Transduction d'arborescences : application aux manipulations de formules sur ordinateur

Jacques Chauché

N° D'ordre

THESE

présentée à

L'UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR DE TROISIEME CYCLE

"Informatique"

par

Jacques CHAUCHÉ

Transduction d'arborescences

Application aux manipulations de formules sur ordinateur

Thèse soutenue le 30 Avril 1971 devant la commission d'examen :

Monsieur J. KUNTZMANN	Président
Messieurs B. VAUQUOIS	Examinateur
N. GASTINEL	Examinateur
Y. SIRET	Invité

FACULTE DES SCIENCES
DE GRENOBLE

LISTE DES PROFESSEURS

<> <> <>

PROFESSEURS TITULAIRES

MM.	NEEL Louis	Physique expérimentale
	KRAVTCHENKO Julien	Mécanique rationnelle
	CHABAUTY Claude	Calcul différentiel et intégral
	BENOIT Jean	Radioélectricité
	CHENE Marcel	Chimie papetière
	FELICI Noël	Electrostatique
	KUNTZMANN Jean	Mathématiques Appliquées
	BARBIER Reynold	Géologie appliquée
	SANTON Lucien	Mécanique des fluides
	OZENDA Paul	Botanique
	KOSZUL Jean Louis	Mathématiques pures
	GALVANI Octave	Mathématiques pures
	MOUSSA André	Chimie nucléaire et radioactivité
	TRAYNARD Philippe	Chimie générale
	SOUTIF Michel	Physique générale
	CRAYA Antoine	Hydrodynamique
	REULOS René	Théorie des Champs
	BESSON Jean	Chimie minérale
	AYANT Yves	Physique approfondie
	GALLISSOT François	Mathématiques pures
Mle.	LUTZ Elisabeth	Mathématiques pures
MM.	BLAMBERT Maurice	Mathématiques pures
	BOUCHEZ Robert	Physique nucléaire
	LLIBOUTRY Louis	Géophysique
	MICHEL Robert	Minéralogie et pétrographie
	BONNIER Etienne	Electrochimie-Electrométallurgie
	DESSAUX Georges	Physiologie animale
	PILLET Emile	Physique industrielle
	YOCCOZ Jean	Physique nucléaire théorique
	DEBELMAS Jacques	Géologie générale
	GERBER Robert	Mathématiques pures
	PAUTHENET René	Electrotechnique
	MALGRANGE Bernard	Mathématiques pures
	VAUQUOIS Bernard	Calcul électronique
	BARJON Robert	Physique nucléaire
	BARBIER Jean Claude	Physique expérimentale
	SILBER Robert	Mécanique des fluides
	BUYLE-BODIN Maurice	Electronique

	DREYFUS Bernard	Thermodynamique
	KLEIN Joseph	Mathématiques pures
	VAILLANT François	Zoologie
	ARNAUD Paul	Chimie
	SENGEL Philippe	Zoologie
	BARNOUD Fernand	Biosynthèse de la cellulose
	BRISSONNEAU Pierre	Physique du solide
	GAGNAIRE Didier	Chimie physique
Mme	KOFLER Lucie	Botanique et physiologie végétale
MM.	DEGRANGE Charles	Zoologie
	PEBAY-PEROULA Jean Claude	Physique
	RASSAT André	Chimie systématique
	DUCROS Pierre	Cristallographie
	DODU Jacques	Mécanique appliquée- I.U.T."A"
	ANGLES D'AURIAC Paul	Mécanique des fluides
	LACAZE Albert	Thermodynamique
	GASTINEL Noël	Analyse numérique
	GIRAUD Pierre	Géologie
	PERRET René	Servomécanisme
	PAYAN Jean Jacques	Mathématiques pures
	CAUQUIS Georges	Chimie organique
	RENARD Michel	Thermodynamique
	BONNET Georges	Electrotechnique
	BOLLIET Louis	Informatique- I.U.T."B"

PROFESSEURS ASSOCIES

MM.	RADHAKRISHNA	Thermodynamique
	BULLEMER Bernhard	Physique

PROFESSEURS SANS CHAIRE

M.	GIDON Paul	Géologie et minéralogie- C.S.U.
Mme	BARBIER Marie Jeanne	Electrochimie
Mme	SOUTIF Jeanne	Physique générale
MM.	COHEN Joseph	Electrotechnique
	DEPASSEL Roger	Mécanique des fluides
	GLENAT René	Chimie organique
	BARRA Jean	Mathématiques appliquées
	COUMES André	Electronique- E.I.E.
	PERRIAUX Jacques	Géologie et minéralogie
	ROBERT André	Chimie papetière
	BIAREZ Jean Pierre	Mécanique
	BONNETAIN Lucien	Chimie minérale
	HACQUES Gérard	Calcul numérique
	POULOUJADOFF Michel	Electrotechnique
Mme	KAHANE Josette	Physique
Mme	BONNIER Jane	Chimie générale
MM.	VALENTIN Jacques	Physique nucléaire
	REBECQ Jacques	Biologie- C.S.U.Chambery
	DEPORTES Charles	Chimie minérale

