur unaire "projection" (création d'une relation m-aire à partir d'une relation n-aire, avec m inférieur à n) ; UNION est l'opérateur binaire d'union de deux relations n-aires. ; et A1 et A2 sont deux relations d'attributs a et d et de clé b.

III.3.3. La vérification de requêtes relationnelles

La sémantique de requêtes relationnelles étant relativement simple, le but de cette vérification est principalement de déceler des erreurs de frappe qui auraient échappé aux contrôles lexicographiques et syntaxiques qui doivent précéder cette analyse. Un exemple élémentaire pour illustrer les vérifications qui doivent être faites est la contrainte suivante : si on fait la projection d'une relation R sur une liste d'attributs X il faut que la liste X soit strictement incluse dans la liste d'attributs de la relation R. Ceci peut être exprimé en termes d'une arborescence paramétrée et d'une condition à vérifier, par exemple :



VERIFIER : ATTRIBUTS (&X1) ⊂ ATTR(&&X2)

Eventuellement, il se peut que l'application d'une règle de vérification soit conditionnée par des relations entre les étiquettes des noeuds de l'arborescence paramétrée et/ou qu'en plus de la vérification on veuille faire des évaluations comme, dans cet exemple, l'assignation d'une CLE au noeud PROJECTION.

```
EVALUER : SI(CLE(&&X2) ⊂ ATTRIBUTS(&X1))
                ALORS CLE(&X1) := CLE(&&X2) ;
```

D'une façon générale on peut définir une règle de vérification comme suit :

Définition III.3.3.

Une règle de vérification RV_i est un quadruplet $RV_i = \langle X_i, C_i, V_i, A_i \rangle$ où :

X_i est une ramification paramétrée, $X_i \in RP$, $X_i \neq \Lambda$

C_i et V_i, sont des prédicats logiques portant sur les paramètres dans X_i.

A_i est une expression qui porte sur les étiquettes déterminées par X_i, ce sont les 'effets secondaires'.

La ramification paramétrée X_i et l'ensemble de conditions C_i déterminent les arborescences sur lesquelles on peut appliquer la règle RV_i. X_i spécifie la structure arborescente $S \in \hat{V}$ sur laquelle on appliquera la règle, et C_i est utilisée pour exprimer des relations entre les étiquettes des noeuds déterminés par X_i. V_i est interprété comme une condition que doit remplir S pour être considérée comme correcte. Si la condition V_i n'est pas vérifiée par la structure S, celle-ci sera rejetée ; dans le cas contraire on procède à l'évaluation des effets secondaires A_i.

L'utilisation des ARPC laisse le choix, comme dans le cas de règles de transformation (§ 2.2.3), du moment où la vérification doit être faite (c'est-à-dire l'évaluation de l'expression V_i) : soit immédiatement après avoir reconnu un schéma (X_i, C_i) ; soit après l'analyse des sous-arborescences contenues dans ce même schéma.

Compte tenu du fait que l'expression A_i modifie les étiquettes des noeuds déterminés par les règles, il est préférable de faire les vérifications et d'exécuter les actions correspondantes après l'analyse des dites sous-arborescences.

III.3.4. Optimisation statique des requêtes relationnelles

L'utilisation de transformations d'arborescences pour l'optimisation de requêtes relationnelles, a été déjà proposée par P.A.V. Hall [1975], J.M. Smith et P.Y.T. Chang [1975], J.Y. Caleca [1978], entre autres. Ces travaux se sont intéressés principalement à la définition de règles de transformation et à leur ordre d'application dans le cadre des systèmes sur lesquels ils ont travaillé. En ce qui concerne le système de transformation d'arborescences, ces travaux proposent l'utilisation des procédures spécialisées. Les avantages de l'utilisation des ARPC pour la transformation d'arborescences par rapport aux procédures spécialisées sont les suivants : (i) la séparation des données et des algorithmes donne plus de souplesse au système : les modifications sur les règles de transformation n'entraînent pas de coûts additionnels (recompilation du système) ; (ii) le traitement automatique des règles de transformation permet d'effectuer des factorisations qui, bien qu'il soit possible de les faire à la main sur des procédures spécialisées, compliquent la mise au point de ces procédures.

Les principales opérations effectuées sur une requête relationnelle dans cette phase d'optimisation statique sont décrites en tenant compte des points suivants : (i) la décomposition de relations globales, (ii) les propriétés algébriques des opérateurs relationnels, (iii) la localisation de données, et (iv) l'existence de sous-expressions équivalentes dans la requête. Les ARPC peuvent être utilisés pour les trois premières opérations et la quatrième appelle l'exécution d'une procédure spécialisée.

III.3.4.1. La décomposition des relations globales

Une relation globale étant définie par une expression relationnelle, on appelle décomposition le remplacement de toute relation globale dans une requête par sa définition. Ceci doit être le premier pas dans l'optimisation de la requête, afin de pouvoir réduire l'expression dans les phases ultérieures. Cette décomposition peut s'exprimer, dans un souci

d'homogénéité, comme une règle de transformation $<X_i, Y_i, C_i, A_i>$ où X_i est un sommet, la relation globale, et Y_i est sa définition. C_i et A_i sont nuls.

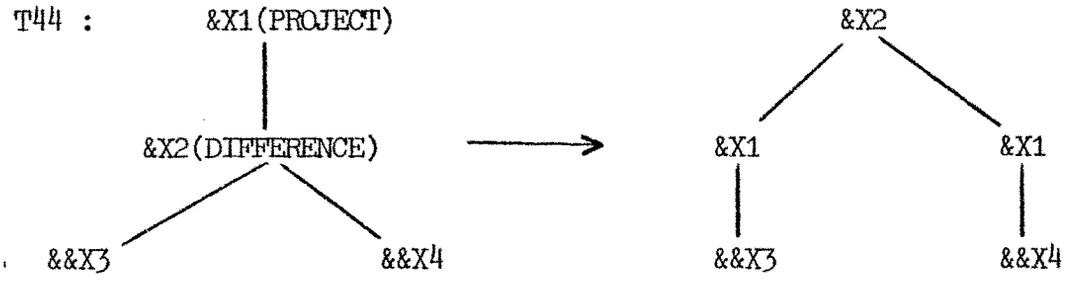
III.3.4.2. Transformations de nature algébrique

Dans le but de normaliser une requête relationnelle et de diminuer son temps d'exécution, des règles de transformation peuvent être construites en utilisant les propriétés algébriques des opérateurs relationnels. J.Y. Caleca [1978] propose un ensemble de telles transformations tout à fait adapté au formalisme des transformations paramétrées. Cet ensemble de transformations (auquel le lecteur peut se référer : § 3.2. [Caleca, 1978]) est divisé en cinq sous-ensembles ; leur application doit se faire séquentiellement pour obtenir de bons résultats. Si les ARPC sont utilisés, ces sous-ensembles sont au nombre de deux, puisque les sous-ensembles (2), (3), (4) et (5) peuvent s'exécuter simultanément. Ceci implique que les transformations comme l'élimination d'opérations inutiles sont exécutées dès leur création.

Puisque les ARPC laissent un choix concernant le moment d'application d'une transformation (avant ou après la transformation de sous-ramifications incluses dans le domaine d'application) il faut faire certaines remarques sur la façon dont nous les avons utilisés :

Parmi les transformations proposées par Caleca on peut donner les suivantes :

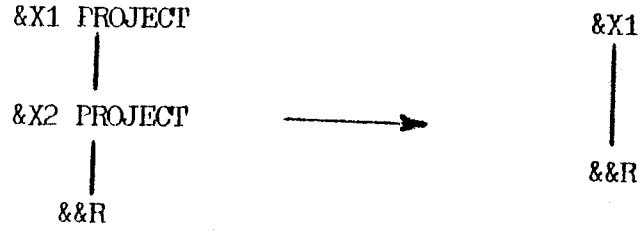
T44 :



C_0 : [ATTR(&X1) ⊃ CLE(&&X3)] et [CLE(&&X4) ⊂ ATTR(&X1)]

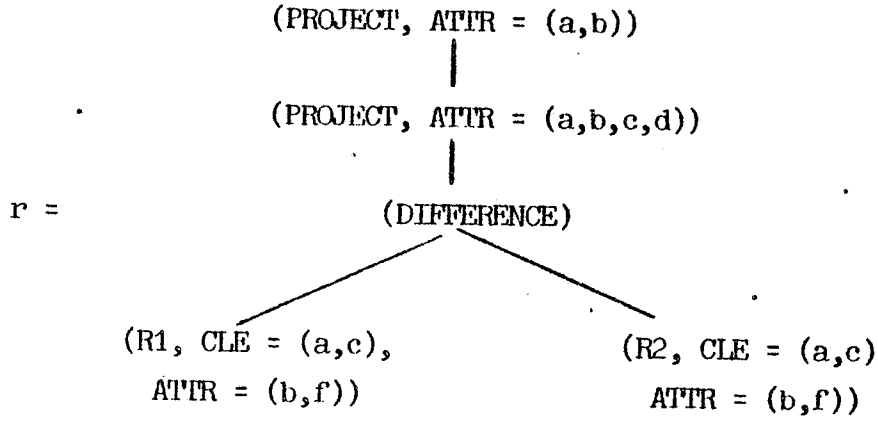
A_0 : ()

T'5 :

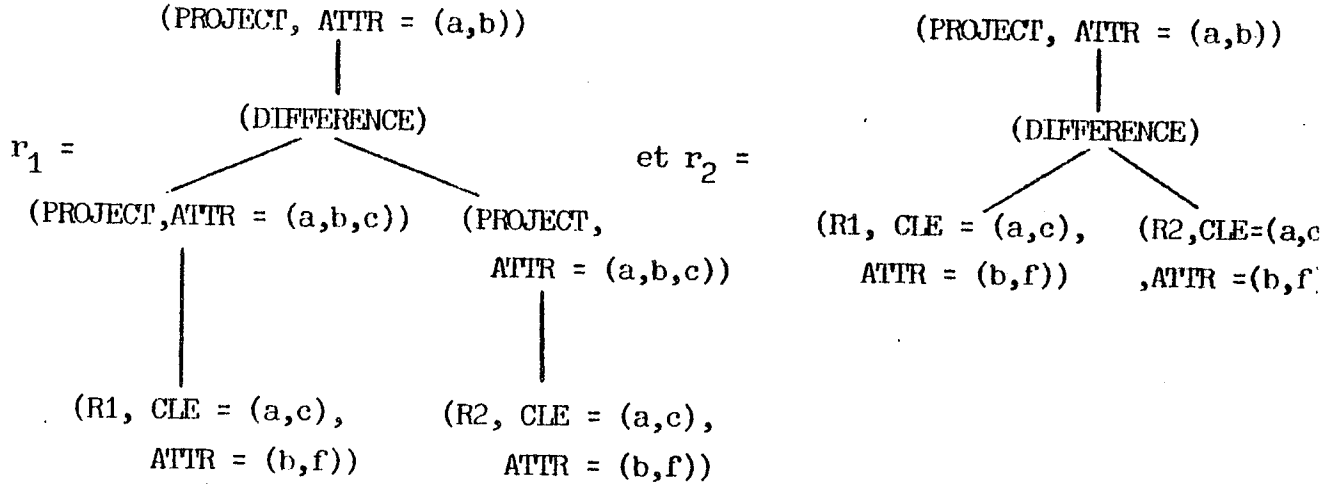


C_i : ()
A_i : ()

L'application de ces transformations sur l'arborescence r :



produit les arborescences r_1 et r_2 :



Donc tout STP qui comporte les règles T44 et T45 n'est pas T-fermé (lemme I.3.1) et s'il n'y a pas deux règles qui transforment r_1 et r_2 vers la même arborescence, il ne possède pas la propriété Church-Rosser.

