HAL
open science

Etude du parallélisme appliqué à la traduction automatisée par ordinateur : STAR-PALE : un système parallèle

José-Nelson Verastegui-Carvajal

To cite this version:

José-Nelson Verastegui-Carvajal. Etude du parallélisme appliqué à la traduction automatisée par ordinateur : STAR-PALE : un système parallèle. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG; Université Joseph-Fourier - Grenoble I, 1982. Français. NNT : . tel-00305347

HAL Id: tel-00305347

https://theses.hal.science/tel-00305347

Submitted on 24 Jul 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

l'Université Scientifique et Médicale de Grenoble
l'Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
"Informatique"

par

José - Nelson VERASTEGUI - CARVAJAL

ETUDE DU PARALLELISME APPLIQUE A LA TRADUCTION AUTOMATISEE PAR ORDINATEUR.
STAR - PALE : un système parallèle.

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

Monsieur	C. BELLISSANT	:	Président
Messieurs	Ch. BOITET Ph. JORRAND G. MAZARE J. VIDART		Examinateurs

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

année scolaire 1980-1981

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

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

PROFESSEURS DE 1ère CLASSE

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

.../...

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

.../...

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

.../...

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

PROFESSEURS DE 2ème CLASSE

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

.../...

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

CHARGES D'ENSEIGNEMENT PHARMACIE

MM.	ROCHAS Jacques	Hygiène et hydrologie
	DEMENGE Pierre	Pharmacodynamie

PROFESSEURS SANS CHAIRE (médecine)

M.	BARGE Michel	Neuro-chirurgie

.../...

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

MAITRES DE CONFERENCES AGREGES (médecine)

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

.../...

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

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1979-1980

Président : M. Philippe TRAYNARD

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

PROFESSEURS DES UNIVERSITES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	BENOIT Jean	Radioéléctricité
	BESSON Jean	Chimie Minérale
	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du Solide - Cristallographie
	BOIS Philippe	Mécanique
	BONNETAIN Lucien	Génie Chimique
	BONNIER Etienne	Métallurgie
	BOUVARD Maurice	Génie Mécanique
	BRISSONNEAU Pierre	Physique des Matériaux
	BUYLE-BODIN Maurice	Electronique
	CHARTIER Germain	Electronique
	CHERADAME Hervé	Chimie Physique Macromoléculaires
Mme	CHERUY Arlette	Automatique
MM.	CHIAVERINA Jean	Biologie, Biochimie, Agronomie
	COHEN Joseph	Electronique
	COUMES André	Electronique
	DURAND Francis	Métallurgie
	DURAND Jean-Louis	Physique Nucléaire et Corpusculaire
	FELICI Noël	Electrotechnique
	FOULARD Claude	Automatique
	GUYOT Pierre	Métallurgie Physique
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du Solide - Cristallographie
	LACOUME Jean-Louis	Géographie - Traitement du Signal
	LANCIA Roland	Electronique - Automatique
	LESIEUR Marcel	Mécanique
	LESPINARD Georges	Mécanique
	LONGEQUEUE Jean-Pierre	Physique Nucléaire Corpusculaire
	MOREAU René	Mécanique
	MORET Roger	Physique Nucléaire Corpusculaire
	PARIAUD Jean-Charles	Chimie - Physique
	PAUTHENET René	Physique du Solide - Cristallographie
	PERRET René	Automatique

.../...

MM.	PERRET Robert	Electrotechnique
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	POLOUJADOFF Michel	Electrotechnique
	POUPOT Christian	Electronique - Automatique
	RAMEAU Jean-Jacques	Chimie
	ROBERT André	Chimie Appliquée et des matériaux
	ROBERT François	Analyse numérique
	SABONNADIERE Jean-Claude	Electrotechnique
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie - Physique
Mme	SCHLENKER Claire	Physique du Solide - Cristallographie
MM.	TRAYNARD Philippe	Chimie - Physique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

CHERCHEURS DU C.N.R.S. (Directeur et Maître 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
	KAMARINOS Georges	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	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.	ALLIBERT Michel	
	BERNARD Claude	
	CAILLET Marcel	
Mme	CHATILLON Catherine	
MM.	COULON Michel	
	HAMMOU Abdelkader	
	JOUD Jean-Charles	
	RAVAINE Denis	
	SAINFORT	C.E.N.G.

.../...

MM. SARRAZIN Pierre
SOUQUET Jean-Louis
TOUZAIN Philippe
URBAIN Georges

Laboratoire des Ultra-Réfractaires ODEILLO

E.N.S.M.E.E.

MM. BISCONDI Michel
BOOS Jean-Yves
GUILHOT Bernard
KOBILANSKI André
LALAUZE René
LANCELOT Françis
LE COZE Jean
LESBATS Pierre
SOUSTELLE Michel
THEVENOT François
THOMAS Gérard
TRAN MINH Canh
DRIVER Julian
RIEU Jean

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

MM. BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

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

MM. BORNARD Guy
DESCHIZEAUX Pierre
GLANGEAUD François
JAUSSAUD Pierre
Mme JOURDAIN Geneviève
MM. LEJEUNE Gérard
PERARD Jacques

E.N.S.H.G.

M. DELHAYE Jean-Marc

E.N.S.I.M.A.G.

MM. COURTIN Jacques
LATOMBE Jean-Claude
LUCAS Michel
VERDILLON André

A Conchita y Viviana
A mi familia
como muestra de afecto y
agradecimiento.

Je tiens à exprimer mes remerciements à:

Monsieur C. BELLISSANT qui s'est intéressé à cette thèse en apportant ses critiques et qui m'a fait l'honneur de présider le jury.

Messieurs P. JORRAND, G. MAZARE et J. VIDART pour avoir bien voulu faire partie du jury et pour l'attention critique qu'ils ont manifestée.

Monsieur Ch. BOITET qui a été mon directeur de thèse et a toujours su me prodiguer les meilleurs conseils tout au long de ce travail.

Je remercie tous les membres du Groupe d'Etudes pour la Traduction Automatique (G.E.T.A), dirigé par Monsieur B. VAUQUOIS, de m'avoir aidé directement ou indirectement pendant ces deux dernières années.

Que tous ceux et celles qui par leurs discussions critiques et suggestions ont contribué à l'amélioration de ce travail, en particulier Madame M. A. MOZOTA et Monsieur F. VELEZ, et tous ceux et celles qui ont contribué à la réalisation matérielle de cet ouvrage, notamment ma femme pour la soigneuse exécution de tous les dessins, trouvent ici un temoignage de reconnaissance.

Nelson VERASTEGUI

TABLE DES MATIERES

Notations. 1
CHAPITRE 1: Introduction 3
1.1 Introduction 4
1.2 Les systèmes de T.A. de seconde génération 4
1.3 Le système de T.A.O. du G.E.T.A. 6
1.4 Situation du travail présenté dans cette thèse 7
CHAPITRE 2: Révision des Principaux Concepts du Parallélisme .9
2.1 Introduction 10
2.2 Définitions et résultats fondamentaux. 12
2.2.1 Processus. 12
2.2.2 Exclusion mutuelle 14
2.2.3 Synchronisation. 15
2.2.4 Communication. 16
2.2.5 Contrôle 17
2.2.5.1 Systèmes centralisés 18
2.2.5.2 Systèmes répartis. 19
CHAPITRE 3: STAR-PALE Système de Transformation
d'ARborescences en PArallèLE. 21
3.1 Introduction 22
3.2 Structures à Transformer 23
3.2.1 Multi-arborescences à tranches 23
3.2.2 Propriétés 26
3.2.3 Relation d'équivalence et d'ordre. 30
3.2.4 Opérations sur les Arborescences 33
3.2.4.1 Eclatement 33
3.2.4.2 Factorisation. 34
3.2.4.3 Extractions. 35
3.3 Règles et Grammaires 41
3.3.1 Quelques notations pour l'ordre et le désordre 42
3.3.1.1 Les notions de base. 42
3.3.1.2 Une notation pour l'ordre total. 46
3.3.1.3 Une notation pour l'ordre partiel. 49
3.3.1.4 Le désordre parmi d'autres notations particulières . 53
3.3.2 Règles 56
3.3.2.1 Schémas. 56
3.3.2.2 Prédicats. 62
3.3.2.3 Image. 63
3.3.2.4 Modifications. 69
3.3.2.5 Contexte d'une règle 70
3.3.3 Grammaires 71
3.3.3.1 Contrôles. 74
3.3.3.2 Ordre et Priorités 76
3.3.3.3 Réseaux de grammaires. 76
3.4 Application d'une grammaire à une structure. 80
CHAPITRE 4: Utilisation du Parallélisme dans une Réalisation 83
4.1 Introduction 84
4.2 STAR-PALE: Un système parallèle. 85
4.2.1 Structures de données internes 86
4.2.2 Description générale 88
4.2.3 Description détaillée. 92
4.2.3.1 Etat Libre 92
4.2.3.2 Etat Test. 92
4.2.3.3 Etat Attente 93
4.2.3.4 Etats Prêt et Révision 94
4.2.3.5 Etat Métamorphose. 95
4.2.3.6 Etats Initialisation et Suppression. 96
4.3 Exemple de transformation. 97
4.4 Remarques sur le comportement de l'algorithme. 107
4.4.1 Commentaires sur l'état actuel 107
4.4.2 Critique, corrections et extensions de l'algorithme. .108
4.4.3 Maquettes à réaliser 111
CHAPITRE 5: Conclusion 115
5.1 Conclusion 116
Bibliographie. 117
Index. 119

Notations

Générales

∈ appartenance à un ensemble

∉ non apartenance à un ensemble

= égal à

≠ différent de

⊂ inclusion stricte dans un ensemble

⊆ inclusion large dans un ensemble

U union ensembliste

I intersection ensembliste

~A complémentaire de l'ensemble A dans un surensemble fixé

A-B différence simple (A I ~B)

|A| cardinalité de A

N ensemble des nombres entiers naturels

≤ inférieur ou égal

< inférieur stict

≥ supérieur ou égal

> supérieur strict

[a,b] intervalle d'entiers { i∈N | a ≤ i ≤ b }

{ a | R(a) } ensemble des éléments a possédant la propriété R

∅ ensemble vide

A**B ensemble des fonctions de B dans A

f:A -> B fonction f de A dans B

f(x) image de x par la fonction f

f[x] image de x par la fonctionelle f, i.e. f:A -> C**D

-f inverse de la fonction f

{a1, a2, ..., an} ensemble fini des éléments a1, a2, ..., an

∀ quel que soit

∃ il existe

& et logique

∨ ou logique

¬ non logique

=> implication logique

<=> équivalence logique

ssi si et seulement si, i.e. équivalence logique

p x le plus petit x

n! factorielle n, i.e. produit des entiers positifs inférieur ou égaux à n

$\binom{n}{m}$ ou C_n^m coefficient binomial: $\frac{n!}{(n-m)!m!}$

□ fin de démonstration, fin d'exemple

Structures de Données

⊥ m.a.t. vide

[n c] noeud d'une m.a.t.

(a1, a2, ..., an) liste finie (ordonnée) des éléments a1, a2, ..., an

<a1, ..., an-1 / an / an+1, ..., am> piste, i.e. liste avec un élément distingué an

Mat ensemble des m.a.t.

Smat ensemble de structures m.a.t.

≅ relation d'équivalence forte entre m.a.t.

≈ relation d'équivalence faible entre m.a.t.

≉ non fortement équivalent

⊏ ordre partiel sur Smat

Permutations

<ai1, ai2, ..., ain> permutation de l'ensemble A={a1, a2, ..., an}, i.e. bijection Ai: A -> [1,n] où Ai(aj)=aij

Π**m ensemble des permutations de [1, m]

ΠA**n ensemble des permutations de A, |A|=n

Ai.Aj concatênation de permutations

πA\B fonction de [1,n]**A -> [1,m]**B

<< être comprise, relation d'ordre entre permutations

⊏ ordre partiel des intervalles fermés

M+ ensemble des occupants explicites

M- ensemble des occupants implicites

π[Ai] permutation naturelle

Expressions d'Ordre

v prédicat de voisinage ordonné

e prédicat d'écartement ordonné

* opérateur de voisinage ordonné

< opérateur d'écartement ordonné

≡ équivalence des expressions

⊥ frontière gauche d'une expression linéaire

T frontière droite d'une expression linéaire

* frontière gauche ou droite d'une expression linéaire

|| opérateur de parallélisme ou interclassement

+ disjonction logique

opérateur de voisinage non ordonné

& opérateur d'écartement non ordonné

~ opérateur de semi-écartement

; opérateur de dissociation

{ } opérateur d'image miroir

Règles et Grammaires

== séparateur syntaxique des parties gauche et droite de règle

/ séparateur syntaxique des composantes des parties gauche et droite de règle

: indicateur syntaxique de schéma vertical

.. indicateur syntaxique de schéma horizontal

↑ priorité verticale vers le haut

↓ priorité verticale vers le bas

< priorité horizontale vers la gauche

> priorité horizontale vers la droite

pp plus prioritaire que, ordre partiel

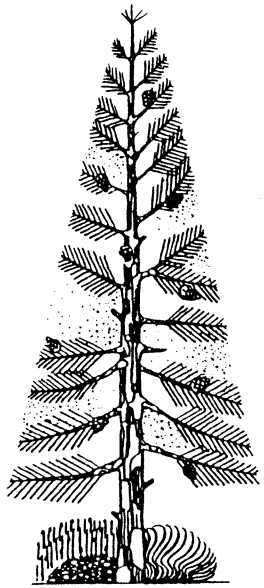
(*) indicateur de point feuille dans un schéma

Occ(C,O,S) occurrence élémentaire C dans O du schéma S

* noeud de sortie d'un système transformationnel

Chapitre I

Introduction



1.1 Introduction

L'idée de traduire des textes écrits avec un ordinateur est apparue il y a environ 30 ans. Depuis le milieu de la décade des années 50 et pendant 12 ans environ, une énorme activité s'est développée dans ce domaine dans différents centres de recherche. Les nécessités de traduction étaient considérées si importantes que n'importe quel mécanisme d'aide était regardé avec attention. A cette époque, la motivation principale était la recherche d'information, mais aujourd'hui les besoins de telles traductions ont augmentés considérablement et en plus il y a une nouvelle orientation des traductions qui consiste à diffuser de l'information. Ce dernier objectif est au moins aussi recherché que le premier [Va 79].

Un système de traduction automatique de première génération, basé sur un grand dictionnaire bilingue, fut terminé vers 1963 sur la base d'un travail commencé en 1954 à l'université de Georgetown. Quelques améliorations au système, conçu initialement pour traduire du russe en anglais, pour traiter d'autres couples de langues, avaient été réalisées par une entreprise privée. Le principe était la traduction mot par mot, avec des recherches contextuelles pour résoudre des ambiguïtés et des sousprogrammes pour changer l'ordre des mots. De plus en plus des sousprogrammes étaient rajoutés pour le traitement de cas particuliers, de même que le contenu des entrées du dictionnaire avait été augmenté pour la prise en compte des valeurs sémantiques dans des domaines particuliers.

De tels systèmes sont nécessairement bornés par la complexité de la programmation d'une si grande quantité de sujets sans faire appel à des modèles logiques et linguistiques structurés. Parallèlement et depuis la période 1956-1959, une idée générale complètement différente était développée qui a permis d'ouvrir un grand domaine de recherche aussi bien linguistique qu'informatique. C'était la base des systèmes de seconde génération qui a évolué le long de toutes ces dernières années.

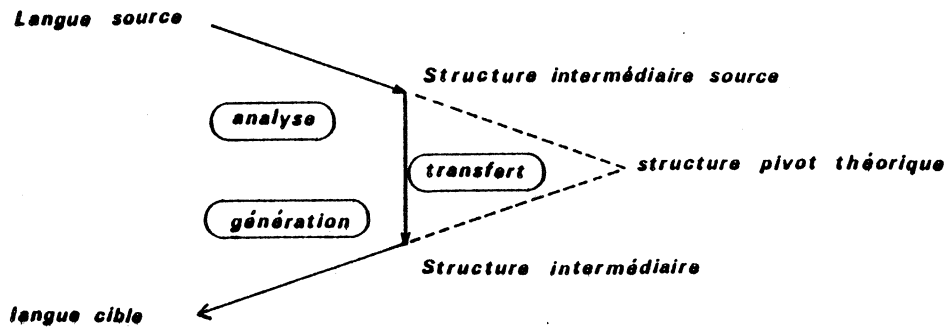
1.2 Les systèmes de T.A. de seconde génération

Les systèmes de traduction automatique de seconde génération sont basés sur trois principes:

a) Détection de "descripteurs structuraux" pour la représentation de toute phrase d'un texte en langue naturelle
b) Modélisation par niveaux
c) Séparation entre les données linguistiques et les programmes

Le processus de traduction d'un texte depuis une langue "source" vers une langue "cible" est divisé en trois étapes logiques principales, comme l'illustre la figure ci-dessous: analyse, transfert et génération. La sortie de l'analyse est un descripteur structural du texte d'entrée, qui est transformé en un descripteur structural équivalent dans la langue cible

par la phase de transfert. Ce descripteur structural cible est alors transformé en texte de sortie par la phase de génération. Dans la conception actuelle, il est essentiel que l'analyse soit effectuée indépendamment de la/des langue(s)-cibles(s). Plus l'analyse est profonde, plus la distance est courte entre les deux descripteurs structuraux. Idéalement, on pourrait imaginer un niveau pivot auquel ils seraient identiques. Mais, il s'agit d'une limite impossible à réaliser.



Entre 1962 et 1972, tous les efforts ont été concentrés sur des descripteurs structuraux représentés par des arbres décorés, comme une conséquence du développement exponentiel des grammaires formelles, théorie des automates, linguistique formalisée et compilateurs pour les langages de programmation de haut niveau. Ces arbres expriment l'organisation syntaxique de la phrase d'une manière considérée comme adéquate pour la phase de transfert. Quelques systèmes de traduction expérimentaux ont été complétés comme suit:

a) Phase d'analyse: Une analyse morphologique optionnelle selon que les unités lexicales du langage source aient une (ou parfois peu de) formes écrites ou au contraire plusieurs mots avec des terminaisons variables. Dans le premier cas, l'analyse est évitée et chaque forme apparaît comme une entrée dans le dictionnaire source. Ceci a été fait assez souvent pour l'anglais. Dans le deuxième cas, le dictionnaire source contient seulement des bases, préfixes, affixes et suffixes. Il est nécessaire donc d'avoir une grammaire pour vérifier la cohérence de la segmentation du mot et pour calculer l'interprétation adéquate de cette segmentation. Ce travail peut être réalisé facilement à l'aide d'un transducteur d'état fini. Par exemple, le français est analysé de cette manière.

Après la consultation des dictionnaires, avec ou sans analyse morphologique, le traitement continue avec l'analyse structurale de chaque phrase. Dans le cas d'un modèle de structure de phrase, la sortie est un arbre décoré dont les

décorations des feuilles sont les mots de la phrase et où les autres noeuds sont décorés avec des symboles non-terminaux tel que "phrase nominale", "phrase prépositionelle", "phrase verbale", etc. Dans le cas d'un modèle de dépendances, la sortie est aussi un arbre décoré, mais chaque noeud est décoré par un mot de la phrase et chaque arc par une relation syntaxique telle que "sujet", "modificateur", etc.

b) Phase de transfert: Elle est divisée généralement en deux parties; d'abord, un transfert lexical au moyen d'un dictionnaire bilingue, puis un transfert structural qui vise à trouver un descripteur structural correspondant de la phrase dans la langue cible.

c) Phase de génération: Il y a aussi deux parties; d'abord, une génération d'une structure syntaxique de surface puis une génération morphologique.

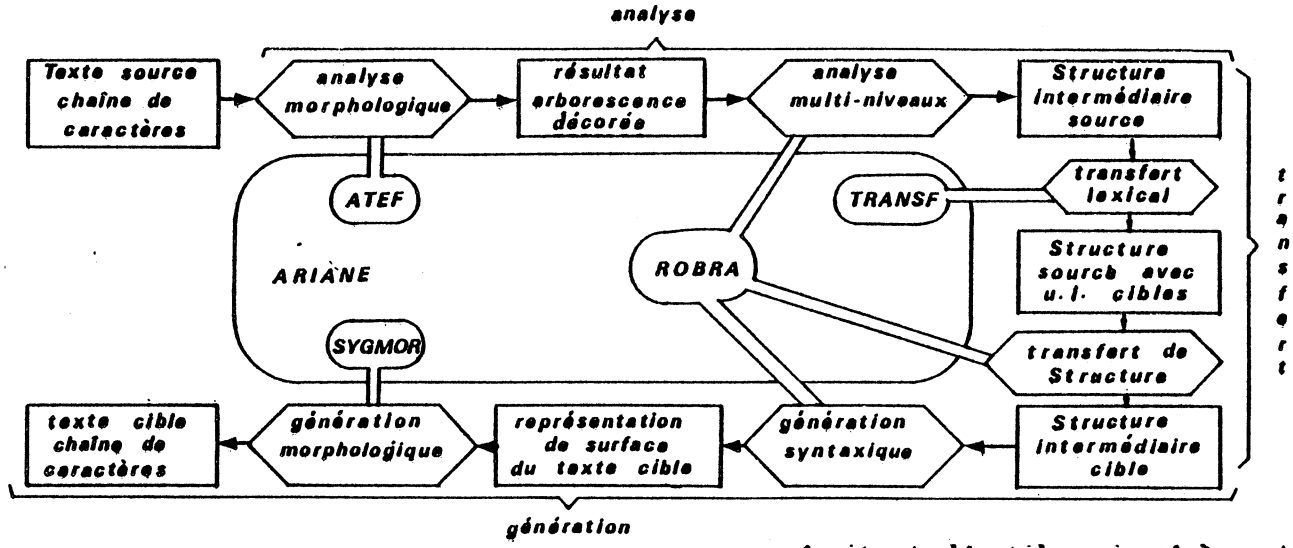
Le rapport ALPAC (1966) a provoqué un ralentissement des activités dans la traduction automatique et même un arrêt complet dans certains pays. En effet, à cette époque les traductions obtenues par les programmes de première génération étaient d'une qualité inférieure à celle escomptée et les performances de ces programmes étant stabilisées, il y avait le doute sur des nouveaux progrès. D'autre part, les systèmes de seconde génération n'avaient pas produit de résultats significatifs à une grande échelle.

Ce n'est qu'à partir de 1972 que les activités ont repris mais cette fois-ci avec des objectifs moins ambitieux. On parle donc de traduction aidée par l'ordinateur ou de traduction automatisée par ordinateur (T.A.O.), et l'idée d'une traduction complètement automatique et sans révisions a été laissée de côté.

1.3 Le système de T.A.O. du G.E.T.A.

Le groupe d'études pour la traduction automatique (G.E.T.A.) de Grenoble a conçu et mis au point un logiciel spécialisé pour les systèmes de traduction automatisée qui vise une traduction multilingue et donne beaucoup de facilités aux linguistes pour la création et mise à jour des grammaires et dictionnaires.

Les premiers analyseurs syntaxiques construits avaient la propriété de combiner les règles d'une grammaire de toutes les manières possibles afin de détecter toutes les structures syntaxiques d'une phrase. Le linguiste était forcé d'écrire les règles et grammaires de façon statique. Le traitement dynamique des règles était réalisé par un algorithme combinatoire et le linguiste n'avait aucun contrôle sur l'exécution. Le GETA a voulu faire un logiciel dans lequel le linguiste n'ait pas besoin de connaître comment programmer un ordinateur, tout en ayant la possibilité de contrôler le traitement dynamique à l'aide d'un langage externe de communication. Il s'agit de l'introduction d'une sorte d'heuristique dans la traduction.



Le logiciel complet, appelé ARIANE-78, comporte les parties suivantes:

a) Un moniteur conversationnel qui conduit toutes les opérations demandées par le linguiste telles que:
   - consulter ou mettre à jour les grammaires et les dictionnaires dans n'importe quelle phase d'analyse, transfert ou génération sur des langues déclarées
   - préparer de textes pour l'experimentation
   - exécuter des phases (ou le traitement complet) de traduction sur des textes d'experimentation, avec (ou sans) traces pour la mise au point
   - imprimer des fichiers, etc.

b) Des composantes algorithmiques
   - ATEF est un transducteur d'état finis non-déterministe. Il s'agit de l'outil qui génère des programmes d'analyse morphologique.
   - ROBRA est un transducteur d'arborescences. Il est utilisé pour l'analyse multi-niveaux (syntaxe et sémantique partielle), pour le transfert structural et finalement, pour la génération syntaxique du langage source.
   - TRANSF est un système pour la consultation de dictionnaires bilingues. Il est utilisé pour le transfert lexical.
   - SYGMOR est un transducteur déterministe d'état finis utilisé pour la génération morphologique.

1.4 Situation du travail présenté dans cette thèse

Les recherches en traduction automatisée se font dans des

directions diverses et à plusieurs niveaux. En gros, il y a les aspects linguistiques qui se concentrent principalement sur l'étude des modèles d'analyse, transfert et génération et la meilleure façon d'organiser les règles, grammaires et dictionnaires pour améliorer la qualité des traductions à l'aide des systèmes existants. De point de vue informatique, il y a une recherche constante de nouveaux algorithmes et de techniques qui augmentent les performances des systèmes de ce type. L'étude de structures de données adéquates est un travail qui se fait plus ou moins dans les deux directions.

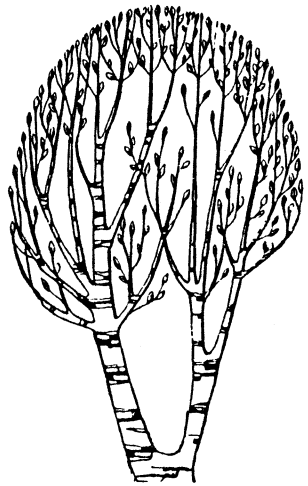
Le but des travaux présentés dans cette thèse était de chercher différentes possibilités d'application du parallélisme à la traduction automatisée à tous les niveaux. Il s'agit de profiter de l'existence du matériel informatique de multitraitement pour la réalisation de travaux simultanés. On peut espérer avec l'application de cette technique des

améliorations comparables, dans un certain sens, au cas où la traduction d'un grand texte par un seul traducteur humain était remplacée par la division du travail entre plusieurs traducteurs humains en coordination.

Le contenu de la thèse est le suivant: Le chapitre 2 résume les concepts principaux en parallélisme. On donne différentes définitions et les résultats trouvés dans la littérature. Le chapitre 3 présente un système, appelé STAR-PALE et basé sur le sous-système ROBRA du GETA, qui a été conçu pour exploiter le parallélisme depuis la structure de données, jusqu'aux règles et grammaires. Il s'agit d'une vision abstraite indépendante de la réalisation éventuelle du système. Le chapitre 4 montre une méthodologie de réalisation pour des systèmes du type précédent sur une architecture orientée au parallélisme. Finalement, on donne un certains nombre de conclusions tirées de ce travail.

Chapitre II

Révision des Principaux Concepts du Parallélisme



2.1 Introduction

Le parallélisme présente un grand intérêt en informatique: c'est un moyen d'augmenter la puissance des systèmes de calcul en faisant le maximum de travail possible simultanément ou d'une façon concurrente. L'idée est d'enlever la restriction imposée par les ordinateurs classiques du type de von Neumann, où l'on ne peut faire qu'une seule chose en même temps, et de concevoir des systèmes informatiques qui exploitent, depuis la conception du matériel jusqu'au système mathématique abstrait en passant par les programmes, algorithmes et systèmes d'exploitation, les possibilités réelles du parallélisme, et qui soient en même temps d'une performance supérieure ou comparable à celles des systèmes actuels. Les recherches se font dans différentes directions parmi lesquelles on peut souligner les suivantes [Ve 81].

L'étude du parallélisme d'une façon abstraite et indépendante des machines et des langages est un domaine fondamental de recherche. Il a été exploré principalement par les chercheurs travaillant sur les systèmes d'exploitation qui ont abouti entre autres à la définition de sémaphores, de mécanismes de synchronisation et de communication entre processus. L'utilisation de modèles mathématiques comme les réseaux de Petri, qui s'avèrent bien adaptés à ce type d'études, a permis d'améliorer la compréhension de ces systèmes.

Il y a plusieurs aspects à mettre en évidence. D'une part, il y a l'objectif de trouver une interprétation mathématique précise du parallélisme qui puisse servir, par exemple, à la preuve de programmes. Il y a aussi la question de connaître les gains réels apportés par son utilisation et, par conséquent, une étude de complexité s'impose, de même qu'une étude de calculabilité. Il existe un rapport entre le parallélisme et le non-déterminisme: il faut faire une analyse de transformations entre modèles mathématiques de ces deux types, ou bien tout simplement entre modèles parallèles et séquentiels. L'examen de problèmes qui requièrent une solution parallèle ou qui soient plus facilement résolus de cette manière aiderait à mieux comprendre les mécanismes du parallélisme.

Une autre orientation est l'étude des langages de programmation par rapport au parallélisme. On trouve des travaux sur l'utilisation de langages classiques dans des systèmes parallèles au niveau des procédures, des instructions d'entrée-sortie, des structures de contrôle et même au plus bas niveau des instructions et des opérateurs. Il y a aussi la conception de langages adaptés au parallélisme qui expriment d'une façon explicite des instructions parallèles, et leur utilisation sur des machines de tout type. On remarque la place privilégiée occupée par les coroutines comme moyen d'expression du parallélisme au niveau des langages. Le but de l'examen des différents types de langages existants, tels que ADA, Pascal

concurrent, LUCID, les systèmes de programmation fonctionnelle de Backus, ALGOL 68, LISP ou le système RUBRA du laboratoire GETA est d'amener à la conception d'un langage approprié ou au moins de définir les caractéristiques désirables ou nécessaires d'un tel langage.

Il y a certainement un changement, dans le domaine des langages de programmation, depuis l'époque où le parallélisme restait caché aux utilisateurs: on passe à une nouvelle période, où l'utilisateur exploite lui même cette possibilité. Il est donc très important de fournir un outil fiable aux programmeurs de sorte qu'ils puissent s'en servir sans se soucier de la réalisation matérielle et sans compliquer le processus de conception de logiciel.

Au niveau du matériel informatique et de l'architecture des machines, l'attention portée à ce sujet est très grande, d'autant plus que l'ordinateur universel d'aujourd'hui reste, dans la majorité des cas, une machine conçue selon le modèle décrit en 1946 par John von Neumann, et où les instructions se déroulent en séquence, les unes après les autres. En effet, pour améliorer la capacité de traitement d'une machine de ce genre, il existe deux moyens: accroître la vitesse d'exécution de chaque instruction ou réaliser des opérations en parallèle. Le parallélisme peut s'obtenir par l'exécution simultanée de plusieurs séquences d'instructions, ou par le traitement simultané de plusieurs données par une séquence d'instructions. En résumé on a le choix entre quatre techniques:

instructions-> données ↓	une seule séquence exécutée	plusieurs séquences exécutées simultanément
une donnée traitée	ordinateur classique modèle UIUD (en anglais SISD, simple instruction simple data)	pipeline modèle PIUD (en anglais MISD, multiple instruction simple data)
plusieurs données traitées simultanément	calculateurs vectoriel modèle UIPD (en anglais SIMD)	multiprocesseur modèle PIPD (en anglais MIMD)

Du point de vue des schémas d'interconnection et des niveaux de parallélisme atteints les possibilités sont nombreuses. Les systèmes à flot de données, qui ont été développés dernièrement, illustrent bien ce type de recherches. Il s'agit tout simplement d'un système commandé par les données de sorte que, dès qu'un module reçoit toutes les données nécessaires à la réalisation d'un calcul, il le fasse indépendamment des autres, et en envoie le résultat dès qu'il est produit. Il est

alors prêt à réaliser immédiatement d'autres calculs.

Finalement, il y a un vaste domaine d'applications au niveau algorithmique qui ont débuté par la conception de systèmes d'exploitation et par la résolution de problèmes combinatoires et heuristiques. Lorsqu'il s'agit de résoudre un problème par division en sous-problèmes indépendants, l'application est immédiate. S'ils ne sont pas indépendants et si l'arbre de décomposition en sous-problèmes peut être parcouru de manière descendante, en parallèle et avec des interactions, la difficulté augmente. Le cas du parcours ascendant parallèle et interactif sera en général plus difficile à résoudre.

Les problèmes de recherche en parallèle sont avantageux dans le cas où k processeurs doivent chercher k éléments différents, car, s'il s'agit du même élément, l'augmentation potentielle de vitesse par rapport à une recherche binaire avec un seul processeur est de O(log2(k+1)) au lieu de O(k) comme on pourrait le croire. Par contre, les tris en parallèle sont très intéressants et même nécessaires lorsqu'il existe un multi-traitement. Les domaines de la traduction automatique, de la reconnaissance de la parole ou des formes et l'analyse de scènes visuelles sont des candidats immédiats à l'application de méthodes parallèles.

On peut dire que la maîtrise du parallélisme et sa mise à la portée de tous les utilisateurs aura un impact énorme dans presque tous les domaines de l'informatique. Ce sera une des voies les plus importantes du développement futur de cette branche.

2.2 Définitions et résultats fondamentaux

Il y a un certain nombre d'éléments qui font partie des concepts fondamentaux, fruits des recherches sur le parallélisme, et que nous allons essayer de résumer dans le présent chapitre. L'ouvrage de base a été [Cr 75] mais il y a d'autres références qui seront données explicitement dans les parties concernées.

2.2.1 Processus

Un processus représente une activité que l'on veut considérer comme élémentaire. Cette activité est l'exécution d'un programme comportant des instructions et des données. Un programme est un ensemble ordonné d'instructions. Une instruction est considérée comme indécomposable (indivisible), c'est-à-dire qu'on s'interdit d'observer le système pendant l'exécution d'une instruction. L'entité, câblée ou non, capable d'exécuter une instruction, est appelée processeur.

Un processus séquentiel, qui correspond à l'exécution d'un programme séquentiel, est une suite temporelle d'exécution d'instructions. Le vecteur d'état d'un processus est l'ensemble

de variables (locales et globales) et des procédures qu'il utilise.

Un système est composé d'un nombre limité de ressources qui seront utilisées par les différents processus au cours de leur existence, par exemple, les données, la mémoire, l'unité centrale, les periphériques, les signaux.

Lorsqu'un processus n'est pas en possession des ressources indispensables à l'exécution d'une instruction, on dit qu'il est dans l'état bloqué. S'il dispose de toutes les ressources dont il a besoin, on dit qu'il est dans l'état actif.

Une ressource est dite locale à un processus si elle ne peut être utilisée que par ce processus. Une ressource qui n'est locale à aucun processus est dite commune. Une ressource commune est dite partageable avec n points d'accès ($n \geq 1$) si cette ressource peut être attribuée, au même instant, à n processus au plus. Une ressource partageable à un point d'accès est dite critique.

Dès qu'on alloue des ressources non sujettes à réquisition à des processus qui s'exécutent concurremment, il apparaît un risque de blocage mutuel de ces processus lorsque, pour certains types de ressource, la demande totale est supérieure au nombre de points d'accès. En effet, les demandes de ressources de différents processus peuvent être satisfaites dans un ordre tel que deux ou plusieurs d'entre eux se bloquent indéfiniment: chacun de ces processus accapare des ressources tout en attendant celles occupées par les autres. On dit que les processus sont interbloqués.

L'interblocage peut mettre en jeu un nombre important de processus et de ressources. Il y a donc deux solutions possibles: la guérison ou la prévention. Le déblocage de tous les processus ou guérison peut être très complexe, voire impossible sans destruction d'une partie ou de la totalité des processus interbloqués. Il est donc intéressant de rechercher des techniques d'allocation évitant l'apparition de l'interblocage, c'est la prévention. Il y a différents algorithmes de détection et de prévention, parmi ces derniers on peut citer l'algorithme du banquier [Di 68].

Des processus sont dits indépendants s'ils n'ont que des ressources locales. Ils sont dits parallèles pour une ressource s'ils peuvent l'utiliser simultanément et en exclusion mutuelle s'il s'agit d'une ressource critique.

Deux processus sont en relation (ne sont pas indépendants) si leurs vecteurs d'état ont une intersection non vide: l'un des processus peut rendre une ressource accessible à l'autre, ou le priver de cette ressource, c'est-à-dire finalement que l'un des processus peut changer l'autre d'état.

Un processus est une entité dynamique qui naît (lors du lancement de l'exécution d'un programme) et meurt (à la fin de cette exécution). Les processus sont créés ou détruits soit à l'initiative du système, soit à celle d'un processus quelconque. Créer un processus, c'est lui donner un nom et définir son vecteur d'état initial. A la destruction d'un processus, son vecteur d'état disparaît, les ressources communes qu'il utilisait sont rendues disponibles pour d'autres processus, ses ressources locales sont détruites.

2.2.2 Exclusion mutuelle

Il s'agit de l'utilisation d'une ressource c à un seul point d'accès par un ensemble de processus de sorte que:

a) A tout instant un processus au plus puisse se trouver en section critique (c'est-à-dire une phase pendant laquelle la ressource c est utilisée et donc inaccessible aux autres processus).

b) Si plusieurs processus sont bloqués en attente de la ressource critique, alors qu'aucun processus ne se trouve en section critique, l'un d'eux doive pouvoir y entrer au bout d'un temps fini.

c) Si un processus est bloqué hors d'une section critique, ce blocage ne doive pas empêcher l'entrée d'un autre processus en sa section critique.

d) La solution doive être la même pour tous les processus, c'est-à-dire qu'aucun processus ne doit jouer de rôle privilegié.

Il y a plusieurs solutions possibles, en particulier: l'attente active, les verrous et les sémaphores.

Pour l'attente active, on déclare une variable p accessible aux processus, de valeur 1 ou 0 suivant que la ressource est occupée ou non; un processus doit consulter p pour pouvoir entrer en section critique, et remettre p à 0 en sortant. En général, la consultation de p se fait à l'aide d'une instruction élémentaire et d'une boucle de test.

Si l'on associe une file d'attente f(p) à la variable précédente, elle est appelée un verrou. Si un processus ne peut entrer en section critique, il entre dans la file d'attente; lorsqu'un processus sort de la section critique, un des processus de la file d'attente est activé, si celle-ci n'est pas vide; il est inutile d'activer tous les processus à la fois car un seul pourra entrer en section critique. La valeur initiale de p est 0. Il existe deux primitives, verrouiller(p) et déverrouiller (p), qui sont utilisées par les processus comme seul moyen d'accès à la ressource.

Un sémaphore est une généralisation des verrous dans laquelle la variable p peut prendre des valeurs entiers quelconques. Un sémaphore s est constitué d'une variable entière e(s) et d'une file d'attente f(s), il est créé par une déclaration qui doit

spécifier la valeur initiale e0(s) de e(s). Cette valeur est nécessairement un entier non négatif. A la création d'un sémaphore, sa file f(s) est toujours initialement vide. On ne peut agir sur un sémaphore s que par les deux primitives, P(s) et V(s), qui sont des opérations indivisibles. La primitive P sert à demander la ressource et la primitive V à la libérer. L'exclusion mutuelle se résout comme suit: on introduit un sémaphore "mutex", initialisé à 1 et chaque processus exécute selon la séquence: <u>début</u> P(mutex); section critique; V(mutex); suite d'instructions; <u>fin</u>.

Les solutions précédentes ne garantissent pas à tout processus une entrée en section critique au bout d'un temps fini: si la file d'attente comporte des priorités, un processus de basse priorité risque d'attendre longtemps, voire indéfiniment. Par contre, si la file est gérée dans l'ordre des arrivées, tout processus entre nécessairement en section critique au bout d'un temps fini.

2.2.3 <u>Synchronisation</u>

Le problème de synchronisation consiste à construire un mécanisme, indépendant des vitesses, permettant à un processus actif (soit p):

- d'en bloquer un autre ou de se bloquer lui-même en attendant un signal d'un autre processus
- d'activer un autre processus (soit q) en lui transmettant éventuellement de l'information. Si q se trouve déjà dans l'état actif, il y a deux possibilités de l'effet de cette opération.
  a) le signal d'activation n'est pas mémorisé, et par conséquent il est perdu si le processus q ne l'attend pas
  b) le signal est mémorisé et le processus q ne se bloquera pas lors de la prochaine opération de blocage concernant ce processus.

Si l'identité des processus est un paramètre de l'opération d'activation (ou de blocage), la synchronisation est dite <u>directe</u>, si l'identité des processus visés peut être inconnue du processus agissant, la synchronisation est alors <u>indirecte</u>. Dans ce dernier cas, la synchronisation met en jeu un ou plusieurs objets intermédiaires connus des processeurs coopérants, et manipulables par eux uniquement à travers des opérations indivisibles spécifiques. Ces objets portent les noms <u>d'événements</u> ou de <u>sémaphores</u> suivant la nature des opérations qui permettent de les manipuler.

Dans un langage de programmation évolué un <u>événement</u> est représenté par un identificateur; il est créé par une déclaration qui fixe sa portée en tant qu'objet du langage. De plus un événement ne peut être manipulé que par le (ou les) processus qui y ont accès, ou bien il peut être déclenché. En outre, un événement peut être mémorisé ou non mémorisé.

On peut imaginer que l'activation d'un processus soit associée à des entités plus complexes qu'un simple évènement, par exemple à l'occurrence conjointe de deux évènements ou à l'occurrence de l'un ou l'autre de deux évènements et plus généralement à une expression booléenne d'évènements.