MM.	SARROT-REYNAULD Jean	Géologie
	BERTRANDIAS Jean Paul	Mathématiques appliquées
	AUBERT Guy	Physique
	GAUTHIER Yves	Sciences biologiques- C.S.U.
	DOLIQUE Jean Michel	Physique des plasmas
	DESRE Pierre	Métallurgie
	LAURENT Pierre	Mathématiques appliquées
	CARLIER Georges	Biologie végétale
	SIBILLE Robert	Construction mécanique-I.U.T."A"

MAITRES DE CONFERENCES

MM.	LANCIA Roland	Physique automatique
Mme	BOUCHE Liane	Mathématiques. C.S.U.Chambery
MM	KAHANE André	Physique générale
	BRIERE Georges	Physique expérimentale
M.	LAJZEROWICZ Joseph	Physique
Mme	BERTRANDIAS Françoise	Mathématiques pures
MM.	LONGEQUEUE Jean Pierre	Physique nucléaire
	ZADWORNY François	Electronique
	DURAND Francis	Métallurgie
	PFISTER Jean Claude	Physique générale
	CHIBON Pierre	Biologie animale
	IDELMAN Simon	Physiologie animale
	BLOCH Daniel	Electrotechnique- I.P.
	MARTIN-BOYER Michel	Chimie-C.S.U.Chambery
	BRUGEL Lucien	Energétique- I.U.T."A"
	BOUVARD Maurice	Mécanique des fluides
	ARMAND Yves	Chimie-I.U.T."A"
	KUHN Gérard	Physique- I.U.T."A"
	RICHARD Lucien	Botanique
	PELMONT Jean	Physiologie animale
	BOUSSARD Jean Claude	Mathématiques appliquées.I.P.
	MOREAU René	Hydraulique. I.P.
	GERMAIN Jean Pierre	Mécanique
	JOLY Jean René	Mathématiques pures
Mle	PIERY Yvette	Biologie animale
MM	CONTE René	Physique I.U.T."A"
	PEFFEN René	Métallurgie I.U.T. "A"
	LE JUNTER Noël	Electronique I.U.T. "A"
	VIALON Pierre	Géologie
	BENZAKEN Claude	Mathématiques appliquées
	MAYNARD Roger	Physique
	BLIMAN Samuel	Electronique E.I.E.
	DUSSAUD René	Mathématiques C.S.U. Chambery
	BELORIZKY Elie	Physique
Mme	LAJZEROWICZ Jeannine	Physique
M.	JULLIEN Pierre	Mathématiques pures
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM	LEROY Philippe	Mathématiques I.U.T. "A"
	BRODEAU François	Mathématiques I.U.T. "B"

MM	ROMIER Guy	Mathématiques I.U.T. "B"
	NEGRE Robert	Mécanique I.U.T. "A"
	BEGUIN Claude	Chimie organique
	BUISSON Roger	Physique I.U.T."A"
	IVANES Marcel	Electricité I.U.T. "A"
	GIDON Maurice	Géologie
	COHEN ADDAD	Spectrométrie physique
	VAN CUTSEM Bernard	Mathématiques appliquées
	GRIFFITHS Michael	Informatique
	MACHE Regis	Physiologie végétale
	GENSAC Pierre	Sciences biologiques
	JOUBERT Jean Claude	Physique du solide I.P
	VEILLON Gerard	Mathématiques appliquées I.P.
	MARECHAL Jean	Physique mécanique I.U.T. "A"
	PECCOUD François	Analyse I.U.T. "B"
	CHIAVERINA Jean	Biologie appliquée E.F.P.

MAITRES DE CONFERENCES ASSOCIES

MM.	CHEEKE John	Thermodynamique
	BOUDOURIS Georges	Radioélectricité
	BENENSON Walter	Physique nucléaire
	GOLDSCHMIDT H.	Mathématiques

Je remercie Monsieur le Professeur KUNTZMANN de m'avoir fait l'honneur de présider le jury de cette thèse.

Je tiens à exprimer ma reconnaissance à Monsieur le Professeur VAUQUOIS pour la façon dont il a bien voulu m'orienter dans ce travail et m'aider de ses conseils.