Rappelons-nous que les algorithmes de contrôle des STP Church-Rosser et T-fermés (§ II.3.2 et § II.3.3.) peuvent être utilisés pour le traitement de STP non Church-Rosser, mais lors de leur application sur une ramification on obtiendra une seule ramification parmi celles qui résulteraient du processus transformationnel non Church-Rosser.

Bien que les ramifications r_1 et r_2 soient équivalentes en ce qui concerne le résultat obtenu, suivant l'implémentation du système relationnel qui exécute la requête, l'une est préférable à l'autre. On peut envisager les solutions suivantes :

i) Diviser l'ensemble des règles qui contient T44 et T45 en deux ensembles (l'un contenant T44, l'autre T45) qui auront la propriété Church-Rosser. L'exécution séquentielle de ces ensembles de règles sur r délivre comme résultat r_1 ou r_2 selon l'ordre d'application. Mais, si l'ensemble qui est exécuté en deuxième lieu est susceptible de créer une ramification sur laquelle une règle du premier ensemble soit applicable, il faudra exécuter le premier encore une fois et ainsi successivement jusqu'à ce qu'on ne puisse plus appliquer aucune règle.

ii) Une solution plus simple consiste à utiliser les propriétés topologiques des règles T44 et T45 pour établir une priorité d'application entre elles..Par exemple, si on veut obtenir la pseudo-arborescence r_2, il faudra donner une priorité d'application à la règle T45 par rapport à la règle T44. Puisque le domaine d'application de la règle T44 se trouve toujours inclus dans le domaine d'application de la règle T45, cette priorité peut être obtenue en appliquant la règle T45 avant d'analyser les sous-ramifications incluses dans son domaine d'application. Réciproquement, si on veut donner une priorité à la règle T44, l'exécution de toutes les règles après l'analyse des sous-ramifications incluses produit le résultat désiré, mais cela implique aussi que la règle T45 soit reconnue comme applicable dans certains cas, mais qu'après elle ne le soit plus ; il faut donc utiliser l'algorithme pour STP Church-Rosser.

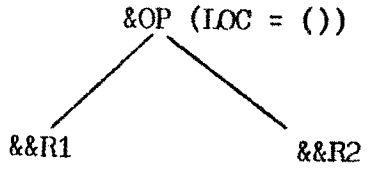
III.3.4.3. La localisation des données

Dans un système de bases de données réparties toute relation locale possède une caractéristique qu'il faut prendre en considération : sa localisation, c'est-à-dire l'endroit où elle est située physiquement. Dans une requête relationnelle déjà normalisée par les phases antérieures, il faut assigner à chaque opérateur relationnel le site où l'opération doit être effectuée, dans le but de réduire le transfert de données sur le réseau. Ceci peut être fait moyennant des règles d'évaluation :

Définition III.3.4.3.

> Une règle d'évaluation RE_i est un triplet $<X_i, C_i, A_i>$ où X_i est une ramification paramétrée, C_i est un prédicat logique et A_i est une expression portant sur les étiquettes déterminées par X_i.

Par exemple :



C : [LOC(&&R1) ∩ LOC(&&R2)] ≠ 0

A : LOC(&OP) = [LOC(&&R1) ∩ LOC(&&R2)]

Cette règle d'évaluation assigne à tout opérateur binaire la localisation de ses opérandes si ceux-ci sont localisés sur le même site.

De cette façon, on pourrait aussi exprimer, dans le cas d'une optimisation statique, l'estimation de la cardinalité des relations globales et en tenir compte pour la localisation.

III.3.5. Exemples

Dans la suite on trouvera des exemples concernant cette application. Dans les sorties de l'ordinateur les textes en minuscule sont l'entrée donnée par l'utilisateur, et les textes en majuscule sont le résultat de l'exécution du système interactif.

Le système permet la déclaration en mode interactif d'un ensemble de règles de transformation au moyen de la fonction AJOUTER. Cette fonction a comme argument le nom du fichier auquel on 'ajoute' les règles définies. Elle demande à l'utilisateur les différents éléments qui composent une règle et réécrit tout ce qui est donné par lui. Si on donne au système la chaîne 'ERROR' on peut corriger le dernier élément donné.

Les arborescences sont écrites par l'utilisateur linéairement en préordre, et dessinées par le système de façon telle que tous les fils d'un sommet sont immédiatement au-dessous de lui, décalés à droite.

```
NIL
>ajouter(base_donn)
ECRIVEZ LE NOM  DE LA REGLE
==>
proj_union
PROJ_UNION
ECRIVEZ LA PARTIE GAUCHE  DE LA REGLE
==>
((projety@er)((unionE
error


ECRIVEZ LE NOM  DE LA REGLE
==>
proj_union
PROJ_UNION
ECRIVEZ LA PARTIE GAUCHE  DE LA REGLE
==>
((&x1 projeter )((&x2 union)(&&x3)(&&x4)))
! &X1- (PROJETER)
         !
         ! &X2- (UNION)
                  ! &&X3-
                  ! &&X4-
```

```
ECRIVEZ LA PARTIE DROITE  DE LA REGLE
==>
(&x2 (&x1 &&x3)(&x1 &&x4))
! &X2-
         !
         ! &X1-
         !          ! &&X3-
         !
         ! &X1-
                    ! &&X4-
ECRIVEZ LA CONDITION  DE LA REGLE
==>
()
NIL
ECRIVEZ LES EFFETS DE BORD  DE LA REGLE
==>
()
NIL
ECRIVEZ LE NOM  DE LA REGLE
==>
proj_select
PROJ_SELECT
ECRIVEZ LA PARTIE GAUCHE  DE LA REGLE
==>
((&x1 projeter)((&x2 selecter)(&&x3)))
! &X1- (PROJETER)
         !
         ! &X2- (SELECTER)
                    ! &&X3-
ECRIVEZ LA PARTIE DROITE  DE LA REGLE
==>
(&x2 (&x1 &&x3))
! &X2-
         !
         ! &X1-
                    ! &&X3-
ECRIVEZ LA CONDITION  DE LA REGLE
==>
(atr (&x2)< atr(&x1))
(ATR (&X2) < ATR (&X1))
ECRIVEZ LES EFFETS DE BORD  DE LA REGLE
==>
()
NIL
ECRIVEZ LE NOM  DE LA REGLE
```

```
ECRIVEZ LE NOM  DE LA REGLE
==>
proj_join
PROJ_JOIN
ECRIVEZ LA PARTIE GAUCHE  DE LA REGLE
==>
((&x1 projeter)((&x2 join)(&&x3)(&&x4)))
| &X1- (PROJETER)
         |
         | &X2- (JOIN)
                  | &&X3-
                  | &&X4-
ECRIVEZ LA PARTIE DROITE  DE LA REGLE
==>
(&x5 (&x1 &&x3)(&x1 &&x4))
| &X5-
         |
         | &X1-
         |          | &&X3-
         |
         | &X1-
                  | &&X4-
ECRIVEZ LA CONDITION  DE LA REGLE
==>
(ou (atr &x1 = atr1 &x2)(atr &x1 = atr2 &x2))
(OU (ATR &X1 = ATR1 &X2) (ATR &X1 = ATR2 &X2))
ECRIVEZ LES EFFETS DE BORD  DE LA REGLE
==>
(si (type &x2 = (ujoin)) alors (nom &x5 = union) sinon
(si (type &x2 = (Ijoin)) alors (nom &x5 = inter))  )
(SI (TYPE &X2 = (UJOIN)) ALORS (NOM &X5 = UNION) SINON (SI (TYPE &X2 = (IJOIN)) ALORS (NOM &X
))
ECRIVEZ LE NOM  DE LA REGLE
==>
fin
FIN
FIN
>
```

L'exécution du système de transformation est activée par l'appel de la fonction ANALYSEZ. Elle a comme argument soit un ensemble de règles, soit une liste d'ensembles de règles qui seront appliquées séquentiellement (chacun sur le résultat délivré par l'ensemble précédent) sur les arborescences données par l'utilisateur. La fonction permet l'impression de l'arborescence donnée chaque fois qu'une transformation lui est appliquée (mode PAS à PAS), ou l'impression du résultat uniquement (mode CONTinu). Les arborescences à transformer peuvent être données soit au terminal, soit par l'intermédiaire d'un fichier. Pour faciliter l'écriture des requêtes, le système permet la déclaration des relations et leurs attributs.

```
NIL
>analysez(base_donn)
PAS A PAS OU CONT ? (PAS OU CONT)
pas
DONNEES ?  (TERM OU NOM DE FICHIER OU FIN)
term
=>
(decl r5 cle(a b ) atr(c d e))
=>
(decl r6 cle(x y) atr (s t u))
=>
((projeter)((union)(r5)(r6)))

! PROJETER-
          !
          ! UNION-
                    ! R5- CLE=(A B);ATR=(C D E);
                    ! R6- CLE=(X Y);ATR=(S T U);
D'ACCORD?
oui
PROJ_UNION
(UNION (PROJETER R5) (PROJETER R6))


! UNION-
          !
          ! PROJETER-
          !          ! R5- CLE=(A B);ATR=(C D E);
          !
          ! PROJETER-
                    ! R6- CLE=(X Y);ATR=(S T U);
```

```
=>
((projeter)((union)((projeter)((union)(r5)(rx )))(r6)))

| PROJETER-
          |
          | UNION-
                    |
                    | PROJETER-
                    |          |
                    |          | UNION-
                    |                    | R5- CLE=(A B);ATR=(C D E);
                    |                    | RX-
                    |
                    | R6- CLE=(X Y);ATR=(S T U);
D'ACCORD?
oui
*PROJ_UNION
(PROJETER (UNION (UNION (PROJETER R5) (PROJETER RX)) R6))
PROJ_UNION
(UNION (PROJETER (UNION (PROJETER R5) (PROJETER RX))) (PROJETER R6))
PROJ_UNION
(UNION (UNION (PROJETER (PROJETER R5)) (PROJETER (PROJETER RX))) (PROJETER R6))


| UNION-
          |
          | UNION-
          |          |
          |          | PROJETER-
          |          |          |
          |          |          | PROJETER-
          |          |                    | R5- CLE=(A B);ATR=(C D E);
          |          |
          |          | PROJETER-
          |                     |
          |                     | PROJETER-
          |                               | RX-
          |
          | PROJETER-
                    | R6- CLE=(X Y);ATR=(S T U);
=>
fin
FIN
```

```
>analysez(base_donn)
PAS A PAS OU CONT ? (PAS OU CONT)
pas
DONNEES ?  (TERM OU NOM DE FICHIER OU FIN)
prbase

! PROJETER- ATR=(X Y);
          !
          ! JOIN- ATR2=(X Y);ATR1=(A B);TYPE=(IJOIN);
                  ! R5- CLE=(A B);ATR=(C D E);
                  ! R6- CLE=(X Y);ATR=(S T U);
PROJ_JOIN
(INTER (PROJETER R5) (PROJETER R6))


! INTER-
          !
          ! PROJETER- ATR=(X Y);
          !          ! R5- CLE=(A B);ATR=(C D E);
          !
          ! PROJETER- ATR=(X Y);
                     ! R6- CLE=(X Y);ATR=(S T U);


! PROJETER- ATR=(A B);
          !
          ! JOIN- ATR2=(X Y);ATR1=(A B);TYPE=(UJOIN);
                  ! R5- CLE=(A B);ATR=(C D E);
                  ! R6- CLE=(X Y);ATR=(S T U);
PROJ_JOIN
(UNION (PROJETER R5) (PROJETER R6))


! UNION-
          !
          ! PROJETER- ATR=(A B);
          !          ! R5- CLE=(A B);ATR=(C D E);
          !
          ! PROJETER- ATR=(A B);
                     ! R6- CLE=(X Y);ATR=(S T U);

DONNEES ?  (TERM OU NOM DE FICHIER OU FIN)
fin
FIN
>fin()
```