Le mécanisme de sémaphores est utilisé pour résoudre des problèmes généraux de synchronisation: un signal d'activation est envoyé par une primitive V, il est attendu par une primitive P. Un sémaphore s est un sémaphore privé d'un processus p si seul ce processus peut exécuter l'opération P(s); les autres processus ne peuvent agir sur s que par V(s). Ainsi un processus dont l'évolution est subordonnée à l'émission d'un signal par un autre processus se bloque, au moyen d'une primitive P, derrière son sémaphore privé initialisé à zéro. Le signal de réveil de ce processus bloqué est obtenu en faisant exécuter par un autre processus une opération V sur le même sémaphore.

Le sémaphore apparaît comme un mécanisme de synchronisation suffisament général pour permettre, à la difference des mécanismes précédents, de mémoriser un nombre quelconque d'activations éventuelles alors que le processus auquel elles sont destinées se trouve encore à l'état actif. Le problème classique des philosophes et des spaghetti [Di 71] peut être resolu à l'aide de cette technique de synchronisation.

2.2.4 Communication

La coopération de plusieurs processus à l'exécution d'une tâche commune nécessite en général une communication d'information entre ces processus. Cette information peut aller de la simple autorisation ou interdiction de continuer l'exécution jusqu'à l'accès à un ensemble de variables globales constituant un univers commun. Les problèmes de sécurité qui se posent mènent à la définition d'un contrôle et des mécanismes spéciaux de communication.

Le modèle du producteur et du consommateur illustre bien les principaux problèmes de la communication entre processus par accès à des variables communes avec synchronisation. On considère deux processus, le producteur et le consommateur, qui se communiquent de l'information à travers une zone de mémoire, dans les conditions suivantes:

- l'information est constituée par des messages de taille constante,
- aucune hypothèse n'est faite sur les vitesses respectives des deux processus.

La zone de mémoire commune a une capacité fixe de n messages (n>0). On souhaite que la communication se déroule suivant les règles:

- l'exécution mutuelle au niveau du message: le consommateur ne peut prélever un message que le producteur est en train de ranger;

- le producteur ne peut pas placer un message dans la zone mémoire si celle-ci est pleine; le producteur doit alors attendre;
- le consommateur doit prélever tout message une fois et une seule;
- si le producteur est en attente parce que la zone mémoire est pleine, il doit être prévenu dès que cette condition cesse d'être vraie; il en est de même pour le consommateur et la condition "zone mémoire vide".

La solution classique à l'aide de sémaphores est très connue. La communication entre processus suivant le schéma dit de la boîte aux lettres est une application directe de ce dernier modèle, où la zone mémoire est la boîte aux lettres.

Parmi les mécanismes spéciaux de communication, on trouve les sémaphores avec messages dont le principe découle de la remarque suivante: lorsqu'un processus p active un processus q par une opération V sur son sémaphore privé, cette activation accompagne souvent une transmission d'information. On peut dès lors songer à inclure cette transmission d'information dans le mécanisme même des primitives de synchronisation.

2.2.5 Contrôle

Lorsqu'il y a un ensemble de processus qui coopèrent à la réalisation d'une tâche commune, ou qui sont en compétition pour l'acquisition de ressources, la nécessité d'un contrôle pour la synchronisation et la communication s'impose. L'entité abstraite à laquelle sont reliés les processus à synchroniser est appelée contrôleur. L'ensemble forme le système de processus (AHV 80).

Si l'on considère des systèmes où l'on peut faire abstraction de la durée des intervalles de temps entre deux événements, chaque processus est représenté par une succession d'événements et la synchronisation dite logique consiste à mettre en concordance plusieurs processus ayant atteint chacun un événement donné. Cette mise en concordance est ponctuelle, les processus reprennent leur propre vitesse ou cadencement jusqu'au prochain point de synchronisation.

Les règles de synchronisation visent deux objectifs complémentaires:

a) assurer un fonctionnement cohérent du système (usage légal des ressources)
b) assurer l'égalité des différents processus (équité), ou au contraire instaurer des privilèges (priorité) et donc des problèmes d'ordonancement apparaissent.

La synchronisation peut être exprimée de façon abstraite à l'aide de langages formels spécifiés soit par un générateur de mots (grammaire), soit par un accepteur (automate). Dans ce sens, un contrôleur est défini par un quadruplet (E, S, C, G)

où:

- E est l'ensemble d'événements significatifs ou des points de synchronisation
- S est l'ensemble de variables d'état
- C:ExS -> {V, F} est une fonction booléenne qui détermine les contraintes de synchronisation
- G est un ensemble de fonctions de gestion de S

Un avantage d'avoir une expression abstraite des problèmes de synchronisation consiste à pouvoir établir des propriétés du système ou effectuer des preuves concernant le comportement du système sous ces contraintes, en particulier le non blocage du système et l'inexistence d'attente infinie pour chaque processus.

La <u>trace temporelle</u> d'un processus p est un mot sur E* qui enregistre les occurrences successives des actions de p. Un contrôleur perçoit les occurrences des évènements significatifs pour la synchronisation et dispose de moyens lui permettant de retarder les processus de manière à ce que la trace temporelle effective soit légale.

Les éléments de l'ensemble E, qui dans l'abstrait ont une durée nulle, vont se traduire, au niveau de la mise en oeuvre, par des actions exécutées par le contrôleur.

Ainsi, pour un point de synchronisation A, il y a lieu de distinguer:

- l'arrivée du processus au point A, qui se traduit par une sollicitation du contrôleur, que l'on appelle une <u>requête</u> ; ce dernier, en général, mémorise les requêtes
- <u>l'autorisation</u> que donne le contrôleur au processus de reprendre son exécution qui correspond au franchissement réel du point de synchronisation
- un certain nombre d'actions effectuées par le contrôleur sur ses variables internes qui peuvent précéder et/ou suivre l'autorisation

Les évènements concrets qui correspondent aux évènements de l'expression abstraite sont donc les autorisations et on peut dire qu'un contrôleur met en oeuvre une expression abstraite de synchronisation si la trace temporelle des autorisations correspond à un mot du langage spécifié.

Le contrôle peut être centralisé ou réparti sur des architectures centralisées ou réparties.

2.2.5.1 <u>Systèmes centralisés</u>: Dans ce cas, il y a un seul exemplaire de chaque variable de S et toutes les variables sont regroupées dans une mémoire unique. Ceci implique un problème <u>d'exclusion mutuelle</u> entre séquences d'instructions accédant aux mêmes variables. L'utilisation de sémaphores peut résoudre le problème, mais lorsque l'on programme avec des sémaphores, rien n'interdit d'utiliser une variable de synchronisation en

dehors d'une séquence en exclusion mutuelle. L'utilisation correcte des variables partagées ne repose que sur une "bonne" programmation, qui introduit les opérations adéquates sur sémaphores pour contrôler les accès aux variables.

Le concept de <u>région critique</u>, introduit en [Ho 72] et [BH 72], remédie à cet inconvénient. Une région critique définit une suite d'instructions, exécutée en exclusion mutuelle et relative à l'utilisation d'une variable déclarée explicitement comme partagée.

On trouve un raffinement supplémentaire dans le concept de <u>moniteur</u> [Ho 74], [BH 73]. Il s'agit d'une entité syntaxique (un module) qui regroupe les variables de synchronisation et les ressources partagées avec les procédures qui les manipulent. On remédie aussi à la dispersion, dans le code des processus, des opérations d'accès aux variables partagées. Par définition, les variables déclarées dans le moniteur ne sont accessibles que par les procédures du moniteur. Parmi ces procédures, certaines sont internes aux moniteurs, d'autres peuvent être appelées de l'extérieur, elles constituent les <u>points d'entrée</u> du moniteur.

2.2.5.2 <u>Systèmes répartis</u>: Un système réparti est défini comme un ensemble de <u>sites</u> connectés. Chaque site possède une mémoire privée et il n'y a pas de mémoire commune. On suppose que:

a) Tout site peut communiquer directement avec tout autre site (maillage logique)
b) Il n'y a pas d'erreurs de transmission
c) Entre tout couple de sites i et j, l'ordre dans lequel i envoie des messages à j est le même que celui dans lequel j les reçoit. En particulier, il n'y a pas de perte de messages
d) La panne (ou l'isolement physique, ce qui revient au même) d'un site est détectée et signalée aux sites qui tentent de communiquer avec lui.

Dans ce cas, comme dans le précédent, il y a deux solutions envisageables pour l'implantation d'un automatisme abstrait de synchronisation: contrôle centralisé ou contrôle réparti.

Le contrôle <u>centralisé</u> consiste à regrouper sur un seul site les variables et les instructions nécessaires à la synchronisation. L'exécution est réalisée par un processus <u>contrôleur unique</u>, auquel les autres processus situés sur les différents sites envoient des messages. On se trouve alors exactement dans la situation d'un système centralisé (pas de parallélisme, fiabilité reposant sur le site privilegié, ...), avec les mêmes possibilités de programmation.

Le contrôle <u>réparti</u> sur les différents sites revient à considérer N processus contrôleurs (un sur chaque site). Pour répartir les variables deux cas sont possibles: certaines variables seront distribuées, c'est-à-dire elles existeront

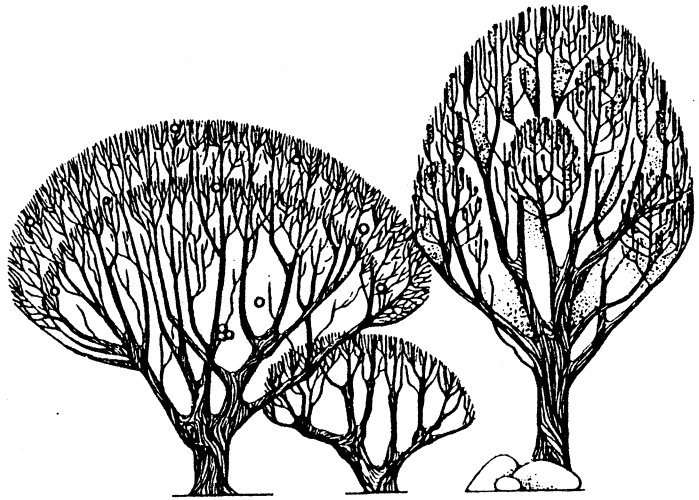
chacune en un seul exemplaire sur un site différent selon les variables, d'autres seront multipliées (plusieurs exemplaires de la même variable sur divers sites et actualisation régulière).

L'exclusion mutuelle est une technique courante dans le contrôle centralisé mais dans le cas réparti on trouve d'autres possibilités. Par exemple, appliquer localement des contraintes de synchronisation suffisament fortes pour que le comportement global du sytème soit correct, ou des algorithmes qui permettent de mettre en oeuvre une datation (logique) unique des évènements du système de sorte que, les contraintes de synchronisation exprimées en termes d'ordre total absolu soient mises en oeuvre en assimilant cet ordre à celui fourni par les dates logiques.

Chapitre III

STAR_PALE

Système de Transformation d'ARborescences en PArallèLE



3.1 Introduction

La traduction d'un texte s'effectue presque toujours en plusieurs étapes, le plus souvent par différentes personnes: une traduction brute suivie d'une ou plusieurs révisions. L'informatique peut être utilisée, soit pour aider l'homme (dans les deux phases), soit pour le remplacer (pour la traduction brute seulement, la révision automatique restant pour l'instant inaccessible).

La traduction automatique aidée par l'homme, ou plus simplement la traduction automatisée, est une méthode dans laquelle il n'y a pas d'interaction homme-machine au cours de la traduction brute, mais certaines parties du texte peuvent être refusées par le système automatique, et donc doivent être traduites par l'homme. C'est le cas des systèmes METEO, TAUM-AVIATION (Montréal), et des systèmes du GETA (Grenoble) et du SFB100 (Saarbrücken). La traduction brute obtenue est révisée si la qualité demandée l'exige. [Co 70], [Ch 74], [TA 79]

Les outils de programmation utilisés pour l'implémentation des systèmes d'automatisation de la traduction ont évolué depuis le début des recherches dans ce domaine. On est passé des programmes écrits en langage machine aux systèmes formels qui appliquent un algorithme unique sur une entrée et un ensemble non structuré des règles qui n'ont aucun contrôle sur cet algorithme. Ensuite, il y a eu la création de modèles où les règles peuvent comporter des contrôles sur l'algorithme général, qui simule toujours un modèle non-déterministe. [Va 75], [Bo 76]

Aujourd'hui on reconnait, après une vingtaine d'années d'experiences, la nécessité de logiciels de base de très haut niveau conçus spécialement pour la traduction automatisée, avec des types de données puissants et adaptés (chaînes, arbres, graphes et réseaux décorés), avec des opérateurs spécifiques, ainsi que des primitives de contrôle pour la programmation non-déterministe, parallèle et heuristique. Il est important que les opérateurs complexes associés aux types de données de base soient adéquats (c'est-à-dire qu'ils doivent permettre d'écrire des processus linguistiques de façon aisée et concise), et que leur complexité soit bornée. La structure de contrôle de leurs appels doit être fondée sur un modèle décidable, de sorte que l'on puisse localiser statiquement les éventuelles sources d'indécidabilité. [Bo 81]

Nous avons voulu faire une approche d'un système de traduction automatisée de ce type sur la base d'une application extensive du parallélisme. Il s'agit en effet d'une extension de ROBRA, un des quatre langages spécialisés pour la programmation linguistique du logiciel de base du laboratoire GETA. Le système proposé est une extension dans deux sens: il permet un contrôle et un pouvoir d'expression plus détaillé et d'autre part il permet la réalisation de beaucoup plus de travaux en

parallèle grâce à la suppression d'un certain nombre de restrictions.

Nous n'avons pas l'intention de donner une description détaillée d'un système de réécriture général mais au moins de décrire les éléments de base et les lignes générales qui, à nctre avis, seraient souhaitables pour une réalisation particulière. Dans une première partie, nous montrons les structures de données qui sont dans un certain sens parallèles. Etant donné que la structure d'arbre apparaît dans de nombreux systèmes de représentation métalinguistique (arbres syntagmatiques, arbres de dépendance, arbres applicationnels, arbres projectifs, ...), ainsi que presque partout en informatique, en particulier dans les langages de programmation, et en algèbre, nous avons cherché une structure pour la manipulation d'arbres qui permette de factoriser une multiplicité de résultats arborescents. Nous définissons plusieurs fonctions et propriétés sur cette classe d'objets qui peuvent être utiles au système.

Dans une deuxième partie, nous présentons les règles et grammaires pour la manipulation de ces structures. Les règles ont un haut degré implicite de parallélisme dans la phase de reconnaissance ainsi que dans la phase de transformation. L'expression de l'ordre relatif entre noeuds frères est raffinée pour permettre un contrôle des choix de patrons et d'images dans les règles. Nous définissons plus précisément le contexte d'une règle pour optimiser le parallélisme et éliminer le risque de destruction de la structure arborescente sur laquelle on travaille.

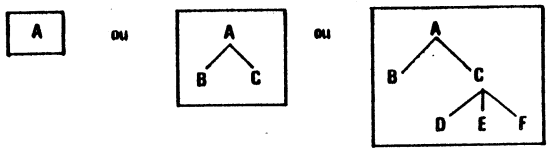
Finalement, nous présentons l'algorithme général d'application d'une grammaire à une structure ainsi que les moyens de composition de grammaires en vue de permettre le regroupement des règles pour des phénomènes linguistiques voisins lors de la programmation d'un système particulier par le linguiste.

3.2 Structures à Transformer

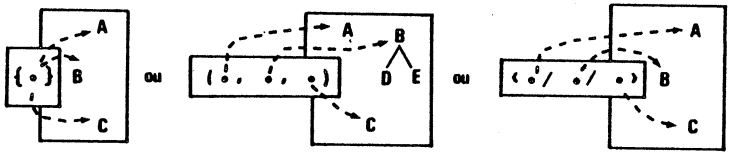
Dans cette section, nous définissons le type de structures manipulées par le système, et les fonctions et les prédicats associés en tant qu'opérateurs. L'idée est de définir une structure de données qui soit, en un certain sens, parallèle. Comme nous avons travaillé dans l'optique de systèmes de manipulation d'arbres, nous avons cherché une structure qui permette de factoriser une multiplicité de résultats arborescents.

3.2.1 Multi-arborescences à tranches

Les multi-arborescences à tranches (m.a.t.) sont des structures arborescentes qui représentent des familles d'arborescences "factorisées" d'une certaine façon. Dans le cas le plus simple, on a les arborescences classiques orientées telles que:

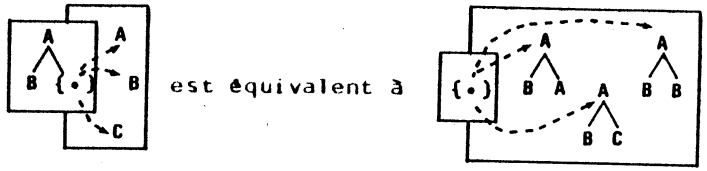


que l'on appelle à une tranche. S'il y a plus d'une tranche, il s'agit d'arborescences du type précédent, mais qui portent au moins une feuille rattachée à une deuxième tranche. Ces feuilles peuvent être de trois types: ensemble, liste ou piste. Si c'est un ensemble, on trouvera dans une deuxième tranche un ensemble d'au moins deux multi-arborescences, qui seront affectées à cette feuille sans ordre spécifique. Si c'est une liste, on aura une séquence non vide et ordonnée de multi-arborescences. Si c'est une piste, on aura une séquence non vide et ordonnée de multi-arborescences avec un élément distingué appelé trace, qui permet de mettre en relief un chemin dans la m.a.t. Voici trois exemples de multi-arborescence à deux tranches:



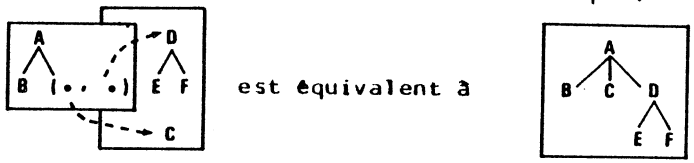
Ce sont respectivement un ensemble, une liste et une piste. La trace de cette piste est l'arborescence B.

Les multi-arborescences de type ensemble permettent d'exprimer d'une façon condensée un ensemble de choix possibles, par exemple:

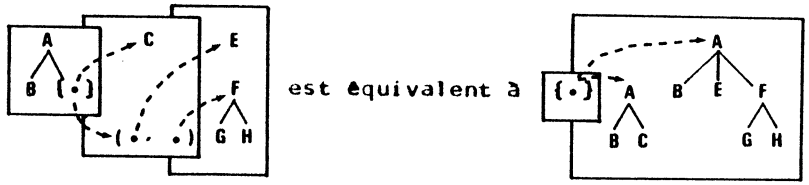


Cela est intéressant pour le traitement des ambiguïtés ou pour l'application des analyseurs "context-free" à programmation dynamique [Ea 70].

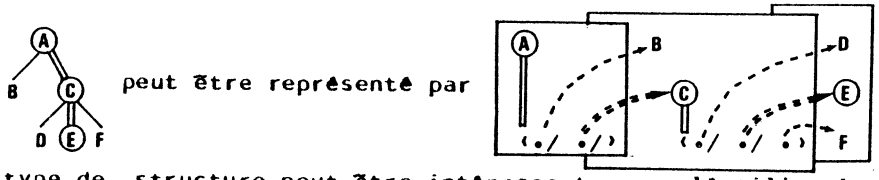
Les listes sont une manière commode de manipuler un sous-ensemble de fils d'un noeud. Par exemple,



Ou bien



Les pistes donnent un moyen de mettre en relief un chemin choisi dans une multi-arborescence. Par exemple, le chemin marqué par les doubles traits sur l'arborescence suivante:



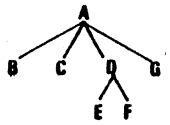
Ce type de structure peut être intéressant pour l'utilisation de grammaires projectives.

On accepte des multi-arborescences à plusieurs tranches mais

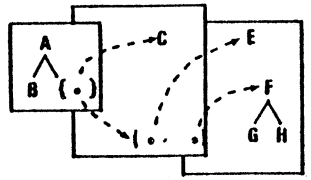
avec un nombre <u>fini</u> de noeuds, et par conséquent un nombre fini de tranches. Il n'y a pas de retour arrière entre les différents liaisons inter-tranches de sorte que les boucles sont interdites. Il y a donc deux types de noeuds dans une m.a.t.: les <u>noeuds simples</u> qui portent une décoration et ne font pas référence à d'autres tranches, et les <u>noeuds composés</u> qui ne portent pas de décoration et font référence à des m.a.t. dans la tranche suivante. Dans les exemples donnés, les lettres A, B, C, etc sont les décorations affectées aux noeuds. Le <u>contenu</u> d'un noeud simple est une décoration et celui d'un noeud composé est une m.a.t.

Afin de bien préciser la différence entre un noeud et son contenu, on donne un <u>nom</u> à chaque noeud d'une m.a.t. Ce nom sera un entier positif unique dans la m.a.t. complète. Chaque noeud d'une m.a.t. est défini par un nom <u>n</u> et un contenu <u>c</u>. Il sera noté [n c]. Ainsi, [1 ;A] est un noeud simple; [1 ([2 ;A], [3 ;B])] est un noeud composé ensemble; [1 ([2 ;A], [3 ;B])] est un noeud composé liste et [1 <[2 ;A]/ [3 ;B]/>] est un noeud composé piste. Cette convention nous permet de transcrire dans une écriture linéaire toute m.a.t., car une arborescence sera écrite sous la forme classique donnée par le parcours préordre, c'est-à-dire

A(B, C, D(E, F), G) à la place de



sachant que seules les feuilles peuvent être des noeuds composés. Par exemple:



peut être écrite comme

[1 ;A]([2 ;B],[3 (,]))
[4 ;C] [5 (,))
[6 ;E] [7 ;F]([8 ;G],[9 ;H])

ou sur une même ligne

[1 ;A]([2 ;B],[3 ([4 ;C],[5 ([6 ;E],[7 ;F]([8 ;G],[9;H])))]))

La syntaxe est la suivante:

<m.a.t> ::= <feuille> | <arbre>
<feuille> ::= <noeud simple> | <noeud composé>
<arbre> ::= <père> <fils>
<noeud simple> ::= [<nom unique> ; <décoration>]
<noeud composé> ::= <ensemble> | <liste> | <piste>
<père> ::= <noeud simple>
<fils> ::= (<liste de m.a.t.>)
<nom unique> ::= <entier positif
unique dans la structure complète>
<ensemble> ::= [<nom unique> (<m.a.t.> , <liste de m.a.t.>]
<liste> ::= [<nom unique> <fils>]
<liste de m.a.t.> ::= <m.a.t>
| <m.a.t> , <liste de m.a.t.>
<piste> ::= [<nom unique> < <traînée gauche>/

<trace> / <traînée droite> >]

<traînée gauche> ::= <m.a.t.> | vide

<traînée droite> ::= <m.a.t.> | vide

<trace> ::= <m.a.t.>

Une décoration est une structure de données générale qui peut être complètement manipulée et définie par l'utilisateur. Dans ce document, les décorations seront des variables simples de type numérique ou texte pour simplifier. L'ensemble des multi-arborescences à tranches, défini par la syntaxe précédente est augmenté d'un élément dit m.a.t. vide, et que l'on note ⊥. L'ensemble des multi-arborescences à tranches y compris ⊥, sera noté Mat. Une arborescence est une m.a.t. qui ne porte que des noeuds simples. Par exemple: Les multi-arborescences

[1 ;([2 ;A], [3 ;B])],

[1 ([2 ;A], [3 ;B])],

[1 ([3 ;B], [2 ;A])], sont toutes différentes car la première est un noeud simple et les deux autres sont deux listes qui n'ont pas le même ordre. La syntaxe des décorations n'est pas définie ici, mais on pourrait supposer des structures générales qui comprennent des m.a.t. comme dans cet exemple.

3.2.2 Propriétés

Nous appellerons propriétés un certain nombre de renseignements associés à chaque noeud d'une m.a.t. et relatifs à l'état de celui-ci. Elles seront connues au travers des fonctions qui dépendent de la m.a.t. à laquelle elles s'appliquent. Ainsi, pour toute m.a.t. a et tout entier n on a:

type (a,n) = [
- 'simple' si "n" est le nom d'un noeud simple de "a"
- 'ensemble' si "n" est le nom d'un noeud composé ensemble de "a"
- 'liste' si "n" est le nom d'un noeud composé liste de "a"
- 'piste' si "n" est le nom d'un noeud composé piste de "a"
- 'libre' si "n" n'est pas dans "a"

contenu (a,n) = [
- ;d si "n" est le nom d'un noeud simple [n ;d] de "a"
- {a1, a2, ..., am} si "n" est le nom d'un noeud composé ensemble [n {a1, a2, ..., am}] de "a"
- (a1, a2, ..., am) si "n" est le nom d'un noeud composé liste [n (a1, a2, ..., am)] de "a"
- <tg/ tr/ td> si "n" est le nom d'un noeud composé piste [n <tg/ tr/ td>] de "a"
- ⊥ si "n" n'est pas dans "a"

La propriété nélements donne la cardinalité d'une m.a.t., i.e. le nombre d'arbres ou de noeuds simples qui sont rattachés directement ou indirectement au noeud "n" de "a".

<u>nélements</u> (a,n)= [1 si "n" est le nom d'un noeud simple de "a"

<u>nélements</u> (a',n) si n est le nom d'un noeud piste {n <tg/ tr/ td>} de "a", et a' est la liste {n (tg, tr, td)}

p + <u>nélements</u> (a, naj1) + ... + <u>nélements</u> (a, najq)
1) si "n" est un noeud ensemble {n {a1, a2, ..., am}} contenant q m.a.t. distinctes {aj1, ..., ajq}, p+q=m, ∀ i ∈ [1, q] aji est un noeud ensemble de nom naji
2) ou bien si "n" est un noeud liste {n (a1, a2, ..., am)} contenant q m.a.t. distinctes {aj1, ..., ajq}, p+q=m, ∀ i ∈ [1, q] aji est un noeud piste ou liste de nom naji

0 si "n" n'est pas dans "a"

On remarque l'associativité particulière de cette propriété pour les ensembles et les listes. On peut voir par exemple que

nélements({1{a1,a2,a3}},1)=nélements({1{{2{a1,a2}},a3}},1)

nélements({1(a1,a2,a3)},1)=nélements({1({2(a1,a2)},a3)},1)

nélements({1{a1,a2,a3}},1)≠nélements({1{{2(a1,a2)},a3}},1)

Cela est dû au fait que les ensembles représentent des choix possibles et les listes ne représentent pas de choix mais une séquence fixe de fils.

La propriété <u>arbs</u> donne le nom du k-ième arbre, noeud simple ou nceud ensemble rattaché directement ou indirectement au noeud "n" de "a" au travers des noeuds liste ou piste. La propriété <u>narbs</u> donne le cardinal du noeud n de a.

<u>narbs</u> (a,n)= [1 si "n" est le nom d'un noeud ensemble de "a"

<u>nélements</u> (a,n) sinon

<u>arbs</u> (a, k, n)= [n si k=1 et "n" est le nom d'un noeud simple ou ensemble de "a"

<u>arbs</u> (a', k, n) si "n" est le nom d'une piste {n <tg/ tr/ td>} de "a", et a' est la liste {n (tg, tr, td)}

<u>arbs</u> (a, k', na i') si "n" est le nom d'une liste {n (a1, a2, ..., am)} et 1 ≤ k ≤ <u>narbs</u> (a, n), où:
∀ j ∈ [1, m] naj est le nom de aj,
i' = p i ≤ m (<u>narbs</u> (a, na1) +...+ <u>narbs</u> (a, nai)) ≥k), et
k' = k -(<u>narbs</u> (a, na1) +...+ <u>narbs</u> (a, nai'-1))

0 si "n" n'est pas dans "a"

On voit qu'un ensemble compte pour un seul "arbre".

Les propriétés <u>fils-apparent</u>, <u>nfils-apparents</u> donnent respectivement, le nom des fils et le nombre de fils qui se trouvent sur une même tranche. Ils seront appelés fils apparents.

<u>nfils-apparents</u> (a, n)= [m si "n" est le nom d'un noeud simple <u>p</u> père d'un arbre a' = p (f1, f2, ..., fm) contenu dans "a"

0 sinon

<u>fils-apparent</u> (a, k, n)= [<u>nfk</u> si "n" est le nom d'un noeud simple <u>p</u> père d'un arbre a' = p (f1, f2, ..., fm) contenu dans "a", 1 ≤ k ≤ m et ∀ i ∈ [1, m] nf i est le nom de fi

0 sinon

Les propriétés <u>fils</u>, <u>nfils</u> donnent respectivement, le nom des

fils d'un noeud simple et le nombre de fils qui se trouvent sur une même tranche ou sur d'autres tranches. Ils seront appelés fils véritables.

<u>nfils</u> (a, n)= [r + narbs(a, nfj1) + ... + narbs(a, nfjq)
si "n" est le nom d'un noeud simple <u>p</u> père d'un arbre a' = p (f1, f2, ..., fm) contenu dans "a" tel qu'il porte r fils apparents de type ensemble, q fils apparents {fj1, ..., fjq},
∀ i ∈ [1, q] fji de nom nfji, qui ne sont pas des noeuds ensemble, avec r + q= m

0 sinon

<u>fils</u> (a, k, n)= [arbs(a, k', nf i') si "n" est le nom d'un noeud simple <u>p</u> père d'un arbre a' = p (f1, f2, ..., fm) contenu dans "a", et 1 ≤ k ≤ <u>nfils</u> (a,n), où:
∀ j ∈ [1, m] nfj est le nom de fj,
i' = p i ≤ m (<u>narbs</u> (a, nf1) +...+ <u>narbs</u> (a, nfi)) ≥k), et
k' =k -(<u>narbs</u> (a, nf1) +...+ <u>narbs</u> (a, nfi'-1))

0 sinon

Il est facile de voir que pour toute arborescence "a"
∀n,k [nfils(a, n)= nfils-apparents(a, n) & fils(a, k, n)= fils-apparent(a, k, n)].

La propriété appelée <u>niveau</u>, qui correspond au niveau des nceuds d'une arborescence classique, est étendue aux m.a.t. Le niveau d'un noeud est sa distance par rapport à la racine de la m.a.t., en comptant 1 pour les arcs à l'intérieur d'une même tranche et zéro pour les arcs joignant deux tranches.

<u>niveau</u> (a, n)= [<u>niveau</u> (a, n') si n' est le nom d'un noeud ensemble [n' {a1, a2, ..., am}], liste [n' {a1, a2, ..., am}] ou piste [n' <tg/ tr/ td>] de "a" tel qu'il existe i ∈ [1, m] et le nom de ai est n ou le nom de tg, tr ou td est n

<u>niveau</u> (a, n') +1 si n' est le nom d'un noeud simple <u>p</u> de "a" et il existe un k tel que fils(a, k, n')= n ou fils-apparent(a, k, n')= n

0 sinon

La propriété <u>tranche</u> détermine la tranche à laquelle appartient un noeud.

<u>tranche</u> (a, n)= [1 si niveau(a, n)=0

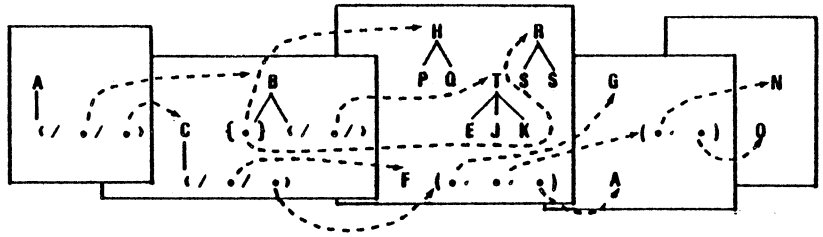
<u>tranche</u> (a, n') si "n'" est le nom d'un noeud simple <u>f</u> de "a" tel que ∄ k fils-apparent(a, k, n')= n

<u>tranche</u> (a, n')+1 si "n'" est le nom d'un noeud ensemble [n' {a1, ..., am}], liste [n' (a1, ..., am)] ou piste [n' <tg/ tr/ td>] de "a" tel que ∄ i ∈ [1, m] le nom de ai est n ou le nom de tg, tr ou td est n

0 sinon

Ces propriétés ne sont pas modifiables directement par l'utilisateur. D'autres propriétés seront définies ultérieurement.

<u>Exemple</u> : Voici les propriétés de la m.a.t. suivante




```
tranche1:[1;A]([2</
tranche2:          [3;B]([4(                    ,                    )),[11</
tranche3:                    [5;H]([6;P],[7;Q]) [8;R]([9;S],[10;S])        [12;T]([13;E],

tranche1:                  /                                                          >))
tranche2:              />)) [16;C]([17</      /                                     >))
tranche3:[14;J],[15;K])                  [18;F] [19(      ,                  ,      ))
tranche4:                                          [20;G] [21(      ,      )) [24;A]
tranche5:                                                   [22;N] [23;O]
```

nom	décoration	type	nélements	narbs	arbs	nfilsA	filsA	nfils	fils	niveau	tranche
1	A	simple	1	1	1	1	2	2	3,16	0	1
2		piste	2	2	3,16	0				1	1
3	B	simple	1	1	3	2	4,11	2	4,12	1	2
4		ensemble	2	1	4	0				2	2
5	H	simple	1	1	5	2	6,7	2	6,7	2	3
6	P	simple	1	1	6	0				3	3
7	Q	simple	1	1	7	0				3	3
8	R	simple	1	1	8	2	9,10	2	9,10	2	3
9	S	simple	1	1	9	0				3	3
10	S	simple	1	1	10	0				3	3
11		piste	1	1	12	0				2	2

12	T	simple	1	1	12	3	13,14,15	3	13,14,15	2	3
13	E	simple	1	1	13	0				3	3
14	J	simple	1	1	14	0				3	3
15	K	simple	1	1	15	0				3	3
16	C	simple	1	1	16	1	17	5	18,20,22,23,24	1	2
17		piste	5	5	18,20,22,23,24	0				2	2
18	F	simple	1	1	18	0				2	3
19		liste	4	4	20,22,23,24	0				2	3
20	G	simple	1	1	20	0				2	4
21		liste	2	2	22,23	0				2	4
22	N	simple	1	1	22	0				2	5
23	O	simple	1	1	23	0				2	5
24	A	simple	1	1	24	0				2	4

□

3.2.3 Relation d'équivalence et d'ordre

Comme on l'a déjà souligné, deux m.a.t. qui ont la même structure et qui ne diffèrent que par leurs noms de noeud sont fortement équivalentes, ainsi

{ 1;A}({2;B},{3{{ 4;C},{5({ 6;E},{ 7;F}({8;G},{9;H}))}}}) et

{61;A}({1;B},{7{{30;C},{2({14;E},{88;F}({5;G},{9;H}))}}})

sont fortement équivalentes. Mais, le fait de permettre une numérotation des noeuds pose quelques problèmes dans le cas des ensembles, car on doit accepter par exemple que

{1{{2;P},{3;Q},{4;R},{5;P},{6;Q},{7;R}}} et

{1{{2;P},{3;Q},{4;R}}}

soient fortement équivalentes, étant donné qu'elles ont toutes les deux la structure {P, Q, R}. Sans la numérotation des noeuds, on ne peut pas accepter de répétitions dans un ensemble.

On veut aussi que deux m.a.t. soient fortement équivalentes par rapport aux fils véritables de chaque noeud, par exemple

{1;A}({2({3;P},{4;Q})},{5<{6;A}/{7;B}/>})}) et

{1;A}({2;P},{3;Q},{4;A},{7;B})

sont fortement équivalentes.

On arrive donc à la définition suivante; soient a1 et a2 deux

multi-arborescences à tranches, on dit que a1 et a2 sont fortement équivalentes et l'on note $a1 \equiv a2$, si et seulement si l'une des conditions suivantes est vérifiée:

1. a1 = a2
2. a1 est une feuille noeud simple [n1; d1], a2 est une feuille noeud simple [n2; d2] et d1=d2
3. a1 est un arbre p1 (f11, f12, ..., f1m1), a2 est un arbre p2 (f21, f22, ..., f2m2), p1=[n1 ;d1]≡[n2 ;d2]=p2, nfils(a1, n1)= nfils(a2, n2) et ∀ k ∈ [1, nfils(a1, n1)] si a1k est la m.a.t. de a1 de nom fils(a1,k,n1) et a2k est la m.a.t. de a2 de nom fils(a2,k,n2) alors a1k≡a2k
4. a1 est un noeud simple ou un arbre, a2 est un noeud ensemble [n2{a21, ..., a2m}] et ∀ k ∈ [1, m] a1≡a2k
5. a1 est un noeud ensemble [n1 {a11, a12, ..., a1 m1}], a2 est un noeud ensemble [n2 {a21, a22, ..., a2 m2}], <u>éléments</u> (a1) = [n1 {b11, ..., b1 m'1}], <u>éléments</u> (a2) = [n2 {b21, ..., b2 m'2}],

   ∀ i ∈ [1, m'1] ∃ j ∈ [1, m'2] b1i≡b2j

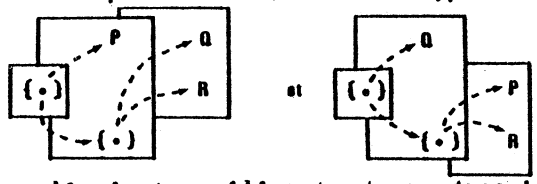
   ∀ j ∈ [1, m'2] ∃ i ∈ [1, m'1] b1i≡b2j

   où: m'1 = <u>néléments</u> (a1, n1), m'2 = <u>néléments</u> (a2, n2) et

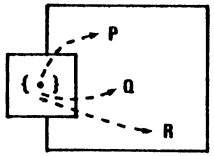
   <u>éléments</u> (a)= [a si a n'est pas un noeud ensemble
   <u>éléments</u> ([n1{a1, ..., ai-1, ai1, ..., aim'1, ai+1, ..., am}]) si a est un noeud ensemble [n1{a1, ..., am}] tel que ∃ i ∈ [1, m] : ai =[n'{ai1, ..., aim'}] est un noeud ensemble
   ([n1{ <u>éléments</u> (a1), ..., <u>éléments</u> (am)}]) si a est un noeud ensemble [n1{a1, ..., am}] dont aucun des ai est un noeud ensemble

6. a1 est un noeud liste [n1 (a11, a12, ..., a1 m1)], a2 est un noeud liste [n2 (a21, a22, ..., a2 m2)], narbs(a1, n1)=narbs(a2, n2)=k, ∀ i ∈ [1, k] arbs(a1, i, n1)≡arbs(a2, i, n2)
7. a1 est une piste [n1 <tg1/ tr1/ td1>], a2 est une piste [n2 <tg2/ tr2/ td2>], [n1(tg1, tr1, td1)]≡[n2(tg2, tr2, td2)]
8. a1 est une piste [n1 <tg/ tr/ td>], a2 est une liste [n2 (a21, a22, ..., a2m2)], [n1(tg, tr, td)]≡a2

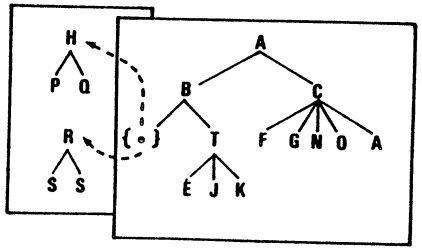
La cinquième condition mérite une explication. Il s'agit de permettre l'identification d'une équivalence forte lorsqu'on a par exemple , deux m.a.t. du type



car l'opérateur éléments donne dans les deux cas



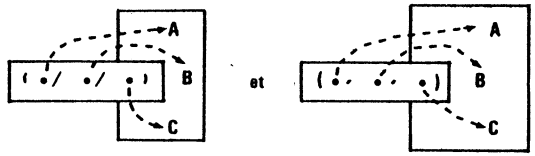
La relation d'équivalence forte nous permet, par exemple, de mettre dans la même classe la m.a.t. de l'exemple donné à la section 3.2.2 et une m.a.t. de structure suivante:



Chaque classe d'équivalence forte sera représentée par une m.a.t. de la classe qui contient le nombre minimum de noeuds et qui porte une numérotation canonique. L'opérateur <u>nnoeuds</u> définit plus précisément le nombre de noeuds d'une m.a.t.