Monsieur le Professeur GASTINEL et Monsieur Yvon SIRET ont bien voulu me faire l'honneur de juger ce travail ; je leur exprime mes profonds remerciements.

Je tiens à remercier également les services de dactylographie et de tirage qui ont assuré la réalisation matérielle de ce travail.

INTRODUCTION

L'étude des langues naturelles ou des langages de programmation à un niveau élevé conduit à un traitement d'arborescences. Ce traitement peut être abordé de différentes manières.

L'étude algébrique conduit à la définition de grammaires transformationnelles qui opèrent par décomposition d'une arborescence en plusieurs arborescences, les ramifications, traitent ces ramifications et par synthèse donnent une arborescence, le résultat du calcul. Cette étude a été faite par Quere et Pair [5] pour l'algèbre des ramifications et par Barbault et Descle [1] pour la définition de ces grammaires. Une autre façon d'aborder le problème est de ne traiter que d'arborescence connexe en donnant une relation entre des sous-arborescences. Ces relations n'étant pas influencées par l'arborescence qui contient cette sous-arborescence.Gladky et Mel'chuk [2] ont défini des grammaires de ce type. L'emploi de ce dernier type de grammaire sur ordinateur pose le premier problème de reconnaître effectivement une sous-arborescence et ensuite de la transformer. Par la définition d'automate équivalent à une règle de transformation et la composition de ces automates on obtient un moyen effectif de traiter les arborescences de cette deuxième manière. Cette façon d'obtenir un traitement d'arborescence est particulièrement utile dans la manipulation des formules algébriques sur ordinateur. Un système simple de traitement construit dans le seul but de montrer cette application possible est donné en exemple.

- CHAPITRE 1 - Rappels

Automates et Arborescences

- CHAPITRE 2 - Transducteurs

Définition et propriétés

- CHAPITRE 3 - Reconnaissance

Définition des schémas de figure et leurs reconnaissances

- CHAPITRE 4 - Transformation

Simulation des transformations élémentaires par des transducteurs

- CHAPITRE 5 - Une application

Manipulation de formules algébriques sur ordinateur.

C H A P I T R E I

RAPPELS SUR LES AUTOMATES ET SUR LES ARBORESCENCES

d) Arborescence d'état fini

On nomme arborescence d'état fini une arborescence dont l'expression homogène peut être générée par un automate d'état fini.
Une telle contrainte implique que la profondeur de l'arborescence doit être finie (i.e. la longeuur du plus long chemin joignant une famille à la racine).

III - Δ - GRAMMAIRES [2]

Gladky et Mel'cuk utilisent des arborescences où les couples de connections sont étiquetés. On peut se ramener au cas où les noeuds sont étiquetés de la façon suivante :

- en donnant un symbole à la racine
- en étiquetant chaque successeur d'un sommet par l'étiquette du couple de connection qui joint ce successeur au sommet.

Un sommet qui n'est pas étiqueté peut avoir n'importe quelle étiquette.

a) Composition d'arborescences

Soit $t_0, t_1, \ldots, t_n$ des arborescences quelconques et $r_1, \ldots r_n$ n noeuds de t_0 non nécessairement différents deux à deux. On définit alors la composition de t_0 avec $t_1 \ldots t_n$ de la façon suivante :

Le résultat de la composition de t_0 avec $t_1, \ldots t_n$ est une arborescence isomorphe à l'arborescence obtenue à partir de t_0 en identifiant les racines de $t_1, \ldots, t_n$ respectivement aux noeuds $r_1, \ldots, r_n$ de t_0.

Notation :

$$T = C\ (t_0\ ;\ r_1, \ldots, r_n \mid t_1 \ldots t_n)$$

b) Définition d'une sous-arborescence

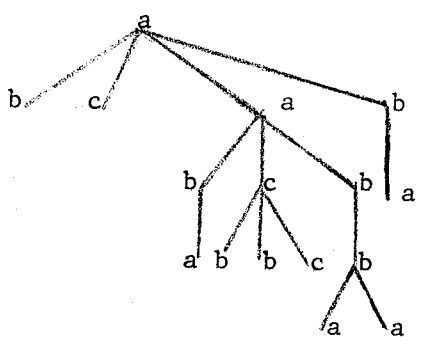
t est une sous-arborescence de T si T est de la forme

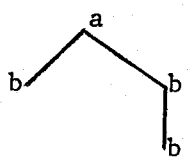
$$T = C\ (T_0, r_0 \mid C\ (t\ ; r_1, \ldots, r_n \mid T_1, \ldots, T_n))$$

Où r_0 est une racine de T_0 et $r_1 \ldots r_n$ une énumération répétitive des noeuds de t.