III.4. UNE APPLICATION AU TRAITEMENT AUTOMATIQUE DE LA LANGUE NATURELLE

Les STP et les ARPC sont utilisés actuellement dans l'élaboration d'un interface de communication avec le système de conception assistée par ordinateur TROPIC (M. LOPEZ, 1979). TROPIC (Latombe, 1977) est essentiellement un système de résolution de problèmes dont la fonction est de construire le modèle d'un appareil satisfaisant un ensemble de conditions données. La fonction de l'interface est de permettre la description dans un sous-ensemble du français de toutes les connaissances nécessaires au système TROPIC.

L'interface est composé de trois étapes principales :
dans la première, on effectue l'analyse morphologique et syntaxique de la phrase traitée et on délivre une ou plusieurs structures de dépendances. La deuxième étape, où interviennent les STP et les ARPC, consiste à valider ces structures et à les transformer en une expression appelée forme réduite, représentation de la signification de la phrase dans le contexte déterminé par TROPIC.

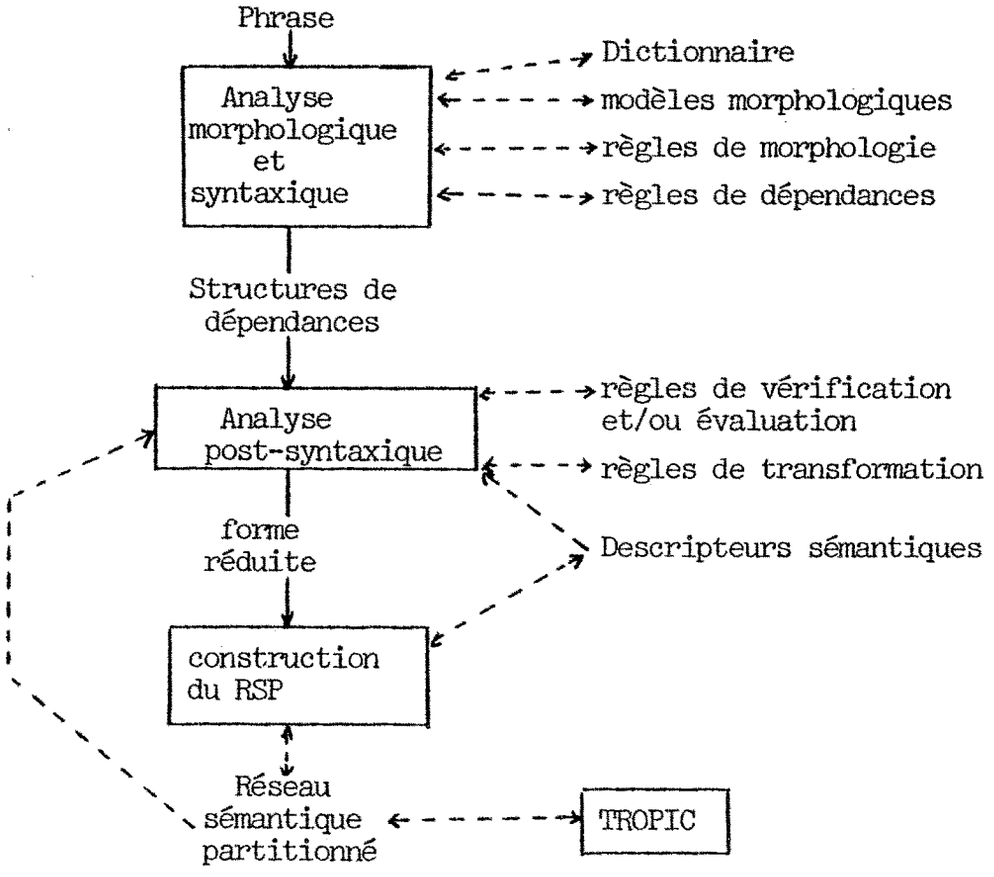


FIG. III.4.

Dans la troisième étape, les nouvelles connaissances apportées par la forme réduite sont incorporées dans un "réseau sémantique partitionné" (Hendrix, 1975), RSP, qui représente toutes les connaissances du système.

Dans ces paragraphes, nous exposerons les différentes opérations qui constituent la phase d'analyse post-syntaxique d'un phrase.

III.4.1. Analyse sémantique

Un langage est un système de corrélations entre forme et signification. Dans la description d'un langage on insiste généralement sur son aspect formel en oubliant la relation phrase-signification. Pour la description de l'aspect formel, on trouve des grammaires morphologiques ou lexicographiques (qui décrivent les éléments d'une phrase), et des grammaires syntaxiques (qui décrivent l'ensemble de phrases du langage). L'exécution d'algorithmes d'analyse lexicale et syntaxique à partir de ces données fournissent la (ou les) structure(s) syntaxique(s) d'une phrase donnée.

On appelle analyse sémantique ou post-syntaxique (d'après Katz et Fodor (1963) : 'semantics is linguistic description minus grammar') l'ensemble des procédés utilisés pour trouver la signification d'une phrase. Ceci suppose la définition d'une représentation de la signification. Dans certains systèmes d'analyse sémantique (Winograd 1971, Woods et al. 1976) la relation existant entre la forme et la signification se trouve dans les procédures de transduction d'une phrase vers sa signification. L'utilisation de systèmes de transformations pour cette analyse (le système du GETA[1] ; de Remer 1974 ; Partee 1975), et les grammaires d'attributs (Knuth 1968 ; Petrick 1973) sont des propositions de formalisation de la relation forme-signification sous la forme d'une grammaire.

III.4.1.1. Modèle linguistique et modèle algorithmique

Tout système de traitement de la langue naturelle est composé de deux modèles indépendants, l'un relevant de la linguistique, l'autre de l'algorithmique. Le premier est un modèle logique qui détermine la représentation de la signification d'une phrase ; il reflète une théorie linguistique.

---

[1] Veillon & Veyrunes & Vauquois, 1967 ; Chauché, 1974 ; Boitet, 1977.

Le deuxième, le modèle algorithmique, est constitué par l'ensemble des techniques utilisées pour obtenir par des moyens informatiques, la représentation fixée par le modèle linguistique.

Si la structure syntaxique d'une phrase et la représentation de sa signification sont sous forme de ramifications, le modèle algorithmique peut consister en un ou plusieurs STP. Ceci indépendamment des théories sémantiques utilisées par le modèle linguistique. Or le fait qu'on ne connaît pas d'algorithme de reconnaissance d'un sous-graphe dans un autre qui ne soit pas de complexité exponentielle, et d'autres facteurs d'efficacité font que l'on cherche à utiliser le plus souvent des représentations arborescentes.

III.4.1.2. Signification d'une phrase

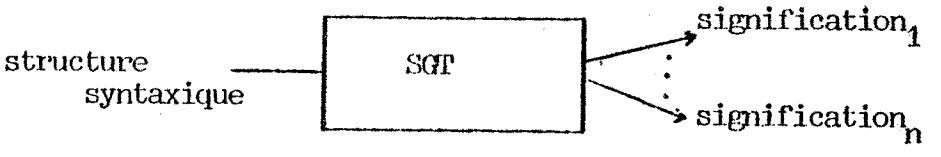
Une hypothèse importante de l'analyse post-syntaxique transformationnelle est que le sens d'une phrase dans un contexte dépend de sa structure et de la signification des mots qui la composent. Si chaque mot d'une phrase avait un seul sens, l'analyse sémantique pourrait être une simple transduction de chaque mot vers sa signification (dans la structure syntaxique). Mais généralement un mot a plusieurs significations, et c'est aussi bien sa position dans la phrase que les mots qui l'entourent qui déterminent sa signification. Si on appelle schème (ou contexte sémantique) la structure formée par un mot et le groupe de mots nécessaire à déterminer sa signification, la signification d'une phrase dans un certain contexte est construite, selon sa structure, à partir des significations de ses schèmes constitutifs.

III.4.1.3. Ambiguïté syntaxique et ambiguïté sémantique

Dans un système d'interprétation sémantique une phrase est considérée comme ambiguë si elle peut être représentée de plusieurs façons différentes. Si elle a plusieurs structures syntaxiques, elle est ambiguë syntaxiquement ; si elle a plusieurs significations, donc si elle a plusieurs

représentations de signification (et vice-versa), elle est ambiguë sémantiquement. Dans le traitement d'une phrase ambiguë au niveau syntaxique, mais non au niveau sémantique il est probable que certaines structures syntaxiques ne mènent pas au sens de la phrase, ces structures sont considérées incorrectes. L'analyse post-syntaxique est donc une phase de filtrage.