<u>nnoeuds</u> (a)= [
- 1 si a est une feuille noeud simple [n ;d]
- 1 + <u>nnoeuds</u> (f1) + ... + <u>nnoeuds</u> (fm) si a est un arbre p(f1, f2, ..., fm)
- 1 + <u>nnoeuds</u> (a1) + ... + <u>nnoeuds</u> (am) si a est un noeud ensemble [n {a1, ... ,am}] ou liste [n(a1, a2, ..., am)]
- 1 + <u>nnoeuds</u> (tg)+ <u>nnoeuds</u> (tr)+ <u>nnoeuds</u>(<u>td</u>) si a est un noeud piste [n <tg/ tr/td>]

Cette numérotation implique le choix d'un ordre sur l'ensemble des décorations, ce qui est toujours possible via un ordre lexicographique induit à partir d'un vocabulaire de base. Il y a un cas où l'on peut trouver un conflit comme une conséquence de la condition 8. Par exemple,

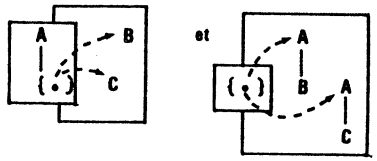


appartiennent à la même classe <u>et</u> ont le même nombre de noeuds. Dans ces cas, on choisit comme représentant le noeud <u>liste</u>. Le représentant d'une classe d'équivalence forte sera appelé structure multi-arborescente à tranches (s.m.a.t.) et l'ensemble de toutes les s.m.a.t. sera noté <u>Smat</u>

Par conséquent, dans une s.m.a.t.:

- il n'existe pas de noeud ensemble [n {a1, ..., am}] tel que $\nexists\ i,j \in [1, m]\ i \neq j\ \&\ ai \equiv aj$
- il n'y a pas de noeud liste de niveau supérieur à 0
- il n'y a pas de noeuds piste

L'ensemble Smat peut être muni d'une relation d'ordre partiel qui permet de rapprocher les s.m.a.t. qui ont une "structure de factorisation" proche, par exemple

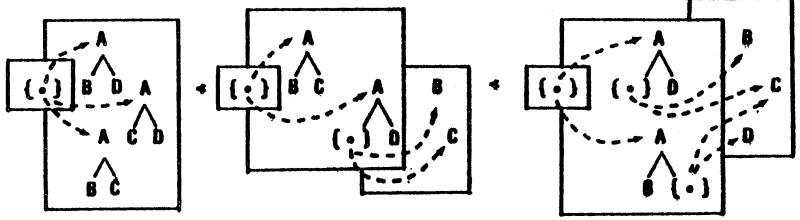


On voit que les deux représentent le fait qu'on a le choix entre deux arborescences de racine A, avec un seul fils chacune: dans un cas B et dans l'autre C. Mais la première m.a.t. factorise A et la deuxième non.

On définit une relation d'ordre partiel, que l'on note $\leq$, sur Smat comme suit: Soient a1 et a2 deux s.m.a.t., a1$\leq$a2 si et seulement si l'une des conditions suivantes est vérifiée:

1. a1 = {n1(a11, a12, ..., a1m)} et a2 = {n2(a21, a22, ..., a2m)} sont deux noeuds liste tels que
   $\forall$ i $\in$ [1, m]: (a1i $\leq$ a2i ou a1i = a2i)
2. a1 = p1 (f11, f12, ..., f1m) et a2 = p2 (f21, f22, ..., f2m) sont deux arbres tels que
   p1 = p2 et $\forall$ i $\in$ [1, m]: (f1i $\leq$ f2i ou f1i = f2i)
3. a1 = {n {b11, b12, ..., b1m}} est un noeud ensemble tel que
   $\forall$ i $\in$ [1, m]: (b1i $\leq$ a2 ou b1i = a2)
4. a2 = {n {b21, b22, ..., b2m}} est un noeud ensemble tel que
   $\exists$ i $\in$ [1, m]: (a1 $\leq$ b2i ou a1 = b2i)

Exemple : Cette relation donne l'ordre suivant aux trois s.m.a.t.



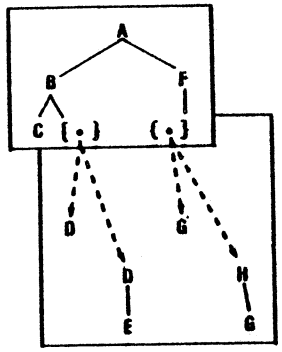
Il y aura évidemment deux noeuds avec décoration C dans la dernière m.a.t., il ne s'agit ici que d'une convention graphique. Cela est imposé par l'affectation d'un nom unique à chaque noeud d'une m.a.t.

□

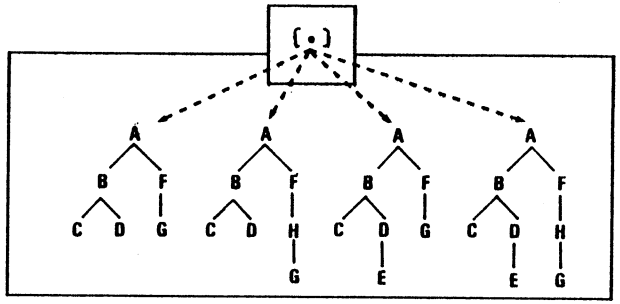
3.2.4 Opérations sur les Arborescences

On définit quelques opérations qui seront utilisées de façon directe ou indirecte par l'utilisateur. Les opérateurs nnoeuds, et éléments ont déjà été définis dans la section précédente.

3.2.4.1 Eclatement : Une m.a.t. représente une famille d'arbres qui ont été "factorisés" d'une certaine façon. Un éclatement consiste à retrouver d'une façon explicite cette famille par une minimisation du nombre de tranches. Par exemple, l'éclatement de:



est égal à



Pour une s.m.a.t. "a", l'opérateur d'éclatement, "E", est le suivant:

E (a)= [b la s.m.a.t. telle que,
b ⊂ a et b= [n {b1, ... ,bm}]
et ∀ i ∈ [1, m] bi est une arborescence,
si a porte au moins un noeud ensemble

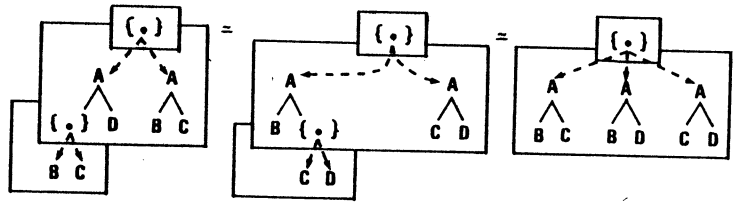
a sinon

Par conséquent, E(a) est une s.m.a.t. à une ou deux tranches.

On introduit une relation d'équivalence, dite faible, qui permet de préciser cette notion de famille d'arbres. Il s'agit d'une extension de la relation d'équivalence forte de 3.2.3.

Soient a1 et a2 deux s.m.a.t., on dit que a1 et a2 sont faiblement équivalentes, et l'on note a1≅a2, si et seulement si E(a1)≅E(a2).

Exemple :



□

3.2.4.2 Factorisation : L'opération inverse de l'éclatement permet de créer une s.m.a.t. "minimale" à partir d'une s.m.a.t. La minimalité s'entend par rapport à la fonction de coût K définie ci-dessous.

K(a)= [1 si a est une feuille noeud simple

K(a1) + ... + K(am)
si a= [n(a1, ... ,am)] est un noeud liste

1 + K(f1) + ...+ K(fm)
si a= p (f1, ..., fm) est un arbre

(K(a1).K(a2). ... .K(am))/(j+1)
si a=[n { a1, ... ,am }]
est un noeud ensemble et niveau(a,n)=j

On veut que les noeuds ensemble se trouvent dans les niveaux les plus grands possibles et qu'ils soient "bien garnis", c'est-à-dire que chaque m.a.t. contenue dans "a" porte le maximum d'informations concernant les différents choix au niveau des feuilles "apparentes". On remarque que la fonction de coût est supérieure ou égale a 1 pour toute s.m.a.t. a.

Propriété: Les s.m.a.t. à l'interieur d'une classe faible (Smat/=) forment un treillis pour la relation d'ordre $\sqsubseteq$.

Démonstration:

1. Soit S(a)={ a' ∈ Smat | a' = a }. Par définition
   ∀ a,a' ∈ S(a), E(a')=E(a) & E(a)$\sqsubseteq$a & E(a)$\sqsubseteq$a'

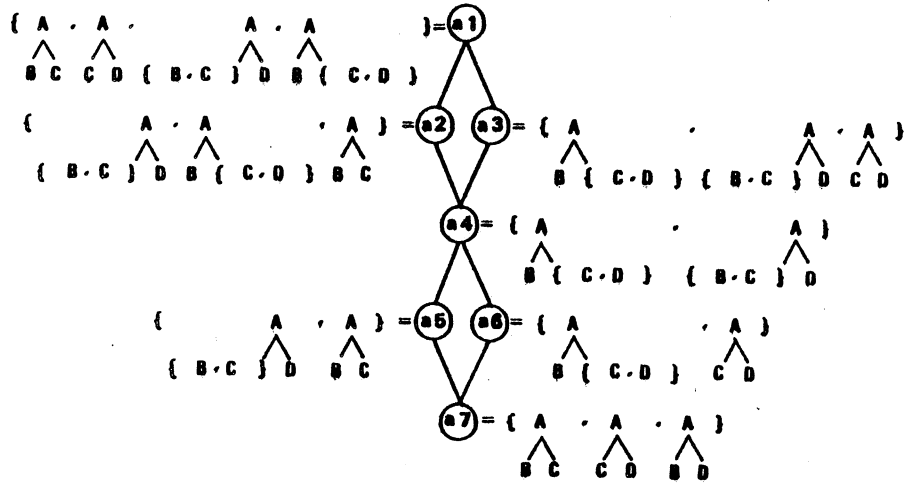
2. Soient a,a' ∈ S(a) tels que a$\not\sqsubseteq$a' alors
   ∃ b ∈ S(a): a$\sqsubseteq$b & a'$\sqsubseteq$ b, car

   a) si a$\sqsubseteq$a' alors b=a'

   b) si ¬(a$\sqsubseteq$a' ou a'$\sqsubseteq$a) alors
      soit a et a' sont deux noeuds ensemble et dans ce cas b est un noeud ensemble qui porte l'union des ensembles a et a';
      soit a est un noeud ensemble mais a' ne l'est pas et b est un noeud ensemble qui porte l'ensemble de a augmenté de a';
      soit dans les autres cas, b est le noeud ensemble {a,a'}
      si a et a' sont deux noeuds liste alors le même raisonnement s'applique pour tout i ∈ {1, n} entre a1i et a2i où a=(a11, a12, ..., a1n) et a'=(a21, a22, ..., a2n)

□

<u>Exemple</u> : Voici un de ces treillis et les coûts associés:



K(a1)=56.25, K(a2)=K(a3)=18.75, K(a4)=6.25, K(a5)=K(a6)=7.5, K(a7)=27

□

Pour une s.m.a.t. "a", l'opérateur de factorisation F est le suivant:

F(a)=[b une s.m.a.t. qui appartient à la classe d'équivalence faible de a, b=a, telle que ∀ b'=a K(b)≤ K(b')

Est-ce qu'elle est unique?. Je n'ai encore pu ni le démontrer d'une façon générale ni trouver de contre-exemple. La question reste ouverte.

3.2.4.3 <u>Extractions</u> : Nous définissons des opérateurs qui permettent de retirer une partie d'informations contenues dans une m.a.t. sans la détruire.

Pour un noeud "a" donné, on peut, à l'aide de l'opérateur nom, extraire son nom, c'est-à-dire que, si a={n; d}, alors <u>nom</u> (a)=n.

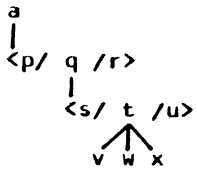
Pour obtenir une sous-m.a.t. d'une m.a.t. donnée "a", on se sert des opérateurs noeud et arbor:

<u>noeud</u> (a,n)=[b si nom(b)=n et b est contenu dans a
⊥ sinon

<u>arbor</u> (a,n)= [b si nom(b)=n, nfils(a,n)=0 et b est contenu dans a
p (f1, f2, ..., fm) si nom(p)=n et p (f1, f2, ..., fm) est un arbre contenu dans a
⊥ sinon

Un <u>chemin</u> dans une m.a.t. peut être mis en relief de manière explicite par des noeuds <u>piste</u>. C'est-à-dire, qu'un <u>chemin</u> est un noeud simple feuille, un noeud ensemble, un noeud liste, ou bien une <u>piste</u> qui a comme trace un <u>chemin</u>, ou bien un arbre qui a comme fils unique un <u>chemin</u>. C'est ainsi que le chemin qui mène de a jusqu'à t dans l'arborescence

peut être indiqué par le chemin



Pour la manipulation des chemins, on définit les opérateurs: dominant, associé, cheminS, cheminG et cheminD. Ils vont respectivement donner le noeud dominant un chemin (a dans l'exemple), la m.a.t. associée au chemin (t(v,w,x) dans l'exemple), et les chemins "strict", "gauche" et "droit" d'un chemin.

<u>dominant</u>(a)= [<u>dominant</u> (tr) si a=[n <tg/ tr/ td>] est une piste
p si a=p (f1, f2, ..., fm) est un arbre
a sinon

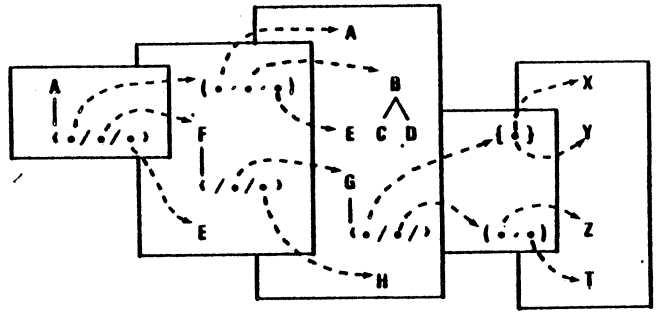
<u>associé</u>(a)= [<u>associé</u> (tr) si a=[n <tg/ tr/ td>] est une piste
<u>associé</u>(pst) si a=p (pst) est un arbre et pst est une piste
a sinon

<u>cheminS</u>(a)= [[n </ <u>cheminS</u>(tr) />] si a=[n <tg/ tr/ td>] est une piste
p (<u>cheminS</u> (pst)) si a= p (pst) est un arbre et pst est une piste
a sinon

<u>cheminG</u>(a)= [[n <tg/ <u>cheminG</u>(tr) />] si a=[n <tg/ tr/ td>] est une piste
p (<u>cheminG</u> (pst)) si a= p (pst) est un arbre et pst est une piste
a sinon

<u>cheminD</u>(a)= [[n </ <u>cheminD</u>(tr) /td>] si a=[n <tg/ tr/ td>] est une piste
p (<u>cheminD</u> (pst)) si a= p (pst) est un arbre et pst est une piste
a sinon

<u>Exemple</u> : Soit "a" la m.a.t. suivante



tranche 1:{1;A}({2< /

tranche 2: {3(, ,)} {9;F}({10</

tranche 3: {4;A} {5;B}({6;C},{7;D}) {8;E} {11;G}({12<

tranche 4: {13(

tranche 1: / >})

tranche 2: / >}) {20;E}

tranche 3: / />}) {19;H}

tranche 4: ,)} {16(,)}

tranche 5:{14;X} {15;Y} {17;Z} {18;T}

L'application des opérateurs dominant, associé et cheminS donne:

dominant(a)={1;A}

associé(a)={16 ({17;Z},{18;T})}

cheminS(a)={1;A}({2</{9;F}({10</{11;G}({12</{16({17;Z},{18;T})}/>})/>})/>})

□

Pour retrouver les noeuds qui se trouvent au premier niveau d'une m.a.t. et les m.a.t. qui commencent en dessous, on a les

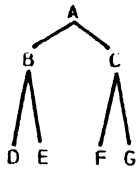
opérateurs Etage et Sous-Etages.

Etage(a)=
- a si a est un noeud simple
- p si a= p (f1, f2, ..., fm) est un arbre
- [n {Etage(a1), Etage(a2), ..., Etage(am)}] si a=[n {a1, a2, ..., am}] est un noeud ensemble
- [n (Etage(a1), Etage(a2), ..., Etage(am))] si a=[n (a1, a2, ..., am)) est un noeud liste
- [n <Etage(tg)/ Etage(tr)/ Etage(td)>] si a=[n <tg/ tr/ td>] est un noeud piste

Sous-Etages(a)=
- [n (f1, f2, ..., fm)] si a=p (f1, f2, ..., fm) est un arbre et nom(p)=n
- [n {Sous-Etages(ai1), ..., Sous-Etages(aij)}] si a=[n {a1, a2, ..., am}] est un ensemble, $1 \leq i1 < i2 < \ldots < ij \leq m$, $j \geq 2$, $\forall k \in [1, m]: k \in \{i1, \ldots, ij\}$ ssi Sous-Etages(aij)≠⊥
- Sous-Etages(ai) si a=[n {a1, a2, ..., am}] est un ensemble, $\nexists i \in [1, m]$: $\forall k \in [1, m]: k \neq i$ ssi Sous-Etages(ak)=⊥
- [n (Sous-Etages(ai1), ..., Sous-Etages(aij))] si a=[n (a1, a2, ..., am)] est une liste, $1 \leq i1 < i2 < \ldots < ij \leq m$, $j \geq 1$, $\forall k \in [1, m]: k \in \{i1, \ldots, ij\}$ ssi Sous-Etages(ak)≠⊥
- [n <stg/ str/ std>] si a=[n <tg/ tr/ td>] est une piste où: str= Sous-Etages(tr) ≠⊥
  - stg= vide si Sous-Etages(tg)=⊥; Sous-Etages(tg) sinon
  - std= vide si Sous-Etages(td)=⊥; Sous-Etages(td) sinon
- ⊥ sinon

Exemple :Soient les m.a.t. a et b suivantes

a=[1;A]([2;B]([3;D],[4;E]),[5;C]([6;F],[7;G]))



Etage(a)=[1;A]

Sous-Etages(a)=[1([2;B]([3;D],[4;E]),
[5;C]([6;F],[7;G])))

Etage(Sous-Etages(a))=[1([2;B],[5;C])]

Sous-Etages(Etage(a))=⊥

..... b =.....

tranche1:[1{ , , ,

tranche2: [2;A] [3;B]([4;C]) [5;C]([6{ , }]

tranche3: [7;D] [8;E]([9;F])

tranche1: ,

tranche2:[10(,)],[14< /

tranche3: [11;G] [12;H]([13;I]) [15;A]([16;B])

tranche1: ,

tranche2: / >] [20< / /

tranche3:[17;J]([18;K]) [19;L] [21;A]([22;B]) [23;J]

tranche1: }]

tranche2: >

tranche3:[24;L]([25;K])

{A, B, C, (G, H), <A/ J/ L>, <A/ J/ L>}
C {D, E} I B K B K
F

Sous-Etages(b)={1{

{3({4;C})},

{5({6({7;D},{8;E}({9;F}))})})},

{10({12({13;I})})})},

{14<{15({16;B})}/{17({18;K})}/>}})

Sous-Etages(Sous-Etages(b))={5({8({9;F})})}

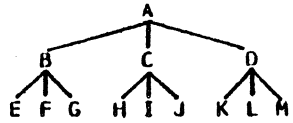
□

Les opérateurs listeD et listeG donnent la liste des m.a.t. qui se trouvent dans une m.a.t. au même niveau à droite ou à gauche, respectivement, qu'un noeud donné y compris celui-ci.

listeD(a,nl)=
- a si nom(a)=nl
- listeD({n(tg, tr, td)}, nl) si a={n <tg/ tr/ td>} est une piste et noeud(a, nl)≠⊥
- listeD(ai, nl) si a={n {a1, ..., am}} est un ensemble et noeud(ai, nl)≠⊥
- {n (listeD(ai, nl), a i+1, ..., am)} si a={n (a1, a2, ..., am)} est une liste, noeud(ai, nl)≠⊥ et niveau(a, n)= niveau(a, nl)
- listeD(Sous-Etages(a),nl) si noeud(a,nl)≠⊥ et niveau(a,nom(a))≠niveau(a,nl)
- ⊥ sinon

listeG(a,nl)=
- a si nom(a)=nl
- listeG({n(tg, tr, td)}, nl) si a={n <tg/ tr/ td>} est une piste et noeud(a, nl)≠⊥
- listeG(ai, nl) si a={n {a1, ..., am}} est un ensemble et noeud(ai, nl)≠⊥
- {n (a1, ..., a i-1, listeG(ai, nl))} si a={n (a1, a2, ..., am)} est une liste, noeud(ai, nl)≠⊥ et niveau(a, n)= niveau(a, nl)
- listeG(Sous-Etages(a),nl) si noeud(a,nl)≠⊥ et niveau(a,nom(a))≠niveau(a,nl)
- ⊥ sinon

Exemple : Soient les m.a.t. c et d suivantes



c={1;A}({2;B}({3;E},{4;F},{5;G}),

{6;C}({7;H},{8;I},{9;J}),

{10;D}({11;K},{12;L},{13;M}))

listeD(c,1)=c

listeD(c,8)={1({6({8;I},{9;J})},

{10({11;K},{12;L},{13;M})})})

listeG(c,6)={1({2;B}({3;E},{4;F},{5;G}),

{6;C}({7;H},{8;I},{9;J}))})

d={<(A, B)/ C/>, (B, C, A)}

d={1{{2<{3({4;A},{5;B})}/{6;C}/>},{7({8;B},{9;C},{10;A})}}}

listeG(d,9)={7({8;B},{9;C})}

listeD(d,5)={2({3({5;B})},{6;C})}

□

L'opérateur d'extraction de listes permet de trouver la liste des sous-m.a.t. d'une m.a.t. qui sont limitées par deux noeuds donnés. Si les limites se trouvent au même niveau, le résultat est une liste, mais si elles se trouvent à des niveaux différents et s'il existe un chemin qui conduit de la limite gauche à la limite droite, alors le résultat est le chemin construit avec une chaîne de pistes. Pour une m.a.t. a et deux entiers n1, n2, <u>extraire</u>(a,n1,n2) est définit par:

arbor(a,n1) si n1=n2 et noeud(a,n1)≠⊥

{n1 (extraire(ai, nom(ai), n2))} si a= {n1 {a1, a2, ..., am}} est un ensemble, ∄ i ∈ {1, m}: noeud(ai, n2)≠⊥

extraire(ai, n1, n2) si a={n {a1, a2, ..., am}} est un ensemble, ∄ i ∈ {1, m}: noeud(ai, n2)≠⊥ & noeud(ai, n1)≠⊥

listeG(a, n2) si a={n1 (a1, a2, ..., am)} est une liste, niveau(a, n1)= niveau(a, n2), ∄ i ∈ {1, m}: noeud(ai, n2)≠⊥

{n1 <tg/ extraire(ai, nom(ai), n2)/ td> si a={n1 (a1, a2, ..., am)} est une liste, niveau(a, n1) < niveau(a, n2), ∄ i ∈ {1, m}: noeud(ai, n2)≠⊥
où:

$$tg = \begin{cases} \underline{\text{vide}} & \text{si } i=1 \\ a1 & \text{si } i=2 \\ \{ntg(a1, \ldots, a\ i-1)\} & \text{sinon} \end{cases}$$

$$td = \begin{cases} \underline{\text{vide}} & \text{si } i=m \\ am & \text{si } i=m-1 \\ \{ntd(a\ i+1, \ldots, am)\} & \text{sinon} \end{cases}$$

ntg et ntd sont de nouveaux noms uniques

extraire(ai, n1, n2) si a={n (a1, a2, ..., am)} est une liste, ∄ i ∈ {1, m}: noeud(ai, n2)≠⊥ & noeud(ai, n1)≠⊥

{n (listeD(ai), a i+1, ..., listeG(aj))} si niveau(a, n1)= niveau(a, n2)= niveau(a, n), a={n (a1, a2, ..., am)}, ∄ i,j ∈ {1, m}: i ≠ j & noeud(ai, n1)≠⊥ & noeud(aj, n2)≠⊥

p (extraire(pst, nom(tr), n2)) si a= p (pst) est un arbre, pst= {n <tg/ tr/ td>} est une piste, nom(p)= n1, noeud(tr, n2)≠⊥

p({np <tg/ extraire(fi, nom(fi), n2)/ td> si a=p (f1, f2, ..., fm)} est un arbre, nom(p)=n1, niveau(a, n1)< niveau(a, n2), ∄ i ∈ {1, m}: noeud(fi, n2)≠⊥
où:

$$tg = \begin{cases} \underline{\text{vide}} & \text{si } i=1 \\ f1 & \text{si } i=2 \\ \{ntg(f1, \ldots, f\ i-1)\} & \text{sinon} \end{cases}$$

$$td = \begin{cases} \underline{\text{vide}} & \text{si } i=m \\ fm & \text{si } i=m-1 \\ \{ntd(f\ i+1, \ldots, fm)\} & \text{sinon} \end{cases}$$

np, ntg et ntd sont de nouveaux noms uniques

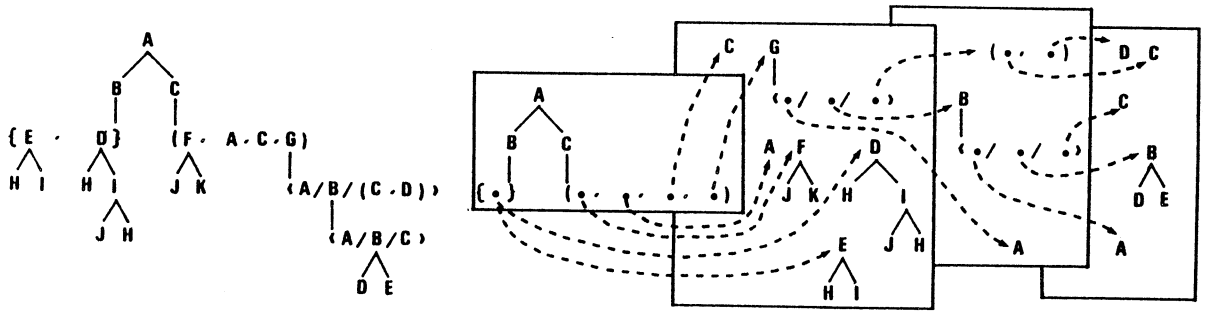
extraire({n(tg, tr, td)}, n1, n2) si a={n <tg/ tr/ td>} est une piste, noeud(a, n1)≠⊥ et noeud(a, n2)≠⊥

extraire(Sous-Etages(a), n1, n2) si niveau(a, n1)= niveau(a, n2) & noeud(a, n1)≠⊥ & noeud(a, n2)≠⊥
niveau(a, n)≠ niveau(a, n1), Sous-Etage(a)≠⊥

extraire(arbor(a, n1), n1, n2) si niveau(a, n1)< niveau(a) & noeud(a, n1)≠⊥ & noeud(arbor(a, n1), n2)≠⊥

⊥ sinon

<u>Exemple</u> : Soit e la m.a.t. suivante:



```
Tranche1:[1;A]([2;B]([3(                                  ,                    ])),
Tranche2:                  [4;D]([5;H],[6;I]([7;J],[8;H])) [9;E]([10;H],[11;I])

Tranche1:[12;C]([13(                      ,      ,      ,
Tranche2:            [14;F]([15;J],[16;K]) [17;A] [18;C] [19;G]([20<      /
Tranche3:                                                          [21;A] [22;B]([23<

     Tranche1:                                                          ))))
     Tranche2:                                        /                  >))
     Tranche3:      /                      /      >)) [29(      ,      ))
     Tranche4:[24;A] [25;B]([26;D],[27;E]) [28;C]        [30;C] [31;D]

extraire(e,25,25)=[25;B]([26;D],[27;E])
extraire(e,3,9)=[3([9;E]([10;H],[11;I]))]
extraire(e,5,6)=[4([5;H],[6;I]([7;J],[8;H]))]
extraire(e,13,17)=[13([14;F]([15;J],[16;K]),[17;A])]
extraire(e,9,11)=[9;E]([32<[10;H]/[11;I]/>])
extraire(e,13,16)=[13</[14;F]([32<[15;J]/[16;K]/>])/[33([17;A],[18;C],arbor(e,19))]>]
extraire(e,15,16)=[14([15;J],[16;K])]
extraire(e,17,18)=[13([17;A],[18;C])]
extraire(e,19,24)=[19;G]([20<[21;A]/[22;B]([23</[24;A]/[32(arbor(e,25),[28;C])]>])/arbor(e,29)>])
extraire(e,25,28)=[23([25;B]([26;D],[27;E]),[28;C])]
extraire(e,4,9)=extraire(e,4,11)=⊥
□
```

3.3 Règles et Grammaires

Un utilisateur du système définit un réseau de grammaires qu'il veut appliquer aux structures de données de la section précédente. Le système applique ces grammaires à une m.a.t. objet et produit comme résultat une m.a.t. cible. Dans cette section, on précise le type de grammaires et de règles qui les composent. Le principe général est inspiré de ROBRA. La première partie sera consacrée à l'étude des restrictions d'ordre et son expression qui sera utilisée dans le système.

3.3.1 Quelques notations pour l'ordre et le désordre

L'expression de l'ordre et du désordre est importante dans la plupart des domaines de l'informatique, en particulier dans les systèmes de reconnaissance de formes et pour l'étude du parallélisme et du non-déterminisme dans les algorithmes. C'est la raison pour laquelle on s'intéresse ici à ce problème. Il a été déjà traité en mathématiques du point de vue des permutations et des ordres partiels, et en informatique il existe des langages qui incorporent des notations particulières, par exemple PLANNER [He 72] ou le système ROBRA du GETA [BGQ 78].

Le but de cette étude est d'essayer de répondre aux questions suivantes:

Comment exprimer le fait qu'une permutation d'un ensemble B est "comprise" dans une permutation d'un ensemble A tel que B est contenu dans A, dans le sens où l'ordre des éléments de B donné par la permutation de B est le même que celui donné par la permutation de A?

Etant donné une permutation de B, comment identifier l'ensemble des permutations de A qui "comprennent" la permutation de B?

Une permutation de B définit un ordre total des éléments de B. Comment se posent les problèmes précédents si, au lieu de prendre une permutation pour B, on prend un ensemble de permutations de B, et si l'on demande la vérification d'une expression logique formée à l'aide de ces permutations et des opérateurs logiques $\neg$, $\&$, $\vee$ (négation, conjonction et disjonction)?

Quelle notation pourrait-on définir pour décrire toutes ces expressions et ensembles d'une façon simple, claire et générale?

Dans la première partie, on étudie les permutations et on définit le vocabulaire employé, puis on étudie l'ordre total, ensuite l'ordre partiel et finalement on introduit d'autres notations particulières.

3.3.1.1 Les notions de base: Dans cette section, on donne un certain nombre de définitions et des démonstrations qui vont permettre de préciser la notion de contenance pour les permutations.

Un intervalle fermé de N ensemble de nombres entiers naturels, que l'on note [i, j], est défini pour tous i, j ∈ N comme: {k ∈ N | i ≤ k ≤ j}.

Exemple : [2, 2]={2}, [3, 8]={3, 4, 5, 6, 7, 8}, [2, 1]=ϕ (vide)

Pour tout ensemble A fini de n éléments, |A|=n, une permutation de A est une bijection Ai de A dans [1, n], que l'on note <ai1, ai2, ..., ain>, telle que Ai(aij)=j pour tout aij ∈ A.

Par exemple: A={a, b, c, d, e, f}, Ai=<b, a, c, e, f, d>, Ai(a)=2, -Ai(5)=f (inverse de Ai)

L'ensemble des bijections de [1, m] dans [1, n] pour tous m, n ∈ N est noté Fm,n. L'ensemble Fm,m est l'ensemble des permutations de [1, m], que l'on note FA**m. L'ensemble des permutations de A, |A|=n, est noté FA**n et contient n! éléments

Par exemple: A={a, b, c}, FA**3={<a, b, c>, <a, c, b>, <b, a, c>, <b, c, a>, <c, a, b>, <c, b, a>}

L'algorithme 3.3.2-P de [Kn 69] page 59 pour l'analyse d'une permutation, définit une bijection f:FA**n -> [0, n! -1] pour tout ensemble A fini non vide par rapport à un ordre total donné à A. Donc, lorsque l'on parle de la i-ème permutation de A et que l'on note Ai, on fait référence implicite à une bijection telle que f (c'est-à-dire, -f(i)=Ai).

Soient Ai une permutation de A et Bj une permutation de B telles que A et B soient disjoints. La concaténation de Ai et Bj, que l'on note Ai.Bj, est définie comme:

$$Ai.Bj(x)=\begin{cases} Ai(x) & \text{si } x \in A \\ Bj(x)+|A| & \text{si } x \in B \end{cases}$$

Par exemple: A={a, b, c}, B={d, e, f}, <a, c, b>.<f, d, e>= <a, c, b, f, d, e>, Ai(b)=3, Bj(d)=2, Ai.Bj(d)=5

Il est facile de démontrer que:

1. La concaténation de Ai et Bj est une permutation de A U B si A et B sont disjoints
2. La concaténation est associative sur des ensembles disjoints. Soient A, B, C mutuellement exclusifs alors Ai.(Bj.Ck)=(Ai.Bj).Ck pour toutes Ai, Bj, Ck permutations de A, B, C.

Soient A et B deux ensembles tels que B ⊆ A (B contenu dans ou égal à A), |B|=m et |A|=n pour m,n ≥1. On définit la fonction "A\B: [1, n]**A -> [1, m]**B comme

$\forall x \in B$ ($\mathbb{T}A\backslash B[Ai](x)=|Pg(x)|+1$) où $Pg(x)=\{x' \in B \mid Ai(x') < Ai(x)\}$

On peut montrer que: $\mathbb{T}A\backslash B[Ai]$ est une permutation de B, dite <u>la permutation de B induite par Ai</u>:

Démonstration:

a) $0 \leq |Pg(x)| < m \Rightarrow 1 \leq \mathbb{T}A\backslash B[Ai](x) \leq m$

b) Si $x0, x1 \in B$ et $x0 \neq x1$ alors $Ai(x0) \neq Ai(x1) \Rightarrow Pg(x0) \neq Pg(x1) \Rightarrow \mathbb{T}A\backslash B[Ai](x0) \neq \mathbb{T}A\backslash B[Ai](x1)$ donc $\mathbb{T}A\backslash B[Ai]: B \rightarrow [1, m]$ est bijective

□

<u>Exemple</u> : $Ai=\langle c, a, e, b, d\rangle$, $B=\{b, c, e\}$

$Pg(b)=\{c, e\}$, $Pg(c)=\phi$, $Pg(e)=\{c\}$

$\mathbb{T}A\backslash B[Ai]=\langle c, e, b\rangle = \{(c, 1), (e, 2), (b, 3)\}$

□

Pour toutes permutations Ai, Bj de A et B telles que $B \subseteq A$, $|B|=m$ et $|A|=n$, il existe une fonction f de Fm,n telle que $\forall x \in B: [f\ Bj(x)= Ai(x)]$. En effet, $f=Ai\ (-Bj)$ par exemple, si $Bj=\langle b, c, e\rangle$ et Ai est la même que dans l'exemple précédent $Ai\ (-Bj)=\{(1, 4), (2, 1), (3, 3)\}$.

Ai (-Bj) est strictement croissante si et seulement si Bj est la permutation de B induite par Ai.

Démonstration:

=>: Soit $bjk= -Bj(k)$. Par hypothèse,

$j\ k-1 = Ai(-Bj(k-1))=Ai(bjk) < AiBj(k)= Ai(bjk)= jk$

alors, pour $\forall l \in [1, k-1]$: $-Ai(jl)=bjl$

$\Rightarrow Pg(bjk)=\{bj1, \ldots, bj\ k-1\}$

$\Rightarrow \mathbb{T}A\backslash B[Ai]\ (bjk)= |Pg(bjk)|+1= k-1+1= Bj(bjk)$

<=: Soient $x0 < x1$ et $x0, x1 \in [1, m]$

$\Rightarrow \nexists\ b, b' \in B: x0=Bj(b') < Bj(b)=x1$

$\Rightarrow |Pg(b')| < |Pg(b)|$ car $Bj =\mathbb{T}A\backslash B[Ai]$

$\Rightarrow Ai(b') < Ai(b)$ car $Ai(b) < Ai(b') \Rightarrow Pg(b) \subseteq Pg(b')$

$\Rightarrow Ai(-Bj)(x0) < Ai(-Bj)(x1)$ donc $Ai(-Bj)$ est strictement croissante

□

Soient X et Y deux ensembles finis, et Xi et Yj deux permutations. Si $Xi(-Yj)$ est définie et croissante, on dit que Yj est <u>comprise</u> dans Xi et l'on note $Yj \ll Xi$. Si $Xi(-Yj)$ n'est pas définie alors Xi et Yj ne sont pas comparables par $\ll$.

<u>Exemple</u> :

$\langle b, c, e\rangle$ et $\langle c, a, e, b, d\rangle$ ne sont pas comparables par $\ll$

$\mathbb{T}A\backslash B[Ai]= \langle c, e, b\rangle \ll \langle c, a, e, b, d\rangle= Ai$ où $B=\{b, c, e\}$

□

La relation $\ll$ est un ordre partiel pour l'ensemble $S=\cup\ \mathbb{T}B\backslash|B|$ sur tout B sous-ensemble de A, et l'ensemble S muni de cette relation d'ordre est un demi-treillis inférieur avec la

permutation de l'ensemble vide comme minimum de S.

On définit un ordre partiel $\ll$ sur l'ensemble des intervalles fermés de N comme $[i, j] \ll [i', j']$ si et seulement si $i \leq j < i' \leq j'$, lu $[i, j]$ est <u>à gauche</u> de $[i', j']$. Cette relation, transitive, asymétrique et irréflexive va nous permettre de donner un ordre aux permutations comprises dans une autre. Pour cela, on définit une <u>place</u> de permutation Ai de A comme un ensemble $[k, l]$ tel que $[0, 0] \ll [k, l] \ll [n+1, n+1]$ où $n=|A|$. L'ensemble <u>occupant</u> la place $[k, l]$ de Ai, que l'on note $-Ai[k, l]$, est $\{a \in A \mid Ai(a) \in [k, l]\}$. Par extension, on dit que: si $[k, l]$ et $[k', l']$ sont deux places de Ai

$-Ai[k, l] \ll -Ai[k', l']$ si et seulement si $[k, l] \ll [k', l']$

<u>Exemple</u> :

$Ai=\langle b, c, d, a, f, e\rangle$, $-Ai[1, 3]=\{b, c, d\}$, $-Ai[4, 4]=\{a\}$, $\{b, c, d\} \ll \{a\}$

¤

On peut voir que pour tout ensemble occupant $O=-Ai[k, l]$ de Ai

$$^{m}A\backslash O[Ai]\ (x)= Ai(x)-k+1 \text{ pour tout } x \in O$$

et par conséquent :

$$\forall\ x \in -Ai[1, l]=O : {}^{m}A\backslash O[Ai]\ (x)=Ai(x)$$

L'ensemble des plus grands (ensembles) occupants de Ai qui sont dans B ou dans A-B, et que l'on note M(Ai, B) où $B \subseteq A$, est une partition de A, dite <u>partition maximale</u> de Ai par B. C'est-à-dire:

$M(Ai, B)=\{ -Ai[k, l] \in N(Ai, B) \mid \forall\ k', l' : [k, l] \subset [k', l'] \Rightarrow -Ai[k', l'] \notin N(Ai, B)\}$

$N(Ai, B)=\{ -Ai[k, l] \mid -Ai[k, l] \subseteq B$ ou $Ai[k, l] \subseteq A-B\}$

$\{ x \in O \mid \forall\ O \in M(Ai, B)\}=A$ et $\forall\ O, O' \in M(Ai, B) : O \neq O' \Rightarrow O$ et O' sont disjoints.

<u>Exemple</u> :

$Ai=\langle a, b, c, d, e, f, g, h\rangle$, $B=\{b, c, e\}$

$M(Ai, B)=\{\{a\}, \{b, c\}, \{d\}, \{e\}, \{f, g, h\}\}$

$N(Ai, B)-M(Ai, B)=\{\{b\}, \{c\}, \{f\}, \{g\}, \{h\}, \{f, g\}, \{g, h\}, \{f, g, h\}\}$

¤

La partition maximale de Ai par B est composée d'occupants implicites et explicites. L'occupant d'une place $[k, l]$ de Ai qui se trouve dans M(ai, B) et dans l'ensemble des parties de B est dit <u>occupant explicite</u>, et s'il se trouve dans M(Ai, B) et dans l'ensemble des parties de A-B, il est dit <u>occupant implicite</u>. M+(ai, B) et M-(Ai, B), les ensembles des occupants explicites et implicites des places de Ai sont des partitions de B et A-B.

$M+(Ai, B)=\{-Ai[k, l] \in M(Ai, B) \mid -Ai[k, l] \subseteq B\}$

$M-(Ai, B)=\{-Ai[k, l] \in M(Ai, B) \mid -Ai[k, l] \subseteq A-B\}$

<u>Exemple</u> : Soient Ai et B les mêmes que ceux de l'exemple précédent

M+(Ai, B)={{b, c}, {e}}

M−(Ai, B)={{a}, {d}, {f, g, h}}

□

Il existe une permutation de M(Ai, B), dite <u>naturelle</u>, définie comme

ⁿn[Ai]: M(Ai, B) -> [1, |M(Ai, B)|]

ⁿn[Ai](O)=|P'g(O)|+1 où P'g(O)={ O' ∈ M(Ai, B)| O' ≺ O}

<u>Exemple</u> : Pour les mêmes Ai et B qu'à l'exemple précédent

ⁿn[Ai]=<{a}, {b, c}, {d}, {e}, {f, g, h}>

□

Il est facile de constater que:

1. La relation ≺ est un ordre total pour l'ensemble M(Ai, B)
2. O ≺ O' si et seulement si ⁿn[Ai](O) < ⁿn[Ai](O')
3. ⁿn[Ai](O) < ⁿn[Ai](O') si et seulement si O ∈ P'g(O')
4. ∀ O, O' ∈ M(Ai, B): (O≠O' et (O∪O' ⊆ B ou O∪O' ⊆ A−B)) => |ⁿn[Ai](O) − ⁿn[Ai](O')| >1

Par conséquent, toute permutation Ai de A peut être décrite par rapport à un sous-ensemble B de A, comme la concaténation des permutations des occupants explicites et implicites des places de Ai induites par Ai et dans l'ordre indiqué par la permutation naturelle de la partition maximale de Ai par B.