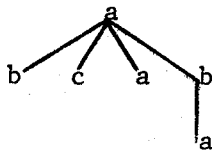
Exemple :

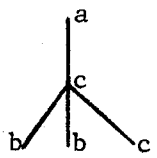
Soit l'arborescence A =



Alors t =  est une sous-arborescence de A

On a : $A = C\ (T_0\ ;\ a \mid C\ (t\ ;\ b\ a\ b \mid T_1\ T_2\ T_3))$

T_0 = 

T_1 =  T_2 =  T_3 = 

On peut numéroter les sommets pour ne pas avoir d'ambiguité dans la composition.

c) Transformation élémentaire

Une transformation élémentaire (ET) est un triplet ordonné (t_1, t_2, f) où :

- t_1 et t_2 sont des arborescences
- f est une fonction de l'ensemble de tous les noeuds de t_1 dans l'ensemble des noeuds de t_2.

T' est dit être le résultat de T par l'application de la transformation si T et T' peuvent être mis sous la forme

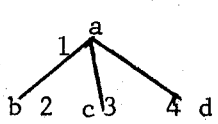
$$T = C\ (T_0\ ;\ r_0 \mid C\ (t_1\ ;\ r_1, \ldots, r_n \mid T_1 \ldots T_n))$$

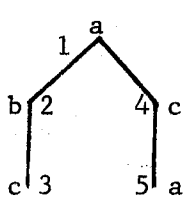
$$T' = C\ (T_0\ ;\ r_0 \mid C\ (t_2, f(r_1), \ldots, f(r_n) \mid T_1 \ldots T_n))$$

Donc T' est le résultat de T si

- t_1 est une sous-arborescence de T
- t_2 est une sous-arborescence de T'
- La racine de T_i dans T' est l'image par f de la racine de T_i dans T.

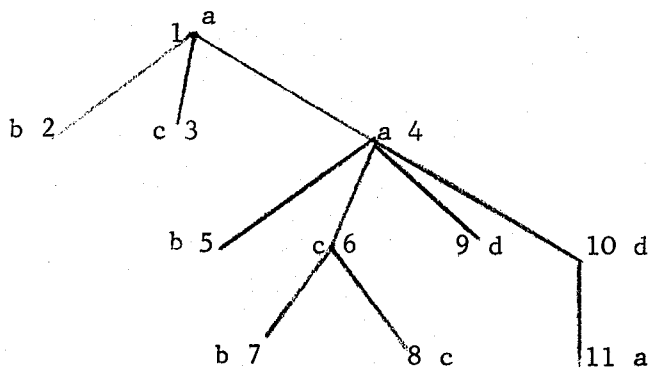
Exemple

t_1 = 

t_2 = 

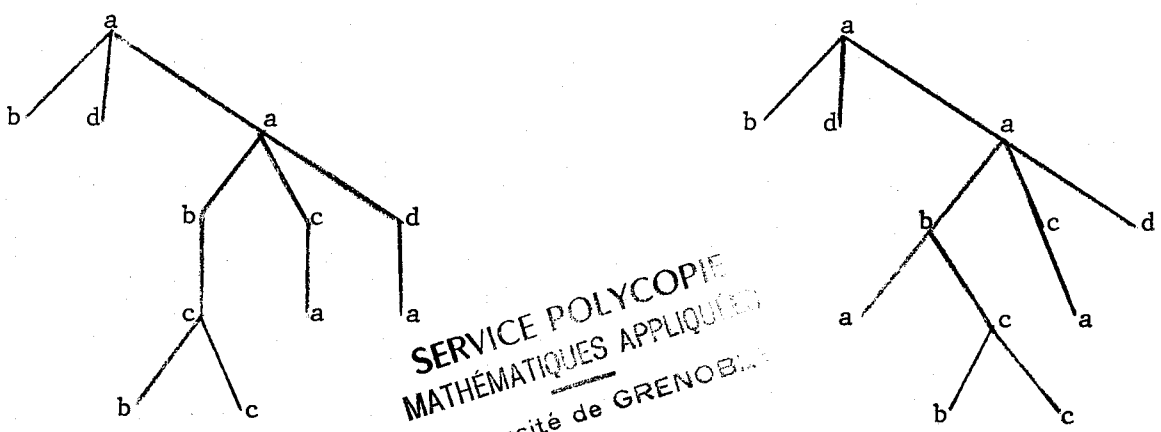
f(1) = 1
f(2) = 3
f(3) = 3
f(4) = 2

Soit l'arborescence :