Au cas où la phrase est ambiguë sémantiquement, il se peut qu'une structure syntaxique corresponde à plusieurs significations. Si l'analyse post-syntaxique est effectuée par un SGT, il ne possède pas forcément



la propriété de Church-Rosser. Si l'on associe aux phrases ambiguës sémantiquement plusieurs structures syntaxiques de façon que, pour chaque structure syntaxique, il existe au plus une signification, alors on pourra utiliser un STP (ou une séquence de STP) de Church-Rosser pour effectuer les phases de filtrage et de réduction vers la représentation de la signification. Ceci explique la tendance (non seulement dans le 'transformationnalisme') à répercuter au niveau syntaxique les ambiguïtés sémantiques des phrases. Par exemple, les constructions "persuade NP to VP" et "expect NP to VP" considérées comme semblables par Chomsky (1957) sont traitées différemment au niveau syntaxique et sémantique dans Rosenbaum (1967). Un autre exemple est le traitement de certaines classes d'adverbes (ceux de quantité, entre autres) dans Lakoff (1970).

III.4.2. Utilisation des ARPC et des STP

Dans l'analyse post-syntaxique les STP et les ARPC peuvent être utilisés pour effectuer les opérations suivantes sur une structure syntaxique : (i) vérification sémantique, (ii) évaluation des variables syntaxiques ou sémantiques, (iii) traitement de paraphrases, (iv) résolution des

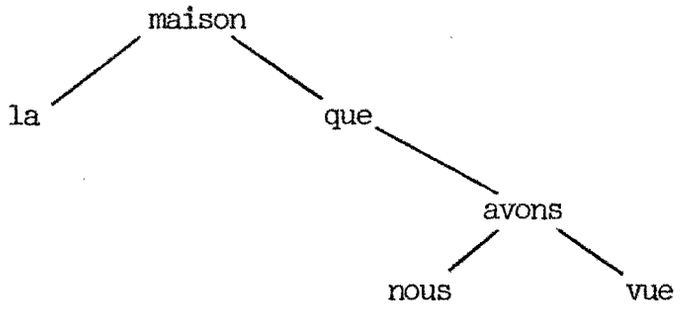
ambiguïtés. L'application de l'ensemble de ces opérations sur une structure syntaxique donne comme résultat une transduction de la structure vers une forme qui représente sa signification. Dans les paragraphes qui suivent on trouvera une description de chacune de ces opérations et des exemples d'utilisation.

III.4.2.1. Vérification et évaluation

Au paragraphe III.1.3. on a décrit schématiquement le modèle de contrôle des variables du système PIAF. Son but est de vérifier certaines relations entre les valeurs des variables syntaxiques des mots de la phrase, pour ainsi accepter ou rejeter une structure de dépendances.

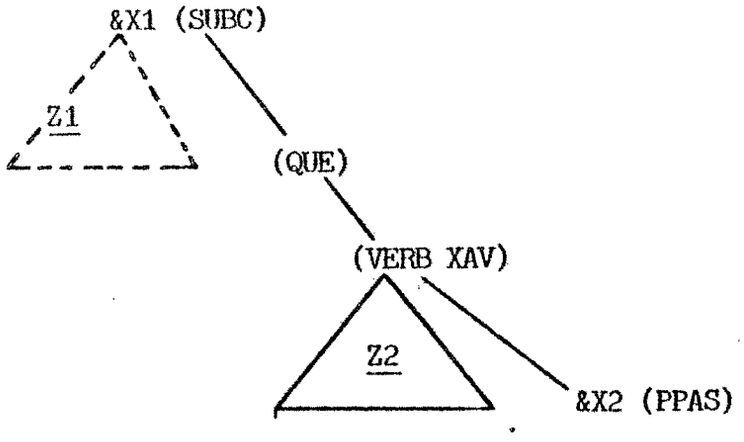
Les règles de contrôle de variables de ce modèle sont des règles binaires qui s'appliquent sur les deux éléments (gouverneur, dépendant) de toute relation de dépendance. Si on veut vérifier une relation entre les valeurs de deux éléments qui ne sont pas joints par une relation de dépendance, ceci oblige à créer des variables dites "véhiculaires" (Veillon, 1970). Les variables véhiculaires sont utilisées pour transmettre dans la structure de dépendances (du bas vers le haut) les valeurs des variables syntaxiques, jusqu'à ce qu'une règle binaire puisse être appliquée pour vérifier une certaine relation entre la valeur de la variable véhiculaire et celle d'une autre variable (qui peut être véhiculaire elle aussi).

Par exemple dans le cas de phrases comme : "la maison que nous avons vue".



il faut vérifier l'accord de genre et nombre entre le substantif "maison" et le participe passé "vue". Deux variables véhiculaires seront créées pour transmettre le genre et le nombre du participe passé au verbe 'avons', et puis au pronom relatif 'que'. Finalement une règle de contrôle de variables vérifiera l'accord des variables du substantif avec les variables véhiculaires du pronom relatif. Notons que la transmission de variables véhiculaires est faite au moyen de règles du modèle de contrôle de variables.

Les règles de vérification définies au § III.3.3. peuvent être utilisées pour éviter ainsi la création des variables véhiculaires et alléger la manipulation ; par exemple :



Condition : ().
Vérifier : GNR(&X1) > GNR(&X3) et
NBR(&X1) > NBR(&X3)

Evaluer : GNR(&X1) = GNR(&X3) ;
NBR(&X1) = NBR(&X3) ;

Notons que cette règle, outre la vérification d'accord en genre et nombre entre le participe passé et le substantif, utilise les effets secondaires pour effectuer simultanément une évaluation des valeurs des variables genre et nombre du substantif.

Bien entendu, de façon similaire à l'évaluation des variables syntaxiques et/ou sémantiques, on peut évaluer la signification d'une phrase. Petrick (1973) par exemple parcourt la structure syntaxique (normalisée par un système de transformations) de haut en bas jusqu'aux feuilles, et puis il remonte à la racine, évaluant les attributs des sommets d'après une grammaire d'attributs de Knuth (1968).

Le processus est répété un nombre fixe de fois[1] (trois) en respectant les contraintes suivantes : (i) à tout attribut on affecte une valeur une fois que les valeurs des attributs dont il dépend (d'après les règles) ont été déterminées, et (ii) une fois déterminée, la valeur d'un attribut ne peut pas être modifiée. A la fin du processus, la représentation de la signification de la phrase est la valeur d'un des attributs de la racine de la structure syntaxique.

Ce processus d'évaluation de la signification a des inconvénients d'ordre pratique principalement. Les règles affectent un attribut à un sommet d'après les attributs de ses fils, et alors certains attributs ont la même fonction que les variables véhiculaires du modèle de contrôle de variables. Outre l'augmentation du nombre de règles pour manipuler ces attributs véhiculaires, ceci ajouté à la deuxième restriction a pour conséquence l'existence d'attributs "copies" d'autres qui ne peuvent pas être modifiés.

Un autre procédé d'évaluation de la signification est celui de Joloboff (1978) décrit au paragraphe III.1.4. Dans ce modèle, les règles d'évaluation sont écrites sous la forme de procédures et elles affectent des valeurs aux variables sémantiques des sommets. A la fin du processus d'évaluation, la représentation de la signification est une structure isomorphe à la structure syntaxique normalisée, construite d'après ses variables sémantiques.

---

[1] On ne comprend pas le pourquoi de cette restriction puisque les contraintes d'application (cf. ci-après) garantissent que pour un nombre fini d'attributs le processus s'arrête. La condition d'arrêt serait donc de faire un parcours de l'arborescence sans affecter de valeur à aucun attribut.

III.4.2.2. Traitement des paraphrases

Les règles de transformation offrent la possibilité de déclarer des équivalences au niveau de la signification de différentes formes syntaxiques. Si une structure syntaxique C_1 est équivalent à une autre structure C_2, l'équivalence peut être exprimée, soit par la règle de transformation $C_1 \rightarrow C_2$, ou soit par $C_2 \rightarrow C_1$, ou soit par les deux règles $C_1 \rightarrow C_0$ et $C_2 \rightarrow C_0$ où C_0 est une autre structure.

En général, dans une langue naturelle, deux structures syntaxiques ne sont jamais tout à fait équivalentes, il y a toujours des nuances, qui dans certains cas peuvent changer la signification. Tout de même, il s'avère assez utile d'écrire des règles qui transforment deux structures assez proches vers une même structure et traduisent les différentes nuances au niveau de variables sémantiques.

Par exemple, les phrases (1) et (2) proposées par Lakoff,

(1) Beaucoup d'hommes lisent peu de livres
(2) Peu de livres sont lus par beaucoup d'hommes

syntaxiquement correspondent à la même phrase exprimée sous une forme active (1) ou passive (2), malgré le fait qu'elles aient respectivement les significations différentes[1] (S1) et (S2).

(S1) Il y a beaucoup d'hommes qui lisent peu de livres
(S2) Il y a peu de livres qui soient lus par beaucoup d'hommes.

Sous la forme d'une expression formelle, ces deux significations peuvent

---

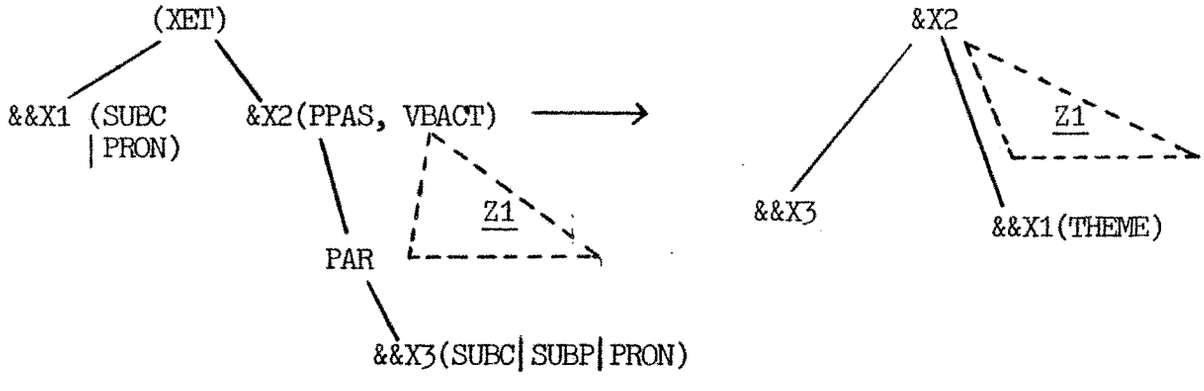
[1] Lakoff (1971) et Schank (1972) les considèrent comme deux phrases indépendantes sur lesquelles les transformations à la forme passive (ou à la forme active) ne fonctionnent pas.

s'écrire comme :

$$(S1)\ (\exists_b \in h \in \text{hommes} : (\exists_p \ell \in \text{livres} : h \text{ lit } \ell))$$
$$(S2)\ (\exists_p \in \ell \in \text{livres} : (\exists_b \ h \in \text{hommes} : h \text{ lit } \ell))$$