C'est-à-dire:

Ai= X1 j1 • X2 j2 • ... • XL jL où L=|M(Ai, B)|

∀ k ∈ [1, L]: Xk ∈ M(Ai, B) & ⁿn[Ai](Xk)=k & Xk jk= ⁿA\Xk[Ai]

<u>Exemple</u> : Avec les mêmes Ai et B qu'à l'exemple précédent, on a:

Ai= ⁿA\{a}[Ai] • ⁿA\{c, b}[Ai] • ⁿA\{d}[Ai] • ⁿA\{e}[Ai] • ⁿA\{h, f, g}[Ai]

= <a> • <b, c> •<d> • <e> • <f, g, h>

= <a, b, c, d, e, f, g, h>

□

3.3.1.2 <u>Une notation pour l'ordre total</u>: Une permutation Ai de A est un ordre total pour les éléments de A. Dans la section précédente, on a défini un ordre total ≺ pour les occupants explicites et implicites des places de Ai par rapport à un sous-ensemble B de A. Dans cette section, nous voulons préciser une notation qui permette d'exprimer ces deux ordres totaux, et étudier un peu comment exploiter cette notation.

Soient a, b ∈ A et Ai une permutation de A, on définit deux prédicats qui vont permettre d'établir si a est à gauche de b, et plus précisément si a est tout de suite à gauche de b ou bien s'il peut y avoir d'autres éléments entre eux. Soient donc <u>v</u> et <u>e</u> les prédicats

<u>v</u>(a, b, i)≡ (a ≠ b)&(Ai(b) = Ai(a)+1)

<u>e</u>(a, b, i)≡ (a ≠ b)& ∃ k>1(Ai(b) = Ai(a)+k)

et on utilise une notation infixe (a $\underline{v}$ b)i et (a $\underline{e}$ b)i respectivement. On dit que a est <u>voisin à gauche</u> de b et que b est <u>voisin à droite</u> de a dans Ai si (a $\underline{v}$ b)i est vrai. On dit que a est <u>écarté à gauche</u> de b dans Ai et que b est <u>écarté à droite</u> de a dans Ai si (a $\underline{e}$ b)i est vrai. S'il n'y a pas ambiguïté, on écrit a $\underline{v}$ b et a $\underline{e}$ b à la place des notations précédentes.

Il est facile de voir que:

1. a $\underline{v}$ b <=> ∀ c ∈ A (c≠a & c≠b => a $\underline{e}$ c ∨ c $\underline{e}$ b)
2. a $\underline{e}$ b<=> ∄ c, c' ∈ A ({c, c'} et {a, b} sont disjoints & a $\underline{v}$ c & c' $\underline{v}$ b)
3. (a $\underline{v}$ b ∨ a $\underline{e}$ b)&(b $\underline{v}$ c ∨ b $\underline{e}$ c) => a $\underline{e}$ c
4. {a} ∝ {b} <=> a $\underline{v}$ b ∨ a $\underline{e}$ b
5. O ∈ M+(Ai, B) & O' ∈ M-(Ai, B) => ∀ a, a' ∈ O ∀ b ∈ O' (a $\underline{e}$ b ∨ b $\underline{e}$ a)&(a $\underline{e}$ b => a' $\underline{e}$ b)&(b $\underline{e}$ a => b $\underline{e}$ a')

<u>Exemple</u> :

Soit Ai =<b, c, d, a, f, e, g>, alors l'expression logique suivante est vraie, (b $\underline{e}$ b)&(b $\underline{v}$ c)&¬(a $\underline{v}$ d)&(f $\underline{e}$ g)&(d $\underline{e}$ g)

□

L'ordre total d'un ensemble A défini par une permutation Ai de A peut être exprimé par une expression logique formée par la conjonction de n-1 relations de voisinage où n=|A|. Soit Ai =<ai1, ai2, ..., ain>, l'expression correspondante est

(ai1 $\underline{v}$ ai2)&(ai2 $\underline{v}$ ai3)& ...&(ai n-1 $\underline{v}$ ain)

dite <u>expression naturelle</u> d'une permutation et sera notée

ai1 * ai2 * ... * ai n-1 * ain

L'ordre total des éléments de B, sous-ensemble de A, défini par la permutation induite par Ai peut être exprimé par une expression logique formée par la conjonction de n-1 relations de voisinage et d'écartement. Soit Bj="A\B|Ai|, on prend l'expression naturelle de la permutation Bj et pour chaque relation (bjk $\underline{v}$ bj k+1) on regarde si {bjk, bj k+1} est contenu dans un des occupants explicites de M+(Ai, B). Si c'est le cas, on passe à la relation suivante, sinon on remplace la relation par (bjk $\underline{e}$ bj k+1). La conjonction résultante est dite <u>expression linéaire</u> induite par Ai sur B, (bj1 o1 bj2)&(bj2 o2 bj3)& ...&(bj n-1 o n-1 bjn) où ok est $\underline{v}$ ou $\underline{e}$ et sera notée

bj1 o'1 bj2 o'2 ... o'n-2 bj n-1 o'n-1 bjn

où o'k =(* si ok=$\underline{v}$; < si ok=$\underline{e}$)

<u>Exemple</u> : Ai=<b, c, d, a, f, e, g>, Bj=<c, a, f, g>, Bj << Ai

c * a * f * g est l'expression naturelle de Bj

c < a * f < g est l'expression linéaire induite par Ai sur B

□

D'après cette définition et d'après les résultats de la section 3.3.1.1, on voit qu'une expression linéaire induite aura autant de relations d'écartement que le nombre d'occupants implicites de M-(Ai, B) <u>internes</u> à la permutation Bj. Un occupant implicite O est <u>interne</u> à Bj s'il existe deux occupants

explicites O1 et O2 tels que O1 « O « O2. Les relations d'écartement laissent des "trous" qui sont remplis par les occupants implicites. Il est intéressant de noter que deux permutations distinctes Ai et Aj de A peuvent induire la même expression linéaire sur B ⊆ A.

<u>Exemple</u> : c < a * f < g est induite par toutes les permutations suivantes de A

<c, b, a, f, d, g, e>	<c, d, a, f, e, g, b>
<c, b, a, f, d, e, g>	<c, d, a, f, e, b, g>
<c, b, d, a, f, e, g>	<c, d, e, a, f, b, g>
<b, c, d, a, f, e, g>	<d, c, e, a, f, b, g>
<c, b, a, f, e, g, d>	<c, e, a, f, b, g, d>
<c, b, a, f, e, d, g>	<c, e, a, f, b, d, g>
<c, b, e, a, f, d, g>	<c, e, b, a, f, d, g>
<b, c, e, a, f, d, g>	<e, c, b, a, f, d, g>
<c, d, a, f, b, g, e>	<c, e, a, f, d, g, b>
<c, d, a, f, b, e, g>	<c, e, a, f, d, b, g>
<c, d, b, a, f, e, g>	<c, e, d, a, f, b, g>
<d, c, b, a, f, e, g>	<e, c, d, a, f, b, g>

¤

On dit qu'une expression linéaire d'un sous-ensemble B de A a une <u>occurrence</u> dans une permutation Ai de A si elle est égale à l'expression linéaire induite par Ai sur B. Par extension, on peut dire qu'une expression linéaire a une occurrence dans l'expression naturelle de la permutation Ai.

Le nombre des permutations de A qui ont une occurrence d'une expression linéaire x d'un sous-ensemble B de A est

$$\binom{n-m+1}{k+1}(n-m)!$$

où n=|A|, m=|B| et k= nombre de relations d'écartement dans x

Démonstration:

En effet, dans chacun des k trous de l'expression linéaire, il faut placer au moins un des éléments de A-B. C'est-à-dire que les éléments qui se trouvent entre deux relations d'écartement consécutives doivent se placer entre deux éléments de A-B. Soit Yl une permutation de A-B

Yl=<yl1, yl2, ..., yl n-m>

x= x1 < x2 < ...< xk < x k+1

les sous-expressions x1, x2, ..., x k+1 peuvent se placer à gauche de yl1, à droite de yl n-m, ou entre yli et yl i+1 pour ∀ i ∈{1, n-m-1}. Donc, on choisit k+1 positions pour les xj parmi les n-m+1 possibilités et cela pour chacune des (n-m)! permutations Yl de A-B.

¤

Dans l'exemple précédent, on a n=7, m=4, k=2 et donc 24 possibilités qui sont celles listées plus haut.

On remarque que si k>n-m alors il n'existe pas de permutation de A qui contienne une occurrence de <u>x</u>, car les restrictions d'écartement ne peuvent pas être remplies. Par exemple, pour

A={a, b, c, d, e, f, g} et $\underline{x}$ égale à c < a < f < g < b.

On peut augmenter la puissance des expressions linéaires par l'introduction de deux prédicats FG et FD

FG(a, i)≡{Ai(a)=1}

FD(a, i)≡{Ai(a)=|A|}

qui permettent de demander que "a" soit <u>voisin à la frontière gauche</u> (ou <u>droite</u> respectivement), de la permutation Ai. Evidemment, les seuls candidats d'une expression linéaire à être voisins d'une frontière sont les éléments qui se trouvent aux "extrémités" de l'expression. On note donc ces conditions sur une expression linéaire $\underline{x}$ comme:

$\underline{\perp}$ * x pour la frontière gauche

et x * T pour la frontière droite

La négation de ces prédicats n'étant importante que pour les éléments extrêmes de x, on écrira

$\underline{\perp}$ < x pour l'écartement de la frontière gauche

et x < T pour l'écartement de la frontière droite

S'il n'y a pas de confusion, on écrit * à la place de $\underline{\perp}$ ou T.

<u>Exemple</u> : * < c < a * f < g * * est induite par <b, c, d, a, f, e, g> mais non par <c, b, a, f, d, g, e>

□

Il est facile de constater que le nombre de permutations de A qui contiennent une occurrence d'une expression linéaire $\underline{x}$ d'un sous-ensemble B de A est

$$\binom{n-m+1-(p+q)}{k-(p+q)+1}(n-m)!$$

où n=|A|, m=|B|, k=nombre de relations d'écartement dans a, p=nombre de conditions d'écartement de frontière, q=nombre de conditions de voisinage de frontière.

Par exemple, pour le cas précédent, on a n=7, m=4, k=3, p=1, q=1 et il y a donc 6 possibilités.

3.3.1.3 <u>Une notation pour l'ordre partiel</u>: Dans cette section, on s'intéresse aux prédicats que l'on peut définir par la <u>conjonction</u> de relations de voisinage et d'écartement de la section précédente étendues aux frontières des permutations, et à des expressions possibles pour les représenter.

Il s'agit d'une approche en "sens inverse", c'est-à-dire qu'on prend une expression logique et qu'on essaie de trouver l'ensemble des permutations qui vérifient cette expression. Il se peut que cet ensemble soit vide, par exemple si on a le prédicat

$$(a \underline{\vee} b) \,\&\, (b \underline{\vee} a)$$

qui est faux pour toute paire a, b d'éléments de A, et toute permutation Ai de A.

L'algorithme de tri topologique 2.2.3.T de [Kn 73] donne un

moyen de reconnaître s'il existe des "boucles" du type précédent dans un prédicat donné. On peut le modifier de sorte que l'on puisse de surcroît vérifier si le prédicat ne viole pas les propriétés 1 et 2 des relations de voisinage et d'écartement de la section précédente, et s'il ne contient aucun couple de relations (a $\underline{v}$ b) et (a $\underline{e}$ b).

Si le prédicat est déclaré <u>conforme</u> par cette procédure, il définit un ordre partiel sur les éléments de A auxquels il fait référence par la relation <u>être à gauche de</u> de la section 3.3.1.1, notée « pour les ensembles. Dans les expressions linéaires, on notera {a} « {b} par a $\leq$ b donc la propriété 4 de la section 3.3.1.2 peut être écrite comme:

$$a \leq b \iff a * b \ \vee \ a < b$$

<u>Exemple</u> :

a * b & c * d & a * d,

a * b & c < a & b * c

ne sont pas conformes et

a < b & a < c & a < d & c < d,

a < b & b < c & d * e & e < f

sont conformes

¤

Soit B le sous-ensemble des éléments de A auxquels on fait référence dans un prédicat conforme. Si l'on ajoute pour chacun des éléments b $\in$ B les relations $\perp \leq$ b et b $\leq$ T, et si l'on retire ensuite toutes les <u>relations transitives</u>, c'est-à-dire que, si (a o1 b), (b o2 c) et (a o3 c) sont dans le prédicat, on retire (a o3 c) pour oi égal à *, < ou $\leq$, le prédicat résultat est équivalent au prédicat original.

<u>Exemple</u> : a * b & a < c & a < d & b * c & b < d & c * d est équivalent à

a * b & a < c & a < d & b * c & b < d & c * d & $\perp \leq$ a & $\perp \leq$ b & $\perp \leq$ c & $\perp \leq$ d & a $\leq$ T & b $\leq$ T & c $\leq$ T & d $\leq$ T

et équivalent à

$\perp \leq$ a & a * b & b * c & c * d & d $\leq$ T

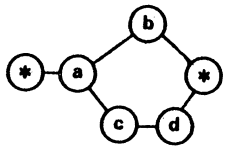
qui peut être écrit

$\perp \leq$ a * b * c * d $\leq$ T ou a * b * c * d

¤

Un prédicat conforme définit un <u>treillis</u> sur l'ensemble B des éléments auxquels il fait référence augmenté des éléments frontière ($\perp$, T), par la relation être à gauche de ($\leq$); les frontières gauche et droite sont le minimum et le maximum du treillis.

<u>Exemple</u> : Pour le prédicat a < b & a < c & a < d & c < d, on a le treillis



¤

Si l'on fait un tri topologique d'un treillis, c'est-à-dire si l'on écrit les éléments dans un ordre linéaire qui ne viole pas l'ordre partiel du treillis, chacun des résultats possibles peut être écrit comme une expression linéaire.

<u>Exemple</u> : Pour le treillis de l'exemple ci-dessus, on a

$$a < b \leq c < d$$
$$a < c \leq b \leq d$$
$$a < c < d \leq b$$

ou en remplaçant le symbole $\leq$, par * ou < on a

$$a < b * c < d \qquad a < b < c < d$$
$$a < c * b * d \qquad a < c * b < d$$
$$a < c < b * d \qquad a < c < b < d$$
$$a < c < d * b \qquad a < c < d < b$$

Pour le prédicat $a < b$ & $a < c$ & $a < d$ & $c * d$ on a les expressions

$$a < b * c * d \qquad a < b < c * d$$
$$a < c * d * b \qquad a < c * d < b$$

ou bien $a < b \leq c * d \qquad a < c * d \leq b$

car la relation entre c et d empêche de mettre un autre élément entre les deux.

□

L'exemple précédent montre que la relation $\leq$ est une notation qui permet d'alléger l'écriture des expressions en augmentant leur puissance. On définit un opérateur <u>d'interclassement</u> ou <u>parallélisme</u> que l'on note ||, qui permet de décrire les expressions linéaires possibles issues du tri topologique d'un treillis.

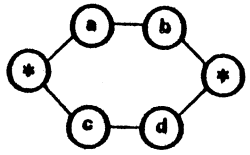
Si x et y sont deux expressions linéaires qui n'ont que des relations * < ou $\leq$, on définit

x||y={z| z est une expression linéaire sur les éléments de x et y qui ne viole ni l'ordre de x ni celui de y}

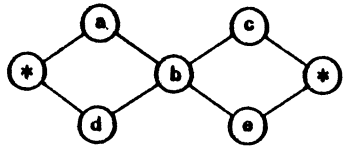
<u>Exemple</u> : $(a < b)||(c < d)$ est égal à l'ensemble des relations suivantes

$$a < b \leq c < d \qquad c \leq a < b \leq d$$
$$a \leq c \leq b \leq d \qquad c \leq a \leq d \leq b$$
$$a \leq c < d \leq b \qquad c < d \leq a < b$$

et représente le treillis



$(a < b < c)||(d < b < e)$ est représenté par



□

On peut étendre la définition de l'opérateur || de façon récursive comme suit: Soient xi, yj des expressions linéaires qui n'ont que des relations * < ou $\leq$, alors

$\{x1, x2, \ldots, xn\}||\{y1, y2, \ldots, ym\} \equiv \cup_i \cup_j (xi||yj)$

$\{x1, x2, \ldots, xn\} \leq \{y1, y2, \ldots, ym\} \equiv \cup_i \cup_j (xi \leq yj)$

$\{x1, x2, \ldots, xn\} \& \{y1, y2, \ldots, ym\} \equiv \cup_i \cup_j (xi \& yj)$

$\forall a, b \in A : a||b \equiv \{a \leq b, b \leq a\}$

<u>Exemple</u> :

$(a || d) < b < (c || e)$

$\equiv \{a \leq d, d \leq a\} < b < \{c \leq e, e \leq c\}$

$\equiv \{a \leq d, d \leq a\} < b \ \& \ b < \{c \leq e, e \leq c\}$

$\equiv \{a \leq d < b, d \leq a < b\} \ \& \ \{b < c \leq e, b < e \leq c\}$

$\equiv \{a \leq d < b < c \leq e, a \leq d < b < e \leq c, d \leq a < b < c \leq e, d \leq a < b < e \leq c\}$

$\equiv (a < b < c)|||(d < b < e)$

□

Un ensemble d'expressions linéaires est interprété comme la <u>disjonction</u> des prédicats qui sont représentés par chacune de ses expressions. Il existe donc un ensemble de permutations de A qui vérifient ce prédicat. Autrement dit, elles contiennent une <u>occurrence</u> d'au moins une de ces expressions.

L'opérateur || peut définir un ensemble vide si les ordres de x et y sont contradictoires. C'est le cas de $(a * b)|||(c * b)$ ou de $(a < b < c)|||(c < d < a)$. On est sûr que $x||y$ n'est pas vide si x et y sont deux expressions linéaires <u>disjointes</u>, c'est-à-dire si Bx et By sont disjoints, avec Bx et By dénotant respectivement les ensembles des éléments de x et de Y.

Deux expressions linéaires x et y sont <u>superposables</u> si Bx et By sont égaux.

Deux expressions linéaires sont <u>entremêlées</u> si elles ne sont ni superposables ni disjointes.

Si x et y sont disjointes, le nombre d'éléments de l'ensemble $x||y$ exprimés à l'aide de la relation $\leq$ est

$\binom{p+q}{q}$ où q est le nombre de relations d'écartement de x plus 1

p est le nombre de relations d'écartement de y plus 1

Par exemple, $(a < b)|||(c < d)$ contient 6 éléments

Si x1, x2, ..., xn sont des expressions linéaires telles que xi et xj soient disjointes pour tout $i \neq j$, le nombre d'éléments de $x1|| x2|| \ldots || xn$ exprimé à l'aide de la relation $\leq$ est

$$\binom{p1+ p2+ \ldots+pn}{p1}\binom{p2+ p3 + \ldots+ pn}{p2}\cdots\binom{pn}{pn} =$$

$$\binom{p1+ p2+ \ldots+ pn}{p1, p2, \ldots, pn} = \frac{(p1+ p2+ \ldots+pn)!}{p1!\ p2!\ \ldots\ pn!}$$

où pi est le nombre de relations d'écartement de xi plus 1

Par exemple, $(a < b)|||(c < d)|||(e < f)$ contient 90 éléments.

Si x1, x2, y1, y2 sont des expressions linéaires disjointes qui n'ont pas l'élément b, $(x1 \leq b \leq x2)|||(y1 \leq b \leq y2)$ contient $|x1||y1|$ par $|x2||y2|$ éléments.

Si x est une expression linéaire qui contient n relations $\leq$, elle représente un ensemble de 2 puissance n (2**n) expressions linéaires qui n'ont pas de relation $\leq$.

Par exemple, $a \leq b \leq c \Leftrightarrow \{a * b < c, a * b * c, a < b * c, a < b < c\}$

Evidemment, il n'existe pas d'expression unique formée par des relations *, <, $\leq$ et || pour tout treillis donné (voir les exemples données dans cette section). Mais cela permet de décrire d'une façon "semi-linéaire" tous les treillis possibles et avec un ensemble réduit d'opérateurs.

3.3.1.4 <u>Le désordre parmi d'autres notations particulières</u>: Les résultats de la section précédente sont en rapport avec l'analyse du flot de données [DA 79] et les transformations entre programmes parallèles et programmes non-déterministes [AM 71], car l'organigramme d'un programme parallèle peut être représenté comme un treillis avec la relation d'ordre partiel a <u>doit être exécuté avant</u> b.

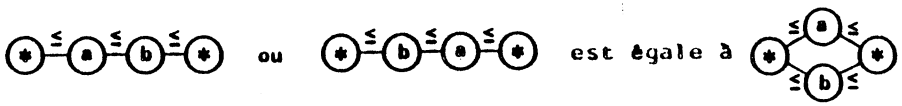
Dans cette section, on essaie de donner d'autres notations particulières qui peuvent être utiles à l'expression d'un prédicat général formé à l'aide des relations *, <, $\leq$ et ||, et des opérateurs logiques &, V et ¬. On exploite le fait que tout prédicat général de ce type est équivalent à un prédicat <u>disjonctif</u>, c'est-à-dire de la forme Vn(P1, ..., Pn), ou chaque Pi (i $\in$ {1, n}) est de la forme &ni(Pi1, ..., Pini) et où chaque Pij (j $\in$ {1, ni}) est formé d'une seule relation d'ordre *, <, $\leq$, ou d'une relation d'ordre précédée de la négation ¬. Autrement dit, le prédicat est une disjonction de conjonctions de relations ou de leurs négations.

Les formes Vn et &n sont équivalentes à:

Vn(P1, ..., Pn)$\equiv$(...((P1 V P2) V P3) ... V Pn)

&n(P1, ..., Pn)$\equiv$(...((P1 & P2) & P3) ... & Pn)

Si le prédicat disjonctif ne contient pas de négations, chacune des conjonctions conformes peut être exprimée par une expression semi-linéaire à l'aide de l'opérateur || et les conjonctions non-conformes peuvent être supprimées du prédicat sans affecter sa sémantique. Il y a des cas où la disjonction peut être simplifiée davantage, par exemple:



mais

ne peut pas être simplifié, au sens de la réduction des symboles différents de *, dans un seul treillis car les ordres sont incompatibles.

Le <u>désordre</u> d'un ensemble B $\subseteq$ A peut être exprimé comme b1 || b2 || b3 || ...|| bm où B={b1, b2, ..., bm} et représente la

disjonction des m! expressions linéaires possibles de B où toutes les relations sont $\leq$, et les m! par 2 puissance m-1 expressions linéaires possibles de B où toutes les relations sont * ou <

Une première possibilité est la <u>factorisation</u> des expressions, c'est-à-dire que, si l'on exprime la disjonction avec le signe + et si l'on utilise des parenthèses on peut faire, comme en algèbre élémentaire, les opérations suivantes:

$$(xoy1 + xoy2 + \dots + xoyn) \equiv xo(y1 + y2 + \dots + yn)$$

$$(y1ox + y2ox + \dots + ynox) \equiv (y1 + y2 + \dots + yn)ox$$

où o est un opérateur & , * , < , $\leq$ ou ||. Il faut remarquer que pour les relations * , < , $\leq$ cette opération de factorisation n'est pas commutative, c'est-à-dire

$$x.(y1 + y2) \neq (y1 + y2).x$$

<u>Exemple</u> :

$a \leq ((b \leq (c || d) \leq e) || (f \leq g)) \leq i$

$\equiv a \leq (b \leq (c \leq d + d \leq c) \leq e \leq f \leq g$

$+b \leq (c \leq d + d \leq c) \leq f \leq (e \leq g + g \leq e)$

$+b \leq (c \leq f \leq d + d \leq f \leq c) \leq (e \leq g + g \leq e)$

$+b \leq (c \leq f \leq g \leq d + d \leq f \leq g \leq c) \leq e$

$+b \leq f \leq (c \leq d + d \leq c) \leq (e \leq g + g \leq e)$

$+b \leq f \leq (c \leq g \leq d + d \leq g \leq c) \leq e$

$+b \leq f \leq g \leq (c \leq d + d \leq c) \leq e$

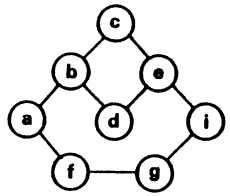
$+f \leq b \leq (c \leq d + d \leq c) \leq (e \leq g + g \leq e)$

$+f \leq b \leq (c \leq g \leq d + d \leq g \leq c) \leq e$

$+f \leq b \leq g \leq (c \leq d + d \leq c) \leq e$

$+f \leq g \leq b \leq (c \leq d + d \leq c) \leq e) \leq i$

qui représente le treillis



□

Ce type d'opération a déjà été donné en [GE 80] pour les E-graphes et dans [KN 73] pour indiquer justement les différentes linéarisations des tris topologiques.

Une autre possibilité est l'utilisation de notations particulières mais assez générales pour exprimer un grand nombre de cas. On définit les opérateurs suivants:

$x \# y \equiv (x * y) \vee (y * x)$ lu x est <u>voisin</u> de y

$x \ast y \equiv (x < y) \vee (y < x)$ lu x est <u>écarté</u> de y

$x \sim y \equiv (x * y) \vee (y < x)$ lu x est <u>semi-écarté</u> de y

$x ; y \equiv (x \# y) \vee (x \ast y) \equiv (x \sim y) \vee (y \sim x)$ lu x est <u>dissocié</u> de y

$\{x1\ o1\ x2\ o2 \dots on\text{-}1\ xn\} \equiv (x1\ o1\ x2\ o2 \dots on\text{-}1\ xn) \vee (xn\ on\text{-}1 \dots o2\ x2\ o1\ x1)$, dite <u>image miroir</u>

On définit donc d'une façon générale l'opération suivante: Soient xi, yj des expresions linéaires qui n'ont que des

relations * , < ou ≤ alors

$$\{x1, x2, \ldots, xn\} o \{y1, y2, \ldots, ym\} \equiv \cup i \cup j (xi \; o \; yj)$$

où o est l'opérateur ||, #, *, ~ ou ; , et

$$\{\{x1, x2, \ldots, xn\}\} \equiv \{\{x1\}, \{x2\}, \ldots, \{xn\}\}$$

([ici, comme dans la section précédente, $\{x1, x2, \ldots, xn\} = Vn(x1, x2, \ldots, xn)$]).

<u>Exemple</u> :

$(a \# b) \# c \equiv (a * b + b * a) \# c$

$\equiv (a * b) \# c + (b * a) \# c$

$\equiv a * b * c + c * a * b + b * a * c + c * b * a$

$(a * b)^{\sim} c \equiv (a < b + b < a)^{\sim} c$

$\equiv (a < b)^{\sim} c + (b < a)^{\sim} c$

$\equiv a < b * c + c < a < b + b < a * c + c < b < a$

$(a < b) ; (c < d) \equiv a < b * c < d + c < d * a < b + a < b < c < d + c < d < a < b$

$a < (b ; c) < d \equiv a < b * c < d + a < c * b < d + a < b < c < d + a < c < b < d$

$\equiv a < (b \| c) < d$

$(a < b) \| (c < d) \equiv a < b \leq c < d + a \leq c \leq b \leq d + a \leq c < d \leq b + c \leq a < b \leq d + c \leq a \leq d \leq b + c < d \leq a < b$

□

On remarque que les opérateurs de dissociation (;) et de parallélisme (||) sont différents sauf lorsqu'il s'agit des éléments simples et non de relations. Ainsi le <u>désordre</u> d'un ensemble $B \subseteq A$ peut être exprimé comme b1 ; b2 ; ... ; bm où $B = \{b1, b2, \ldots, bm\}$. L'idée de la preuve est donnée dans l'exemple suivant:

$(a ; b) ; c \equiv a * b ; c + a < b ; c + b < a ; c$

$\equiv a * b * c + a < b * c + b * a * c + b < a * c$

$+ a * b < c + a < b < c + b * a < c + b < a < c$

$+ c * a * b + c * a < b + c * b * a + c * b < a$

$+ c < a * b + c < a < b + c < b * a + c < b < a$

$\equiv a \leq b \leq c + b \leq a \leq c + c \leq a \leq b + c \leq b \leq a$

$\equiv (a \| b) \leq c + c \leq (a \| b)$

$\equiv (a \| b) \| c$

Pour éviter l'abus des parenthèses dans ces expressions, on définit une priorité d'application des opérateurs comme dans les expressions des langages de programmation. On a donc les niveaux

1. les expressions entre opérateurs miroir { }
2. les expressions entre parenthèses ( )
3. les relateurs * et <
4. la négation ¬
5. le relateur ≤
6. les conjonctions &
7. les opérateurs ; et ||
8. les opérateurs # , * et ~
9. les disjonctions V

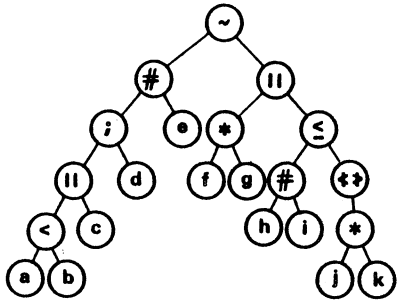
qui donnent une priorité à l'opérateur o1 sur l'opérateur o2 si

le niveau de o1 est inférieur au niveau de o2 ou s'ils ont le même niveau mais o1 est plus à gauche que o2 dans l'expression

Exemple : L'expression

a < b || c ; d # e ~ f * g ||(h # i)≤ { j * k }

représenté comme un arbre donne



La sémantique de ces expressions peut être définie, par exemple, d'une façon dénotationelle [Ve 79] ou par un système de réduction [Ro 73]. Pour la négation, on trouve les expressions suivantes

¬ a * b ≡ a < b ou b ≤ a ≡ b * a ou a * b ≡ b ~ a ou b < a

¬ a < b ≡ a * b ou b ≤ a ≡ b < a ou a # b ≡ a ~ b ou b * a

¬ a ≤ b ≡ b ≤ a

¬ a ~ b ≡ b ~ a

¬ a # b ≡ a * b

Un prédicat général qui peut être écrit comme la disjonction d'expressions linéaires mutuellement superposables est dit plein. Un prédicat général qui n'est pas plein est dit fragmentaire. Cette classification n'est pas exactement la même que celle donnée dans [Me 66] page 28, car elle demande la non répétition des littéraux dans chaque membre de la disjonction. Elle est importante pour l'utilisation des expressions d'ordre dans les règles.

3.3.2 Règles

Une règle indique d'une façon précise comment reconnaître les différents noeuds d'une m.a.t. sur lesquels on doit réaliser un certain nombre de transformations. Elle est composée de deux parties: une première de reconnaissance et une deuxième de transformation. La partie de reconnaissance se compose d'un schéma qui impose les conditions géométriques ou structurales et d'un prédicat simple ou composé à vérifier par les valeurs des décorations des noeuds repérés et par leur ordre relatif. Cela détermine l'applicabilité locale d'une règle, c'est-à-dire, indépendamment des autres règles de la grammaire utilisée. La partie de transformation indique le schéma image qui sera suivi pour la nouvelle disposition des noeuds repérés et des noeuds créés, et toutes les valeurs des décorations des noeuds explicites de l'image ainsi que les nouvelles places données aux descendants implicites du schéma.
La syntaxe est la suivante:

<règle>::=<nom de règle> : <reconnaissance> == <transformation>
<reconnaissance>::=<schéma> / <prédicat> | <schéma>
<transformation>::=<image> / <modifications> | <image>

3.3.2.1 Schémas: C'est la partie d'une règle qui définit les conditions géométriques ou structurales demandées aux sous-ensembles de noeuds d'une m.a.t. objet O pour être choisis et liés à l'ensemble de "points" qui forment une structure arborescente S. Cette structure ou schéma S peut être de deux types: un schéma de premier niveau qui est testé seulement au premier niveau de O si la m.a.t. objet est un noeud composé liste; ou un schéma général qui est une arborescence où les noeuds, appelés points, peuvent être liés à des noeuds simples de O, à des listes de noueds de O ou à des pistes de noeuds de O. Si un point p est lié à un noeud q, les fils de p dans S seront liés à des descendants de q qui ne seront pas forcément des fils de q car l'existence de schémas verticaux laisse une liberté dans le choix du niveau du descendant et l'existence de schémas horizontaux peut lier un point à plusieurs fils de q. Les différents sous-ensembles ainsi repérés peuvent être partiellement ordonnés grâce aux priorités verticales et horizontales affectées aux points du schéma. Un schéma peut porter des précisions sur les décorations que l'on attend trouver dans les noeuds correspondants. Il définit aussi un ordre partiel sur les positions relatives des descendants d'un point. La syntaxe est la suivante:

<schéma>::=<schéma de 1er niveau> | <schéma général>
<schéma de 1er niveau>::=[<point nommé> <schémas descendants>]
<schéma général>::=<schéma ramifié> | <schéma vertical>
<schémas descendants>::=(<liste de schémas généraux>) | (<prédicat d'ordre de schémas généraux>)
<schéma ramifié>::= <point familial> <schémas descendants> | <point de repère>
<schéma vertical>::= :<priorité verticale> <schéma ramifié>
<liste de schémas généraux>::=<schéma> | <schéma horizontal> | <schéma> , <liste de schémas généraux> | <schéma horizontal> , <liste de schémas généraux>
<point familial>::=<priorité horizontale> <point>
<point de repère>::=<point> | <point feuille>
<priorité verticale>::= ↑ | ↓ | vide
<schéma horizontal>::= ..<point nommé>
<priorité horizontale>::= < | > | vide
<point>::=<point nommé> | <point décoré>
<point feuille>::=<point> (*)
<point nommé>::= <nom unique>
<point décoré>::= [<nom unique> ; <décoration>]

Exemple : Parmi les schémas, on trouve les suivants
1, <11, >45, 1(*), <5(*), >7(*), [1;A], >[1;A](*), <[3;B], :[2;B], :↓3, :↑<[4;C], :↓>[10;D](*), 1(2(3, 4), 5), [1(..2, 8, 11, 4)], <5([1;A](>[11;M](*), :↓3), :↑<[10;D](*), 7(..24))
□

Le premier point donné dans le schéma est appelé racine du schéma. Elle est au niveau le plus petit du schéma et tout autre point de celui-ci en est le descendant.

Afin d'éviter l'apparition de schémas et de points facultatifs, on n'accepte ici que des prédicat d'ordre qui soient pleins. Ainsi chaque permutation réalisable fait référence à tous et chacun des schémas descendants. D'autre part, pour éviter d'alourdir l'écriture des règles, on doit permettre l'utilisation de fonctions ou procédures pour le test de ces prédicats qui peuvent être très complexes et pouvoir de cette manière n'écrire qu'une seule fois chaque schéma descendant au moment de l'appel de la procédure correspondante. Un autre avantage est de pouvoir partager ces procédures entre les différentes règles d'une grammaire.

S'il n'existe pas de sous-ensemble non vide de noeuds d'une m.a.t. objet O qui vérifient un schéma S, on dit que la règle n'est pas applicable, sinon on dit que le schéma est applicable. Chacun de ces sous-ensembles, s'il y en a, définit une occurrence élémentaire du schéma qui est une arborescensce C de la même forme que le schéma, où les décorations portent le nom du point du schéma qui lui correspond et les noms des nceuds de O qui sont liés à ce point du schéma. Plus précisément, une arborescence C est une occurrence élémentaire dans O d'un schéma S, ce que l'on note Océ(C, O, S), si l'une des conditions suivantes est vérifiée.

1. S est un point de repère, C est un noeud simple {n;(n1, n2)}, le nom de S est n1, n2 est le nom d'un noeud simple de O et en plus

   1.1. si S est un point feuille alors nfils(O, n2)=0

   1.2. si S est un point décoré alors sa décoration est égale à celle du noeud n2 de O

   (On dit que n2 est lié, ou correspond à n1 par C)

2. S est un schéma vertical : v S1, v est sa priorité verticale, C est telle que Etage(C)={n;(n1, n2, n3)}, le nom du point familial de S1 est n1, a=extraire(O, n2, n3) est un arbre ou piste qui représente un chemin dans O et de plus

   2.1. si S1 est un point repère alors C est un noeud simple et {n;(n1, n3)} est une occurrence élémentaire dans O de S1

   2.2 si S1 est un schéma ramifié alors C est un arbre p f et {n;(n1, n3)} f est une occurrence élémentaire dans O de S1

   (On dit que n2, n3 correspondent ou sont liés à n1 par C)

3. S est un schéma ramifié p1 f1, C est un arbre p2 f2, p2={n;(n1, n2)} est une occurrence élémentaire dans O de p1, le nombre de fils véritables de n2 est différent de zéro (nfils(O, n2) $\geq$ m), f1 est composé des schémas de X={S1, S2, ..., Sm}, f2=(f21, f22, ..., f2m), il existe une permutation Xi de X telle que

   a) Chaque f2j soit une occurrence élémentaire dans O d'un Si unique de X, c'est-à-dire: $\forall j \in [1, m]$

      si Xi(j) est un schéma horizontal ..Y alors

      f2j est un noeud simple {n';(n21, n22, n23)}, le nom de Y est n21, extraire(O, n22, n23) est une liste, n22 et n23 sont les noms de 2 fils véritables du

noeud de O de nom n2 (lié à p1 par p2) liés à n21 par f2j

si Xi(j) est un schéma vertical alors

Océ(f2j, O, Xi(j)) telle que Etage(f2j)={n';(n21, n22, n23)}, n22 est un descendant de n2 dans O, n23 est un descendant de n22 dans O (c'est-à-dire, arbor(arbor(O, n2), n22)≠⊥ & arbor(arbor(O, n22), n23)≠⊥), n22 et n23 sont liés à n21 par f2j

sinon Océ(f2j, O, Xi(j)) telle que Etage(f2j)={n';(n21, n22)}, n22 est un fils véritable de n2 dans O lié à n21 par f2j (c'est-à-dire, ∃ k ∈ {1, nfils(O, n2)} : fils(O, k, n2)= n22)

b) L'ordre (f21, f22, ..., f2m) correspond à l'ordre dans O des noeuds auquels ils sont liés, c'est-à-dire

Soit L2j l'ensemble des noeuds de O liés à Xi(j) par f2j alors ∀ j, j' ∈ {1, m} j < j' => ∀ nj ∈ L2j ∀ nj' ∈ L2j' ∃ k, k' ∈ {1, nfils(O, n2)} tels que

k < k' et nj soit un descendant du k-ième fils de n2 (c'est-à-dire arbor(arbor(O, fils(O, k, n2)), n)≠⊥) et nj' soit un descendant du k'-ième fils de n2

c) La permutation de X induite par l'ordre des fils de n2 liés à X au travers de f2 vérifie le <u>prédicat d'ordre</u> associé à X, c'est-à-dire : Soient g1 et g2 les fonctions suivantes

$$g1(j)=\begin{cases} g1(j-1)+1 & \text{si } j>1 \text{ et } \nexists\, nj \in L2j,\ \nexists\, n'j \in L2\ j-1,\ \nexists\, k \in \{1, nfils(O, n2)\} \text{ tels que: } n'j \text{ et } nj \text{ soient descendants du } k\text{-ième et du } k+1\text{-ième fils de } n2 \text{ respectivement (c'est-à-dire, ils sont voisins)} \\ k & \text{si } L2j=\{nj\} \text{ et } nj \text{ est descendant du } k\text{-ième fils de } n2 \text{ ou } L2j=\{nj, n'j\} \text{ et } nj \text{ est descendant du } k\text{-ième fils de } n2 \text{ et } n'j \text{ est descendant du } l\text{-ième fils de } n2 \text{ et } k \leq l \text{ (c'est-à-dire, } nj \text{ est écarté à droite de } L2\ j-1 \text{ ou } j=1) \\ 0 & \text{sinon} \end{cases}$$

$$g2=\begin{cases} g1(m) & \text{si } \nexists\, nm \in L2m : nm \text{ est descendant du dernier fils de } n2 \\ g1(m)+1 & \text{sinon} \end{cases}$$

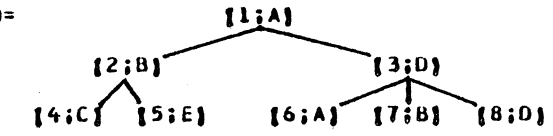
La permutation <g1(1), g1(2), ..., g1(m)> comprise dans <1, 2, ..., g2> induit sur X la permutation <S g1(1), ..., S g1(m)>

d) Le prédicat d'ordre associé à une liste de schémas généraux (S1, S2, ..., Sm) est par défaut *≤S1*S2* ... *Sm≤*, c'est-à-dire l'ordre total spécifié et l'exclusion d'occupants implicites internes

4. S est un schéma de premier niveau {n p}, C est une liste {mi(n, n1)} f, niveau(O, n1)=0, Océ(C, O, S') où S'= n p est un schéma général

<u>Exemple</u> : Soit la m.a.t. objet O suivante

O=



Soient les schémas suivants

S1=3 S5={3;A}(*) S9 =1(2(*), 3)

S2=3(*) S6=:[3;D] S10=1(2, :3)

S3=:3 S7=:[3;D](*) S11=1(..2)

S4=[3;A] S8=1(2, 3) S12=1(..2, ..3)

et soient les arborescences

C1=[1;(3, 1)] C4=[1;(3, 4)] C7=[1;(3, 1, 3)])

C2=[1;(3, 2)] C5=[1;(3, 5)] C8=[1;(3, 2, 5)])

C3=[1;(3, 6)] C6=[1;(3, 1, 6)] C9=[1;(3, 3, 8)])

C10=[1;(3, 8, 8)]

C11=[2;(1, 1)]([1;(2, 2)], [5;(3, 3)]

C12=[3;(1, 2)]([8;(2, 4)], [1;(3, 5)]

C13=[4;(1, 3)]([2;(2, 6)], [3;(3, 7)]

C14=[5;(1, 1)]([1;(2, 2)], [8;(3, 3, 3)]

C15=[1;(1, 1)]([5;(2, 2)], [7;(3, 3, 8)]

C16=[4;(1, 1)]([1;(2, 2, 3)])

C17=[2;(1, 3)]([6;(2, 6, 7)])

C18=[6;(1, 3)]([2;(2, 6, 8)])

C19=[3;(1, 1)]([5;(2, 2, 2)], [1;(3, 3, 3)])

C20=[7;(1, 3)]([3;(2, 6, 7)], [1;(3, 8, 8)])

C21=[7;(1, 3)]([4;(2, 6, 6)], [1;(3, 7, 8)])

On peut trouver les occurrences élémentaires suivantes

Oc̄(C1, O, S1) Oc̄(C1, O, S4) Oc̄(C12, O, S9)

Oc̄(C2, O, S1) Oc̄(C3, O, S4) Oc̄(C14, O, S10)

Oc̄(C3, O, S1) Oc̄(C3, O, S5) Oc̄(C15, O, S10)

Oc̄(C3, O, S2) Oc̄(C9, O, S6) Oc̄(C16, O, S11)

Oc̄(C4, O, S2) Oc̄(C10, O, S6) Oc̄(C17, O, S11)

Oc̄(C5, O, S2) Oc̄(C10, O, S7) Oc̄(C18, O, S11)

Oc̄(C6, O, S3) Oc̄(C11, O, S8) Oc̄(C19, O, S12)

Oc̄(C7, O, S3) Oc̄(C12, O, S8) Oc̄(C20, O, S12)

Oc̄(C8, O, S3) Oc̄(C13, O, S8) Oc̄(C21, O, S12)

□

Les priorités affectées aux différents points d'un schéma S, applicable à une m.a.t objet O, donnent un ordre partiel à l'ensemble des occurrences élémentaires dans O de S. L'interprétation des priorités se fait par rapport à l'ordre partiel défini par les niveaux des noeuds et par rapport à l'ordre partiel défini par le fait que les arborescences soient orientées. Ainsi, si n1 et n2 sont les noms de deux noeuds de O, on dit

n1 est <u>plus haut</u> que n2 si niveau(O, n1) < niveau(O, n2)

n1 est <u>plus à gauche</u> que n2 si n1 et n2 sont descendants d'un noeud n de O (c'est-à-dire, arbor(arbor(O, n), n1) $\neq \perp$ & arbor(arbor(O, n), n2)$\neq\perp$) et n1 est descendant du k-ième fils (ou élément) de n et n2 est descendant du k'-ième fils (ou élément) de n et k<k'

Etre plus haut et plus à gauche sont des relations d'ordre partiel.