Elle contient deux sous-arborescences

On obtient par application de la ET aux deux sous-arborescences les arborescences :



d) Δ - grammaire

Une Δ-grammaire est une paire ordonné $\Gamma = \langle V, \Pi \rangle$ où V est un ensemble fini de symboles (étiquettes des noeuds) et Π un ensemble fini de ET, les règles de la grammaire.

e) Transformations élémentaires spéciales

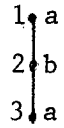
Une transformation élémentaire spéciale (SET) est une transformation élémentaire des trois types suivants :

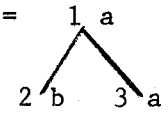
- Eclatement d'un noeud

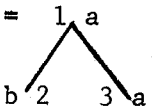
t_1 = 1 a $\qquad$ t_2 = 

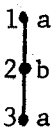
$\qquad$ f(1) = 1 ou 2

- Transfert d'un noeud

t_1 = 

$\qquad$ t_2 = 

t_1 = 

$\qquad$ t_2 = 

Dans les deux cas $f(1) = 1 \quad f(2) = 2 \quad f(3) = 3$

- Absorption d'un noeud

t_1 = 

$\qquad$ t_2 = a $\qquad$ f(1) = 1
f(2) = 1

- Simulation

Une ET peut être simulée par un ensemble fini M de ET, s'il existe une séquence finie $m_1 \ldots m_n$ d'éléments de M, tel que pour une arborescence quelconque T et l'arborescence déduite T' de T par la ET, l'arbre T' peut être déduit de T par application en séquences des ET $m_1 \ldots m_n$ à T et ses conséquents.

- Propriété

Théorème 1 :

Toute transformation élémentaire peut être simulée par un ensemble fini de SET.

IV - ARBORESCENCES DOUBLEMENT ORIENTEES

a) Définition de l'ordre $\succ_G$:

On traite en général d'arborescences doublement orientées obtenues en munissant l'ensemble des descendants directs d'un point d'un ordre total : $\succ_G$

Si a et b sont deux descendants directs d'un point de l'arborescence : $a \underset{G}{\succ} b$ si et seulement si a est à gauche de b.

Cet ordre peut être étendu à un ordre partiel sur l'ensemble des points de l'arborescence de la façon suivante :

- Deux éléments a et b sont comparables si et seulement si l'un n'est pas un descendant de l'autre.

- Soit a et b deux points comparables, alors il existe deux points a' et b' possédant les propriétés

  . a est un descendant de a'

  . b est un descendant de b'

  . a' et b' sont les descendants directs d'un même point.

- Si a et b sont comparables :

$a \underset{G}{\succ} b$ si et seulement si $a' \underset{G}{\succ} b'$

b) Définition

Soient deux points a et b d'une arborescence. On définit alors un troisième point Anc (a,b) par :

- Anc (a,b) est la racine de la sous-arborescence intersection de toutes les sous-arborescences contenant a et b.

Dans le cas où Anc (a,b) n'est ni a ni b, on dit que Anc (a,b) est simple.

Si Anc (a,b) est simple, il existe deux points notés $\mathcal{F}_a$ (a) et $\mathcal{F}_a$ (b) définis par :

$\mathcal{F}_a$ (a) est la racine d'une sous-arborescence contenant a

$\mathcal{F}_a$ (b) est la racine d'une sous-arborescence contenant b

$\mathcal{F}_a$ (a) et $\mathcal{F}_a$ (b) sont des descendants directs de Anc (a,b)

c) Ordre sur l'arborescence

On peut alors étendre l'ordre $\underset{G}{\succ}$ à un ordre partiel sur l'arborescence par :

- Deux points sont comparables si et seulement si Anc (a,b) est simple

- Si deux points sont comparables

$$a \underset{G}{\succ} b \Longleftrightarrow \mathcal{F}_a(a) \underset{G}{\succ} \mathcal{F}_a(b)$$

d) Sous-arborescence orientée

Une sous-arborescence t de T est une sous-arborescence orientée si

$$T = C\ (T_0\ ;\ r_0 \mid C\ (t\ ;\ r_1, \ldots, r_n \mid T_1 \ldots T_n))$$

et si on a la propriété :

Si deux points a,b de t sont comparables, alors ils le sont dans T et ont le même ordre.

e) Transformation élémentaire orientée

Soit une ET (t_1, t_2, f)

T' est le résultat de l'application de la ET orientée (t_1, t_2, f) à T si et seulement si :

- t_1 et t_2 sont des sous-arborescences orientées de T et T'.