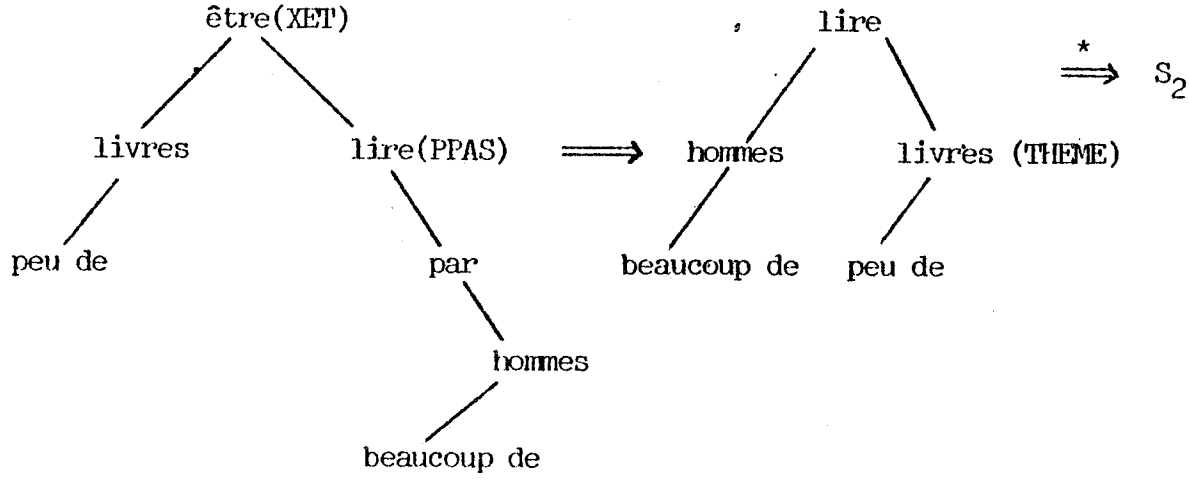
où $\exists_b$ se lit : "il existe beaucoup de", et $\exists_p$ comme : "il existe peu de". Nous notons que sous cette forme les deux significations sont différenciées par la priorité des quantificateurs $\exists_p$ et $\exists_b$, ce qui est exprimé en langue naturelle sous la forme déclarative ou passive.

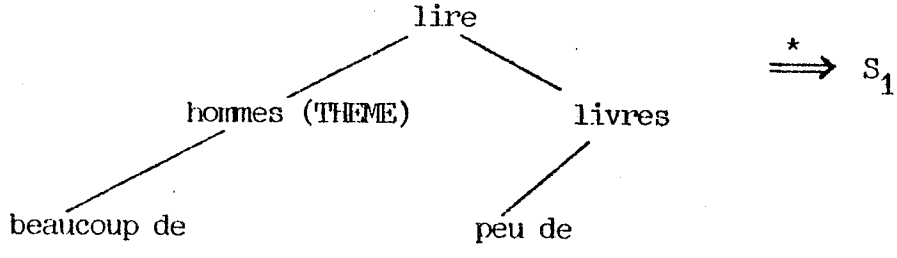
Nous considérons que la structure syntaxique d'une phrase sous la forme active et celle sous la forme passive sont équivalentes, sauf en ce qui concernen le THEME de la phrase. En conséquence, on aura la règle suivante de transformation de la forme passive à la forme active d'une phrase :



où VBACT est une variable qui indique qu'un verbe est transitif. Ainsi la phrase : "il est venu par Paris" ne sera pas interprétée comme étant à la forme passive. Nous notons que la présence du groupe nominal &&X3 est obligatoire ; s'il n'est pas présent dans la phrase, cette règle ne sera pas appliquée. Il faut écrire une autre règle prévoyant ce cas et exprimant l'attitude à prendre (par exemple : création d'un sujet implicite "quelqu'un" ou "on").

L'application de cette règle à la phrase (2) produit une structure semblable à celle de la phrase (1) et ce sera alors la variable THEME de la phrase qui désigne le quantificateur prioritaire au niveau de la représentation de la signification.





Un exemple plus complet où on voit un ensemble de règles de transformation et les structures transformées est donné dans la suite, à l'aide de sorties d'ordinateurs. Comme dans l'exemples du § III.2.4. une structure de dépendance est écrite par l'utilisateur linéairement en préordre (en séparant les dépendants à droite de ceux de gauche par le symbole "*"), et elle est dessinée par le système d'après les conventions suivantes : (i) les dépendants d'un sommet sont déclarés à droite par rapport à lui, (ii) les dépendants à gauche sont situés au-dessus de leur gouverneur et (iii) les dépendants à droite sont situés au-dessous de lui.

Un ensemble de règles peut être additionné à un autre ensemble quelconque (même l'ensemble vide) au moyen de la fonction AJOUTER. Cette fonction demande interactivement à l'utilisateur tous les éléments qui composent une règle et construit l'ARPC correspondant.

Dans l'exemple suivant on définit un ensemble de règles appelé PARAPHRASE. Ces règles définissent une équivalence entre les structures suivantes : (i) une phrase substantivée introduite par l'expression "le fait que" et la phrase même, (ii) la forme passive et la forme active d'une phrase, (iii) la substantivation des verbes et la phrase correspondante en forme active, et (iv) le passé composé d'un verbe avec l'indicateur PASSE.

```
NIL
>ajouter(paraphrase)
ECRIVEZ LE NOM  DE LA REGLE
==>
phrase_subst
PHRASE_SUBST
ECRIVEZ LA PARTIE GAUCHE  DE LA REGLE
==>
(fait le * (que (&&x2 vers)))
          ! LE-
! FAIT-
          !
          !          ! &&X2- (VERS)
          ! QUE-
ECRIVEZ LA PARTIE DROITE  DE LA REGLE
==>
error
ECRIVEZ LA PARTIE GAUCHE  DE LA REGLE
==>
(fait le * (que * (&&x2 vers)))
          ! LE-
! FAIT-
          !
          ! QUE-
                    ! &&X2- (VERS)
ECRIVEZ LA PARTIE DROITE  DE LA REGLE
==>
((?ind &&x2 (phsubst)))
! &&X2- ((PHSUBST))
ECRIVEZ LA CONDITION  DE LA REGLE
==>
()
NIL
ECRIVEZ LES EFFETS DE BORD  DE LA REGLE
==>
()
NIL
ECRIVEZ LE NOM  DE LA REGLE
==>
forme_passive
FORME_PASSIVE
$ECRIVEZ LA PARTIE GAUCHE  DE LA REGLE
==>
((&x1 xet) (&&x2 subc ! subp) * ((&x3 ppas)*(par * (&&x4 subc ! subp))))
          ! &&X2- (SUBC ! SUBP)
! &X1- (XET)
          !
          ! &X3- (PPAS)
                    !
                    ! PAR-
                              ! &&X4- (SUBC ! SUBP)
ECRIVEZ LA PARTIE DROITE  DE LA REGLE
==>
(&x3 &&x4 (?ind &&x2 (theme)))[ (&x3 &&x4 * (?ind &&c@x2 (theme)))
          ! &&X4-
! &X3-
          ! &&X2- ((THEME))
ECRIVEZ LA CONDITION  DE LA REGLE
==>
()
NIL
ECRIVEZ LES EFFETS DE BORD  DE LA REGLE
==>
()
```

```
ECRIVEZ LE NOM  DE LA REGLE
==>
verbe_subst
VERBE_SUBST
ECRIVEZ LA PARTIE GAUCHE  DE LA REGLE
==>
((&x1 verbnom) (&x2 artd) * ((@de (@&@*(&&x3 subc ! subp)) (par * (&&x4 subc ! subp)))
          ! &X2- (ARTD)
! &X1- (VERBNOM)
          !
          ! DE-
          !          ! &&X3- (SUBC ! SUBP)
          !
          ! PAR-
                     ! &&X4- (SUBC ! SUBP)
ECRIVEZ LA PARTIE DROITE  DE LA REGLE
==>
(&x1 &&x4 * &&x3)
          ! &&X4-
! &X1-
          ! &&X3-
ECRIVEZ LA CONDITION  DE LA REGLE
==>
()
NIL
ECRIVEZ LES EFFETS DE BORD  DE LA REGLE
==>
()
NIL
ECRIVEZ LE NOM  DE LA REGLE
==>
passe_comp
PASSE_COMP
ECRIVEZ LA PARTIE GAUCHE  DE LA REGLE
==>
(&@(&x1 xav) * (&&x2 ppas)))
! &X1- (XAV)
          ! &&X2- (PPAS)
ECRIVEZ LA PARTIE DROITE  DE LA REGLE
==>
((?ind &&x2 (passe\)))
! &&X2- ((PASSE\))
ECRIVEZ LA CONDITION  DE LA REGLE
==>
()
NIL
ECRIVEZ LES EFFETS DE BORD  DE LA REGLE
==>
()
NIL
ECRIVEZ LE NOM  DE LA REGLE
==>
fin
FIN
FIN
\
```

La fonction ANALYSEZ exécute un ou plusieurs ensembles de règles sur les structures de dépendances des phrases données interactivement. Si l'on veut appliquer plusieurs ensembles de règles, ceux-ci sont appliqués séquentiellement (dans l'ordre donné), chacun sur le résultat délivré par celui qui le précède.

Cette fonction permet : (i) l'impression de la structure de dépendances de la phrase (mode ARBRE), (ii) l'impression de l'arborescence chaque fois qu'une transformation lui est appliquée (modes ARBRE et TRACE) et (iii) l'impression de la structure résultante du processus transformationnel (modes ARBRE, TRACE et REP). Enfin, il ne faut pas oublier que les structures de dépendances des phrases sont le résultat des analyses morphologiques et syntaxiques effecutées par le système PIAF.