Soient C1 et C2 deux occurrences élémentaires dans O de S, on dit que C1 est <u>pricritaire</u> par rapport à C2, et on note C1 <u>pp</u> C2, si une des conditions suivantes est vérifiée:

1. Etage(C1)=[n1; (n, n12)], Etage(C2)=[n2; (n, n22)] et

1.1 niveau(O, n12)= niveau(O, n22) et

1.1.1 Ou bien, n porte une priorité horizontale vers la gauche

(<) et n12 est plus à gauche que n22

1.1.2 Ou bien, n porte une priorité horizontale vers la droite (>) et n22 est plus à gauche que n12

2. Etage(C1)={n1; (n, n', n13)}, Etage(C2)={n2; (n, n', n23)} et

2.1 Ou bien, niveau(O, n')= niveau(O, n13)= niveau(O, n23) et n13 est plus à gauche que n23 (c'est-à-dire, la liste la plus courte est prioritaire)

2.2 Ou bien, "n'" est plus haut que n13 et n13 est plus haut que n23 et n porte une priorité vers le haut (↑)

2.3 Ou bien, "n'" est plus haut que n23 et n23 est plus haut que n13 et n porte une priorité vers le bas (↓)

2.4 Ou bien, "n'" est plus haut que n13 et niveau(O, n13)= niveau(O, n23) alors les conditions de 1 s'appliquent à n13 et n23

3. Etage(C1)={n1; (n, n12, n13)}, Etage(C2)={n2; (n, n22, n23)} et n12 et n22 sont descendants d'un noeud n' et n12≠n22 alors les conditions de 2 s'appliquent à n12 et n22

4. C1=p1 (f11, ..., f1m), C2=p2 (f21, ..., f2m) et

4.1 Ou bien, p1 <u>pp</u> p2

4.2 Ou bien, p1 = p2 et ∄ k ∈ {1, m} : ((1 ≤ i < k => f1i = f2i) et f1k <u>pp</u> f2k)

<u>Exemple</u> : Pour la même m.a.t. objet O de l'exemple précédent, le schéma 1(..→2, :↓→3) définit l'ensemble des occurrences élémentaires

C1=(1,1) ∧ (2,2,2)(3,3,3) C2=(1,1) ∧ (2,2,2)(3,3,6) C3=(1,1) ∧ (2,2,2)(3,3,7) C4=(1,1) ∧ (2,2,2)(3,3,8)

C5=(1,2) ∧ (2,4,4)(3,5,5) C6=(1,3) ∧ (2,6,6)(3,7,7) C7=(1,3) ∧ (2,6,7)(3,8,8) C8=(1,3) ∧ (2,7,7)(3,8,8)

Elles forment les chaînes

C4 <u>pp</u> C3 <u>pp</u> C2 <u>pp</u> C1

C8 <u>pp</u> C6 <u>pp</u> C7

C5

qui peuvent être représentées par une m.a.t.

{(C4, C3, C2, C1), C5, (C8, C6, C7)}

□

Le test d'un schéma sur une m.a.t. objet produit comme résultat <u>1</u> si le schéma n'est pas applicable, sinon il produit une s.m.a.t. qui porte toutes les occurrences élémentaires triées par priorités sous la forme de listes qui représentent des chaînes d'ordre. Ce résultat est appelé la <u>conjecture</u> du schéma et peut être écrit d'une manière très compacte à l'aide de l'opérateur de factorisation.

<u>Exemple</u> : Supposons que l'ensemble d'occurrences élémentaires d'un schéma est {C1, C2, C3, C4, C5, C6, C7} et qu'il forme la conjecture {(C1, C2, C4, C7), (C1, C2, C5, C7), (C1, C3, C6, C7), (C1, C3, C5, C7)} alors la factorisation de cette conjecture nous donne

(C1, <u>F</u>({(C2, <u>F</u>({C4, C5})), (C3, <u>F</u>({C5, C6}))}), C7)

où F est l'opérateur de factorisation

¤

3.3.2.2 Prédicats: Il s'agit d'une expression logique formée à l'aide de prédicats simples et des opérateurs logiques ¬, &, V (négation, conjonction et disjonction). On peut supposer que sa forme est une disjonction de conjonctions de prédicats simples ou de négation de prédicats simples. Chaque prédicat simple est une expression de comparaison entre des expressions du même type. On accepte au moins les opérateurs de comparaison =, ≠, <, ≤, >, ≥, les opérateurs ensemblistes ∈, c, ⊆, I, U, les opérateurs arithmétiques +, −, *, /, les opérateurs de manipulation de décorations, et les propriétés et opérateurs sur des m.a.t.

Une expression logique peut être quantifiée universellement ou existentiellement pour des variables représentant des éléments liste ou des éléments noeud appartenant à des chemins stricts.

Les opérateurs seront appliqués aux noms des points du schéma. Chaque fois que l'on fait référence à un nom de schéma horizontal on suppose une référence à la liste de points qui lui sont liés. Chaque fois que l'on fait référence à un nom de schéma vertical précédé de deux points (:), on suppose une référence au chemin complet associé au schéma et si l'on utilise le nom d'un schéma vertical sans les deux points on fait référence à la m.a.t. associée au chemin.

Si dans l'utilisation d'une proriété ou opérateur de m.a.t., on ne donne pas la m.a.t. par rapport à laquelle on l'applique, on suppose la m.a.t. objet 0.

Exemple : Voici quelques prédicats simples

contenu(1) = contenu(2),

1 = 2,

néléments(2) > néléments(3) + néléments(4),

narbs(arbor(3), 4) = 5,

niveau(5) ≥ 3,

nnoeuds(arbor(5)) < nnoeuds(arbor(4)),

arbor(3) ⊆ arbor(5),

arbor(1) ≠ arbor(2),

1 ∈ arbor(4),

cheminS(:4)= :4,

niveau(dominant(:4)) > niveau(associé(:4)),

niveau(associé(:4)) − niveau(dominant(:4)) > 5,

néléments(..3) > 1,

Etage(1) = Etage(Sous-Etages(8))

¤

Les conditions posées sur les points d'un schéma sont testées sur les noeuds associés à ces points sur chaque occurrence élémentaire de la conjecture du schéma. Les occurrences qui ne vérifient pas le prédicat sont retirées de la conjecture. Le résultat final sera l'ensemble des occurrences triées par

prioritées pour lesquelles le prédicat est vrai. Il est appelé la <u>conjecture locale</u> de la règle qui sera égale à $\perp$ si aucune des occurrences n'a vérifié le prédicat et dans ce cas on dit que la règle n'est pas applicable.

Si la partie reconnaissance ne comporte pas de prédicat, on suppose l'existence d'un prédicat toujours <u>vrai</u>.

3.3.2.3 <u>Image</u>: Cette partie décrit le futur modèle pour la réalisation éventuelle de la transformation structurale des parties de la m.a.t. objet repérées par le schéma de la règle. L'image permet de construire une m.a.t. à partir des noeuds liés aux points du schéma et qui remplacera la m.a.t. repérée par le schéma. De cette façon le résultat est encore une m.a.t.

L'image présente une forme semblable aux schémas avec quelques variations. Elle porte, comme les schémas, des points qui ont un nom. Celui-ci peut apparaître plusieurs fois dans l'image et il peut être égal à un des noms de point du schéma correspondant. Il n'est pas obligatoire que tous les points du schéma soient retrouvés dans l'image. Si l'ensemble des noms de l'image est T, l'ensemble des noms de points du schéma est R et I est l'opérateur d'intersection ensembliste, on dit que les points dans R - (R I T) disparaissent et que les points dans T - (R I T) sont créés.

Dans l'image on peut mettre en jeu des <u>listes implicites</u> du schéma qui sont les parties de la m.a.t. objet qui se trouvent à l'intérieur de la zone touchée par la transformation mais qui n'ont pas été mentionées explicitement au travers d'un point du schéma. Plus précisément, pour tout point p d'un schéma S, on peut définir ses listes implicites de la façon suivante

1. Si p est une feuille ou un schéma horizontal il ne porte qu'une liste implicite vide.
2. Si p est un point de repère non-feuille, il porte une seule liste implicite qui correspond à la liste de m.a.t. fils du noeud de la m.a.t. objet O qui est lié au point de repère.
3. Si p est un schéma de premier niveau ou le point familial d'un schéma ramifié et si n est le nombre de schémas généraux qui forment les schémas descendants de p alors p porte n+1 listes implicites.

   C'est-à-dire, quelle que soit la permutation <p1, p2, ..., pn> donnée dans une occurrence C du schéma et pour $\forall$ i $\in$ [2, n] la liste implicite i est composée des m.a.t. dans O, fils du noeud de O associé à p et qui se trouvent entre le noeud le plus à droite associé à p i-1 et le noeud le plus à gauche associé à pi;

   la liste implicite 1 est composée des m.a.t. dans O, fils du noeud de O associé à p qui se trouvent à gauche du noeud le plus à gauche associé à p1;

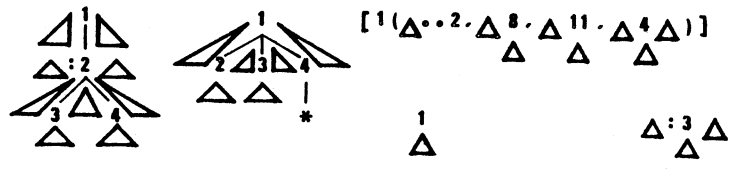
   la liste implicite n+1 est composée des m.a.t. dans O, fils du noeud de O associé à p qui se trouvent à droite du noeud le plus à droite associé à pn;

Parmi ces n+1 listes il peut y avoir des listes vides.

4. Si p est un schéma vertical : v p1, on trouve, en plus des listes implicites de p1, deux listes propres à p qui correspondent aux traînées gauche et droite de la piste qui a comme trace la m.a.t. associée au chemin. La traînée gauche est la liste implicite 1 et la droite est la 2.

Un schéma qui ne porte que des listes implicites vides est dit <u>complet</u>.

<u>Exemple</u> : On indique par des triangles les listes implicites des schémas suivants



□

Les listes des m.a.t. associées aux schémas horizontaux seront appelées <u>listes explicites</u>. Les éléments appartenant aux listes explicites et implicites, et aux chemins associés aux schémas verticaux, ne pourront pas être manipulés de manière individuelle mais comme un tout à l'aide des opérateurs appropriés. Ainsi on pourra faire appel aux opérateurs cheminS, cheminD, cheminG, Etage et Sous-Etages. Dans le cas des chemins, on se réfère à la m.a.t. associée au chemin par le nom du schéma vertical sans les deux points (:) et au chemin sans la m.a.t. associée par le nom du schéma vertical suivi de deux points de telle sorte qu'un schéma :p soit équivalent dans l'image à :p ou à p: p. Dans ce dernier cas on décompose explicitement le chemin.

Les listes implicites et explicites sont une généralisation des occupants implicites et explicites de la section 3.3.1.2.

Pour finir, on accepte dans l'image des expressions d'ordre sur les images descendantes dans les structures ramifiées en tant que moyen de donner une famille d'images comme résultat de la transformation. Pour chaque permutation admise dans l'expression on aura une image "fixe". La syntaxe est la suivante:

<image> ::= <image ramifiée> | <image verticale> |
<image horizontale>

<image ramifiée> ::= <point d'adoption> <images adoptives> |
<point adoptif>

<image verticale> ::= <chemin sans issue>: <image> |
:<chemin achevé> <images adoptives> | <point adoptif>

<image horizontale> ::= .. <liste implicite> |
..<liste explicite> | (<point nommé> <images adoptives>
)

<point d'adoption> ::= <point>

<images adoptives> ::= (<liste d'images>) |
(<expression d'ordre d'images>)

<point adoptif> ::= <point> | <point feuille>

<chemin sans issue> ::= <chemin provenant d'un schéma vertical>

<chemin achevé> ::= <chemin provenant d'un schéma vertical>

<liste implicite> ::= (<nom d'un point du schéma> ,
<numéro d'ordre>)

<liste explicite> ::= <liste provenant d'un schéma horizontal>

<point> ::= <point nommé> |<point décoré>

<liste d'images> ::= <image> | <image> , <liste d'images>

<point feuille> ::= <point> (*)

<chemin provenant d'un schéma vertical> ::=
<nom d'un point schéma vertical> |
< cheminS , <nom d'un point schéma vertical> > |
< cheminD , <nom d'un point schéma vertical> > |
< cheminG , <nom d'un point schéma vertical> >

<liste provenant d'un schéma horizontal> ::=
<nom d'un point schéma horizontal> |
< Etage , <nom d'un point schéma horizontal> > |
< Sous-Etages , <nom d'un point schéma horizontal> >

<point nommé> ::= <nom de point>

<point décoré> ::= { <nom de point> ; <décoration> }

<nom de point> ::= <nom unique du schéma> |
<nom créé pour l'image>

Le premier point donné dans l'image est appelé racine de l'image. Elle est au niveau le plus petit de l'image et tout autre point de celle-ci en est le descendant.

Pour garder la symétrie de l'image et du schéma correspondant, on demande que les expressions d'ordre soient, comme dans le schéma, pleines. Ainsi chaque image "fixe" fait référence à toutes et chacune des images descendantes. On prévoit aussi l'utilisation de procédures pour les expressions d'ordre qui pourraient être partagées avec les schémas.

Un point créé dans l'image ne porte comme fils que les points correspondants aux images adoptives qui lui sont affectées. Les points supprimés du schéma disparaissent avec toutes leurs listes implicites qui n'ont pas été mentionnées explicitement dans l'image. Un point du schéma qui est dans l'image, amène avec lui toutes ses listes implicites s'il y a au moins un "trou" dans l'expression d'ordre affectée à ses images descendantes où il puisse les placer. Si l'expression d'ordre n'a pas laissé de "trous" pour ses listes implicites, elles disparaîtront de l'image sauf celles qui ont été placées ailleurs de façon explicite.

L'algorithme de placement des listes implicites d'un point qui est gardé dans l'image est basé sur les suppositions suivantes. Soit p un point du schéma qui porte "n" listes implicites dont "n'" vides. Soit m-1 le nombre d'images adoptives qu'il porte dans l'image. Il y aura une expression d'ordre associée qui utilise au moins m opérateurs d'ordre. S'il n'y a pas d'expression d'ordre explicite, on suppose l'expression

$$* \leq i1 * i2 * \dots * im-1 \leq *$$

S'il n'a pas d'images adoptives alors s'il est un point feuille l'opérateur est * sinon l'opérateur est $\leq$.
Etant donné que l'expression est pleine, elle peut être écrite sous une forme disjonctive d'expressions linéaires superposables telle que chaque ordre relatif donné aux images descendantes n'apparaisse qu'une seule fois. Par exemple, (a * b) V (a < b) est remplacée par (a $\leq$ b), voir aussi les exemples de la section 3.3.1.4. Soit donc l'expression résultante
(e1 V e2 V ... V er) où $\forall$ i $\in$ [1, r]: ei ne porte que des opérateurs *, < et $\leq$, et définit une permutation unique et fixe

des images descendantes. On note l1, l2, ..., ln les listes implicites de p, t1, t2, ..., tm les opérateurs d'ordre d'une expression ei et p1, p2, ..., pn les places données à chaque liste. L'algorithme est le suivant:

```
entiers: i, j, k, nvides
j :=1; k :=1; nvides :=0
Tantque j ≤ n & k ≤ m faire
     (On a encore des listes et des places à distribuer)
     Si lj = ⊥ & tk = '<'
     alors (On met la liste vide à cette place et on cherche une liste non vide pour remplir ce trou)
          pj := k; nvides := nvides + 1; j := j + 1
     sinon
          Si lj ≠ ⊥ & tk = '*'
          alors (On cherche un trou pour mettre cette liste car cette place n'est pas disponible)
               k := k + 1
          sinon (On peut mettre cette liste dans cette place car soit (lj = ⊥ & tk='*' ou '≤') ou soit (li ≠ ⊥ & tk='<' ou '≤'))
               Si lj = ⊥ alors nvides := nvides + 1 finsi
               pj := k; j := j + 1; k := k + 1
          finsi
     finsi
fintantque
Si k > m
alors (On a fini les places et on cherche un emplacement pour les listes restantes)
     Si nvide = 0 & n' = n - j
     alors (Toutes les listes qui restent à placer sont vides)
          pour j := j jusqu'à n
               pj := k - 1
          finpour
     sinon
          Si ∃ k'∈ [1, m] (tk' ='<' ou '≤'  & ∀ k" ∈ [k' + 1, m]:tk" ='*')
          alors (On met les listes qui restent dans le dernier trou utilisé)
               pour j := j jusqu'à n
                    pj := k'
               finpour
          sinon (Il est impossible de placer les listes implicites restantes par conséquent elles sont éliminées)
          finsi
     finsi
sinon
     Si k ≤ m & j > n
     alors (On a fini les listes mais il restent des places)
          Si ∃ k' ∈ [k, m] : tk'= '<'
          alors  (Il est impossible de réaliser la permutation demandée faute de listes implicites pour remplir les trous.
               L'expression ei est éliminée de l'expression d'ordre)
          sinon (On a rempli les places restantes avec des listes vides)
          finsi
     sinon (Les listes et les places sont épuisées en même temps et l'expression a été vérifiée)
     finsi
finsi
```

Cet algorithme détermine un sous-ensemble d'expressions linéaires ei contenues dans l'expression d'ordre associé dans l'image, pour lesquelles il est possible de satisfaire, à l'aide de listes implicites, les restrictions de voisinage

demandées.

Si ce sous-ensemble est vide alors il n'y a pas assez de listes implicites non vides pour remplir les trous et on n'affecte au point dans l'image ni ses listes implicites ni ses images adoptives. Par conséquent, il sera un point feuille et tous les points et listes du schéma qui devraient être gardés comme descendants de ce point dans l'image et qui d'autre part ne se trouvent pas ailleurs dans l'image vont disparaître. Cela peut amener dans certains cas à la production d'une image complètement vide, par exemple dans le cas d'une image horizontale de niveau zéro. Il appartient donc à l'utilisateur de bien étudier les expressions d'ordre qu'il met en jeu et les prédicats sur les contenus des listes implicites pour être sûr du résultat voulu. Par ailleurs, il est aussi intéressant d'avoir la possibilité d'écrire des règles à image vide comme un moyen direct d'élimination de gros morceaux d'une m.a.t. En ROBRA, par exemple, cette possibilité n'existe pas mais on peut aboutir au même résultat par le biais du chapeau des schémas qui permet de décrire un contexte.

Si le sous-ensemble a plus d'une affectation possible, alors on aura différents ordres relatifs pour les images adoptives et chacun avec une unique affectation de listes implicites. Etant donné que ces ordres sont incompatibles, il seront réalisés au moyen d'un noeud ensemble qui porte chaque possibilité une fois.

Autrement, le sous-ensemble n'a qu'un seul ordre relatif possible et une affectation unique des listes implicites. On aura, comme dans ROBRA, un point adoptif avec les listes implicites et les images adoptives dans l'ordre donné, comme fils.

L'algorithme présenté ici a l'avantage de trouver une affectation si et seulement si elle existe et de donner la même place aux listes implicites que dans le schéma si l'image est égale au schéma. C'est-à-dire qu'il produit la transformation identité, ce qui est important lorsque l'on ne veut modifier que les décorations et non la structure. Il y a évidement d'autres algorithmes possibles, notamment l'utilisation des priorités horizontales, comme dans le schéma, pour contrôler les choix.

On remarque que le fait de permettre la copie des éléments dans l'image implique la nécessité de pouvoir éliminer d'une façon explicite un élément si l'on veut réaliser un déplacement du type de la fonction de transfert de ROBRA. Cela est possible par le biais des images complètes, c'est-à-dire lorsqu'on écrit tous les éléments que l'on veut affecter à un point et qu'on ne laisse pas de trous dans l'expression d'ordre correspondante.

Exemple : Soient la m.a.t. objet O, le schéma S et l'occurrence élémentaire C suivantes

```
      O=  {1;A}                                          C=  (1,1)
   {2;B}  {9;E}   {3;D}                  S=  1          (2,2,2)(3,3,6)
{4;C} {5;E} {6;A} {7;B} {8;D}            ..>2 ≤ :>3
```

Voici quelques images et les m.a.t. correspondantes (indiquées par =>) construites à partir de S et de C avec des éléments de O:

```
1  => {1;A} ,   1   =>   {1;A}
      |         |
    {9;E}       *                          ..(1, 2)  => {10({9;E})}

      1     =>     0
  ..2 ≤ :3

      1     =>     {1;A}
  ..2 ≤..2       {2;B} {9;E} {12;B}
            {4;C} {5;E} {14;C} {15;E}

                1     =>                {1;A}
..<Etage, 2>  ..<Etage, 2>        {2;B} {12;B} {9;E}

                1       =>                {1;A}
** ..<Etage, 2>  ..<Etage, 2> **        {2;B} {12;B}

  3:    =>  {3;D}
  |         |
..2         {10</{2;B}/{7;B},{8;D}>}
            {4;C}  {5;E}

  :3    =>  {3;D}
              |
              {10</{6;A}/{7;B},{8;D}>}

  :3    =>  {3;D}
  |         |
..2         {10</{6;A}/{7;B},{8;D}>}
                |
              {2;B}
            {4;C} {5;E}

:{3;F}  =>  {3;D}
              |
              {10</{6;F}/{7;B},{8;D}>}          3 => {6;A}

                1     =>                {1;A}
..<Etage, 2> ≤ ..(:3, 2)          {2;B} {9;E} {10({7;B},{8;D})}

<cheminS, 3>:   => {3;D}
|               |
<cheminD, 3>:     {10</{13;D}/>}
|                   |
..(:3, 2)           {11</{12({17;B},{18;D})}/{7;B},{8;D}>}

                1
..<Etage, 2> ≠ ..(:3, 2)
=> {10{ {1;A}  ,                                                {11;A} }}
  {2;B} {9;E} {13({7;B},{8;D})} {23({17;B},{18;D})}{19;E}{12;B}

{4( 1, ..(1, 2))}   =>  {10(  {1;A}  ,  {19;E})}
..2 ..(1, 1)                {2;B} {9;E}
                          {4;C} {5;E}
```

□

Avec les conventions proposées, la fonction de transfert de RUBRA n'est plus nécessaire car tout ce qu'elle peut réaliser est déjà possible dans notre définition. Cependant, elle peut être encore utile pour simplifier l'écriture des déplacements de dépendants dans l'image et pouvoir combiner en même temps copies et transferts. Cette nouvelle fonction et ses implications restent à définir.

L'introduction d'une notation spéciale pour les listes implicites dans l'image peut surcharger l'écriture notamment dans les expressions d'ordre. Une facilité d'écriture pourrait être appliquée lorsque les expressions d'ordre sont données à un ensemble de listes implicites d'un même point du schéma. Par exemple, si A est le nom d'un point du schéma et on a dans l'image

..(A,1) * ..(A,2)

on devrait pouvoir simplifier par ..(A,1 * 2) ou bien par ..(A,1,2).

Une expression du type ..(A,1,2,3,4,5) pourrait être remplacée par ..(A,1, ..., 5) ou ..(A,1-5), et on devrait avoir la possibilité de noter la dernière liste implicite d'un point d'une façon spéciale par exemple avec $. Ainsi, une expression telle que ..(A,1,$) ferait référence à la première et la dernière liste implicite de A et ..(A,1-$) exprime toutes les listes implicites de A. Des notations de ce type, qui n'ont pas d'effet sur le comportement interne du système, peuvent faciliter le travail de l'utilisateur. Elles restent à définir avec plus de précision.

3.3.2.4 <u>Modifications</u>: C'est la dernière partie d'une règle et elle indique les valeurs de décorations des points de l'image. C'est un moyen de modifier partiellement les décorations des noeuds liés aux points du schéma gardés dans l'image ou d'affecter les décorations d'un noeud avec les valeurs des décorations d'un autre noeud. On remarque que dans l'image on peut donner d'une façon explicite les décorations d'un noeud.

Les modifications ne sont acceptées ni pour les listes ni pour les chemins, étant donné qu'il n'est pas possible de repérer ses éléments isolés, mais elles sont acceptées pour le noeud associé aux chemins.

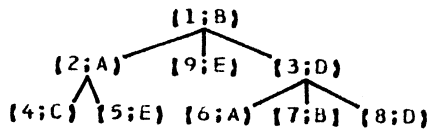
Il reste à résoudre le problème qui se pose lorsque les valeurs de décorations ne sont données ni dans l'image ni dans les modifications. Pour les noeuds liés au schéma et gardés dans l'image les décorations sont les mêmes mais pour les noeuds créés ce n'est pas défini. En réalité, cela dépend d'une définition détaillée des décorations, ce qui n'est pas l'objectif de notre exposé. Si par exemple, on définit une <u>décoration vide</u> on peut accepter ce type de règles et travailler sur des structures "semi-décorées".

Les modifications peuvent être du type: dec(1):=dec(2) ou bien

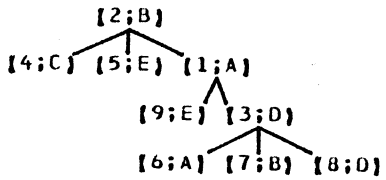
dec(1):=dec(2)+4, ou encore dec(1):='faux'.

<u>Exemple</u>: Soit la m.a.t. objet O de l'exemple précédent. La règle R1:[1;A]([2;B])==[1;B]([2;A]), ne produit pas le même effet sur O que R2:[1;A]([2;B])==2(1).

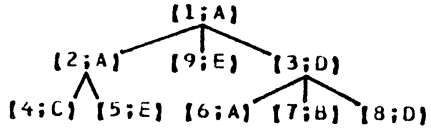
En effet, R1 et R2 ont toutes les deux la conjecture locale C=[1;(1,1)]([2;(2,2)]) mais R1 produit comme résultat



et R2



La règle R3:[1;A]([2;B])==1(2)/dec(1):=:dec(2) est équivalente à R1 si :=: est l'opérateur d'échange. La règle R4:[1;A]([2;B])==1(2)/dec(1):=dec(2),dec(2):=dec(1) produit



n'est pas égale à R1

Finalement, R5:[1;A]([2;B])==2(**1(**..(2,1)**),..(1,2)**) n'est pas égale à R1 mais produit le même résultat que celle-ci dans ce cas particulier.

n

3.3.2.5 <u>Contexte d'une règle</u>: Il s'agit de la zone de la m.a.t. objet à laquelle on fait référence explicite ou implicite mais qui n'est pas modifiée par la transformation. Sur le contexte d'une occurrence de la règle R on peut appliquer d'autres règles en parallèle sans risque de conflit avec R. L'important est d'assurer la survie de la m.a.t. dans le sens que le résultat soit encore une m.a.t.

Dans cette optique, un schéma est divisé en deux parties: une passive et une autre active. Un point associé à un chemin par un schéma vertical, un point familial ou un point de repère est <u>actif</u> si au moins une des conditions suivantes est réalisée:

- il n'apparaît pas dans l'image
- il change de père ou de liste
- sa décoration est modifiée
- l'ordre de ses fils ou de ses éléments est modifié
- il porte un nouveau fils ou élément dans l'image
- il perd un de ses fils ou éléments

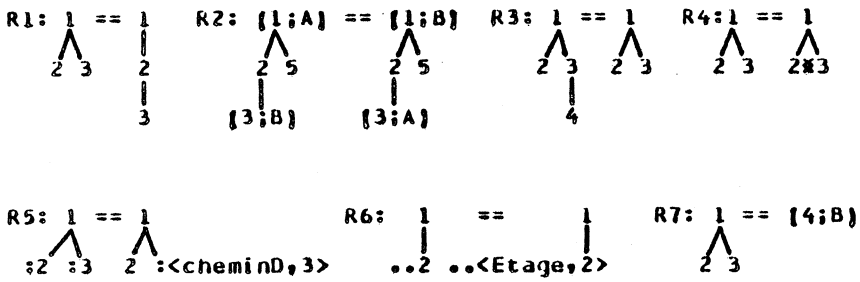
Les points correspondant à des schémas verticaux ou horizontaux sont tous actifs si leurs noms n'apparaissent pas dans l'image. Les points correspondant à des listes implicites qui n'apparaissent pas explicitement dans <u>l'image</u> et qui ne peuvent pas être gardés par le point auquel ils sont associés dans le schéma à cause de l'expression d'ordre affectée à celui-ci dans

l'image, ou bien parce qu'il disparaît, sont tous actifs.

Tous les points correspondant à l'étage d'un schéma horizontal ainsi que tous les points appartenant au chemin strict d'un schéma vertical sont actifs si le nom du schéma est associé à un opérateur Etage, Sous-Etages ou chemin dans l'image. Les points qui disparaissent à cause de ces opérateurs sont actifs.

Un point qui n'est pas actif est <u>passif</u>. Par extension, on dit qu'une m.a.t. est <u>active</u> si elle porte au moins un point actif sinon elle est passive. L'ensemble de points passifs d'un schéma est appelé <u>contexte</u>. On voit donc que, au contraire de ROBRA, le contexte peut être non connexe et qu'il peut être déterminé statiquement.

<u>Exemple</u>: Soient les 7 règles suivantes



D'après les définitions précédentes on peut classer les points comme ceci:

-pour la règle R1, 1 et 2 sont actifs, 3 et toutes les listes implicites sont passifs

-pour la règle R2, 1 et 3 sont actifs, 2, 5 et toutes les listes implicites sont passifs

-pour la règle R3, 3 et 4 de même que les listes implicites de 4 sont actifs

-pour la règle R4, 1 est actif

-pour la règle R5, 1, 2 et tous les points appartenant au chemin de 2 de même que tous ceux appartenant au chemin gauche de 3 sont actifs, 3 en tant que point associé au chemin est passif

-pour la règle R6, 1 et tous les noeuds associés au schéma horizontal sont actifs

-pour la règle R7, tous les points du schéma sont actifs, 4 n'est ni actif ni passif car il n'est pas dans le schéma

¤

3.3.3 <u>Grammaires</u>

Une grammaire est composée d'un ensemble de règles, muni d'une relation d'ordre partiel, et d'un ensemble d'options de contrôle. La relation d'ordre donne une priorité d'application aux règles en cas de conflit <u>hors des contextes</u>, et les options indiquent la façon dont ces règles seront utilisées dans le processus de transformation d'une m.a.t.

La composition des grammaires permet de créer des <u>systèmes transformationnels</u> (ST), c'est-à-dire un ensemble de grammaires structuré en réseau ou graphe orienté et ordonné qui porte des conditions de parcours sur les arcs et des conditions

d'application d'une grammaire sur les noeuds.

Le système prend un ST et une m.a.t. O et donne comme résultat la m.a.t. produite par l'application du ST sur O. Cela est réalisé par une séquence d'applications des grammaires du ST sur les transformations successives de O. Les options de contrôle sur les grammaires et les réseaux essaient entre autres de rendre le système décidable. Le système transformationnel le plus simple est composé d'une seule grammaire avec un prédicat toujours vrai comme condition de parcours. La syntaxe est la suivante:

<système transformationnel>::= <nom de ST> : (<réseau>)

<réseau>::= <nom de noeud> <arc> <noeud> ; <réseau>

| <nom de noeud> <arc> <noeud>

<arc>::= -- <condition de parcours> -->

| --->

<noeud>::= { <nom de noeud> ; <nom de grammaire> <conditions d'application> }

| <nom de noeud> | *

<condition de parcours>::= <boucles> : <reconnaissance>

| <boucles> : ¬(<reconnaissance>)

<condition d'application>::=(<passage> <retour arrière> <attente>) | vide

<boucles>::= B(<entier positif>) | E | vide

<passage>::= I | F | vide

<retour arrière>::= Z | N | vide

<attente>::= & | o | vide

<grammaire>::=<nom de grammaire> <contrôles> : <règles>

<contrôles>::= (<passages> <dépendances> <interdiction par application> <interdiction automatique>) | vide

<règles>::= <nom de règle possible> ≤ <règles> |

<nom de règle possible> || <règles> |

<nom de règle possible> | (<règles>)

<passages>::= E | B (<entier positif>) | vide

<dépendances>::= T | C(↑) | C(↓) | vide

<interdiction par application>::= L | P | D | S | vide

<interdiction automatique>::= A | V(↑,<) | V(↑,>) | V(↓,<) | V(↓,>) | vide

<nom de règle possible>::=<nom de règle> | <règle récursive>

<règle récursive>::=<nom de règle> (<nom de SST> / <points de récursion>)

<nom de SST>::= <nom de ST> | <nom de ST> (<nom de noeud>) | <nom de grammaire> | <règles>

<points de récursion>::=<noms de points de l'image de la règle>

Une règle récursive est composée d'un nom de règle suivi d'un couple: nom de Sous-Système Transformationnel et points de récursion. Les points de récursion sont les noms de points de l'image de la règle qui correspondent à des points actifs du schéma non supprimés dans l'image, ou à des points créés dans l'image. Un point passif du schéma (conservé dans l'image) ne peut pas être un point de récursion. Le <u>contexte de la récursion</u> est composé de l'ensemble de noeuds liés à des points

de l'image qui ne sont pas des points de récursion.

Une règles récursive s'applique comme suit:

- La règle nommée participe à la définition d'une conjecture globale de la même façon que les autres règles non-récursives.
- Lorsqu'elle est applicable et choisie dans la conjecture globale, elle est appliquée normalement comme les autres. mais une fois qu'on a le résultat O1 de la transformation, il sera "découpé" temporairement à partir de la racine P de l'image produite. Ensuite il sera soumis au SST nommé dans l'appel récursif de la règle.
- L'application du SST sur le résultat partiel diffère d'une application normale par l'existence du contexte de la récursion. Aucun noeud appartenant à ce contexte ne peut appartenir à une partie active d'une occurrence de règle, au cours de l'application du SST. Par conséquent, à la fin de l'application du SST, on les retrouve encore dans la m.a.t. produite.
- A ce moment là, O2 sera remis à la place P qu'aurait occupé normalemant O1, Cela complète l'application de la règle récursive.

Avant de présenter les différents composants d'une grammaire et d'un ST, on définit les notions de <u>passage</u> d'une grammaire et de <u>marques</u> sur les noeuds d'une m.a.t.

Soit G une grammaire composée des règles R1, R2, ..., Rm et soit O une m.a.t. que l'on veut transformer par G. Chacune des m règles définit une conjecture locale sur O que l'on note CLi pour i $\in$ [1,m]. Si aucune des règles n'est applicable, alors toutes les CLi seront égales à $\perp$ et le résultat de l'application de G sur O est O, on dit qu'il y a eu un nombre de <u>passages</u> égal à zéro. Sinon, un sous-ensemble d'occurrences sera choisi parmi tous les CLi, tel qu'il n'y ait pas d'intersections des parties actives (hors du contexte) de chaque occurrence. L'algorithme de choix sera donné dans la section 3.4. Il détermine la <u>conjecture globale</u> de chaque règle, conjecture sans conflits avec les autres. On applique en parallèle les transformations indiquées; avec les appels récursifs éventuels et on retrouve une m.a.t. O' comme résultat. Cet ensemble d'opérations est appelé un <u>passage</u>. L'itération contrôlée, par les options de la grammaire, peut continuer jusqu'à l'arrivée dans un <u>état d'arrêt</u>, avec une m.a.t. O" comme résultat final, après un nombre de passages supérieur ou égal à zéro.

L'algorithme de transformation utilise un système de marquage des noeuds qui suit les options de contrôle. Après chaque passage et sur chaque noeud, les règles sont divisées en deux groupes: les règles candidates et les règles interdites. Celles qui sont interdites ne seront pas utilisées dans le passage suivant, même si elles sont applicables. On ne se sert que des règles candidates à chaque passage. L'application d'une

grammaire s'arrête lorsqu'il n'y a plus de règles candidates sur aucun noeud de la m.a.t. Cette classification et d'autres qui seront données dans le chapitre suivant sont appelées <u>marques</u>.

3.3.3.1 <u>Contrôles</u>: On utilise à peu près les mêmes options de contrôle que dans ROBRA, à savoir:

$$\begin{bmatrix} E \\ B(n) \end{bmatrix} \begin{bmatrix} T \\ C(\{\uparrow\ \downarrow\}) \end{bmatrix} \begin{bmatrix} L \\ P \\ S \\ D \end{bmatrix} \begin{bmatrix} A \\ V(\{\uparrow\ \downarrow\},\{\leftarrow,\rightarrow\}) \end{bmatrix}$$

a) <u>Borné à n passages</u> (B(n)): Pour un entier positif n, si l'application de la grammaire considérée ne s'est pas arrêtée avant l'arrivée au n-ième passage, elle sera arrêtée automatiquement après le n-ième passage, donnant comme résultat la m.a.t. produite à ce moment-là.

<u>Exhaustif</u> (E):Il n'y a pas de limite sur le nombre de passages

En l'absence d'un de ces deux paramètres, on suppose l'option B(1).

b) <u>Coupe haute ou basse</u> (C(↑) ou C(↓)): On demande que la conjecture globale de chaque passage d'une grammaire ne porte pas d'occurrences qui soient descendantes les unes des autres. Pour cela, elles seront choisies le plus haut possible ((↑) niveau minimum) ou le plus bas possible ((↓) niveau maximum)

<u>Total</u> (T): Il n'y a pas de restriction sur les dépendances des occurrences choisies. (Il n'y a pas d'indication de choix (↑ ou ↓) puisqu'on applique le parallélisme).

En l'absence d'un de ces deux contrôles on suppose C(↓)

c) <u>Dessous</u> (D): A chaque passage, une fois qu'une règle R est appliquée sur une m.a.t. contenue dans la m.a.t. objet du passage, le noeud correspondant à la racine de l'image aura pour le passage suivant la règle R et toutes les règles plus prioritaires que R (i.e. inférieures ou à gauche de R dans la relation d'ordre) avec une marque d'interdiction.

<u>Dessus</u> (S): Comme dans le cas précédent, mais au lieu d'interdire les plus prioritaires on interdit les moins prioritaires que R. (Cela ressemble un peu à la fonction / de PROLOG).

<u>Ponctuel</u> (P): Comme dans le cas précédent, mais n'interdit que la règle R.

<u>Libre</u> (L): Pas d'utilisation de marques et pas d'interdiction de règles à la fin d'un passage après leur application.