Soient deux points comparables a et b dans T, leurs images dans T' σ_a et σ_b ont la propriété

* Si $\mathcal{F}_a(\sigma_a)$ et $\mathcal{F}_a(\sigma_b)$ n'appartiennent pas à t_2 alors

$a \geq_G b \iff \mathcal{F}(\sigma_a) \geq_G \mathcal{F}(\sigma_b)$

* Si $\mathcal{F}_a(\sigma_a) \in t_2$ et $\mathcal{F}_a(\sigma_b) \notin t_2$ alors

$\mathcal{F}_a(\sigma_a) \geq_G \mathcal{F}_a(\sigma_b)$

* Si ils ne sont pas comparables l'un des deux au moins (σ_a ou σ_b) appartient à t_2.

f) Transformations élémentaires spéciales :

Les transformations élémentaires spéciales deviennent alors :

- éclatement d'un noeud

$t_1 =$

$t_2 =$ 

$f(1) = 1$ ou 2

- Absorption d'un noeud

$t_1 =$ 

$t_2 = 1$. a

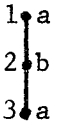
$f(1) = 1$

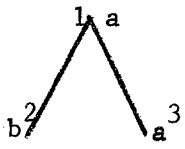
$f(2) = 1$.

- Transfert d'un noeud

Il y a quatre transferts :

Transfert à droite

$t_1 =$ 

$t_2 =$ 

I - AUTOMATES

a) Machine de Turing

- Définition

L'automate de base est la machine de Turing. Elle est définie par [1] :

$T = (K, \Sigma, \Gamma, \delta, q_0, F)$ où :

K est un ensemble fini d'états

Γ est un ensemble fini de symboles : les symboles de bandes. Un de ces symboles est particulier c'est le blanc qui est noté B.

Σ est un sous-ensemble de Γ ne contenant pas B : l'ensemble des symboles d'entrée

δ est une fonction de transition qui est une application :

$$K \times \Gamma \longrightarrow K \times (\Gamma - \{B\}) \times \{L, R\}$$

q_0 est un élément distingué de K, l'état de départ

F est un sous-ensemble de K, les états finaux.

Remarque :

Le fait que T_m ne peut imprimer le symbole B n'est pas une restriction, il suffit d'avoir dans Γ un symbole B' qui joue le même rôle. Cette notation est plus commode pour définir la configuration de l'automate.

- Configuration

La configuration d'une machine de Turing T est donnée par le triplet (q, α, i) où :

q est l'état courant de T, $q \in K$

α est une chaîne sur $(\Gamma - \{B\})^*$, la partie de bande imprimée

i est un entier qui définit la position de la tête de lecture de la machine T sur la chaîne α, par le nombre de symboles qui existe entre l'extrémité gauche de α et celle-ci.

- Mouvement de T

La définition des mouvements de T s'opère en définissant une relation $\vdash_T$ entre les configurations.

Soit $(q, A_1 A_2 \ldots A_n, i)$ une configuration

Si $\delta(q, A_i) = (p, A, R)$ et $1 \leqslant i \leqslant n$

Alors $(q, A_1 \ldots A_n, i) \vdash_T (p, A_1 \ldots A_{i-1} A A_{i+1} \ldots A_n, i+1)$

Si $\delta(q, A_i) = (p, A, L)$ et $2 \leqslant i \leqslant n$

Alors $(q, A_1, A_2, \ldots A_{n-1} i) \vdash_T (p, A_1 \ldots A_{i-1} A A_{i+1} \ldots A_n, i-1)$

Cas des extrémités

T ne peut "déborder" à gauche de α . Seul le cas où T déborde à droite induit une relation.

Si $\delta(q, B) = (p, A, R)$

Alors $(q, A_1 \ldots A_n, n+1) \vdash_T (p, A_1 \ldots A_n A, n+2)$

Si $\delta(q, B) = (p, A, L)$

Alors $(q, A_1 \ldots A_n, n+1) \vdash_T (p, A_1 \ldots A_n A, n)$

Si deux configurations sont en relations alors on dit que la deuxième résulte de la première par un mouvement de la machine de Turing T.

Si une configuration résulte d'une autre par un nombre fini de mouvements alors ces deux configurations sont dans la relation $\vdash_T^*$.