Dans les exemples qui suivent on montre des paraphrases pour lesquelles l'analyse transformationnelle délivre le même résultat mettant en évidence leur équivalence.

```
NIL
>analysez(paraphrase)
MODE? (ARBRE, TRACE OU REP)
arbre
?
le fait que Gaetane est etonne\ par la destruction de Beauceville
?
par Benoit est constate\ par Richard.
TROP DE DEPENDANTS ?


                  ! LE- (ARTD MAS SIN)
          ! FAIT- (SUBC MAS SIN)
          !         !
          !         ! QUE- (PRL ACC INT)
          !                 !
          !                 !         ! GAETANE- (SUBP MAS SIN)
          !                 ! EST- (VERS IND PRE XET SIN TRE)
          !                         !
          !                         ! ETONN- (PPAS MAS SIN)
          !                                 !
          !                                 ! PAR- (PEPC)
          !                                         !
          !                                         !          ! LA- (ARTD FEM SIN)
          !                                         ! DETRU- (SUBC VERBNOM FEM SIN)
          !                                                   !
          !                                                   ! DE- (PEPC)
          !                                                   !         ! BEAUCEVILLE- (SUBP FEM S
          !                                                   !
          !-------------------------------------------------------------!
          !                                                   ! PAR- (PEPC)
          !                                                           ! BENOIT- (SUBP MAS SIN)
! EST- (VERB IND PRE XET SIN TRE)
          !
          ! CONSTAT- (PPAS MAS SIN)
                    !
                    ! PAR- (PEPC)
                            ! RICHARD- (SUBP MAS SIN)

VERBE_SUBST
(EST (FAIT LE * (QUE * (EST GAETANE * (ETONN * (PAR * (DETRU BENOIT * BEAUCEVILLE)))))) * (CONSTAT *
(PAR * RICHARD)))
FORME_PASSIVE
(EST (FAIT LE * (QUE * (ETONN (DETRU BENOIT * BEAUCEVILLE) * GAETANE))) * (CONSTAT * (PAR * RICHARD)
))
PHRASE_SUBST
(EST (ETONN (DETRU BENOIT * BEAUCEVILLE) * GAETANE) * (CONSTAT * (PAR * RICHARD)))
FORME_PASSIVE
(CONSTAT RICHARD * (ETONN (DETRU BENOIT * BEAUCEVILLE) * GAETANE))


          ! RICHARD- (SUBP MAS SIN)
! CONSTAT- (PPAS MAS SIN)
          !
          !         !
          !                   ! BENOIT- (SUBP MAS SIN)
          !         ! DETRU- (SUBC VERBNOM FEM SIN)
          !         !         ! BEAUCEVILLE- (SUBP FEM SIN)
          ! ETONN- (PPAS THEME PHSUBST MAS SIN)
                    ! GAETANE- (SUBP THEME MAS SIN)
```

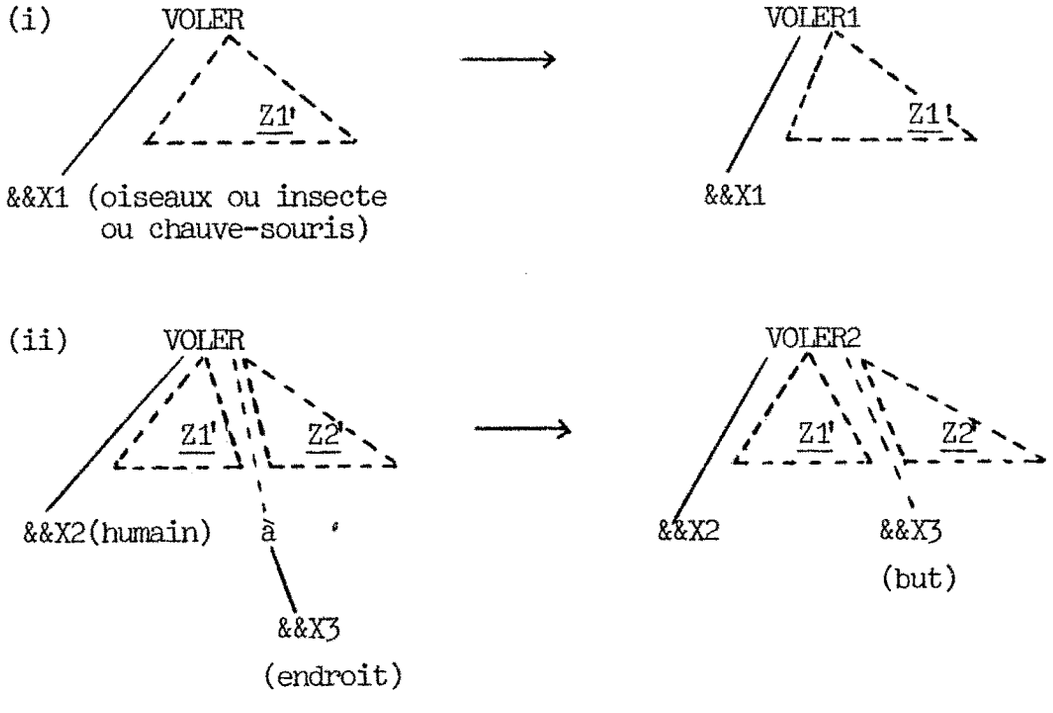
```
NIL
>analyse2(paraphrase)
MODE? (ARBRE, TRACE OU REP)
trace
?
Richard constate l'etonnement de Gaetane par la destruction de
?
 Beauceville par Benoit.
TROP DE DEPENDANTS ?
$VERBE_SUBST
(CONSTAT RICHARD * (ETONN L' * (DE * GAETANE) (PAR * (DETRU BENOIT * BEAUCEVILLE))))
VERBE_SUBST
(CONSTAT RICHARD * (ETONN (DETRU BENOIT * BEAUCEVILLE) * GAETANE))


          ! RICHARD- (SUBP MAS SIN)
! CONSTAT- (VERB IND PRE VERB IMP VERB SUB PRE SIN SIN SIN UNO TRE DOS UNO TRE)
          !
          !          !
          !                    ! BENOIT- (SUBP MAS SIN)
          !          ! DETRU- (SUBC VERBNOM FEM SIN)
          !          !          ! BEAUCEVILLE- (SUBP FEM SIN)
          ! ETONN- (SUBC VERBNOM MAS SIN)
                    ! GAETANE- (SUBP MAS SIN)

?
l'etonnement de Gaetane par le fait que Beauceville est detruite
?
 par Benoit est constate\ par Richard.
$$FORME_PASSIVE
(EST (ETONN L' * (DE * GAETANE) (PAR * (FAIT LE * (QUE * (DETRUI BENOIT * BEAUCEVILLE))))) *
* (PAR * RICHARD)))
PHRASE_SUBST
(EST (ETONN L' * (DE * GAETANE) (PAR * (DETRUI BENOIT * BEAUCEVILLE))) * (CONSTAT * (PAR * R
)
VERBE_SUBST
(EST (ETONN (DETRUI BENOIT * BEAUCEVILLE) * GAETANE) * (CONSTAT * (PAR * RICHARD)))
FORME_PASSIVE
(CONSTAT RICHARD * (ETONN (DETRUI BENOIT * BEAUCEVILLE) * GAETANE))


          ! RICHARD- (SUBP MAS SIN)
! CONSTAT- (PPAS MAS SIN)
          !
          !          !
          !                    ! BENOIT- (SUBP MAS SIN)
          !          ! DETRUI- (PPAS PHSUBST FEM SIN)
          !          !          ! BEAUCEVILLE- (SUBP THEME FEM SIN)
          ! ETONN- (SUBC THEME VERBNOM MAS SIN)
                    ! GAETANE- (SUBP MAS SIN)

?
fin.
FIN
```

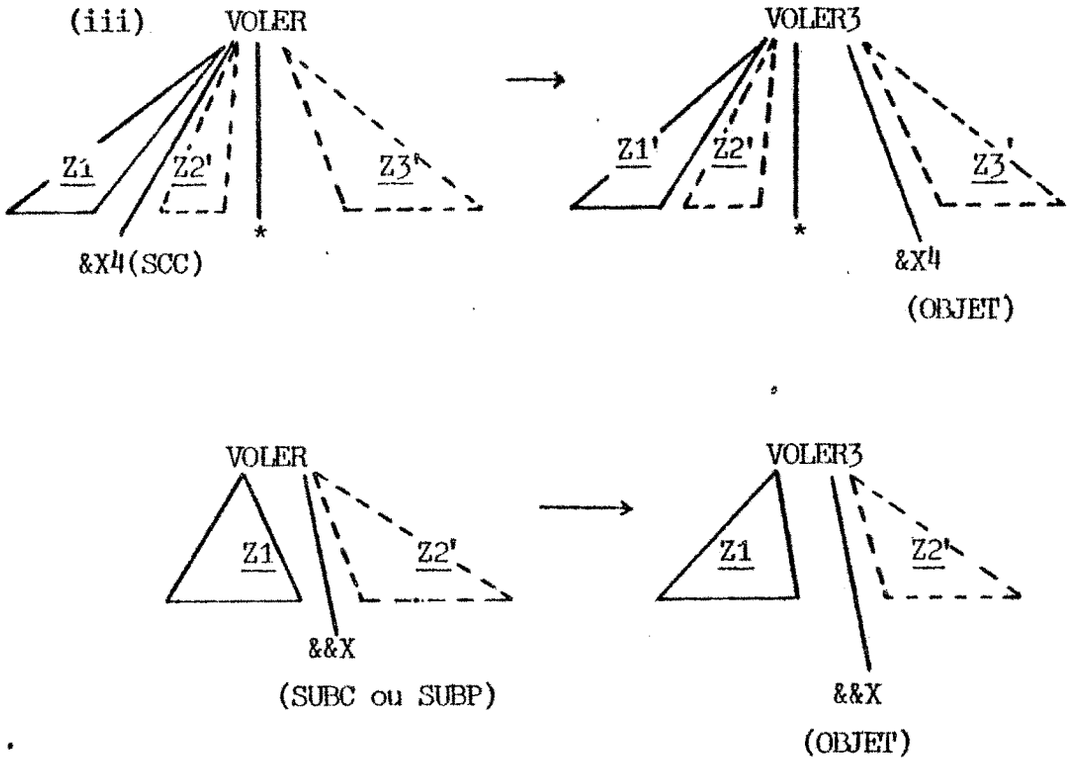
Finalement notons que les règles de transformation pour le traitement de paraphrases ne sont pas absolument nécessaires, mais leur utilisation diminue le nombre total de règles dans un système pour l'analyse post-syntaxique, et en conséquence réduit aussi sa complexité.

III.4.2.3. Résolution d'ambiguïtés sémantiques

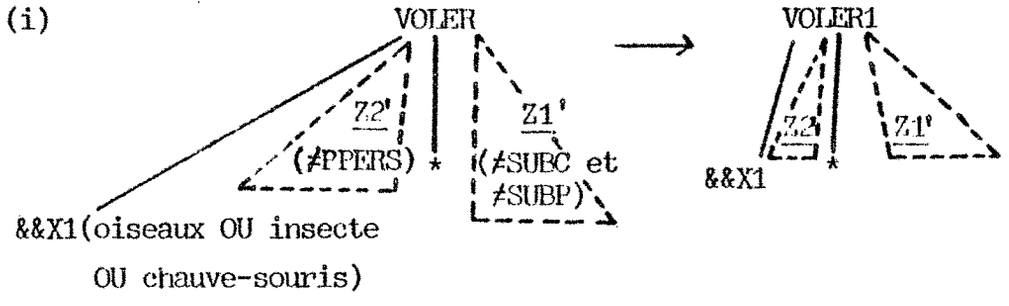
Si, dans un certain contexte, pour déterminer la signification d'un mot, on doit avoir recours à l'analyse d'une sous-structure de la phrase dont elle fait partie, alors ce mot est considéré sémantiquement ambigu. La sous-structure nécessaire pour déterminer sa signification est appelée le contexte sémantique du mot. L'utilisation de règles de transformation (et des traits sémantiques des mots de la phrase) permet d'exprimer le contexte sémantique d'un mot et la signification associée.