En l'absence d'une de ces quatre possibilitées on suppose D

d) <u>Convergent haut ou bas et gauche ou droit</u> (V(↑ ou ↓, ← ou →): Avant de commencer chaque nouveau passage et de façon

automatique, si le nombre de noeuds pour lesquels une règle R est candidate, dit le <u>nombre de candidatures</u> de R, n'est pas strictement inférieur au nombre de candidatures de R dans le passage antérieur, alors on réduit le nombre de candidatures de R de manière automatique. C'est-à-dire, R sera interdite sur le noeud le moins prioritaire par rapport aux priorités internes au schéma. Si le schéma porte des points sans priorité (parallélisme permis), on suppose pour ces points les priorités (↓, ↑, <, >) données dans cette option. La candidature de R est donc munie d'un ordre total. Cela est fait pour toutes les règles de façon itérative jusqu'à ce que le nombre de candidatures de chaque règle soit acceptable.

Pour l'application des règles récursives, la condition de convergence est appliquée en plus par rapport au dernier appel récursif de la règle pendant le passage courant mais seulement sur la sous-m.a.t. de récursion. On ne considère pas comme finie l'application d'une règle récursive tant que l'application du sous-système transformationnel correspondant n'est pas terminée.

S'il ne reste plus de règles candidates sur une m.a.t., le passage ou la récursion s'arrête et donne comme résultat la m.a.t. produite au dernier passage ou récursion.

<u>Autonome</u> (A): Pas d'interdiction automatique au début de chaque passage ou récursion.

En l'absence d'une de ces deux possibilités, on suppose V(↓,>).

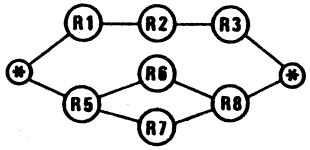
Ce type d'options doit être étudié plus en détail car c'est le moyen le plus direct de contrôler le système. Dans certains cas, on pourrait permettre ce type de décisions au niveau des règles comme dans le groupe (c). Ainsi une règle pourrait interdire l'utilisation d'autres règles et même arriver à manipuler les marques au moment de la transformation. On pourrait imaginer d'autres contrôles, tels que l'application de chaque règle limitée à un maximum ou des cas spéciaux pour les règles parallèles qui échappent aux contrôles D, S ou P. C'est un domaine très intéressant qui reste à decouvrir. On peut cependant penser que l'utilisation n'en serait pas évidente.

Lors de l'application d'une règle récursive appartenant à une grammaire G1 en mode de contrôle m1, et qui fait un appel à une grammaire G2 en mode de contrôle m2, le contrôle réellement utilisé est le minimum entre m1 et m2 (donné par l'ordre de bas en haut de la figure précédente). Ainsi il y aura un <u>mode de contrôle courant</u> qui sera utilisé effectivement. (Cela pourrait faire partie des options de contrôle). On remarque donc qu'une grammaire avec option B ou V s'arrête et produit un résultat après un nombre fini de passages.

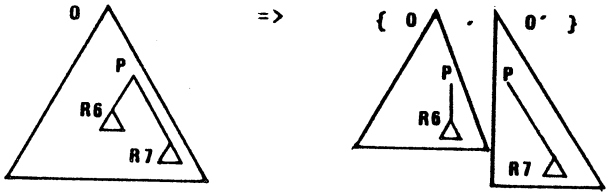
Les options par défaut sont les plus restrictives et le mode

d'application sans restrictions correspond à ETLA. Sans tenir compte des valeurs de n, on retrouve 5! modes d'application qui correspondent aux différents choix possibles. Pourtant, si B(1) est choisi, les options (c) et (d) n'ont pas d'effet sur le résultat.

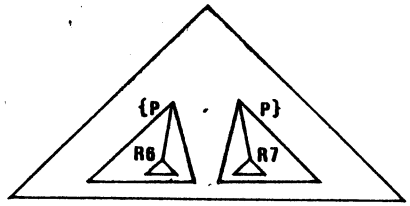
3.3.3.2 Ordre et Priorités: La relation d'ordre partiel affectée aux règles d'une grammaire est donnée par une expression linéaire qui n'utilise que les opérateurs ≤ et ||. Ainsi par exemple, on peut avoir la grammaire G1(ETLA): R1 ≤ R2 ≤ R3 || R5 ≤ (R6 || R7) ≤ R8 c'est-à-dire:



Si dans un passage les règles R1 et R2 ont un conflit à cause d'une intersection non contextuelle, la règle R1, étant plus prioritaire, sera choisie et R2 sera refusée. Par contre, s'il s'agit des règles R6 et R7 qui ont un conflit, on crée une m.a.t. O' équivalente qui remplacera la m.a.t. originale O par un noeud ensemble composé des m.a.t. O et O' tel que sur O on applique R6 et sur O' on applique R7



ou d'une façon factorisée à partir du plus proche ascendant commun p



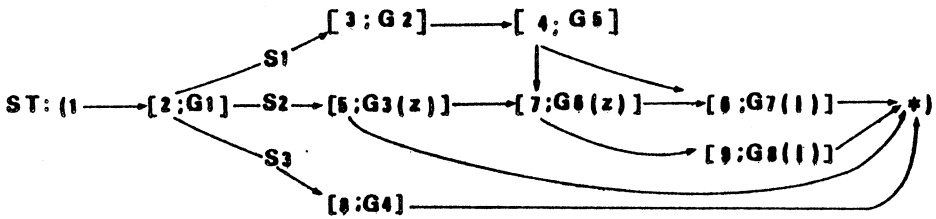
L'ordre total est spécifié lorsqu'il n'y a pas d'opérateur de parallélisme.

3.3.3.3 Réseaux de grammaires: Pour regrouper des transformations successives et particulières, on réalise la composition de grammaires sous la forme d'un graphe orienté et ordonné (éventuellement avec des cycles). Chaque arc définit une liaison entre le noeud nommé à sa gauche et le noeud nommé à sa droite. Le noeud à gauche est le noeud dit de départ et celui à droite est dit le noeud d'arrivée. Les noms des noeuds sont uniques dans un ST. Il existe un noeud distingué (*) dit de sortie, qui n'apparaît jamais à gauche d'un arc du réseau. Tout noeud qui n'apparaît jamais à droite d'un arc est dit d'entrée et par conséquent, il ne porte pas de grammaire. Les autres noeuds sont dit internes et portent une grammaire qui sera donnée dans la partie droite d'un seul des arcs du réseau. Tout autre noeud différent du noeud de sortie qui n'apparaît jamais à gauche d'un arc a de façon implicite un arc qui mène, sans condition de parcours, au noeud de sortie. Un système transformationnel peut ne pas avoir de noeuds d'entrée.

On suppose un ordre total sur l'ensemble des noms des noeuds du

réseau, tel que le noeud * soit toujours le plus grand. Les arcs issus d'un noeud sont ordonnés selon leur ordre d'apparition dans la description du ST. Dans ce sens, on parlera des noeuds ou arcs les plus petits ou les plus grands ou inférieurs ou supérieurs entre eux. On dit qu'un noeud N est <u>accessible</u> à partir d'un noeud M s'il existe un chemin direct ou indirect qui mène de M à N.

<u>Exemple</u>: Voici un système transformationnel ST:(1--->{2;G1}; 2--S1->{3;G2}; 2--S2->{5;G3(Z)}; 2--S3->{8;G4}; 3--->{4;G5}; 4--->{6;G7(I)}; 4--->{7;G6(Z)}; 5--->7; 5--->*; 7--->6; 7--->{9;G8(I)}; 6--->*; 8--->*; 9--->*) ou sous la forme d'un graphe:



On pourrait donner quelques facilitées d'écriture telles que 2--->(S1->{3;G2}; S2->{5;G3(Z)}; S3->{8;G4}), ou (5; 6; 8; 9)->*, où l'on factorise des parties communes aux arcs.

Le noeud 1 est le seul point d'entrée, le ST n'a pas de cycles, S1, S2 et S3 sont des conditions de parcours. Si l'arc 8--->* n'est pas donné, le système l'aurait créé de même que pour 6--->* et 9--->*

¤

L'application d'un ST sur une m.a.t. O est réalisée selon les indications initiales de l'utilisateur qui donne le nom du ST et des noeuds de <u>commencement</u> qui sont un sous-ensemble de noeuds du ST à partir desquels on lance l'exécution. Le système peut à ce moment-là indiquer, après une analyse statique, la liste des noeuds des ST qui sont inaccessibles soit dans le parcours, soit dans les appels récursifs. En l'absence d'indication explicite, on suppose le plus petit noeud du premier ST donné comme noeud de commencement.

Le ST fonctionne, comme dans ROBRA, de manière <u>heuristique</u> en cherchant à partir de chaque noeud de commencement le premier chemin conduisant à * et qui sera traversé par les transformations successives d'une copie de la m.a.t. objet O. A un moment donné, il peut y avoir plusieurs m.a.t. qui traversent les réseaux. Les différentes conditions portées par les arcs et les grammaires contrôlent le flux des m.a.t. au travers du ST. Il y aura une seule copie de la m.a.t. objet par noeud de commencement défini, et le système peut identifier, pendant l'exécution, le noeud de commencement d'où provient chaque m.a.t. qui traverse le réseau, de même que les sous-m.a.t. qui correspondent à des appels récursifs et qui traversent le réseau sans interférer avec les autres traversées.

Un arc qui ne porte pas de condition de parcours peut être

traversé autant de fois que l'on veut par n'importe quelle m.a.t. au cours de la transformation. S'il existe une condition de parcours, alors, si l'option de boucle est donnée, on aura deux possibilités:

- Borné à n passages (B(n)): De façon analogue aux options de contrôle des grammaires, on ne pourra pas traverser cet arc plus de n fois avec une m.a.t. provenant d'un même noeud de commencement. Donc si m est le nombre de noeuds de commencement, le noeud sera traversé au maximum nxm fois au cours d'une application du réseau.
- Exhaustif (E): Il n'y a pas de limite au nombre de passages par l'arc.

En l'absence d'une des deux options, on suppose E.

Si la condition de boucle est vérifiée par une m.a.t. O', alors il faut que l'option de reconnaissance soit testée sur la m.a.t. Cette option a la même syntaxe et sémantique que la partie gauche d'une règle, et sa forme la plus réduite est un schéma avec un seul point. L'arc sera traversé si la reconnaissance est vérifiée et qu'elle n'est pas précédée de la négation (¬) ou si elle n'est pas vérifiée et qu'elle est précédée d'une négation.

Une m.a.t. O", résultat de la transformation d'une m.a.t. O' par la grammaire G affectée à un noeud N, prendra le premier arc issu de N qu'elle peut traverser et sera transformée éventuellement par la grammaire G' affectée au noeud d'arrivée N' de l'arc (N--->N') pris. Sous sa nouvelle forme, elle continuera son parcours.

Si aucun des arcs ne peut être choisi, O" disparaît et on revient en arrière avec O' jusqu'au noeud M où l'on avait choisi l'arc (M--->N), pour essayer de trouver un autre chemin.

Si le retour arrière n'est pas possible, comme dans certains cas qu'on verra plus loin, le ST produit comme résultat O' accompagnée d'un message indiquant le noeud N où l'on était arrivé à une impasse.

L'application d'une grammaire à une m.a.t. est sujette aux conditions données sur le noeud N où elle se trouve, à savoir: conditions de passage, de retour arrière et d'attente

a) Impératif (I): Pour considérer que la grammaire a été appliquée et continuer la traversée avec le résultat, il faut que le nombre de passages de règles soit supérieur à zéro. Autrement, on fait comme si aucun des arcs issus de N ne pouvait être choisi.

   Facultatif (F): Il n'y a pas de restrictions sur le nombre de passages de règles

   En l'absence d'un des deux, on suppose F.

b) Zéro (Z): Il est défendu de faire un retour arrière à partir de ce noeud-ci. En cas d'impasse futur et de retour arrière

jusqu'au noeud actuel N, le résultat sera la m.a.t. Q' avant la transformation avec le message d'avertissement correspondant.

<u>Normal</u> (N): Il n'y a pas de barrière au non-déterminisme de l'heuristique de recherche de chemins sur ce noeud-ci.

En l'absence d'une des deux, on suppose N. Les noeuds d'entrée sont par défaut avec option Z car il est impossible d'aller plus loin.

c) <u>Et</u> (&): Avant d'appliquer la grammaire, on laisse la m.a.t. en <u>attente</u> s'il existe des m.a.t., qui ne sont pas en attente et qui sont en train de traverser des noeuds depuis lesquels N est accessible. Dès que cette condition n'est plus vérifiée, toutes les m.a.t. en attente sur N seront <u>factorisées</u> dans une seule m.a.t. qui pourra continuer sa traversée comme une unité. Pour les options de boucle qu'elle rencontrera, on tiendra compte du maximum de nombres de passages par l'arc de chacune de ses composantes.

<u>Ou</u> (o): Avant d'appliquer la grammaire, on supprime toutes les m.a.t. qui, en attente ou non, sont en train de traverser des noeuds depuis lesquels N est accessible. Pour éviter la destruction mutuelle de deux m.a.t. qui se trouvent dans un cycle et qui arrivent <u>en même temps</u> à deux noeuds avec option o, on donne la priorité suivante:

- la m.a.t. qui se trouve sur le noeud le plus grand est gardée.
- sinon, la m.a.t. qui provient du noeud de commencement le plus petit est gardée.
- sinon, la m.a.t. qui a comme composante une m.a.t. qui provient d'un noeud de commencement plus petit est gardée.

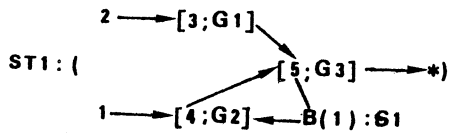
Etant donné qu'il y a une seule m.a.t. par noeud de commencement, il y aura toujours un choix possible et une m.a.t. comme résultat.

<u>En l'absence</u> d'une des deux, il n'y a ni attente ni suppression et pour chaque noeud de commencement on aura un résultat indépendant. Les différents résultats ne seront pas pour autant forcément différents.

<u>Exemple</u>: Pour le ST de l'exemple précédent, soit le parcours 1, 2, 3, 4, 6 et supposons que G7 ne s'applique pas (zéro passages). Comme elle est impérative (I), on revient en arrière et le système entre dans 7. Si G7 et G8 ne s'appliquent pas, on ne peut pas revenir en arrière (à 4, 3, 2 puis essayer G3, etc.), mais on donne comme résultat la m.a.t. transformée par G6.

Si par contre S1 n'est pas vérifiée, on testera S2, d'où l'exécution 2, 5 et la condition Z de G3 interdit de revenir plus tard à 2 et d'essayer G4. Si S1 et S2 ne sont pas vérifiés, on utilisera G4.

Pour illustrer les options d'attente, soit le ST suivant:



où l'arc 5---→4 est plus petit que 5---→*. il y aura deux m.a.t. qui vont essayer de traverser ST1, une à partir de 1 et l'autre à partir de 2. Comme toutes les grammaires sont en option FN et que les conditions de parcours sont vides, à un moment donné, elles vont arriver au noeud 5. Si la reconaissance S1 est vérifiée, la m.a.t. passera à 4 et puis reviendra à 5, mais cette fois-ci, la seule possibilité sera le noeud de sortie, à cause de la condition de boucle B(1), qui empêche le parcours.

On aura alors quatre résultats possibles, qui se résument en (G3G1(O) ou G3G2G3G1(O)) et (G3G2(O) ou G3G2G3G2(O)), selon que S1 est vérifié ou non.

Supposons maintenant que le noeud 5 donne l'option Et (&) à la grammaire G3. Dans ce cas, quelle que soit la m.a.t. qui arrive la première à 5, elle attend l'arrivée de l'autre, et G3 sera appliquée à la factorisation (F) des deux. On aura donc deux résultats possibles qui se résument en G3(F{G1(O), G2(O)}) ou G3G2G3(F{G1(O), G2(O)}).

Finalement, si le noeud 5 donne l'option Ou (o) à la grammaire G3, il y a quatre résultats possibles:

- Si la m.a.t. en provenance de 1 arrive à 5 la première ou si elles arrivent en même temps, on aura G3G2(O) ou G3G2G3G2(O) comme résultat
- Si la m.a.t. en provenance de 2 arrive à 5 la première, on aura G3G1(O) ou G3G2G3G1(O) comme résultat

□

Les options par défaut sont les moins restrictives, et les options B(1) sur chaque arc et IZO sur chaque noeud donnent le maximum de restrictions.

3.4 Application d'une grammaire à une structure

Dans cette section, on montre comment, lors de l'application d'une grammaire à une m.a.t., est définie une conjecture globale pour chaque passage de règles à partir des conjectures locales de chacune d'elles et on donne un algorithme général d'application.

Supposons que l'on se trouve au début du premier passage d'une grammaire G composée des règles R1, R2, ..., Rn. Soient CL1, CL2, ..., CLn les conjectures locales correspondantes. Chaque conjecture locale donne un ordre partiel aux occurrences élémentaires d'une même règle et l'expression d'ordre affectée aux règles de G donne un ordre partiel à l'ensemble des occurrences élémentaires de toutes les règles. Dans ce sens, on pourra savoir si une occurrence Oi est inférieure, supérieure

ou égale en priorité, à une autre occurrence Oj indépendamment des règles en question.

Les occurrences élémentaires donnent une liaison entre les points du schéma d'une règle et les noeuds de la m.a.t. Chaque règle donne une classification des points de son schéma en deux groupes: points actifs et points passifs. Deux occurrences qui affectent à deux points actifs un même noeud de la m.a.t. sont dites en <u>conflit hors du contexte</u>, et il est interdit de les appliquer en même temps. C'est l'ordre des occurrences qui décide laquelle des deux appliquer: l'occurrence la plus petite est prioritaire. Si l'ordre n'est pas défini entre ces deux occurrences en conflit, elles seront appliquées en <u>parallèle</u> sur deux copies de la m.a.t. Par conséquent, un noeud de la m.a.t. ne peut pas être en même temps l'occurrence de deux points actifs différents, mais un noeud peut être l'occurrence d'un point actif et de plusieurs points passifs, ou bien seulement de points passifs. Dans ces cas, on dit que les conflits sont <u>contextuels</u> et permis.

Une fois qu'une occurrence élémentaire est choisie, toutes les occurrences <u>moins prioritaires</u> qui ont un conflit hors du contexte avec elle sont <u>éliminées</u> du passage. L'ensemble des occurrences non éliminées (car incomparables), mais en conflit non contextuel avec une des occurences déjà choisies sera utilisé sur une autre copie de la m.a.t. Sur la copie en question, avec les nouvelles conjectures (ensemble réduit), on recommence, et ceci jusqu'à l'épuisement des conjecturres.

Donnons quelques notations:

Oij est la j-ième occurrence élémentaire de la règle i (dans la conjecture CLi)
CGik est le sous-ensemble des occurrences élémentaires de la conjecture CLi choisi pour la k-ième copie de la m.a.t. objet du passage
CG est la conjecture globale composée des CGik
Conflit(X,Y) est le prédicat ∃ i, i' ∈ {1,n} : X ∈ CLi & Y ∈ CLi' & X et Y ont un conflit hors du contexte
mj est le nombre d'éléments dans la conjecture CLi au départ.
M vaut m1xm2x...xmn. C'est le nombre maximum de copies de la m.a.t., dans le cas où toutes les occurrences élémentaires ont des conflits hors du contexte et où il n'y a pas de priorités.
<u>pp</u> est l'ordre partiel (lu, plus prioritaire que) défini dans la section 3.3.2.4 pour les occurrences élémentaires
"avant" est l'ordre total sur les occurrences définit comme suit:
O' "avant" O" si et seulement si (O' <u>pp</u> O" ou
(O' et O" ont la même priorité &
(nombre de points de O' < nombre de points de O" ou
nombre de noeuds liés à O' < nombre de noeuds liés à O" ou
(nombre de noeuds liés à O' > nombre de noeuds liés à O" &
(Il existe un noeud x lié à O' et non lié à O" tel que: x est plus petit (dans l'ordre induit par un parcours préordre) que tout noeud y lié à O" et non lié à O")))))
Ppt(Cli', O') est le prédicat pour la plus petite occurrence élémentaire d'une conjecture locale par rapport à l'ordre "avant", c'est-à-dire, O' ∈ Cli' & (∀ O" ∈ Cli' : O' ≠ O" => O' "avant" O")
Choix(Cli) est la fonction de choix d'une occurrence élémentaire pour une conjecture globale
O' si X={O' ∈ Cli' | ∀ i" ∈ {1, n} ∀ O" ∈ CGi"k: (O' ≠ O" => ¬Conflit(O', O"))} & Ppt(X, O')
⊥ sinon

Voici l'algorithme de construction d'une conjecture globale:

```
(initialisation de CG)
 pour k:=1 jusqu'à M faire
    pour i:=1 jusqu'à n faire
       CGik:=φ
    finpour
 finpour
 k:=1
 tantque ∃ i ∈ [1,n]: CLi ≠ ⊥ faire
    (Il existe des occurrences élémentaires à utiliser)
    itérer
       i' := 0 ; O' := ⊥
       tantque i' < n & O' = ⊥ faire
          i' := i' + 1
          O' := Choix(Cli')
       fintantque
    arrêt: (O' = ⊥)
       (On peut choisir O' pour l'ajouter à CGi'k)
       CGi'k:=CGi'k ∪ {O'}
       (On élimine les occurrences qui ne peuvent plus être
            choisies)
       pour l:=1 jusqu'à n faire
          T:={O" ∈ CLl | O' pp O" & Conflit(O',O")}
          CLl:=CLl - T
       finpour
    finitérer
    k:=k+1
 fintantque
 k:=k-1
```

La valeur finale de k donne le nombre de copies nécessaires pour réaliser le maximum d'applications. S'il existe un ordre total des occurrences élémentaires, il y aura une seule m.a.t. et k sera égale à 1 à la sortie.

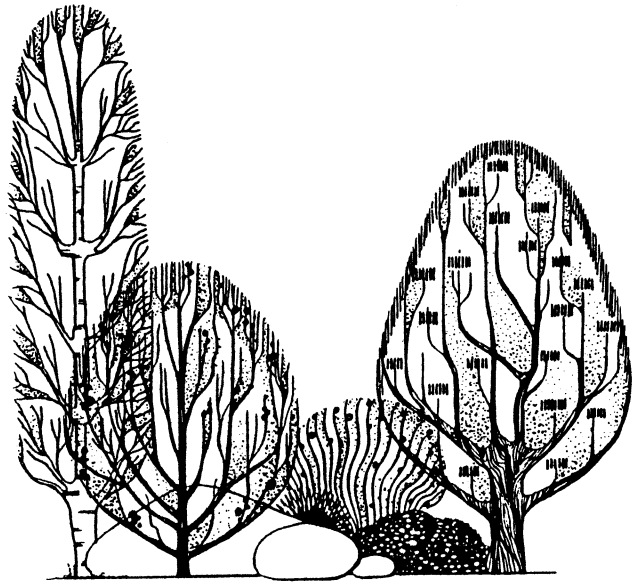
L'algorithme général d'application d'une grammaire est le suivant:

1. lecture de la m.a.t. objet et factorisation, de façon à simplifier sa structure
2. initialisation des marques. Au départ, toutes les règles sont candidates sur tous les noeuds
3. npassages:=0 (compteur de passages)
4. itérer
   5. détermination des conjectures locales de toutes les règles
   6. détermination de la conjecture globale
   7. application du contrôle de dépendances, c'est-à-dire s'il y a une option de coupe (C), on ne garde que les occurrences de niveau maximum ou minimum indépendantes les unes des autres
   8. si la conjecture globale n'est pas vide alors
      9. npassages:=npassages+1
      10. on produit l'éclatement nécessaire pour la production de copies si c'est le cas. Cet éclatement peut être partiel si les conflits sont locaux
      11. transformation en parallèle et attente de la finalisation des appels récursifs, s'il y en a
      12. actualisation des marques
      13. factorisation du résultat de sorte que les ensembles de marques de deux noeuds qui doivent être remplacés par un seul noeud soient unis dans un seul ensemble de marques
   14. finsi
15. arrêt: (npassages > borne de contrôle ou fin de règles candidates)
   16. application du contrôle d'interdiction automatique si l'option de convergence est donnée
17. finitérer
18. production du résultat

Dans le chapitre suivant, on donnera un algorithme qui exploite mieux les possibilités du parallélisme.

Chapitre IV

Utilisation du Parallélisme dans une Réalisation

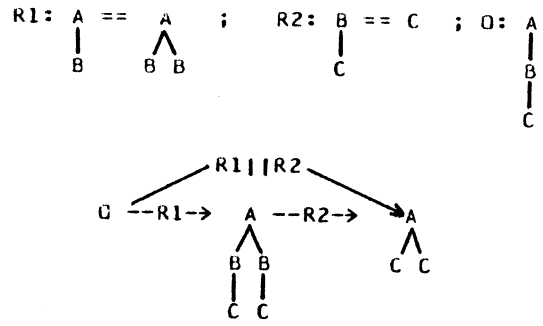


4.1 Introduction

Dans ce domaine, on peut appliquer le parallélisme de différentes façons et à des niveaux divers. Il y a le cas très simple de la division d'un texte à traduire en un certain nombre de sous-textes ou paragraphes, qui peuvent être traduits indépendamment les uns des autres. Il y a aussi le cas de RUBRA, qui réalise des transformations d'arborescences en parallèle. On pourrait ajouter encore d'autres exemples, comme la simultanéité possible des différentes phases de traduction, comme l'analyse morphologique en parallèle avec l'analyse structurale, ou la génération morphologique en parallèle avec la génération syntaxique. Ceci n'a jamais été réalisée dans un système de traduction automatisée par ordinateur, mais seulement dans certains compilateurs munis de coroutines. Il existe certainement des problèmes de temps de réponse de chacune de ces étapes, qui empêchent une réduction pratique des temps de traduction. D'un côté, la consultation des dictionnaires pourrait être améliorée par une recherche simultanée de plusieurs chaînes, qui ne sont pas des préfixes ni des suffixes les unes des autres, car une organisation particulière des dictionnaires pourrait donner directement tous les préfixes d'une chaîne sans recourir au parallélisme, et il peut y avoir encore d'autres opérations susceptibles d'être exécutées simultanément.

Le type de parallélisme qui nous intéresse en particulier dans

ce chapitre est l'application simultanée de règles de réécriture. Par exemple, les règles R1 et R2 appliquées à la m.a.t. O suivante:



On pourrait dire qu'il suffit de définir une seule règle à partir des deux qui veulent s'appliquer en parallèle, par exemple R3:A(B(C))==A(C,C), et continuer à travailler de manière séquentielle, mais l'ensemble de règles serait trop grand pour considérer toutes les possibilités, notamment lorsqu'on utilise des expressions d'ordre complexes, des schémas verticaux ou horizontaux. D'autre part, il y a des cas où l'application parallèle n'a pas le même effet que l'application séquentielle, par exemple si R2 s'applique avant R1.

On propose dans ce chapitre une méthode d'implémentation pour STAR-PALE, mais applicable d'une façon plus générale à d'autres systèmes comme les systèmes-Q, [Co 70], et à des systèmes généraux de transformations sur des structures telles que les E-graphes par exemple, [GE 80]. L'idée de base est de ne regarder ni les règles ni les arborescences comme des éléments

passifs qu'on manipule, mais comme des processus qui coopèrent à la réalisation d'une tâche commune. Ils sont donc en communication entre eux et développent un travail simultané. On pourrait contrôler leur exécution en restreignant le parallélisme de certaines règles, ou en évitant la manipulation simultanée de certaines zones de la structure à transformer. Dans notre cas, il s'agit plutôt de ne pas interrompre un processus qui se trouve en train de modifier la structure d'appel de processus ou l'existence d'autres processus.

Cette idée peut être généralisée au niveau des grammaires dans un système transformationnel, en ajoutant une communication entre les processus qui représentent les grammaires.

Dans une première partie, nous donnons un algorithme simplifié pour la réalisation du parallélisme. Il est présenté d'abord d'une manière générale, puis plus en détail. La deuxième partie contient un exemple de transformation avec l'algorithme précédent. En conclusion, nous apportons des précisions sur les problèmes rencontrés, et sur les modifications et les possibilités envisagées à partir de cette première approche.

4.2 STAR-PALE: Un système parallèle

Le système consiste en un ensemble de règles muni d'une relation d'ordre partiel, qui s'applique sur une structure à transformer. Dans notre cas, il s'agit d'une multi-arborescence à tranches (m.a.t.). Une règle comporte une partie gauche, dite "de reconnaissance", et une partie droite, dite "de transformation". Dans la partie gauche, on trouve un schéma et un prédicat sur les valeurs des décorations associées aux noeuds de la m.a.t. La partie droite est constituée d'une m.a.t. image et des modifications des décorations portées par les noeuds de la m.a.t image. Pour tout noeud de la m.a.t. objet, chaque règle définit une énumération canonique des occurrences possibles de cette règle, enracinées sur ce noeud, ce qui donne une priorité de ces occurrences entre elles: soit priorité vers la gauche ou vers la droite, ou vers le haut ou vers le bas si elles partent du même niveau ou pas.

Un schéma est divisé en deux parties: une passive et une autre active. Un point associé à un chemin par un schéma vertical, un point familial ou un point de repère est actif si l'une au moins des conditions suivantes est réalisée:

- il n'apparaît pas dans l'image
- il change de père ou de liste
- sa décoration est modifiée
- l'ordre de ses fils ou éléments est modifié
- il porte un nouveau fils ou élément dans l'image
- il perd un de ses fils ou éléments

Les points correspondant à des schémas verticaux ou horizontaux sont tous actifs si leurs noms n'apparaissent pas dans l'image. Les points correspondant à des listes implicites du schéma qui

n'apparaissent pas explicitement dans l'image et qui ne peuvent pas être gardées par le point auquel elles sont associées dans le schéma (à cause de l'expression d'ordre affectée à celui-ci dans l'image), ou bien parce qu'il disparaît, sont tous actifs.

Tous les points correspondant à l'étage d'un schéma horizontal ainsi que tous les points appartenant au chemin strict d'un schéma vertical sont actifs si le nom du schéma est associé à un opérateur Etage, Sous-Etages ou chemin dans l'image. Les points qui disparaissent à cause de ces opérateurs sont actifs. Voir exemples de la section 3.3.2.5.

Un point qui n'est pas actif est passif. Par extension, on dit qu'une m.a.t. est active si elle porte au moins un point actif, sinon elle est passive. L'ensemble des points passifs d'un schéma est appelé contexte. On voit donc que, le contexte peut être non connexe (au contraire de ROBRA) et qu'il peut être déterminé statiquement.

L'utilisation des point actifs et passifs sert à préserver la structure d'arborescence lors d'une transformation, en permettant en même temps un maximum de parallélisme. C'est pourquoi on fait une extension de la définition à certains noeuds non explicites dans le schéma, et qu'un point qui n'a qu'une seule modification correspondant à un changement de père est défini comme actif même si lui-même il n'est pas modifié.

4.2.1 Structures de données internes

Le comportement de base du système ainsi que les structures de données propres à l'algorithme seront introduites ici.

Pour chacune des règles, on active un processus qui sera chargé de questionner les noeuds de la m.a.t. pour savoir si la règle est applicable à cet endroit, c'est-à-dire s'il existe une occurrence de la règle enracinée sur ce point. Il y a deux étapes dans chaque processus: il faut reconnaître un schéma dans la structure, puis transformer une partie de celle-ci. Chacune de ces étapes est faite en parallèle, mais on ne commence la transformation que lorsque la reconnaissance globale a été réalisée pour tous les noeuds et toutes les règles, en tenant compte de toutes les restrictions d'application entre les règles, c'est-à-dire lorsqu'une conjecture globale est trouvée. Par contre, la reconnaissance peut démarrer pour le passage suivant des règles s'il y en a, en parallèle avec la transformation précédente.

L'algorithme parallèle proposé ici utilise des structures de données qui se trouvent principalement au niveau de chaque noeud. A chaque noued Ni de la m.a.t. sont attachés

- une file des messages reçus et en attente
- une liste de renseignements concernant les messages et réponses qu'il a envoyés au cours de chaque passage.
- une pile d'états dont le haut correspond à l'état courant.

Au cours du traitement, chaque noeud est dans un état qi appartenant à l'ensemble (libre, test, attente, prêt, révision, métamorphose, initialisation, suppression).

- un compteur de passages de règles qui permet de définir un niveau de communication entre processus. Une condition nécessaire à la communication entre deux noeuds est qu'ils aient la même valeur dans leurs compteurs de passage.
- une distance de modification qui donne la différence de niveau entre le noeud Ni et le noeud descendant le plus proche (vers le bas) qui porte une liste d'applicabilité modifiée après son utilisation par Ni et où son contenu actuel, inconnu par Ni et ses ascendants, risque d'affecter leur listes d'applicabilité. Elle sera utile dans la phase de définition de la conjecture globale du passage.
  La lecture de la distance de modification d'un noeud par ses ascendants n'est possible que si le noeud ne se trouve pas dans les états d'initialisation, d'attente ou de révision, afin d'assurer une lecture correcte de la valeur.
- un témoin de nouveau noeud qui peut avoir deux états: éteint ou allumé. Il sera allumé lorsque le noeud qui le porte sera créé par un autre noeud au cours d'une transformation, et il sera éteint par son père quand celui-ci aura reconnu ce nouveau fils. Autrement, il est éteint.
- un indicateur du nom du noeud qui le remplace qui sera mis dans la phase de transformation et servira à répondre aux questions posées à ce sujet par ses ascendants.
- une liste d'intergiction qui porte les noms des règles qui ne doivent plus s'appliquer si elles sont enracinées sur ce noeud.
- une liste de candidature contenant le nom des règles qui ont le droit d'être testées sur le noeud.
- deux listes d'applicabilité qui contiennent les occurrences de règles applicables à partir du noeud, ou qui le contiennent. L'une d'elles est la liste courante, que l'on appellera liste d'applicabilité, et l'autre est une copie de la première. Elle n'est utilisée que dans l'état de révision, et sert à mémoriser temporairement la première.
- une liste de non-applicabilité qui contient les occurrences de règles non applicables à partir de Ni, soit parce que la reconnaissance a échoué, soit parce qu'il y a eu un conflit hors du contexte avec une règle ou occurrence plus prioritaire.
  - Les occurrences portent, en plus du nom de la règle et des noms des noeuds liés à chaque point du schéma correspondant,
    - un témoin d'application partielle qui sera allumé si l'occurrence ne lie pas le noeud Ni à la racine du schéma, et donc si elle est enracinée au niveau de ses ascendants. Sinon, il sera éteint.

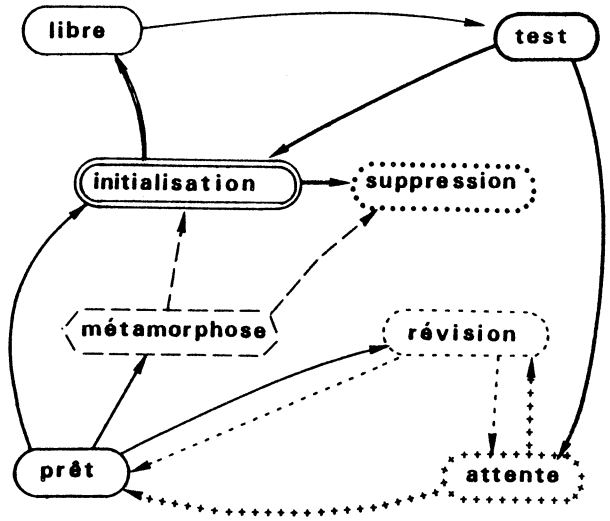
- un indicateur d'occurrences responsables du refus qui indique, pour les occurrences dans la liste de non applicabilité, les noms des occurrences de la liste d'applicabilité courante qui sont en conflit hors du contexte avec elle. Il est vide pour une occurrence qui n'a pas de tel conflit.

Au cours du traitement et dans chaque noeud, on trouve chaque règle de la grammaire qu'on applique dans au moins une des listes de margues du noeud (c'est-à-dire d'interdiction, de candidature, d'applicabilité ou de non applicabilité).

Au départ, chaque noeud porte:
la file des messages reçus et la liste de renseignements concernant les messages envoyés vides,
la pile avec le seul état libre,
le compteur de passages et la distance de modification égaux à zéro,
le témoin de nouveau noeud éteint,
l'indicateur du nom du noeud, les listes d'interdiction, d'applicabilité et de non-applicabilité vides
et la liste de candidature avec le nom de toutes les règles.

4.2.2 Description générale

Pour réaliser une transformation, le système utilise deux principes qui seront décrits d'une manière générale dans cette section. Chaque processus représentant un noeud de la m.a.t. se trouve dans un état particulier, depuis lequel il est capable d'envoyer et de recevoir des messages. Ces messages contrôlent la synchronisation et le changement d'état de chaque processus.



Le graphe précédent ilustre les possibilités de transitions d'état d'un noeud en STAR-PALE.

L'état libre est affecté aux noeuds de la m.a.t. qui sont "inactifs", soit parce qu'ils viennent d'être crées et n'ont pas encore commencé l'étape de reconnaissance ou de transformation, soit parce qu'ils viennent de finir une reconnaissance ou une transformation et qu'ils sont prêts à

recevoir des messages. Ils ont aussi le rôle de "retransmettre" des messages qui vont en direction de leurs descendants.

L'état test est le succeseur de l'état libre et indique que le nceud est en train de déterminer la conjecture locale des règles. Le résultat des opérations réalisées dans cet état détermine si le noeud se trouve dans la zone d'action d'au moins une règle applicable. Si ce n'est pas le cas, il peut se préparer pour le prochain passage de règles et pour cela il passe à l'état initialisation. Autrement, il doit attendre la définition de la conjecture globale du passage pour savoir le sort qui lui est réservé.

L'état attente sert à déterminer la conjecture globale partielle par une recherche de conflits de bas en haut de la m.a.t. Une fois finie cette opération pour le noeud, il passe à l'état prêt de façon directe ou bien au travers de l'état révision. Il s'agit d'une conjecture partielle, parce qu'on ne tient compte que des conflits depuis le niveau du noeud vers le bas.

L'état prêt est un état intermédiaire où le noeud a une conjecture globale partielle (vers le bas) et attend soit un message qui lui indique qu'elle est bien définitive, soit des messages indiquant des modifications à lui faire avant de passer à la transformation. S'il y a des changements, il revient à l'état attente en passant par l'état révision. Sinon, une fois connue la conjecture globale définitive, il change d'état pour la phase de transformation. Il passe dans l'état métamorphose s'il participe d'une façon active aux changements, et sinon directement dans l'état initialisation.

L'état révision sert à constater des modifications de la conjecture globale partielle du noeud à un moment donné. Il s'agit d'un état intermédiaire et obligatoire dans les deux sens entre les états prêt et attente, en cas de "retour arrière" jusqu'à ce dernier.

L'état métamorphose est réservé aux noeuds racines des règles applicables. Ces noeuds seront chargés d'effectuer la transformation. Un noeud dans cet état a le droit de créer ou de supprimer d'autres noeuds et de modifier leurs décorations. Il peut s' "autosupprimer" à la fin de la transformation si c'est nécessaire, sinon il passe à l'état initialisation

L'état initialisation est un état d'attente d'autorisation d'aller au passage suivant de règles. En effet, un noeud dans cet état peut encore être supprimé. Dans ce cas, il passe à l'état suppression, sinon, il arrive à l'état libre pour un nouveau passage de règles. Le cycle est ainsi fermé.

L'état suppression est un état final où un noeud est maintenu pour indiquer le nom du noeud qui le remplace, ou simplement pour noter qu'il ne doit plus être pris en compte. Une fois

qu'il ne sera plus nécessaire, il pourra être éliminé réellement, en libérant la place occupée.

Le tableau suivant montre, pour chaque message du système, les états possibles de l'expéditeur, les états dans lesquels le destinataire peut traiter le message et les réponses possibles, s'il y en a.

MESSAGE	ETAT DE L'EXPEDITEUR	ETAT DU DESTINATAIRE	REPONSES
reconnaissance d'une règle ou partie	règle* test	libre test	"oui" "non"
départ	attente	test	aucune
confirmation de partie de règle	attente prêt	prêt	"refus" "confirmation"
prévention d'utilisation d'une règle	prêt	prêt	aucune
reconfiguration	prêt révision	prêt	"modification" "pas de modif"
ordre de transformation	libre test attente prêt métamorphose	libre prêt	aucune
suppression ponctuelle ou totale	métamorphose initialisation	initialisation	aucune
nouveau nom	métamorphose	initialisation	aucune
identification	initialisation	initialisation suppression	nouveau nom du destinataire "supprimé"
fin de transformation	initialisation libre métamorphose	initialisation	aucune

* règle n'est pas un état, mais l'indication que l'expéditeur du message est une règle

Les messages représentent des signaux ou des questions et peuvent produire des changements d'état des destinataires.

Le message reconnaissance est une demande au destinataire pour savoir si la règle est applicable à partir de ce noeud. Le résultat sera oui ou non selon le cas. Cette applicabilité correspond à la conjecture locale.

Le message départ est un signal indiquant au destinataire que l'expéditeur a fini sa conjecture locale et attend que ses descendants trouvent leurs conjectures globales partielles pour qu'il puisse le faire à son tour.

Le message confirmation est une demande au destinataire pour savoir si la règle ou partie de la règle a des conflits avec les conjectures globales de ses descendants. Le résultat sera refus s'il existe déjà une occurrence plus prioritaire choisie par un de ses descendants et en conflit hors du contexte avec celle-ci. Sinon la réponse sera confirmation. De toutes façons, les descendants ne modifient pas leurs conjectures à cause de ces messages.

Le message prévention est un signal qui indique aux destinataires que la règle en question a été choisie de manière définitive par un ascendant. Cela permet de mémoriser le fait pour compléter la conjecture globale des noeuds.