Configuration d'arrêt

Une configuration est une configuration d'arrêt notée $[q, \alpha, i]$ si il n'existe pas de configuration (p, β, j) de T tel que $(q, \alpha, i) \vdash_T (p, \beta, j)$

Si (q, α, i) est une configuration d'arrêt, alors $(q, \alpha, i) \vdash_T [q, \alpha, i]$

- Langage accepté par une machine de Turing

Le langage accepté par une telle machine est un sous-ensemble de Σ^* défini par

$$L = \left\{ W \mid W \in \Sigma^* \wedge (q_0, W, 1) \vdash_T^* [q, \alpha, i] \ , \ q \in F, \ \alpha \in \Gamma^* \ , \ i \in N \right\}$$

c'est-à-dire que T reconnaît un langage L si T s'arrête dans un état final sur tous les mots du langage.
Autrement, il est possible que T ne s'arrête pas.

b) Automate à pile

- Définition

Une première restriction importante à la machine de Turing est l'automate linéaire borné. Ce n'est pas une restriction suffisante pour pouvoir l'employer couramment sur ordinateur en gardant une certaine efficacité. Une restriction intéressante est celle de l'automate à pile qui sera employée par la suite.

Un automate à pile est défini par [3] :

$P = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

où K est un ensemble fini d'états

Σ est un ensemble fini, l'alphabet d'entrée

Γ est un ensemble fini, l'alphabet de pile

δ est une application de $K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ dans un sous-ensemble fini de $K \times \Gamma^*$

q_0 est un élément distingué de K, l'état initial

Z_0 est un élément distingué de Γ le symbole apparaissant en pile au départ

F est un sous-ensemble de K, les états finaux

- Configuration

Une configuration de l'automate est un doublet (p, β) où p est l'état courant de l'automate et β une chaîne sur Γ^*, la chaîne de pile.

Relation entre configuration :

Cette relation est définie sur $\Sigma \times C$ où C est l'ensemble des configurations.

Soit $\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2) \cdots (p_n, \gamma_n)\}$

Alors $\forall_i \ 1 \leqslant i \leqslant n$:

$$a\,(q, Z\beta) \vdash_p (p_i, \gamma_i \beta)$$

Si $\delta(q, \varepsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2) \cdots (p_n, \gamma_n)\}$

$\forall_a \in \Sigma$

$$\forall_{i \ 1 \leqslant i \leqslant n} \quad a\,(q, Z\beta) \vdash_p (p_i, \gamma_i \beta)$$

Dans ce dernier cas, l'automate n'avance pas. On peut définir alors une relation sur $\Sigma^* \times C$:

Soit $a_1 \cdots a_n$ une chaîne sur $(\Sigma \cup \{\varepsilon\})^*$ et deux suites $q_1 \cdots q_{n+1}$, $\gamma_1 \cdots \gamma_{n+1}$ d'états et de chaînes de piles.

Si pour tout i $1 \leqslant i \leqslant n$ on a $a_i\,(q_i, \gamma_i) \vdash_p (q_{i+1}, \gamma_{i+1})$

Alors $a_1 \cdots a_n\,(q_i, \gamma_i) \vdash_p^* (q_{n+1}, \gamma_{n+1})$

Soit $b_{2_1} \cdots b_{n_1}$ défini par $a_1 \cdots a_n = b_{1_1} \cdots b_{1_\ell}\, b_{2_1} \cdots b_{i_j} \cdots b_{n_k}$

$$\text{ou } b_{i_j} = \begin{cases} \varepsilon \text{ si } i = 1 \vee (i \geqslant 2 \wedge j \neq 1) \\ \neq \varepsilon \text{ sinon} \end{cases}$$

Alors $b_{2_1} \ldots b_{n_1} (q_1, \gamma_1) \overset{*}{\underset{p}{\vdash}} (q_{n+1}, \gamma_{n+1})$

Par convention $\epsilon (q_1, \gamma_1) \overset{*}{\underset{p}{\vdash}} (q_1, \gamma_1)$

- Langage accepté

Le langage accepté par un tel automate est le sous-ensemble de Σ^* défini par

$$L = \{ W \mid W \in \Sigma^* \wedge W (q_0, Z_0) \vdash (q_1 \gamma) , \gamma \in \Gamma^* , q \in F \}$$

II - ARBORESCENCES

a) Définition

Une arborescence est un arbre dont on a choisit un élément appelé racine.

Un arbre est un réseau d'articulations tel que :

1 - il contient au moins deux articulations

2 - il est connexe

3 - il perd sa connexité si on lui enlève un de ses couples de connection.

A cette définition nous ajouterons :

- le réseau vide, c'est-à-dire qui ne contient pas d'articulation
- les réseaux ne contenant qu'une articulation.

b) Propriétés

Si à chaque point du réseau on affecte une étiquette, symbole d'un vocabulaire fini V, on obtient une arborescence étiquetée.