Par exemple, pour le verbe 'voler', on peut considérer les sginifications suivantes : (i) se déplacer dans l'air au moyen d'ailes (oiseaux, insectes et quelques mammifères), (ii) se déplacer dans l'air au moyen d'un engin spécial (êtres humains), (iii) dérober quelque chose. Les deux premières significations pouvant être reconnues par les traits sémantiques du sujet et la troisième par l'usage transitive du verbe. On peut écrire les règles de transformation :

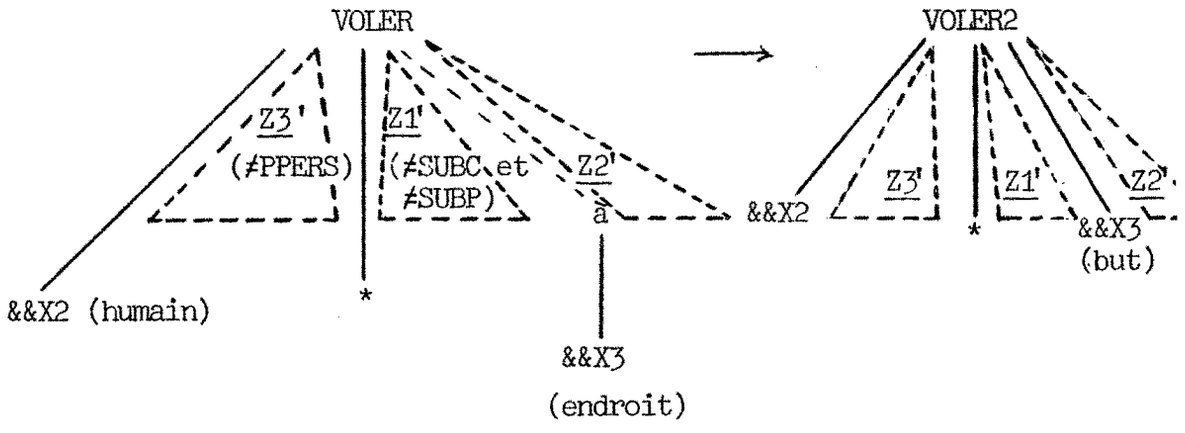




où VOLER1, VOLER2, VOLER3, sont les différentes significations de 'voler', et ACC indique les pronoms personnels qui remplacent l'objet direct de la phrase. En ce qui concerne l'application de ces règles, il faut donner une priorité d'application aux règles (iii) en les insérant dans un STP dont l'application précède celle du STP qui contient les règles (i) et (ii). Ceci correspond à vérifier d'abord la transitivité du verbe et puis les traits sémantiques du sujet. Si on veut écrire un seul STP sans priorités on peut réécrire les règles (i) et (ii) comme suit :



(ii)



où PPERS indique la classe lexical pronom personnel.

Ces règles correspondent à la vérification simultanée des traits sémantiques du sujet et de l'intransitivité du verbe.

CONCLUSION

Nous avons étudié les propriétés des systèmes de transformation de ramifications et présenté un formalisme pour l'expression de règles paramétrées. L'étude effectuée nous a permis,d'introduire la définition de la propriété T-fermeture, plus forte que les propriétés Fermeture et Church-Rosser. Bien qu'un algorithme de vérification de la T-fermeture d'un ensemble de règles n'ait pas été développé jusqu'à présent, nous estimons que cela est possible. Ceci pourrait être l'objet d'un travail ultérieur.

Nous avons résolu le problème de la reconnaissance d'une sous-ramification dans une ramification donnée en définissant les ARP (issus des automates d'états finis), et une règle de composition de ces objets. Outre le fait que la recherche est guidée par la ramification donnée (philosophie ascendante) ceci permet de reconnaître toutes les sous-structures recherchées en un seul parcours de la ramification. L'étude développée laisse percevoir des similitudes entre les ARPC et les transducteurs d'états finis du type de ceux utilisés en morphologie. Une suite de ce travail pourrait être l'étude et l'évaluation d'un transducteur général qui regroupe les ARPC et le transducteur d'états finis du système PIAF.

On a montré grâce aux diverses applications effectuées que les systèmes de transformation se placent plutôt dans le domaine des modèles algorithmiques que dans celui des modèles linguistiques. Ces applications ont servi aussi à définir des règles de vérification et d'évaluation, cas particuliers des règles de transformation.

Enfin, il serait nécessaire que les systèmes exposés ici, soient expérimentés et évalués sur un système opérationnel. Les applications abordées ici, bien que représentant un travail nécessaire et utile, restent cependant au niveau de maquettes expérimentales.

BIBLIOGRAPHIE

[1] ANDREEWSKY A., FLUHR C. : Indexation automatique. Construction automatique des thésaurus. Classification automatique.
Centre d'Etudes Nucléaires de Saclay. Note C.E.A. - N - 1795, Juin 1975.

[2] BACH, Emmon : An introduction to transformational grammars.
Holt, Rinehart and Winston Inc, 1964.

[3] BERLIOUX, Pierre : Etude d'automates et de grammaires de ramifications.
Thèse de 3ème cycle, Université de Grenoble, Juillet 1972.

[4] BOBROW Daniel G., FRASER Bruce : An augmented state transition network analysis procedure.
Proc. International Joint Conf. on Artificial Intelligence, Washington D.C. 1969 pp. 557-567.

[5] BOITET, Christian : Un essai de réponse à quelques questions théoriques et pratiques liées à la traduction automatique. Définition d'un système prototype.
Thèse d'Etat, Université de Grenoble, Avril 1976.

[6] BOITET, Christian : Problèmes actuels en Traduction Automatique : Un essai de réponse.
COLING, 1976.

[7] BOITET, Christian ; GUILLAUME, P.; QUEZEL-AMBRUNAZ, M. : Manipulations d'arborescences et parallèlisme : le système ROBRA
COLING, Août 1978.

[8] BRAINERD Walter Scott : The generating systems and free automata Purdue University, Ph.D., 1967 Mathematics. University Micro-films, Inc., Ann Arbor, Michigan.

[9] CALECA, Jean-Yves : Projet POLYPHEME. L'expression et la décomposition de transactions dans un système de bases de données réparties. Thèse de 3ème Cycle, Université de Grenoble, Septembre 1978.

[10] CHAUCHE, Jacques : Transducteurs et Arborescences. Etudes et réalisations de systèmes appliquées aux grammaires transformationnelles. Université Scientifique et Médicale de Grenoble, Thèse d'Etat, Décembre 1974.

[11] CHAUCHE, Jacques : Vers un modèle algorithmique pour le traitement automatique des langues naturelles. COLING, 1976.

[12] CHASSAGNE, Claudine : Etude et réalisation d'une méthode de transport : traduction de programmes PL360 en LP80. Thèse Docteur-Ingénieur, INPG, Janvier 1979.

[13] CHIARAMELLA, Yves : Détection automatique des variations orthographiques sur des noms propres. Définition d'un transducteur morpho-phonétique interactif. Conférence Internationale sur le Traitement des Langues. Ottawa, Juin - Juillet 1976.

[14] CHOMSKY,Noam, 1955 : Transformational Analysis. Dissertation, University of Pennsilvania

[15] CHOMSKY, Noam, 1957 : Syntactic Structures The Hague, 1957.

[16] CHOMSKY , Noam : On certain formal properties of grammars. Information and Control 2, 137-167 (1959)

[17] CHOMSKY, N. : Context-free grammars and pushdown storage. Quart. Prog. Dept. n°65, MIT Res. Lab. Elect. 187-194, 1962

[18] CHURCH, Alonzo, ROSSER J.B. : Some properties of conversion Transactions of the American Mathematics Society 39 (1936), 472-482.

[19] CODD E.F. : A relational model of data for large shared data banks Comm. of the ACM. V.13, n°6, Juin 1970.

[20] COLMERAUER Alain : Les grammaires de métamorphose. Groupe d'I.A., Marseille-Luming, Novembre 1975.

[21] COLMERAUER, Alain : Les systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur.

[22] CONWAY, M.E. : Design of separable transition diagram compiler. CACM V6, N7, 396-408, July 1963.

[23] GINGSBUR S., GREIBACH S., et HARRISON H. 1967 : One way stack automata. Journal of the ACM, 14, pp. 389-418. 1967

[24] COURTIN, Jacques : Un analyseur syntaxique interactif pour la communication homme-machine.
Article présenté à la Conférence Internationale sur le traitement automatique des langues (PISE 27 août - 1er septembre 1973).

[25] COURTIN, Jacques : Un système d'analyse des langues naturelles : Application à la correction Interactive de textes.
Séminaire de Programmation, Université de Grenoble, Avril 1975.

[26] COURTIN, Jacques : Utilisation des redondances pour l'analyse et le contrôle automatiques d'énoncés en langue naturelle.
Conférence ICCL, Ottawa, Juin 1976.

[27] COURTIN, Jacques : Algorithmes pour le traitement interactif des langues naturelles.
Thèse d'Etat, Université de Grenoble, Octobre 1977.

[28] COURTIN, Jacques : Applications du traitement assisté des langues : documentation, Braille abrégé, traduction de programmes.
COLING, Août 1978.

[29] COURTIN, Jacques ; DUJARDIN D. : Paramètres linguistiques de la morphologie française dans le système PIAF.
Université de Grenoble, Laboratoire d'Informatique, Décembre 1976.

[30] COURTIN J., DUJARDIN D., GRANDJEAN E. : Editeur lexicographique pour les langues naturelles.
U.S.M.G. - I.R.M.A. Mai 1973 (Nouvelle version) 1976.

[31] COURTIN J., GRANDJEAN E. : Editeur lexicographique pour les langues naturelles.
U.S.M.G. - I.R.M.A., Séminaire de Programmation, Mai 1973.

[32] COURTIN J., GRANDJEAN E., VEILLON G. : PIAF : un système de manipulation de données en langue naturelle.
Colloque International de terminologie, Juin 1976.

[33] DAMERAU Fred J. : Advantages of a Transformational Grammar for Question Answering.
IJCAI 1977.

[34] DE REMER F.L. : Transformational Grammars.
dans Lecture notes in Computer Sciences. N°21. Compiler Construction, an advanced course. Ed. G.Goos and J. Hartmanis Springer Verlag,Berlin - Heidelberg, New-York (Chap. 2E) 1974. pp. 121-145.

[35] FILLMORE C.J. : The case for case
dans Universals in linguistic Theory. Holt, Rinehart & Wilson, 1968

[36] FRIEDMAN Joyce : Essential variables in mathematical and Computational models of transformational grammar.
Natural Language processing. Edité par Randall Rustin. Décembre 1971.