Le message reconfiguration est un signal et une demande en même temps. C'est un signal indiquant que toutes les règles qui pourraient affecter la conjecture globale du destinataire ont été déjà choisies et indiquées à l'aide des messages de prevention précédents. C'est une demande pour savoir s'il y a eu des modifications sur la conjecture globale d'au moins un de ses descendants. La réponse sera modification ou pas de modification.

Le message ordre de transformation est un signal indiquant aux destinataires que les informations sur leurs conjectures globales ne sont plus nécessaires et qu'ils peuvent commencer la phase de transformation.

Le message suppression ponctuelle ou totale est un signal qui sert à indiquer au destinataire qu'il a été supprimé individuellement, ou bien avec tous ses descendants.

Le message nouveau nom est un signal qui donne au destinataire le nom du noeud qui va le remplacer et qu'il transmettra ultérieurement si nécessaire.

Le message identification est une demande adressée au destinataire afin de connaitre son nom. La réponse peut être supprimé.

Le message fin de transformation est un signal qui donne

l'autorisation au destinataire de participer au passage suivant de règles.

On suppose que le système de communication des messages se comporte de sorte que, lorsqu'un noeud envoie plusieurs messages à un même noeud, ils seront reçus par le destinataire dans l'ordre de leur expédition.

Dans l'étape de reconnaissance, les communications entre les noeuds se font au travers des descendants, mais, dans la partie transformation, elles peuvent se réaliser de façon directe, notamment dans les messages de nouveau nom qui doivent arriver le plus vite possible. Cela implique la connaissance par les noeuds racine de transformations, des noms de tous les points actifs et passifs correspondant à la règle dans la m.a.t.

4.2.3 Description détaillée

Dans la suite, on présente un premier algorithme parallèle comme base de traitement des applications des grammaires à des m.a.t. La section 4.3 donne un exemple complet de son utilisation avec une trace. La présentation est commentée pour chaque état des noeuds.

4.2.3.1 Etat Libre: Dans l'étape de reconnaissance, les noeuds répondent aux questions adressées par les règles ou par d'autres noeuds, en général des ascendants. Un noeud qui rentre dans un état libre (s'il est la racine de la structure à transformer, il s'envoie à lui même un message de fin de transformation), accepte des requêtes de reconnaissance de n'importe quel expéditeur, mais, lorsqu'il y a plusieurs demandes en attente, c'est la priorité affectée aux règles qui résout le conflit. Le noeud met l'état libre dans sa pile d'états et passe dans l'état test. Pendant qu'il se trouve dans l'état libre, il envoie des ordres de transformation à tous ses fils chaque fois qu'il reçoit un ordre de transformation.

4.2.3.2 Etat Test: Tant qu'il reste dans l'état test, il vérifie les conditions posées sur lui-même, sauf si la règle demandée se trouve dans sa liste d'interdiction, car il répond alors non tout de suite. Si ces conditions ne sont pas vérifiées, il répond non à son questionneur. S'il vérifie les conditions, alors, s'il y a des conditions sur les fils, il demande à chacun d'eux de les tester, sinon il répond oui à son demandeur. A ce moment-là, il peut demander plusieurs combinaisons en même temps ou dans un ordre donné. Il attend donc ces réponses et, lorsqu'il ne reçoit que des combinaisons de réponses non-acceptables, il répond non a son demandeur. Finalement, s'il trouve une configuration vérifiant les conditions de l'appel, il répond oui à son demandeur.

Les prédicats qui mettent en rapport des décorations de noeuds différents, dites conditions intersommets, seront testés par le

noeud racine du schéma.

Si l'appel était envoyé directement par une règle, elle sera retirée de la liste de candidature et ses occurrences seront ajoutées à la liste d'applicabilité ou de non applicabilité selon que la réponse sera oui ou non (respectivement).

S'il n'y à plus de règles dans la liste de candidature, on connaît la situation globale d'applicabilité sans restrictions du noeud: celui-ci continue à traiter tous les messages de reconnaissance qui lui sont envoyés, et, dès qu'il aura reçu un message de fin de transformation, de départ ou un ordre de transformation, il prendra la décision suivante avant de répondre à ce message; si le noeud a donné au moins une réponse oui, il met un état prêt dans la pile et passe à un état attente, sinon il passe à l'état initialisation en donnant des ordres de transformation à ses fils. La pile sert principalement à faire le lien entre les états prêt, révision et attente.

4.2.3.3 Etat Attente: Un noeud en attente envoie un message de départ à chacun de ses fils (à condition que sa pile ait un état de test en haut de pile), puis il vérifie pour chacune des règles applicables et dans l'ordre de priorité associé, si elles sont encore applicables par rapport aux restrictions imposées par l'applicabilité des règles au niveau de ses descendants. Les règles applicables qui ont leur témoin d'application partielle allumé n'ont pas besoin de confirmation et ont par l'effet de cet état du témoin une priorité supérieure à toutes celles qui ne l'ont pas.

Pour cela, le noeud demande pour chacune des occurrences acceptables et dans un ordre compatible avec les priorités, une réponse de confirmation de chacun des descendants actifs qui composent l'occurrence (où un descendant est actif par rapport à une règle s'il correspond à un point actif du schéma) et qui ont une distance de modification égale à zéro. Chaque fois qu'une occurrence est refusée, il ôte celle-ci de la liste d'applicabilité et la met dans sa liste de non applicabilité. Chaque fois qu'une occurrence est confirmée, il faut s'assurer qu'elle n'a de conflit avec aucune des autres occurrences déjà confirmées dans la liste d'applicabilité. Il y a donc une deuxième phase de confirmation, dans laquelle on vérifie chaque couple d'occurrences de règle formé par une occurrence déjà confirmée et l'occurrence testée.

Pour cela, il faut que le noeud appartienne au contexte d'une des deux occurrences ou des deux, et, si c'est le cas, il faut vérifier que l'intersection des ensembles de descendants actifs par rapport à chacune d'elles est vide. Si le couple est refusé, c'est à dire si au moins un des descendants est actif pour les deux, l'occurrence testée est refusée, sinon elle est confirmée pour le couple. Si tous les couples sont vérifiés sans conflit, l'occurrence testée est confirmée applicable et

on continue le test pour l'occurrence suivante. Lorsqu'une occurrence de règle Oi ne peut pas être confirmée à cause d'un conflit, elle aura dans son indicateur un nouvel élément: le nom de l'occurrence responsable du refus, et les occurrences de règles non applicables qui étaient en conflit avec l'occurrence de règle Oi seront passées à la liste d'applicabilité afin de les essayer à nouveau, à condition que leur indicateur d'occurrences n'ait pas des noms qui correspondent à des occurrences confirmées dans la liste d'applicabilité.

Lorsqu'on aura fini la vérification pour tous les éléments de la liste d'applicabilité, le noeud connaît donc sa situation partielle d'applicabilité vers le bas, il s'envoie à lui même un ordre de transformation (à condition qu'il n'ait pas envoyé de réponses oui à des messages de ses ascendants lors de son dernier passage par l'état de test et que sa pile ait un état de test en haut de pile), et passe à l'état contenu au sommet de la pile, en la dépilant.

4.2.3.4 Etats Prêt et Révision: Dans l'état prêt un noeud répond aux demandes de confirmation. Si dans la liste d'applicabilité il y a une occurrence O d'une règle où le noeud n'est pas dans le contexte, alors, si elle est plus prioritaire que la règle demandée, il envoie à son demandeur un refus de confirmation. Autrement, c'est-à-dire si le noeud est dans le contexte de toutes les règles de la liste, ou si la règle demandée est plus prioritaire que l'occurrence O de la liste, il vérifie qu'il n'y a pas de conflit avec les règles de la liste d'applicabilité qui sont plus prioritaires que celle-ci et c'est le résultat de ce test qui décide de la réponse qu'il envoie à son expéditeur.

Si, pendant qu'il est dans l'état prêt, le noeud reçoit un ordre de transformation et si sa liste d'applicabilité contient au moins une règle avec le témoin d'application partielle éteint, il passe à l'état de métamorphose après avoir envoyé aux descendants actifs par rapport à chacune des règles, et dont le témoin d'application partielle est allumé, un message de prévention d'utilisation de la règle suivi d'un dernier message de reconfiguration par l'intermédiaire de ses fils racines des sous-m.a.t. actives (le message de reconfiguration ne sera envoyé que s'il y a au moins un message de prévention), sinon il envoie des ordres de transformation à tous ses fils et passe à l'état initialisation.

S'il est dans un état prêt et reçoit un message prévention, il met la règle de l'appel dans la liste d'applicabilité avec le témoin d' application partielle allumé (s'il est actif par rapport à cette règle) et passe des messages de prévention aux descendants actifs par rapport à la règle par l'intermédiaire de ses fils.

S'il est dans un état prêt et reçoit un message de reconfiguration, il empile l'état et passe à un état de

révision ou il envoie des messages de reconfiguration à tous ses fils auxquels il a envoyé au moins un message de prévention depuis le dernier message de reconfiguration reçu ou depuis le début du traitement s'il n'a pas encore reçu ce type de message. Puis, il attend une réponse modification ou pas de modification de chacun d'eux. Dès qu'il aura reçu toutes les réponses, c'est-à-dire dès que chacun de ses descendants aura actualisé une situation partielle d'applicabilité vers le bas, il garde une copie de sa liste d'applicabilité, met l'état révision dans la pile et passe à l'état attente. A son retour à l'état révision il compare son ancienne liste d'applicabilité (copie) avec la nouvelle (courante):

- Si elles sont différentes ou s'il avait ajouté au moins une règle avec témoin d'application partielle allumé à la liste d'applicabilité pendant son dernier passage par l'état prêt, il met sa distance de modification à zéro;
  sinon il la met au minimum plus un des distances de ses fils, s'il en a, ou à infini s'il est une feuille.
- Si elles sont différentes ou s'il a reçu une réponse de modification de ses fils, il répond modification
  sinon il répond pas de modification, puis il passe à l'état de la pile en la dépilant.

4.2.3.5 Etat Métamorphose: Un noeud qui passe dans l'état métamorphose envoie des ordres de transformation à chacun de ses fils et commence l'application des opérations indiquées par les règles de sa liste d'applicabilité qui n'ont pas allumé leur témoin d'application partielle. Il effectue donc les opérations suivantes: Pour chacune de ces règles, il envoie des messages suppression totale aux noeuds racine des sous-m.a.t. qui seront supprimées et des messages suppression ponctuelle aux noeuds qui seront supprimés indépendamment de ses fils.

Pour chacun des autres points actifs de ces règles qui subiront au moins une modification différente d'un changement de père, et pour les nouveaux noeuds des parties droites des règles, il crée un noeud avec les noms des noeuds qui seront ses fils et dans l'ordre indiqué par la règle (avec son témoin de nouveau noeud allumé, ce qui indique que ce nom correspond à un noeud créé par le noeud qui réalise la transformation), il initialise sa décoration, rend son compteur de passages égal à celui du noeud qui opère la transformation et met sa distance de modification à zéro. Ses listes d'applicabilité, non applicabilité, candidature et interdiction et sa pile d'états seront copiées du noeud actif qu'il va remplacer ou seront initialisées s'il s'agit d'un nouveau noeud créé par la règle.

Au fur et à mesure que l'on aura fini ces opérations pour un noeud, ce noeud sera mis dans un état initialisation et activé tout de suite. Chacun des noeuds actifs qui sera remplacé par un des noeuds que l'on vient de créer est informé du nom du noeud qui le remplace, dès que ce nom est connu. On envoie des messages de fin de transformation aux noeuds explicites de la m.a.t. image qui correspondent à des noeuds actifs du schéma ou

à de nouveaux noeuds.

Une fois finie cette suite d'opérations pour toutes les règles, si le noeud lui-même est un noeud actif pour une des règles qu'il a utilisée, il mémorise le nom du noeud racine de la m.a.t. image de cette règle puis, s'il est supprimé ou remplacé par un autre noeud d'après la règle, ou s'il a reçu un message de nouveau nom, il passe à l'état suppression. Autrement il passe à l'état initialisation.

4.2.3.6 Etats Initialisation et Suppression: Dans l'état initialisation, un noeud passe dans l'état suppression dès qu'il a reçu un message de nouveau nom. Tant que cela n'arrive pas, il demande à ses fils qui n'ont pas allumé leur témoin de nouveau noeud de s'identifier, et il éteint les témoins de nouveaux noeuds des autres fils. Au fur et à mesure qu'ils répondent, il actualise les noms de ses fils et leur ordre par la voie des messages d'identification (peut-être y a-t-il des fils supprimés ou des fils remplacés par des listes). Puis il se met à attendre un message suppression, nouveau nom ou fin de transformation.

S'il reçoit un message de suppression, il passe le même message au nouveau noeud qui le remplace si c'est le cas, puis s'il s'agissait d'un message suppression de type ponctuel, il passe à l'état de suppression, et si c'était un message de suppression totale, il envoie ce même message à chacun de ses fils et passe dans l'état de suppression.

S'il reçoit un message de fin de transformation, il envoie ce même message à chacun de ses fils qui n'ont pas changé de nom lors de sa réponse au dernier message d'identification qu'il a envoyé. S'il a reçu des messages demandant son identification, alors:

- s'il a mémorisé dans l'état métamorphose le nom de la racine de la m.a.t. image qui doit le remplacer, il répond ce nom,
- s'il a reçu un message de nouveau nom, il envoie ce nom comme réponse,
- et sinon il donne son nom actuel comme réponse.

Finalement, il retire de ses listes les règles qui portent allumés leur témoins d'application partielle et réorganise les règles qui restent selon un critère donné, met sa distance de modification à zéro, puis passe à l'état de suppression s'il a reçu un message de nouveau nom, sinon à l'état de la pile en incrémentant son compteur de passages sans tenir compte des messages qui attendent mais qui correspondent à la valeur précédente du compteur de passages.

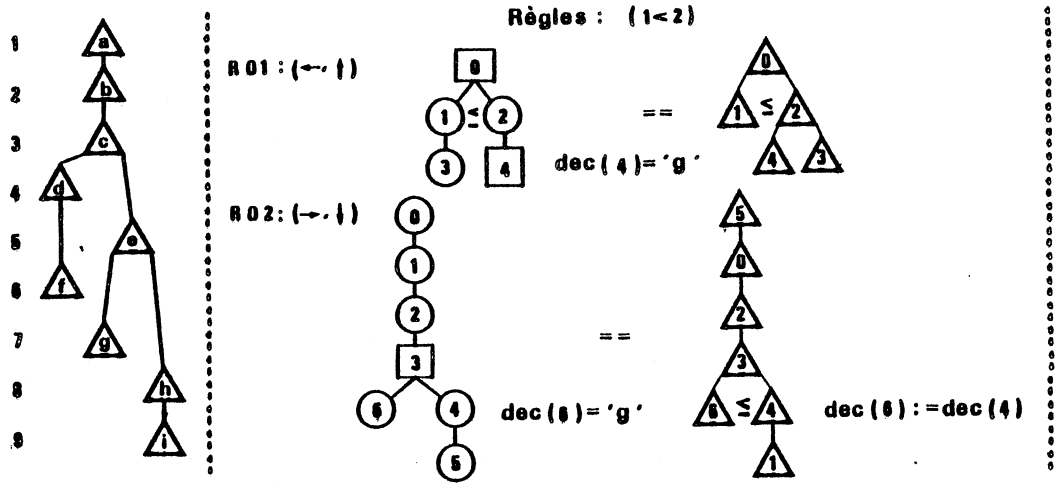
Un noeud en état suppression répond le nom de la racine de la sous-m.a.t. image s'il l'a mémorisé, ou le nouveau nom qui lui a été envoyé, si c'est le cas. Sinon, il répond supprimé aux demandes d'identification. Il envoie tous les autres messages qu'il reçoit au nouveau noeud qui le remplace, s'il y en a un.

4.3 Exemple de transformation

Dans l'exemple suivant, on a un arbre composé de neuf noeuds, chacun avec une variable simple comme décoration associée, et deux règles à appliquer en parallèle. La règle 1 est plus prioritaire que la règle 2. S'il y a plusieurs occurrences de la règle 1 qui s'intersectent sur un même noeud qui est actif pour les deux, on prend la plus à gauche (<) et la plus haute (↑), et si c'est la règle 2 on prend la plus à droite (>) et la plus basse (↓).

La règle 1 contient 3 points actifs et 2 points passifs, on demande que la valeur du point 4 soit "g", et on veut que les noeuds 4 et 3 soient adjacents dans l'image. La règle 2 contient 6 points actifs et 1 point passif, on demande que les points 6 et 4 soient adjacents et que la valeur du point 6 soit "g", et on veut que la valeur du noeud 6 soit égale à celle du noeud 4 dans l'image.

Comme on peut le voir dans la figure suivante, ces deux règles sont applicables en parallèle sur les noeuds 3 et 1 de l'arborescence, mais si on les applique dans un ordre séquentiel, seulement une des deux pourra être utilisée.



4.3 Exemple de transformation

La liste suivante montre les messages envoyés dans le système pendant le premier passage des règles. La première colonne donne l'identification du message par un numéro séquentiel, la deuxième colonne contient le message en question: par exemple, le message 20 est une demande de reconnaissance de la règle 02 à partir du point 01 et le message 127 est une prévention d'utilisation de la règle 02 aux noeuds 03, 07, 08 et 09. La troisième colonne indique l'expéditeur du message, la quatrième colonne indique l'état de l'expéditeur du message, la cinquième est le nom du destinataire, puis on trouve une liste de numéros de messages qui doivent précéder le message en question, par exemple le message 21 est précédé par les messages 2 et 1.

Finalement, on trouve le (numéro de) niveau du message, défini comme zéro s'il n'est précédé par aucun message, et comme le maximum plus un des (numéros de) niveaux des messages qui le précèdent sinon. Cette liste est triée par numéro de niveau, expéditeur, état et destinataire. Les messages qui portent le même numéro de niveau sont susceptibles d'être envoyés en parallèle, et si l'on suppose l'existence d'un nombre arbitrairement grand de processeurs et si les opérations réalisées par un noeud entre deux messages prennent à peu près la même quantité de temps (ce qui n'est pas vrai en général), on peut dire que le niveau maximum correspond au nombre de pas nécessaires pour réaliser la transformation.

No	Message	Exp	Etat	Dest	Prédécesseurs		Niveau	Commentaires
1	fin de trans.	N01	L	N01			0	<- démarrage de la phase de reconnaissance du premier passage
2	reconnaître R01	R01		N01			0	
3	reconnaître R01	R01		N02			0	<- lancement de la recherche de R01 sur tous les noeuds
4	reconnaître R01	R01		N03			0	
5	reconnaître R01	R01		N04			0	
6	reconnaître R01	R01		N05			0	
7	reconnaître R01	R01		N06			0	
8	reconnaître R01	R01		N07			0	
9	reconnaître R01	R01		N08			0	
10	reconnaître R01	R01		N09			0	
11	reconnaître R02	R02		N01			0	<- lancement de la recherche de R02 sur tous les noeuds
12	reconnaître R02	R02		N02			0	
13	reconnaître R02	R02		N03			0	
14	reconnaître R02	R02		N04			0	
15	reconnaître R02	R02		N05			0	
16	reconnaître R02	R02		N06			0	<- les abréviations pour les états sont: T=test, L=libre, A=attente, R=révision, P=prêt, S=suppression, I=initialisation et M=métamorphose
17	reconnaître R02	R02		N07			0	
18	reconnaître R02	R02		N08			0	
19	reconnaître R02	R02		N09			0	
20	reconnaître R02P01	N01	T	N02	11	1	1	<- tous les noeuds passent a l'état test
21	"non"	N01	T	R01	2	1	1	<- rechercher R01 sur le noeud N01 est impossible l'envoie du NON est immédiat
22	reconnaître R02P01	N02	T	N03	12		1	
23	"non"	N02	T	R01	3		1	
24	reconnaître R01P01	N03	T	N04	4		1	<- la recherche continue vers le bas étant donné que N03 peut être racine de R01
25	reconnaître R02P01	N03	T	N04	13		1	
26	reconnaître R01P02	N03	T	N05	4		1	
27	reconnaître R02P01	N03	T	N05	13		1	
28	reconnaître R02P01	N04	T	N06	14		1	
29	"non"	N04	T	R01	5		1	
30	reconnaître R01P01	N05	T	N07	6		1	
31	reconnaître R02P01	N05	T	N07	15		1	
32	reconnaître R01P02	N05	T	N08	6		1	
33	reconnaître R02P01	N05	T	N08	15		1	
34	"non"	N06	T	R01	7		1	
35	"non"	N06	T	R02	16		1	
36	"non"	N07	T	R01	8		1	
37	"non"	N07	T	R02	17		1	
38	reconnaître R02P01	N08	T	N09	18		1	
39	"non"	N08	T	R01	9		1	
40	"non"	N09	T	R01	10		1	
41	"non"	N09	T	R02	19		1	
42	reconnaître R02P02	N02	T	N03	20		2	
43	reconnaître R02P02	N03	T	N04	22		2	
44	reconnaître R02P02	N03	T	N05	22		2	
45	reconnaître R01P03	N04	T	N06	24		2	
46	reconnaître R02P02	N04	T	N06	25		2	
47	reconnaître R01P04	N05	T	N07	26		2	
48	reconnaître R02P02	N05	T	N07	27		2	
49	reconnaître R01P04	N05	T	N08	26		2	

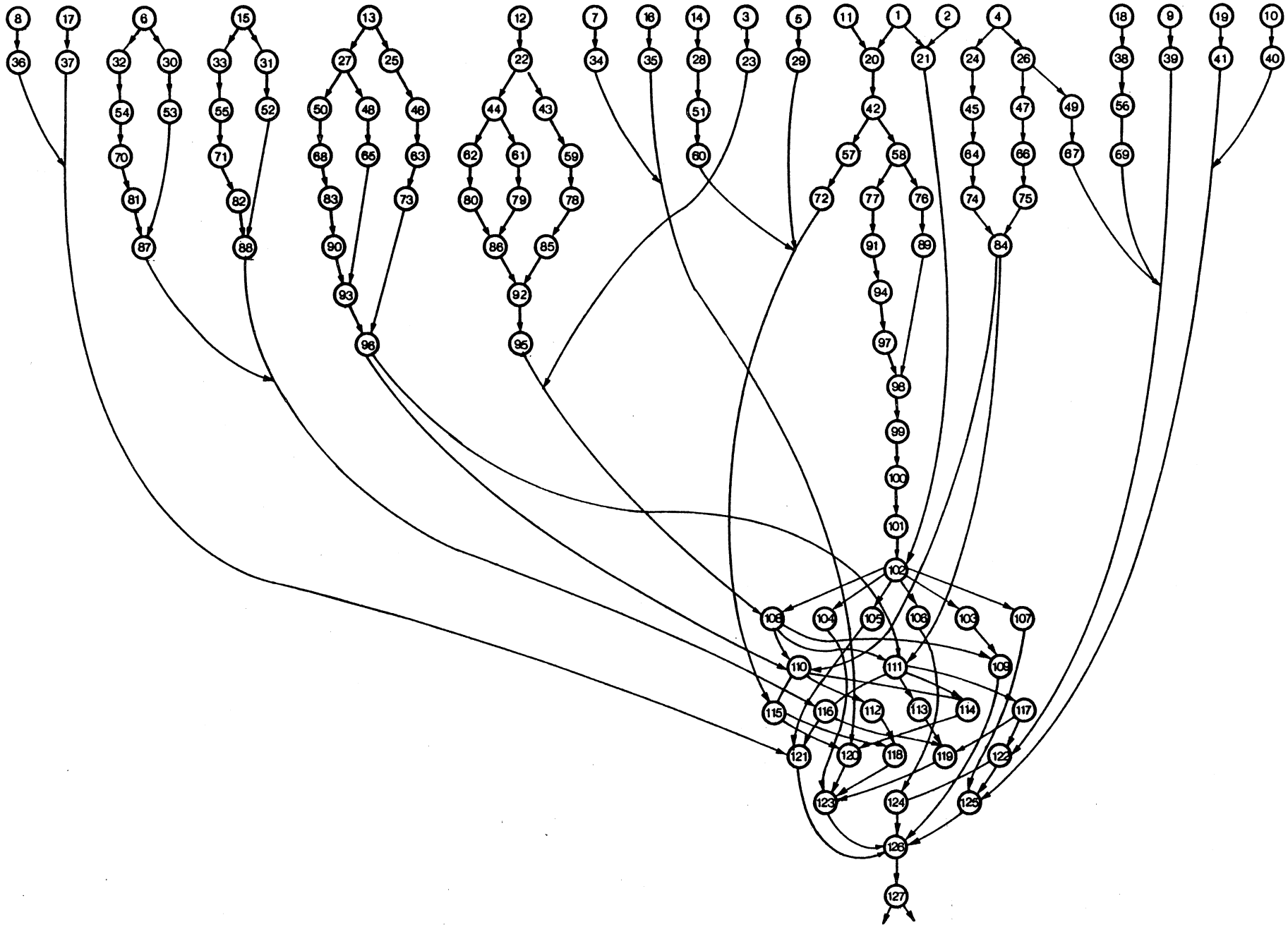
50	reconnaître R02P02	N05	T	N08	27		2	
51	"non"	N06	T	N04	28		2	
52	"non"	N07	T	N05	31		2	
53	"non"	N07	T	N05	30		2	
54	reconnaître R01P04	N08	T	N09	32		2	
55	reconnaître R02P02	N08	T	N09	33		2	
56	"non"	N09	T	N08	38		2	
57	reconnaître R02P03	N03	T	N04	42		3	
58	reconnaître R02P03	N03	T	N05	42		3	
59	reconnaître R02P03	N04	T	N06	43		3	
60	"non"	N04	T	R02	51		3	
61	reconnaître R02P03	N05	T	N07	44		3	
62	reconnaître R02P03	N05	T	N08	44		3	
63	"non"	N06	T	N04	46		3	
64	"oui"	N06	T	N04	45		3	
65	"non"	N07	T	N05	48		3	<- la règle 1 est applicable partiellement sur le noeud 6
66	"oui"	N07	T	N05	47		3	
67	"non"	N08	T	N05	49		3	
68	reconnaître R02P03	N08	T	N09	50		3	
69	"non"	N08	T	R02	56		3	
70	"non"	N09	T	N08	54		3	
71	"non"	N09	T	N08	55		3	
72	"non"	N04	T	N03	57		4	
73	"non"	N04	T	N03	63		4	
74	"oui"	N04	T	N03	64		4	<- la règle 2 ne peut pas s'appliquer sur le noeud 4
75	"oui"	N05	T	N03	66		4	
76	reconnaître R02P06	N05	T	N07	58		4	
77	reconnaître R02P04	N05	T	N08	58		4	
78	"non"	N06	T	N04	59		4	
79	"non"	N07	T	N05	61		4	
80	"non"	N08	T	N05	62		4	
81	"non"	N08	T	N05	70		4	
82	"non"	N08	T	N05	71		4	
83	"non"	N09	T	N08	68		4	
84	"oui"	N03	T	R01	74	75	5	<- la règle 1 est applicable enracinée au noeud 3
85	"non"	N04	T	N03	78		5	
86	"non"	N05	T	N03	79	80	5	
87	"non"	N05	T	R01	53	81	5	
88	"non"	N05	T	R02	52	82	5	
89	"oui"	N07	T	N05	76		5	
90	"non"	N08	T	N05	83		5	
91	reconnaître R02P05	N08	T	N09	77		5	
92	"non"	N03	T	N02	85	86	6	
93	"non"	N05	T	N03	65	90	6	
94	"oui"	N09	T	N08	91		6	
95	"non"	N02	T	R02	92		7	
96	"non"	N03	T	R02	73	93	7	
97	"oui"	N08	T	N05	94		7	
98	"oui"	N05	T	N03	89	97	8	
99	"oui"	N03	T	N02	98		9	
100	"oui"	N02	T	N01	99		10	<- la règle 2 est applicable enracinée sur 1
101	"oui"	N01	T	R02	100		11	
102	départ	N01	A	N02	21	101	12	<- les conjectures locales sont complètes, on commence la construction des
103	confirmer R02	N01	A	N02	102		13	

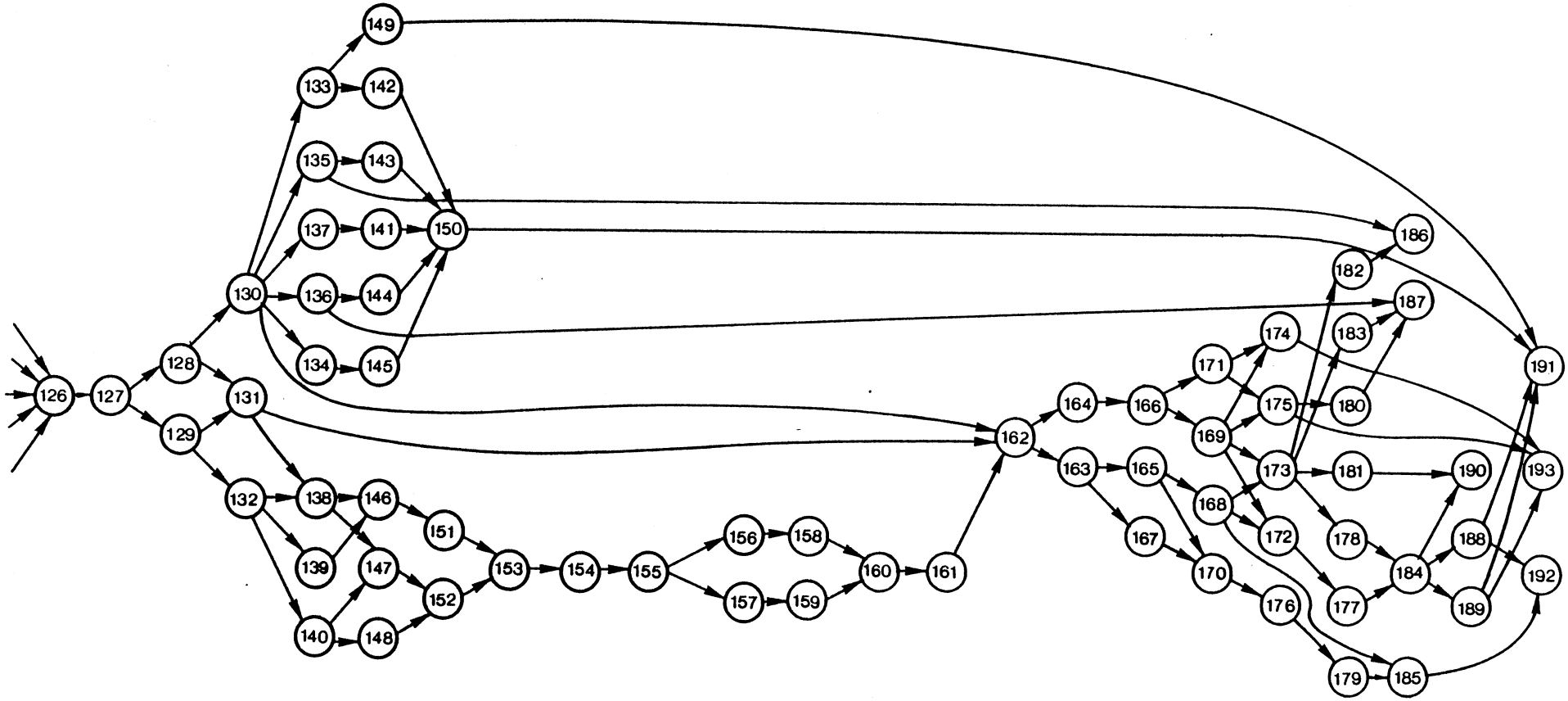
104	confirmer R02	N01	A	N03	102						13	conjectures globales partielles vers le bas
105	confirmer R02	N01	A	N07	102						13	
106	confirmer R02	N01	A	N08	102						13	
107	confirmer R02	N01	A	N09	102						13	
108	départ	N02	A	N03	23	95	102				13	
109	confirmation	N02	P	N01	108	103					14	
110	départ	N03	A	N04	96	84	108				14	<- transmision de messages
111	départ	N03	A	N05	96	84	108				14	
112	confirmer R01	N03	A	N04	110						15	
113	confirmer R01	N03	A	N05	111						15	
114	confirmer R01	N03	A	N06	110	111					15	
115	départ	N04	A	N06	29	60	72	110			15	
116	départ	N05	A	N07	87	88	111				15	
117	départ	N05	A	N08	111						15	
118	confirmation	N04	P	N03	115	112					16	
119	confirmation	N05	P	N03	116	117	113				16	
120	confirmation	N06	P	N03	34	35	115	114			16	
121	confirmation	N07	P	N01	36	37	116	105			16	
122	départ	N08	A	N09	39	69	67	117			16	
123	confirmation	N03	P	N01	108	104	118	119	120		17	
124	confirmation	N08	P	N01	122	106					17	
125	confirmation	N09	P	N01	40	41	122	107			17	
126	ordre de trans.	N01	A	N01	109	121	124	123	125		18	<- la phase de transformation commence
127	prév.R02N03N07N08N09	N01	P	N02	126						19	
128	reconfiguration	N01	P	N02	127						20	
129	prév. R02N07N08N09	N02	P	N03	127						20	
130	ordre de trans.	N01	M	N02	128						21	<- le noeud 1 racine de l'occurrence réalise la
131	reconfiguration	N02	R	N03	129	128					21	transformation correspondant à R02
132	prév. R02N07N08N09	N03	P	N05	129						21	
133	nouveau nom N11	N01	M	N01	130						22	<- création de noeuds
134	nouveau nom N14	N01	M	N02	130						22	
135	nouveau nom N12	N01	M	N07	130						22	
136	nouveau nom N13	N01	M	N08	130						22	
137	nouveau nom N10	N01	M	N09	130						22	
138	reconfiguration	N03	R	N05	132	131					22	
139	prév. R02	N05	P	N07	132						22	
140	prév. R02N09	N05	P	N08	132						22	
141	fin de trans.	N01	M	N10	137						23	<- autorisation de participation au passage
142	fin de trans.	N01	M	N11	133						23	suivant de règles. Le noeud 10 peut commencer
143	fin de trans.	N01	M	N12	135						23	le passage suivant avant la fin du présent
144	fin de trans.	N01	M	N13	136						23	passage pour les autres noeuds
145	fin de trans.	N01	M	N14	134						23	
146	reconfiguration	N05	R	N07	139	138					23	
147	reconfiguration	N05	R	N08	140	138					23	
148	prév. R02	N08	P	N09	140						23	
149	identification	N11	I	N03	133						23	
150	fin de trans.	N01	M	N03	141	142	143	144	145		24	
151	"pas de modif."	N07	R	N05	146						24	<- vérification des conjectures
152	reconfiguration	N08	R	N09	148	147					24	
153	"pas de modif."	N09	R	N08	152	151					25	
154	"pas de modif."	N08	R	N05	153						26	
155	"pas de modif."	N05	R	N03	151	154					27	
156	confirmer R01	N03	A	N04	155						28	
157	confirmer R01	N03	A	N06	155						28	

158	confirmation	N04	P	N03	156	29	
159	confirmation	N06	P	N03	157	29	
160	"pas de modif."	N03	R	N02	158 159	30	
161	"pas de modif."	N02	R	N01	160	31	
162	ordre de trans.	N02	P	N03	130 131 161	32	
163	prêv. R01N06	N03	P	N04	162 160	33	
164	prêv. R01	N03	P	N05	162 160	33	
165	reconfiguration	N03	P	N04	163	34	
166	reconfiguration	N03	P	N05	164	34	
167	prêv. R01	N04	P	N06	163	34	
168	ordre de trans.	N03	M	N04	165	35	<- début de la transformation correspondant à la règle 1. C'est le noeud 3 qui la réalise
169	ordre de trans.	N03	M	N05	166	35	
170	reconfiguration	N04	R	N06	167 165	35	
171	"pas de modif."	N05	R	N03	166	35	
172	nouveau nom N15	N03	M	N04	168 169	36	<- création de noeuds
173	nouveau nom N16	N03	M	N05	168 169	36	
174	ordre de trans.	N05	P	N07	169 171	36	
175	ordre de trans.	N05	P	N08	169 171	36	
176	"pas de modif."	N06	R	N04	170	36	
177	fin de trans.	N03	M	N15	172	37	<- autorisations pour le passage suivant
178	fin de trans.	N03	M	N16	173	37	
179	"pas de modif."	N04	R	N03	176	37	
180	ordre de trans.	N08	P	N09	175	37	
181	identification	N16	I	N06	173	37	
182	identification	N16	I	N07	173	37	
183	identification	N16	I	N08	173	37	
184	fin de trans.	N03	M	N06	177 178	38	
185	ordre de trans.	N04	P	N06	168 179	38	
186	N12	N07	S	N16	135 182	38	<- réorganisation finale de la m.a.t.
187	N13	N08	S	N16	136 183	38	
188	identification	N03	I	N04	172 173 184	39	
189	identification	N03	I	N05	172 173 184	39	
190	N06	N06	I	N16	184 181	39	
191	N03	N03	I	N11	150 149 188 189	40	
192	N15	N04	S	N03	185 172 188	40	
193	N16	N05	S	N03	174 175 173 155 189	40	<- fin du premier passage

La figure suivante représente le graphe des messages pour l'exemple précédent.

On remarque la ressemblance de certains morceaux du graphe avec la structure de la m.a.t. à transformer. Ceci est une conséquence du fait que les messages sont envoyés au travers des descendants, notamment dans la phase de reconnaissance.

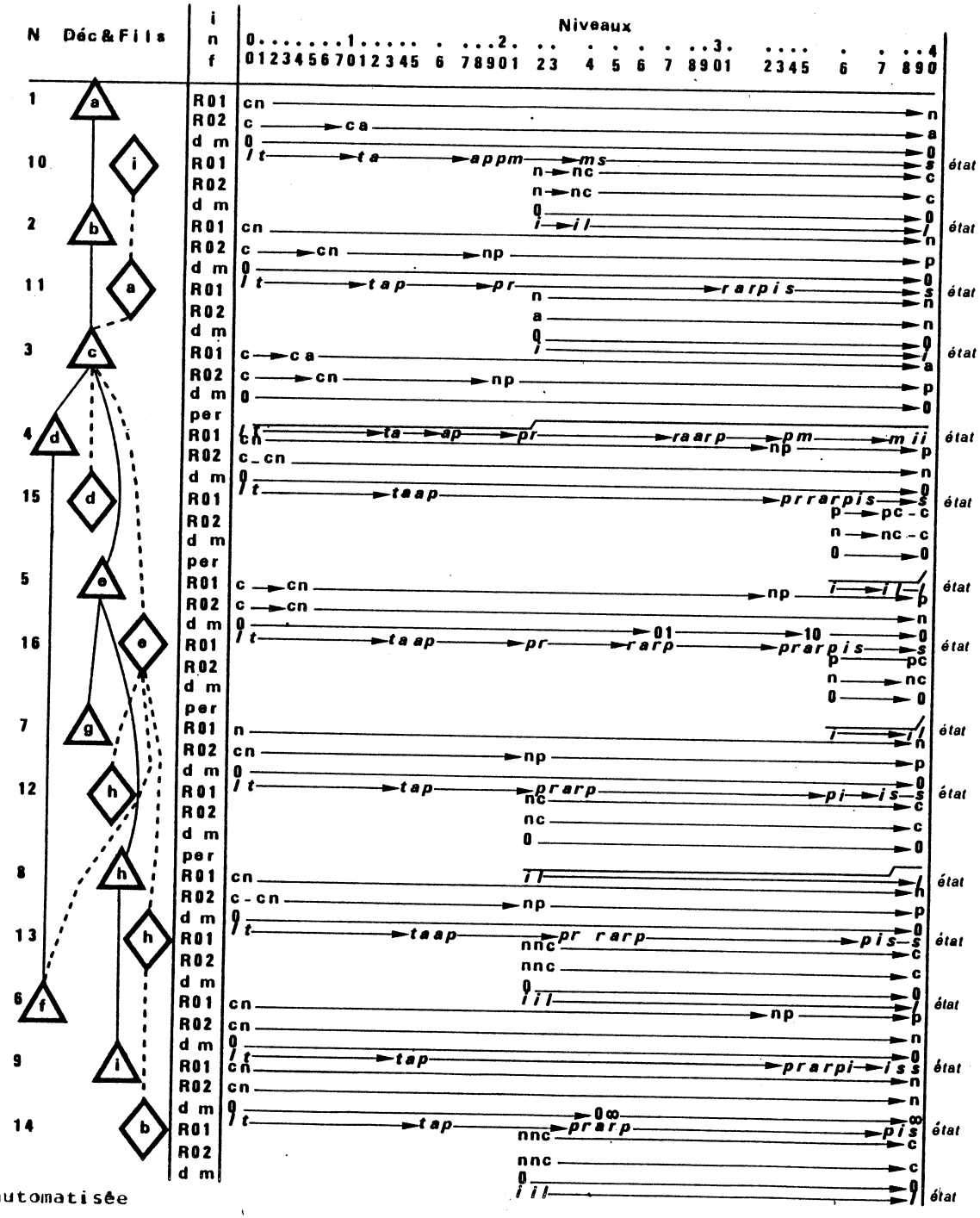

8 17 6 15 13 12 7 16 14 3 5 11 1 2 4 18 9 19 10
36 37 32 30 33 31 27 25 22 34 35 28 23 29 20 21 24 26 38 39 41 40
54 53 55 52 50 48 46 44 43 51 42 45 47 49 56
70 71 68 65 63 62 61 59 60 57 58 64 66 67 69
81 82 83 73 80 79 78 72 77 76 74 75
87 88 90 86 85 91 89 84
93 92 94
96 95 97
98
99
100
101
102
106 104 105 108 103 107
110 111 109
115 116 112 113 114 117
121 120 118 119 122
123 124 125
126
127


126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193

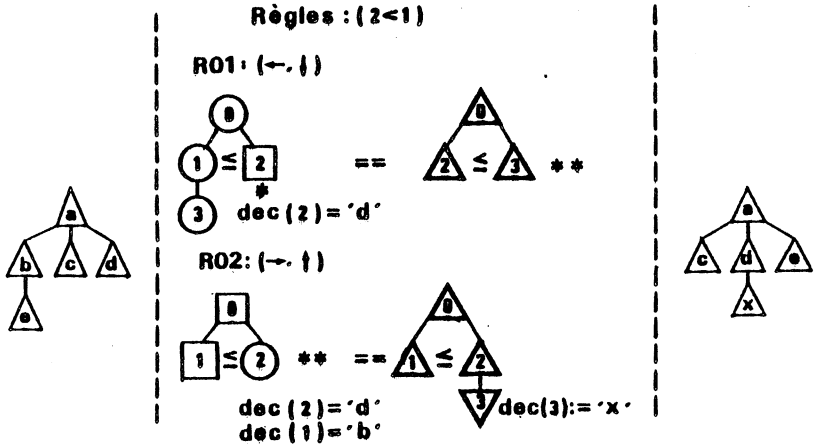
La figure suivante permet de suivre la transformation dans chacun des niveaux. Pour chaque niveau et chaque noeud, on montre: la liste dans laquelle se trouve chacune des règles (C pour candidature, N pour non-applicabilité, A pour applicabilité, I pour interdiction et P pour indiquer si en plus son témoin d'application partielle est allumée), la distance de modification (∞ pour infini) et le moment à partir duquel un noeud change de père (/). Les flèches signifient que la valeur ne change pas entre le début et la fin de chacune, par exemple a-->a est égal à aaaaa. Un blanc signifie que le noeud n'est pas encore créé. Les niveaux 8 et 9 ne sont pas présentés car ils sont identiques au niveau 7. On a laissé certaines colonnes libres entre niveaux, par exemple entre 15 et 16, pour pouvoir montrer des changements d'état pendant lesquels les noeuds ne produissent pas de messages. L'arborescence en lignes continues est la m.a.t. objet et l'autre est le résultat de la transformation pour le premier passage de règles. En bas de chaque ligne de distance de modification (D M), on trouve l'état dans lequel se trouve le noeud.