[37] GLADKY A.V., MELT'CHUK,I.A.:Tree grammars (Λ-grammars)
COLING 1969. Sweden.

[38] GINGSBURG Seymour, PARTEE Barbara : A mathematical model of transformational Grammars.
Information and Control, 15. (pp. 297-334) 1969.
Academic Press Inc. New York

[39] GINGSBURG, Seymour : The mathematical theory of context-free languages.
Mc. Graw-Hill, New-York, 1966

[40] GRANDJEAN Ernest : Conception et réalisation d'un dictionnaire pour un analyseur interactif de langues naturelles.
Thèse Ingénieur CNAM, Grenoble, Février 1975.

[41] GRANDJEAN Ernest : Système PIAF - Détecteur de fautes d'orthographe PIAFDET.
Rapport Interne, Université de Grenoble, Octobre 1975.

[42] GRANDJEAN E. : Algorithmes de construction et de mise à jour d'un dictionnaire.
Université de Grenoble, Rapport de Recherche n°28, Février 1976.

[43] GRANDJEAN E. : Le système PIAFDOC
3ème Congrès Européen sur les systèmes et réseaux documentaires, Luxembourg 3 mai 1977.

[44] GRANDJEAN E. : Application d'un système de traitement de langues naturelles pour l'indexation automatique.
U.S.M.G. Avril 1978.

[45] GRANJEAN E., MATHIEU B. : Traitement assisté des langues naturelles Système PIAF. Application au contrôle de saisie de textes, à la traduction et à l'édition en braille abrégé.
Laboratoire d'Informatique IMAG

[46] HALL, P.A.V. : Optimisation of single expressions in a relational data base system.
IBM Journal of Research and Development - Vol.20, n°3, Mai 1976.

[47] HAYS D.G. : Dependency theory : a formalism and some observations.
Memorandum R.M. 4087 - P.R. The Rand Corporation - 1964.

[48] HENDRIX G. : Expanding the utility of semantic networks through partitioning.
IJCAI, Tbilisi, USSR, 1975.

[49] HOFMANN, Thomas : Description sémantique et dynamique du discours.
Thèse 3ème cycle en linguistique. Université de Paris-Sorbonne, Paris IV, mai 1978.

[50] HOPCROFT, J.E. ; ULLMAN, J.D. : Formal languages and their relation to automata.
Addison - Wesley Publishing Co., 1969.

[51] JOLOBOFF V. : Interprétation sémantique d'énoncés en langue naturelle.
COLING, Août 1978.

[52] JOLOBOFF, V. : Unification d'arborescences. Evaluation sémantique d'énoncés en langue naturelle.
Thèse de Docteur-Ingénieur, INPG, Septembre 1978.

[53] KAIN, Richard, Y. : Automata theory : machines and languages.
Mc. Graw Hill Books,1972.

[54] KATZ J. and FODOR J. : The structure of a semantic theory.
Language, 39 1963.

[55] KENDER John R. : An annotated Bibliography of natural language and speech understanding systems.
Department of Computer Science. Carnegie-Mellon University Pittsburg, P. 15213. Décembre 1977.

[56] KNUTH, Donald E. 1968 : Semantics of Context-free languages.
Mathematical Systems Theory, Vol.2, n°2.

[57] LAKOFF, G. 1970 : Irregularity in Syntax.
Holt, Rinehart and Winston, New-York.

[58] LAKOFF, G. : On generative semantics.
Dans. D.Steinberg and L. Jakobovits. Eds. 1971. pp. 232.296.

[59] LATOMBE, Jean-Claude : Une application de l'intelligence artificielle à la conception assistée par ordinateur TROPIC.
Université Scientifique et Médicale de Grenoble, Thèse d'Etat, Novembre 1977.

[60] LEVY, L.S., JOSHI, A.K. : Some Results in Tree Automata.
Mathematical Systems Theory, Vol. 6, n°4, 1973 Springer-Verlag New York Inc.

[61] LOMET David Bruce : A formalization of transition diagram Systems.
Journal of the ACM, Vol.20, n°2, Avril 1973, pp. 235-257.

[62] LOPEZ Mauricio : Réalisation d'un interface de communication en langue naturelle avec un système de CAO.
Thèse Docteur-Ingénieur, INPG, (en préparation).

[63] LOPEZ MEDINA Julio : Transformations de structures et graphes de transition.
COLING, Août 1978.

[64] LUX, Augustin : Etude d'un modèle abstrait pour une machine LISP et de son implantation.
Thèse de 3ème cycle, Univ. de Grenoble, Mars 1975.

[65] MATHIEU Blaise : Utilisation d'un transducteur général d'états finis pour la saisie de textes et leur traduction en braille abrégé.
Article présenté au Congrès de l'AFCET - Paris 13-15 Novembre 1978.

[66] NGUYEN G.T. : Optimisation de requêtes relationnelles sur des bases de données réparties.
Document PP13, Equipe POLYPHEME, Université de Grenoble, 1978

[67] PAIR C. : Etude de la notion de pile, application à l'analyse syntaxique.
Thèse, Nancy 1965.

[68] PAIR C. et QUERE A. : Définition et étude des bilangages réguliers.
Information and Control 13, pp. 563-593 - 1968.

[69] PAUCHARD Hélène : Rapport de Contrat Convention n°291-961.
Direction de la documentation française. Service Informatique documentaire.

[70] PARTEE Barbara : Montague grammar and transformational grammar.
Linguistic Inquiry, Vol.VI, n°2, 1975, pp.203-300.

[71] PETRICK, Stanley, R. : Semantic Interpretation in the REQUEST System.
Proceedings of the 1973 International Conference on Computational Linguistics, Pisa, Italy.

[72] PETRICK, Stanley, R. : A recognition procedure for the transformational grammars.
MITRE Corporation.

[73] PETRICK, Stanley, R. : Transformational Analysis.
dans "Natural language processing" edité à Randall Rustin, Algorithmic Press Inc. Décembre 1971.

[74] PLATH, Warren, J. : Transformational Grammar and Transformational Parsing in the REQUEST System.
Proceedings of the 1973 International Conference on Computational Linguistics, Pisa, Italy.

[75] QUERE Alain : Etude des ramifications et des bilangages.
Thèse d'Etat - Université de Nancy - 1969.

[76] ROBINSON Jane J. : Dependency structures and transformational rules.
LANGUAGE, Journal of the Linguistic Society of America , Vol. 46, n°2 (1970) pp. 259-285.

[77] ROSEN, BARRY H. : Tree Manipulating Systems and Church-Rosser Theorems.
Journal of the ACM, Vol. 20, n°1, Janvier 1973, pp.160-187

[78] ROSENBAUM , P. : The grammar of english predicate complement constructions
MIT Press, Cambridge, Massachussetts, 1967.

[79] ROUNDS W.C. : Trees transducers and transformations
Ph.D. Dissertation, Standford University, Août 1968.

[80] ROUNDS William C. : Context free grammars on trees
ACM Symposium on theory of computing, Mai 1969, Marina del Rey California.

[81] RUSTIN Randall (Editor) : Natural language processing.
Courant computer Science symposium 8 : Decembre 20-21 1971.
Algorithmics Press, Inc., P.O. Box 97, New York, N.Y. 10012, 1973.

[82] SALOMAA Arto : The generative capacity of transformational grammars of Gingsburg and Partee.
Information and Control 18, 227-232 (1971)

[83] SALTON, Gerald (Editor) : The SMART retrieval system. Experiments in automatic document processing.
Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1971.

[84] SCHANE, Sanford S. A Schema for sentence coordination.
Information sciences Dept., the MITRE Corporation, Avril 1966,MTP-10.

[85] SCHANK Roger C. : 'Semantics' in Conceptual Analysis
LINGUA 30 (1972) pp.101-140. North-Holland Publishing Company.

[86] SCHANK, Roger ; GOLDMAN, Neil ; RIEGER, Charles J., RIESBECK, Chris : MARGIE : memory, analysis, response generation and inference on english.
Thirth International joint Conference on artificial intelligence. IJCAI,1973.

[87] SCHANK R.C. ; COLBY K.M. (eds) : Computer models of thought and Language. W.H. Freeman, and Company. San Francisco, 1973.

[88] SMITH, J.M. and CHANG P. Y-T. : Optimizing the performance of a relational algebra database interface.
Comm. of the ACM, V.18, n°10, Octobre 1975.

[89] STANLEY, R.J. : Finite state representations of context-free languages Quart. Prog. Rept., 76, MIT Res. Lab. Elect. 276-279, 1965.

[90] STEINBERG D. et JAKOBOVITS L. (eds) : Semantics : an interdisciplinary reader.
Cambridge University Press, Cambridge, 1971.

[91] TESNIERE L. : Eléments de syntaxe structurale
Editions Kleinchseick, 1959

[92] THATCHER, J.W. : Transformations and translations from the point of view of generalized finite automata theory.
Proceedings of the ACM Symposium on Theory of Computing, 1969, pp. 129-142.

[93] THATCHER, J.W. : There's a lot more to finite automata theory than you would have thought.
Proceedings of the 4th Annual Princeton Conference on Information Sciences and Systems, 1970.

[94] VAUQUOIS, Bernard : La traduction automatique à Grenoble.
Document de linguistique Quantitative n°24 - ISBN 2-04-009956-5
Editions Dunod, 1975.

[95] VEILLON Gérard : Modèles et algorithmes pour la traduction automatique.
Thèse d'Etat, Université de Grenoble, Avril 1970.

[96] VEILLON G., VEYRUNES J., VAUQUOIS, B : Un métalangage de grammaires transformationnelles.
Centre d'Etudes pour la Traduction Automatique, Doc. G.2300-A, Janvier 1967.

[97] WALTZ David L. : Natural Language Interfaces.
SIGART Newsletter, n°61, Fevr. 1977.

[98] WINOGRAD T. : Procedures as a representation for data in a computer program for understanding natural language.
Ph.D. Thesis, M.I.T., 1971.

[99] WINOGRAD, T. : An A.I. Approach to English Morphemic Analysis.
Memo # 241, M.I.T., A.I. Laboratory, Fevr. 1971.

[100] WOODS W. et al. : Speech understanding systems. Final Technical progress report.
BBN Report No 3438 Décembre 1976. pp.132

[101] WOODS W.A. : Transition Network Grammars for Natural Language Analysis.
Communications of the ACM, Vol.13, n°10, Octobre 1970.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU les rapports de présentation de Messieurs :

- E. ANDRE, Ingénieur de recherche à CII-HB - GRENOBLE -
- G. VEILLON, Professeur à l'Institut National Polytechnique de GRENOBLE

Monsieur Julio Ernesto LOPEZ MEDINA

est autorisé à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR-INGENIEUR, dans la spécialité "Génie Informatique".

Grenoble, le 13 Juin 1979

Le Président de l'I.N.P.G.

Ph. TRAYNARD
Président
de l'Institut National Polytechnique