On n'a pas suivi l'exemple pour le deuxième passage des règles pour ne pas le compliquer. Néanmoins, on voit bien qu'un passage de règles peut commencer sur les états libres avant que le passage précédent ne soit complètement achevé, c'est le cas du noeud 10 dans cet exemple.

On remarque que, dans les conditions idéales de l'exemple, c'est la règle la plus <u>longue</u> qui a défini le temps de reconnaissance et non le nombre de règles à reconnaître ni le nombre de noeuds de l'arbre. Les messages des niveaux 28 et 29 pourraient être évités dans l'exemple si l'on utilise mieux les distances de modification. Il faudrait peut-être mettre à infini les distances de modification de toute une sous-m.a.t. dès que sa racine reçoit un message de reconfiguration. Les messages de modification et pas de modification pourraient être utilisés de sorte que, dans le passage qui suit, on n'ait pas à refaire des reconnaissances des règles ou certaines occurrences qui ont déjà été testées dans le passage précédent sur un noeud, si les noeuds concernés n'ont pas été modifiés. Cela est en rapport avec les critères de passage des règles entre les listes d'un noeud.


N
Déc&Fils
i
n
f
Niveaux
0 1 2 3 4
0123456701234 5 6 78901 23 4 5 6 7 8901 2345 6 7 890
R01
R02
d m
per
état

Dans l'exemple qui suit, on retrouve deux règles qui doivent être appliquées en parallèle comme seul moyen de réaliser les deux transformations. Il illustre aussi un cas de suppression de noeud.



4.4 Remarques sur le comportement de l'algorithme

Afin d'éclairer le comportement de ce système, on donne quelques explications sur les suppositions et possibilités qui ont été envisagées et sur la réalisation d'une maquette.

4.4.1 Commentaires sur l'état actuel

Il est nécessaire de prouver qu'il n'y a pas de modifications simultanées sur un noeud pendant tout le temps de son existence. La définition du contexte prévoit qu'il n'y a pas un noeud actif pour deux occurrences simultanément, ce qui empêche des incohérences au niveau des modifications des noeuds actifs. Mais lorsqu'un noeud N est passif pour une occurrence et lorsque des renseignements sur son contenu ou ses propriétés sont utilisés pour la modification d'autres noeuds actifs, il est nécessaire de sauvegarder ces informations à l'intérieur des listes de marques, car le noeud N peut bien être actif pour une autre règle ou occurrence dans un autre noeud responsable de la transformation, en métamorphose, et ces informations risquent d'être altérées entre temps. Cela doit se faire à la fin de la phase de reconnaissance et juste avant de passer à la phase de transformation. D'ailleurs, les remarques au sujet des distances de modification dans la section 4.2.1 sont en rapport avec ce problème. On suppose donc que les structures de données internes pour les occurrences en tiennent compte.

La place occupée par les m.a.t. peut être importante, et le système qui s'en occupe doit tenir compte du besoin pour les noeuds en état de suppression d'avoir une petite portion des structures de données internes qu'ils portent. La partie inutile pourrait être libérée le plus tôt possible et dès que tous les noeuds qui appartiennent au même niveau de passage sont dans l'état de suppression, ils peuvent être éliminés réellement, libérant ainsi la place occupée.

Le cas serait différent si le système accepte et manipule des pointeurs dans les décorations, car il faudrait maintenir la liaison entre le noeud pointé et le noeud pointeur, peut-être à

l'aide d'un processus supplémentaire. Ce processus ne pourrait être arrêté que par le noeud pointeur et il maintiendrait l'existence du noeud pointé tant que lui-même serait actif. Si l'on assure un nom unique pour chaque noeud du système, les nceuds supprimés et pointés par d'autres noeuds pourraient "suivre" les noeuds pointeurs en incrémentant leurs compteurs de passages, de sorte qu'ils ne soient pas éliminés pendant qu'ils sont pointés. Cette possibilité de liaison entre noeuds supprimés et noeuds non-supprimés pose des problèmes pour les modifications et la cohésion de la structure arborescente, ce qui mérite une étude approfondie. On suppose donc que l'utilisation de pointeurs est de la responsabilité de ceux qui s'en servent au niveau des décorations.

Le système tel qu'il a été défini au chapitre 3 accepte la création de plusieurs copies des noeuds ou des m.a.t. du schéma gauche dans la partie droite des règles. Cela implique la nécessité de créer un opérateur de copie appliqué à la demande par le noeud qui réalise la transformation, mais seulement lorsque toutes les modifications sur cette m.a.t. pendant le présent passage ont été terminées. On pourrait le réaliser par un processus chargé de faire la copie lorsque tous les noeuds de la m.a.t. à copier sont dans l'état d'initialisation, et, après la copie, de leur envoyer le message de fin de transformation. Cela entraîne déjà une modification de l'algorithme pour l'état de métamorphose.

Comme on l'a déjà souligné dans la section 4.3, les distances de modification ne sont pas très bien exploitées par l'algorithme, car elles peuvent servir à éviter la réalisation de reconnaissances inutiles. Cela fait partie des "optimisations" qui peuvent avoir des répercussions sur la complexité du système. Une étude plus complète est nécessaire pour répondre cette question.

4.4.2 Critique, corrections et extensions de l'algorithme

a) Au niveau des noeuds de la m.a.t., il existe des opérations susceptibles d'être réalisées en parallèle. Cela dépend entre autres de la complexité des décorations et des prédicats, car il faut un équilibre entre le contrôle du parallélisme et les opérations "utiles" à la transformation. Si les décorations sont très simples, il peut être préférable de réaliser un travail séquentiel sur toute une sous-m.a.t. que de travailler au niveau des noeuds. D'autre part, si les décorations et prédicats sont très complexes, il pourrait être intéressant de créer plusieurs processus par noeud de la m.a.t. Le choix dépend aussi du nombre de processus et du nombre de règles du système. Ainsi, dans l'étape de reconnaissance, et lorsqu'un noeud se trouve dans l'état de test, on pourrait réaliser plusieurs opérations de reconnaissance en parallèle en créant un processus par couple de noeud et partie de règle à reconnaître, ou bien on pourrait définir un seul processus qui essaie une à une ces règles dans un ordre d'arrivée ou de

priorité.

b) L'algorithme travaille sur des m.a.t. qui sont en principe "éclatées". L'opérateur d'éclatement peut être réalisé en parallèle facilement une fois qu'on a l'opérateur de copie parallèle. C'est un travail qui se fait du haut vers le bas et indépendamment des ascendants, des frères et des descendants des frères du noeud qui éclate au moins dans un premier temps, car il est possible que l'on produise plusieurs m.a.t. équivalentes dans un même noeud ensemble et dans ce cas il faut en laisser une seulement.

Par contre, l'opération de factorisation est beaucoup plus délicate, comme on l'a déjà souligné dans la section 3.2.4.2, car il faut un parallélisme du bas vers le haut qui tienne compte des résultats partiels des branches indépendantes mais descendantes du père du noeud qui factorise. Cela met en évidence le fait que l'algorithme ne suit pas vraiment le modèle défini dans le chapitre 3, car il ne fait pas d'éclatement ni de factorisation dans chaque passage comme prévu. Il doit être corrigé dans une nouvelle version. On souligne en passant que, dans sa version actuelle, STAR-PALE n'accepte pas de règles avec des schémas "ensemble". Cela reste à étudier, et donnerait encore un degré de liberté à l'utilisateur.

c) L'application de règles récursives n'a pas été prévue explicitement dans l'algorithme, mais son inclusion ne pose pas de gros problèmes. En effet, il suffit d'introduire une phase de récursion dans la toute dernière partie de l'étape de transformation dans chaque passage. L'utilisation du parallélisme est immédiate pour des images disjointes et non dépendantes les unes des autres, car les découpages peuvent être simultanés. Si l'on a deux images d'occurrences de règles récursives dont l'une est dépendante de l'autre, il faut découper la plus basse, réaliser la récursion et puis découper la plus haute qui portera déjà les transformations faites par la première récursion. Le parallélisme dans ce cas est impossible et un autre ordre d'application risque de modifier l'image dépendante. Finalement, s'il y a deux images non-disjointes, alors les considérations précédentes sont valables à condition que leurs racines ne soient pas liées au même noeud, mais si c'est le cas on utilise la priorité des règles pour choisir un ordre d'application. Reste le cas où les deux règles ont la même priorité et où leurs racines d'images sont liées au même noeud. Etant donné que, par définition, les points de récursion de chacune d'elles ne peuvent pas partager des noeuds dans l'image, on peut trier les occurrences par rapport au point de récursion lié au noeud de niveau minimum et le plus à gauche dans la m.a.t. Cela nous donne un <u>ordre total</u>, car il n'existe pas d'"images ensemble", et alors on applique dans l'ordre de bas en haut et de gauche à droite.

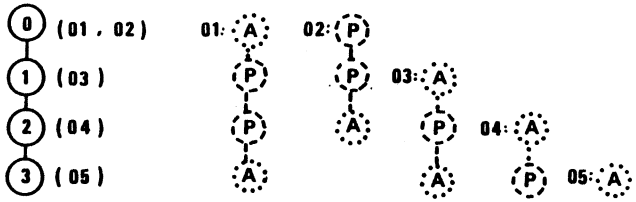
En plus des considérations d'ordre précédentes, il faut modifier les structures de données internes pour mémoriser le contexte de récursion et sauvegarder les valeurs des noeuds "actifs" (liés à des points de récursion) lors des "appels" récursifs.

d) Au début de chaque passage, le premier noeud de la m.a.t. à transformer, dit la racine, entame la phase de reconnaissance en s'envoyant à lui-même un message de fin de transformation car il ne peut pas attendre ce message des ascendants qu'il n'a pas. Les autres noeuds le reçoivent de ses ascendants et l'envoient à leur tour à ses descendants.

La définition de la conjecture globale de chaque passage est réalisée par les noeuds qui se trouvent dans la zone limitée par au moins une occurrence de règle. Cela est fait dans les états "prêt" et "attente" et l'ordre de réalisation n'est pas important si aucune des occurrences dans les listes d'applicabilité ne porte de témoin allumé.

La phase de transformation est lancée par les noeuds qui ne font pas partie d'occurrences ou de chemins dans des occurrences enracinées au niveau de leurs ascendants, qui complètent leur conjecture globale vers le bas et se trouvent dans l'état d'attente. Pour cela, ils s'envoient à eux-mêmes un ordre de transformation avant de passer à l'état de test.

Il y a un problème que l'on va illustrer à l'aide d'un exemple. Supposons que l'algorithme s'applique sur une m.a.t. donnée et qu'à la fin de la phase de reconnaissance on a la situation suivante:



Les noeuds 0, 1, 2 et 3 portent les occurrences élémentaires (01,02), (03), (04) et (05) respectivement. Cela détermine la conjecture locale de chacun d'eux et les noeuds actifs et passifs pour chacune d'elles sont indiqués par A ou P respectivement. Si les priorités des règles sont telles que 03<01, 03<05, 01<05 et 04<02, alors on doit choisir 03, 04 et éliminer 01, 02. Mais comme le noeud 0 passe à l'étape de transformation avant le noeud 1, l'occurrence 01 sera mise avec témoin allumé dans le noeud 3 et, lorsque le noeud 1 voudra confirmer 03, <u>03 sera refusée</u>. On produit ainsi la conjecture 01, 04 qui n'est pas correcte.

La cause de ce conflit est l'ordre d'allumage de témoins d'application partielle. Cela devrait se faire par ordre de priorité des règles, qui n'est pas nécessairement du haut vers le bas. La solution pour notre algorithme pourrait être l'introduction d'un message d'avertissement d'allumage pour chaque règle et qui serait propagé vers le bas. Chaque fois qu'un de ces messages arrive à un noeud qui porte des

occurrences plus prioritaires, il réalise l'avertissement puis l'allumage de ses occurrences prioritaires et après il laisse traverser le message reçu. De toutes façons, il faudrait attendre une réponse au message avant l'allumage correspondant, car celui-ci se fait de manière directe entre le noeud qui demande l'allumage et le descendant actif qui le réalise, sans passer par les noeuds intermédiaires.

e) On remarque que la conjecture globale définie dans la section 3.4 produit une partition de chaque conjecture locale lors de la définition de la conjecture globale de sorte qu'il y aura autant de copies de la m.a.t. que nécessaire pour réaliser les occurrences élémentaires. Ces conjectures seront <u>parallèles</u>, dans le sens où les priorités affectées aux règles n'ont donné de critère complet de choix entre aucune des paires d'occurrences en conflit et qui doivent se trouver sur deux copies différentes.

Cependant, l'algorithme actuel ne crée pas ces copies et ne réalise qu'une seule conjecture globale. Cette anomalie peut être résolue par une utilisation des listes de non-applicabilité qui ont des indicateurs de refus non vides. Il suffit de créer une copie avec les éléments de ces listes, de réaliser l'algorithme tel qu'il est, et puis refaire une autre copie si le résultat contient encore des indicateurs non vides dans les listes de non applicabilité. Cette itération s'arrête après un nombre fini de copies, étant donné que les listes sont finies. Il ne sera évidemment pas nécessaire de tout recopier si l'on se sert de l'opérateur de factorisation.

f) Pour finir nous soulignons la nécessité d'une étude des algorithmes parallèles pour l'application des réseaux de grammaires et pour des opérateurs d'impression de résultats intermédiaires qui seraient utiles aux utilisateurs.

4.4.3 <u>Maquettes à réaliser</u>

Le système STAR-PALE devrait être testé sur une maquette afin de vérifier son comportement. Le travail présenté dans cette thèse a été mené pendant deux ans. La première partie a été consacrée à la mise au courant sur le parallélisme et sur la traduction automatisée par ordinateur. La deuxième partie a été prise par l'étude et définition du système, et la dernière partie par la rédaction. Il aurait été souhaitable de programmer une maquette, mais, vu l'étendue du sujet et le temps disponible, nous avons décidé de faire une définition la plus complète et la plus détaillée possible, plutôt que de commencer une maquette avant de terminer l'étude.

Cependant, on a programmé en PROLOG une partie de l'analyse des expressions d'ordre du chapitre 3. Cela a permis d'étudier la puissance des expressions. Pour l'étude de l'algorithme parallèle donné ici, il a été nécessaire d'utiliser un programme de tri de messages ce qui a permis le test et la mise

au point de l'algorithme tout en faisant des traces à la main.

Pour avoir une idée de la taille d'un système de ce type, on peut regarder le système ARIANE-78 du GETA et un des systèmes opérationnels de traduction russe français qui l'utilise [BN 81]. Il y a trois parties dans une application de traduction automatisée par ordinateur dans ce système: la place en mémoire centrale, à l'exécution, et la place sur disque, qui se divise en deux, celle prise par le logiciel ARIANE-78 et celle prise par l'application linguistique proprement dite (textes, grammaire, dictionnaires, ...). Grosso modo, on a la répartition suivante, en octets:

ARIANE-78	7 Mo
LOGICIEL LINGUISTIQUE et TEXTES	15 Mo
	---
sur disque	22 Mo
A L'EXECUTION (mémoire centrale)	2 Mo

Notons cependant que la place sur disque indiquée (22 Mo) correspond à deux versions de la version RUSSE FRANÇAIS, compilées, ainsi qu'aux corpus de travail (au moins 100 000 mots), avec des résultats de traduction pour la moitié d'entre eux environ, mais sans résultats intermédiaires (en moyenne, la taille d'un arbre est quatre fois plus grande que celle du texte correspondant).

La place mentionée pour le logiciel ARIANE-78 concerne la partie active du système: programmes interprétables (moniteur EXEC), chargeables (TEXT) ou directement exécutables (MODULE). Aucun programme source (ASM360, PLI, PL360) n'y figure. On pourrait en cas de besoin diminuer cette place en supprimant la grande majorité des TEXT, qui ne servent qu'à pouvoir regénérer les MODULES si nécessaire. En ce qui concerne les programmes source de ROBRA (sous-système d'ARIANE-78), on compte environ 12000 lignes en PL360 et 5000 en assembleur (macros et dummy sections inclus).

La réalisation d'une maquette de STAR-PALE peut comporter différents niveaux de modélisation et c'est pour cela que nous parlons de maquettes, c'est-à-dire:

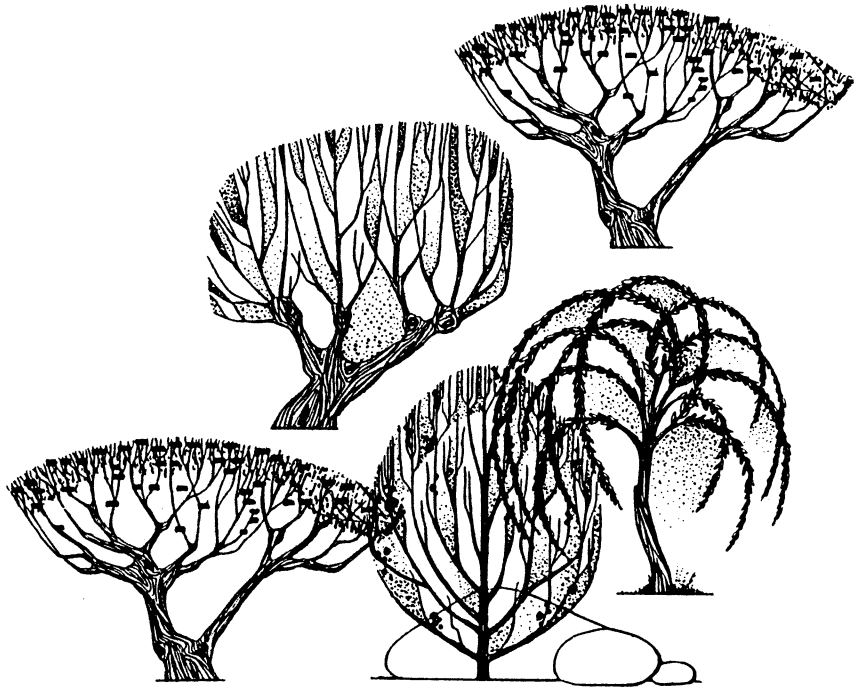
- une maquette générale du type donné dans les algorithmes du chapitre 3
- une maquette qui concerne le problème interne du parallélisme tel qu'il a été exposé dans le présent chapitre
- ces modèles peuvent concerner aussi bien l'application d'une grammaire simple à une m.a.t. que la traversée d'un réseau complet

Le langage de programmation le plus adapté dans une première phase d'étude serait PROLOG, car il permet une programmation non déterministe avec laquelle on peut simuler le parallélisme, son niveau est assez élevé pour laisser de côté certains détails de programmation et sa sémantique claire pourrait

faciliter la preuve des programmes. Une réalisation plus complète pourrait utiliser des langages tels que PASCAL Concurrent ou ADA qui permettent une programmation du parallélisme, gèrent des structures de données complexes et assurent une programmation plus détaillée car, tout en étant de haut niveau, ils sont plus proches des architectures des machines actuelles.

Chapitre V

Conclusion



5.1 Conclusion

Le système présenté ici est plus puissant et plus général que ROBRA. Il s'agit d'un système de transformation de structures arborescentes qui permet une exploitation du parallélisme à des niveaux divers, depuis la définition de la structure de données jusqu'aux contrôles d'application de règles et de grammaires.

On n'a pas défini les décorations utilisées, mais il est évident que le parallélisme peut être appliqué, même au niveau de la manipulation de décorations. De même, on a le cas des dictionnaires, qui n'a pas été mentionné ici, mais qui implique aussi des recherches éventuelles en parallèle. La syntaxe des différents noms utilisés dans le système, tels que noms de nceuds, de points, de règles, de systèmes transformationnels n'a pas été donnée ici.

Les phases de reconnaissance et de transformation sont susceptibles d'utiliser le parallélisme, à cause des expressions d'ordre et de l'usage des priorités horizontales et verticales, ce qui permet en effet la description de plusieurs possibilités avec une seule règle.

Les systèmes transformationnels munis de plusieurs points d'entrée, et des options de contrôle Et et Ou, permettent le test de différentes stratégies en parallèle, la production de différents résultats et leur comparaisons en parallèle, ou la recherche non-déterministe d'une ou plusieurs solutions.

On remarque que le fait d'avoir les m.a.t. d'entrée et de sortie de type liste ou arbre, ainsi que l'introduction de dictionnaires, laisse supposer qu'un système comme STAR-PALE pourrait remplacer non seulement ROBRA mais aussi ATEF et SYGMOR (deux autres sous-systèmes du laboratoire GETA), ce qui reste à démontrer.

Nous avons fait l'étude d'une méthodologie d'implémentation qui a permis de définir un algorithme d'application de règles en parallèle qui a été testé sur des petites arborescences. Mais il n'est pas encore complètement défini et il s'agit plutôt d'une première approche. Son comportement doit être étudié d'une façon théorique et pratique. On doit construire une maquette qui permettrait de comparer ses résultats avec d'autres systèmes, en particulier avec ROBRA. Il est intéressant de voir les gains d'un tel système en fonction du nombre de processeurs disponibles et de la taille de la m.a.t. à transformer.

Les principales questions qui attendent un développement sont: une étude plus complète des m.a.t., une étude des types de décorations et des dictionnaires, une étude des systèmes de marquage et des options de contrôle, et l'étude de la complexité et de la décidabilité du système pour en connaitre la puissance et les limites. L'importance d'une étude de cette nature est grande du point de vue pratique et théorique.

Bibliographie

(AHV 80) F.Andre, D.Herman et J.P.Verjus
"Contrôle du parallélisme et de la répartition", 2ème édition du cours donné à l'école d'été de l'AFCET à Aix-en-Provence, 1980

(AM 71) E.Ashcroft and Z.Manna
"Formalizations of Properties of Parallel Programs", MACHINE INTELLIGENCE 6, (ed.) B.Meltzer and D.Michie, Edinburgh University Press 1971

(BGQ 78) Ch.Boitet, P.Guillaume et M.Quezel-Ambrunaz
"Manipulation d'arborescences et parallélisme: Système ROBRA", Communication présentée à COLING 78 (14-18 Août 78) - Bergen

(BH 72) P.Brinch Hansen
"A comparison of two synchronizing concepts", Acta informatica, 1, pp. 190-199 (1972)

(BH 73) P.Brinch Hansen
"Operating system principles", Prentice-Hall, Englewood Cliff, New-Jersey (1973)

(BN 81) Ch.Boitet, N.Nedobejkine
"Documentation du systeme Russe-Francais: Une nouvelle organisation des grammaires transformationnelles donnant un gain de temps d'environ 40%", Doc. No. 42, G.E.T.A., Grenoble, Juillet 1981

(Bo 76) Ch.Boitet
"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, 1976

(Bo 81) Ch.Boitet
"Tendances futures en traduction automatisée", revue Le langage et l'homme, No. 45, pp. 16-28, janvier 1981

(Ch 74) J.Chauché
"Transducteurs et Arborescences. Etude et Réalisation de Systèmes appliqués aux grammaires transformationnelles", thèse d'Etat, Université de Grenoble, 1974

(Co 70) A.Colmerauer
"Les sytemes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur", Publication interne No. 43 Projet TAUM, Université de Montréal, Septembre 1970

(Cr 75) CROCUS
"Systèmes d'exploitation des ordinateurs", Dunod informatique, Ed. Bordas, Paris 1975

(Dav 79) A.L.Davis
"A Data Flow Evaluation System Based on the Concept of Recursive Locality", AFIPS, National Computer Conference 1979

(Di 68) E.W.Dijkstra
"Co-operating Sequential Processes", (in) Programming Languages, F.Genuys (ed.), Academic Press, New York, 1968, pp 43-112

(Di 71) E.W.Dijkstra
"Hierarchical ordering of sequential processes", Acta Informatica, Vol. 1, No.2, 1971

(Ea 70) J.Earley
"An efficient context-free parsing algorithm", CACM, Vol. 13, No. 2, Fevrier 1970, pp. 94-102

(GE 80) GETA
"Study of Eurotra's Data Structures and C-level Language. Final Report", GETA/EUROTRA/ April 1980

(He 72) C. Hewitt
"Description an Theoretical Analysis (using Schemata) of PLANNER: A language for proving theorems and manipulating Models in Robot", PhD Thesis, AI-TR258, the artificial intelligence laboratory, M.I.T., Cambridge, Massachusets, 1972

(Ho 72a) C.A.R.Hoare
"Toward a Theory of Parallel Programming", (in) Operating Systems Techniques, Academic Press, New York, 1972, pp 61-71

(Ho 74) C.A.R.Hoare
"Monitors: An Operating System Structuring Concept", CACM, Vol. 17, No. 10, (Oct. 74), pp. 549-557

(Kn 69) D.E.Knuth
"The Art of Computer Programming, Vol. 2: Seminumerical algorithms", Addison - Wesley, Reading, Mass., 1969

(Kn 73a) D.E.Knuth
"The Art of Computer Programming, Vol. 1: Fundamental Algorithms", Addison - Wesley, Reading, Mass., Sec. Ed., 1973

(Kn 73b) D.E.Knuth
"The Art of Computer Programming, Vol. 3: Sorting and Searching", Addison - Wesley, Reading, Mass., 1973

(Me 66) E.Mendelson
"Introduction to Mathematical Logic", van Nostrand Reinhold Company, New York, 1966

(Ro 73) B.K.Rosen
"Tree Manipulating Systems and Church-Rosser theorems", JACM, Vol. 20, No. 1, Janvier 1973

(TA 79) TAUM
"Présentation de la chaîne de traduction informatisée TAUM/AVIATION", Université de Montréal, 27 mars 1979

(Va 75) B.Vauquois
"La traduction automatique à Grenoble", Document de Linguistique Quantitative No. 24, Dunod, 184p., 1975

(Va 79) B.Vauquois
"Aspects of mechanical translation in 1979", conference for Japan IBM scientific program, Juillet

1979

(Ve 79) J.N.Verastegui-Carvajal
"Sémantique dénotationnelle et implémentation d'un interpréteur pour un langage applicatif", Mémoire de D.E.A., Université de Grenoble, 1979

(Ve 81) J.N.Verastegui-Carvajal
"Présentation des directions générales de recherche en parallélisme", IMAG, R. R. No. 258, Grenoble, Juillet 1981

(Wo 70) W.A.Woods
"Transition Network Grammars for Natural Language Analysis", CACM, Vol. 13, No. 10, pp. 391-606 (1970)

Index

A

ADA, 113
ALPAC (rapport), 6
algorithme
  analyse d'une permutation, 42
  conjecture globale, 82
  général d'application, 82
  parallèle, 86
  placement des listes implicites, 66
  tri topologique, 50 , 51
ambiguïté, 23
applicabilité, 57
  locale, 56
arbre, 25
  famille de, 33
arc, 72 , 76
ARIANE-78, 7 , 111
  ATEF, 7
  ROBRA, 7
  SYGMOR, 7
  TRANSF, 7
associativité, 27 , 43

C

candidatures (nombre de), 74
chaînes d'ordre, 63
chemin, 23 , 35 , 57 , 62
  achevé, 65
  droit, 36
  m.a.t. associée, 36
  noeud dominant, 36
  gauche, 36
  provenant d'un schéma vertical, 65
  sans issue, 65
  strict, 36
complexité, 82
concaténation, 42
condition
  attente (et, ou), 72
  borné à n passages (B(n)), 78
  boucles (borné à n passages, exhaustif), 72 , 78
  de parcours (boucles), 72
  d'application (passage, retour arrière, attente), 72
  exhaustif (E), 78
  et (&), 79
  facultatif (F), 79
  géométrique ou structural, 56
  impératif (I), 78
  intersommets, 92
  normal (N), 79
  ou (o), 79
  passage (impératif, facultatif), 72
  retour arrière (zéro, normal), 72 , 78
  zéro (Z), 79
conflit (prédicat), 81
conforme (prédicat), 50
conjecture, 62
  globale, 73 , 81 , 86 , 109
  globale partielle vers le bas, 89
  locale, 63 , 88
  locale ou situation globale d'applicabilité sans restrictions, 93
  parallèle, 111
contexte, 86 , 107
  conflit contextuel, 81
  conflit hors du, 71 , 81
  de récursion, 72 , 109
  d'une règle, 70
contrôles, 72
  autonome (A), 74
  borné à n passages (B(n)), 73
  convergent haut ou bas et gauche ou droit (V(↑ ou ↓, ← ou →)), 74
  coupe haute ou basse (C(↑, ↓)), 73
  dépendances (coupe haute ou basse, total), 72
  dessous (D), 73
  dessus (S), 74
  exhaustif (E), 73
  interdiction automatique (dessous, dessus, ponctuel, libre), 72
  interdiction par application (convergent haut ou bas et gauche ou droit, autonome), 72
  libre (L), 74
  mode de contrôle courant, 76
  passages (borné à n passages, exhaustif), 72
  ponctuel (P), 74
  total (T), 73
  copie
    anomalie, 111
    des éléments, 67 , 107
coût (fonction de), 34

D

décidabilité, 82
décoration, 24 , 26 , 69
  structures semi-décorées, 69
  vide, 69
descendant
  actif, 93
  implicite, 56
dictionnaire, 82
dissocié (opérateur), 54

E

éclatement, 33 , 108
écarté, 54
  à droite, 47 , 58
  à gauche, 47
  semi-, 54
ensemble, 23
équivalence
  faible, 34
  forte, 30
état, 26
  attente, 87 , 89 , 93
  courant, 87
  graphe de transitions, 88
  initialisation, 87 , 89 , 96
  libre, 87 , 88 , 92
  métamorphose, 87 , 89 , 95
  pile d', 87
  prêt, 87 , 89 , 94
  retour arrière, 89
  révision, 87 , 89 , 94
  suppression, 87 , 89 , 96
  test, 87 , 88 , 92
expression
  désordre, 42 , 54
  linéaire induite par une permutation, 47
  linéaires disjointes, 52
  linéaires entremêlées, 52
  linéaires superposables, 52
  logique (négation ¬, conjonction &, disjonction V), 42
  naturelle d'une permutation, 47
  ordre, 42
extractions, 35

arbor, 35
associé, 36
cheminD, 36
cheminG, 36
cheminS, 36
dominant, 36
Etage, 37 , 57
extraire, 39 , 57
listeD, 39
listeG, 39
noeud, 35
nom, 35
Sous-Etages, 37

F

factorisation, 23 , 32 , 34 , 54 , 62 , 79 , 108
commutativité, 54
feuille, 25
apparente, 34
ensemble, 23
liste, 23
piste, 23 , 35
fils, 25
apparent, 27
véritable, 28 , 58

G

grammaires, 71
passage d'une grammaire, 72
projectives, 24
réseau de, 42 , 72

H

heuristique, 76

I

image, 63 , 65
adoptives, 65
complète, 67
famille d', 64
horizontale, 65
miroir, 54
ramifiée, 65
verticale, 65
vide, 67
indicateur
du nom du noeud, 87
d'occurrence responsable du refus, 87
intervalle fermé, 42
inverse d'une fonction, 42

L

liaison ou correspondance entre noeuds, 57
libre (type), 26
liste, 23 ,62
d'applicabilité (courante, copie), 87
de candidature, 87
d'interdiction, 87
de non-applicabilité, 87
de renseignements concernant des messages, 87
explicite, 64
implicite, 63
provenant d'un schéma horizontal, 65

M

maquette, 111
modèle de modélisation, 113
marques, 72 , 88
message
d'avertissement, 79
d'avertissement d'allumage, 111
confirmation (question), 91
confirmation ou refus (réponse), 91
départ (signal), 91
fin de transformation (signal), 91
graphe (exemple parallèle), 102
identification (question), 91
liste de (exemple parallèle), 97
modification ou pas de modification (réponse), 91
nouveau nom (signal), 91
nouveau nom ou supprimé (réponse), 91
numéros de niveau (exemple parallèle), 97
ordre de transformation (signal), 91
oui ou non (réponse), 91
parallélisme (exemple), 102
prévention (signal), 91
file de, 87
reconfiguration (question et signal), 91
reconnaissance (question), 91
suppression ponctuelle ou totale (signal), 91
système de communication, 91
tableau de, 89
minimisation, 33 , 34
modification, 69
distance de, 87 , 94 , 107 , 107 , 108
simultanée sur un noeud, 107
multi-arborescence à tranches (m.a.t.), 23 , 85
cardinalité, 27
Mat, 26
m.a.t. active, 71 , 86
m.a.t. cible, 42
m.a.t. en attente, 79
m.a.t. objet, 42
m.a.t. passive, 71 , 86
m.a.t. vide, 26
place occuppée, 107

N

niveau, 56
noeud
accessible, 76
autosuppression, 89
composé, 24
contenu, 24
création, 89
d'arrivée, 76
de commencement, 76
de départ, 76
d'entrée, 76
de sortie, 76
du réseau, 72
explicites de l'image, 56
internes, 76
nom, 24
nom unique, 25
simple, 24
suppression, 89
numérotation canonique, 32

O

occupant, 44
d'une place, 46
explicite, 46 , 64
implicite, 46 , 64
implicite interne, 48 , 62
occurrence, 48 , 87
élémentaire, 57
opérateur
arithmétiques, 62
de comparaison, 62
de copie, 107
d'impression, 111
de manipulation de décorations, 62
d'échange, 70
ensemblistes, 62

interclassement ou parallélisme, 51
ordinateurs
calculateur vectoriel, 11
flot de données, 11
modèle de John von Neumann, 11
multiprocesseur, 11
pipeline, 11
ordre, 58
chaînes d', 63
contradictoire, 52
doit être exécuté avant (relation d'--- partiel), 53
être à gauche (relation d'--- partiel), 44 , 50
être plus à gauche (relation d'--- partiel), 63
être plus haut (relation d'--- partiel), 63
lexicographique, 32
partiel, 32 , 44
total, 47

P

parallélisme
niveau de, 108
et non-déterminisme, 10
processeur, 12
processus, 12
actif, 13
bloqué, 13
communication, 16
boîte aux lettres, 17
messages, 16
modèle du producteur et du consommateur, 16
sémaphores avec messages, 17
contrôle et contrôleurs, 17
attente infinie, 18
blocage, 18
centralisé ou réparti, 18
moniteurs, 19
région critique, 19
requête et autorisation, 18
sites (systèmes répartis), 19
trace temporelle, 18
création, 14
destruction, 14
exclusion mutuelle, 13
attente active, 14
section critique, 14
sémaphore, 14
verrou, 14
indépendant, 13
interblocage, 13
algorithme du banquier, 13
guérison, 13
prévention, 13
parallèles, 13
ressource, 13
commune, 13
critique (partageable avec 1 points d'accès), 13
locale, 13
partageable avec n points d'accès, 13
synchronisation, 15
directe et indirecte, 15
par événements, 15
par sémaphores privés, 16
problème des philosophes et des spaghetti, 16
vecteur d'état d'un, 12
recherche en parallèle, 12
tris en parallèle, 12
PASCAL Concurrent, 113
passage, 86
compteur de, 87
père, 25
permutation, 42 , 58 , 64
bijection, 42
comprise, 42 , 44
contenance, 42
induite, 43
naturelle, 46
partition maximale d'une, 44
place d'une, 44
piste, 23 , 35
point, 56 , 65
actif, 70 , 85
adoptif, 65
créé, 65
d'adoption, 65
de récursion, 72
de repère, 56
décoré, 56 , 65
facultatif, 57
familial, 56
feuille, 56 , 65
nommé, 56 , 63 , 65
passif, 70 , 86
supprimé, 65
pointeurs et décorations, 107
prédicat, 56 , 62
d'ordre, 58
fragmentaire, 56
plein, 56 , 57 , 65
priorité, 63
d'application des opérateurs, 56
et témoin d'application partielle, 93
horizontale, 56
verticale, 56
processus, 85
communication, 85
coopération, 85
démarrage de la phase de reconnaissance, 109
démarrage de la phase de transformation, 109
synchronisation, 88
PROLOG, 111
propriété, 26
arbs, 28
contenu, 26
fils, 28
fils-apparent, 28
narbs, 28
néléments, 27
nfils, 28
nfils-apparents, 28
niveau, 28
nnoeuds, 32
tranche, 28
type, 26

R

racine
de l'image, 65
du schéma, 57
reconnaissance, 56 , 86
règle, 56
candidate, 73
interdite, 73
nom de, 56 , 72
récursive, 72 , 74 , 109
retour arrière, 78
zone d'action, 89

S

schéma, 56
complet, 64
de premier niveau, 56
descendant, 56
ensemble, 108
facultatif, 57
général, 56
horizontal, 56
ramifié, 56
vertical, 56
sémantique, 56

structure multi-arborescente à tranches (s.m.a.t.), 32
minimale, 34
Smat, 32

T

témoin (éteint, allumé),
d'application partielle, 87
d'application partielle et priorité, 93
de nouveau noeud, 87
problème d'allumage, 111
test (procédure), 57
trace, 23 , 25 , 64
traduction
analyse, 4
morphologique, 5
structurale, 5
aidée par l'ordinateur, 6
automatisée par ordinateur (TAO), 6
descripteur structural, 4
génération, 4
morphologique, 6
d'une structure syntaxique de surface, 6
système de première génération, 4
système de seconde génération, 4
transfert, 4
lexical, 6
structural, 6
traînée gauche, droite, 25 , 64
tranche, 23
transfert (fonction de), 69
transformation, 56 , 86
exemple parallèle, 96
identité, 67
nom de SST, 72
nom de ST, 72
nombre de pas nécessaires, 102
sous-système transformationnel (SST), 72
système transformationnel (ST), 71
trace par niveau (exemple parallèle), 102
transitive (relation), 50
treillis, 34 , 44 , 51
expression semi-linéaire, 53
trou, 48 , 65

V

voisin, 54 , 58
à droite, 47
à gauche, 47
à la frontière droite, 48
à la frontière gauche, 48

THESE DE DOCTEUR INGENIEUR
INFORMATIQUE

Auteur: J.Nelson VERASTEGUI C.
Titre: Etude du Parallélisme appliqué à la Traduction Automatisée par Ordinateur. STAR-PALE: Un système parallèle.
Date: 17 Mai 1982, à 16H30, salle D. 001
Mots clés: Parallélisme, Multi-traitement, Traduction Automatisée, Systèmes de réécriture, Transformation d'arborescences.

Résumé:

On fait une courte présentation de la traduction automatisée par ordinateur et des concepts du parallélisme. On présente un système de transformation de structures arborescentes adapté au traitement parallèle. Les structures de données, appelées mat, permettent la manipulation des ambiguïtés et des choix structuraux. Les règles et grammaires du système, appelé STAR-PALE, peuvent contenir un certain nombre d'options de contrôle, de reconnaissance et de transformation. On donne une méthodologie pour l'implémentation de ce type de systèmes.

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

. Ch. BOITET, Professeur
. J. VIDART, Professeur

Monsieur VERASTEGUI-CARVAJAL José Nelson

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, Spécialité "Informatique".

Fait à Grenoble, le 13 mai 1982

Le Président de l'I.N.P-G.