Soit L_1 le langage sur $V \cup \{(,)\}$ (les parenthèses n'appartenant pas à V) défini par $L = (V_T, V_N, S, P)$

Où $V_T = V \cup \{(,)\}$

$V_N = \{S, T, A\}$

P : $S \longrightarrow A\ (T)$

$T \longrightarrow TT$

$T \longrightarrow A$

$T \longrightarrow A\ (T)$

$A \longrightarrow a \qquad \forall a \in V$

$S \longrightarrow a \qquad \forall a \in V \cup \{\wedge\}$

Alors il existe une bijection entre L et l'ensemble des arborescences étiquetées sur V. Le mot de L associé par cette bijection à une arborescence donnée s'appelle expression parenthétique de l'arborescence.

c) Représentation homogène

Dans la représentation linéaire donnée, tous les symboles de V_T ne jouent pas le même rôle suivant qu'ils précèdent ou non une (.
On emploiera une représentation équivalente à celle donnée précédemment qui ne fait intervenir qu'une parenthèse).

On obtient cette représentation en deux temps

Soit L_1 défini par G_1

$G_1 = (V_N, V \cup \{(,)\}, P, S)$

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

P : $S \longrightarrow A\,(B)$ $B \longrightarrow BB$ $B \longrightarrow A\,(B)$ $B \longrightarrow A\,(\,)$
$S \longrightarrow \wedge$, $S \longrightarrow A\,(\,)$ $A \longrightarrow a$ $\forall\, a \in V$

Comme chaque étiquette précède une (on peut alors sans équivoque supprimer la (et on obtient le langage L' défini par G'

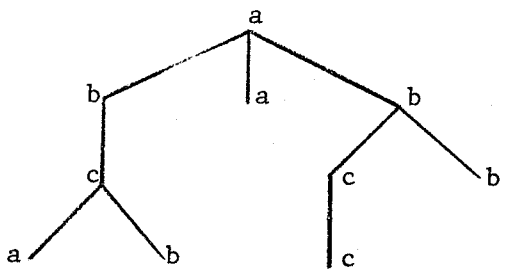
$G' = (V_N, V \cup \{)\}, P, S)$

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

P : $S \longrightarrow A\,B)$ $B \longrightarrow BB$ $B \longrightarrow AB)$ $B \longrightarrow A)$
$S \longrightarrow \wedge$ $S \longrightarrow A)$ $A \longrightarrow a$ $\forall\, a \in V$

Exemple :

Soit le graphe

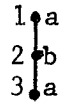


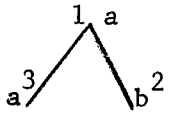
Son expression parenthétique est a(b(c(ab))ab(c(c)b))

Son expression homogène abca)b)))a)bcc))b)))

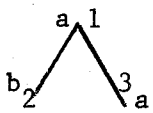
Son expression dans L_1 est a(b(c(a()b()))a()b(c(c())b()))

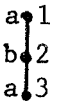
Transfert à gauche

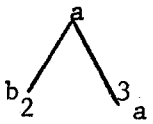
t_1 = 

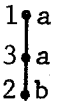
t_2 = 

Transfert inverse

t_1 = 

t_2 = 

t_1 = 

t_2 = 

Dans tous les cas $f(1) = 1,\ f(2) = 2,\ f(3) = 3$

Par la suite nous n'emploierons que ces transformations élémentaires spéciales orientées ainsi que leurs composés.
L'ordre défini de cette façon se retrouve sur les arborescences écrites en représentation linéaire par l'ordre dans lequel les sommets sont lusdans la chaîne.

C H A P I T R E II

LES TRANSDUCTEURS ET LEURS COMPOSES

I - TRANSDUCTEURS

- Définition

Un transducteur T est défini par $T = (V, V_E, V_S, Q, \delta, Q_0, F)$

où : - V est le vocabulaire général dont un des éléments est le blanc noté B

- V_E le vocabulaire d'entrée ($V_E \subset V$) tel que $B \notin V_E$
- V_S le vocabulaire de sortie n'incluant pas le blanc
- Q un ensemble fini d'état
- Q_0 un sous-ensemble de Q les états initiaux
- F un sous-ensemble de Q les états finaux
- δ une fonction de transition quelconque de la forme:

$$Q \times V \longrightarrow Q \times (V - \{B\}) \times \{D, G\} \times V_S^* \times \{D', G'\}$$

- Configuration

La configuration d'un transducteur est donnée par le quadruplet (q, A, A', i) où :

q est l'état courant de T $\quad q \in Q$

A une chaîne sur $(V - \{B\})^*$ la chaîne sur la bande d'entrée

A' une chaîne sur V_S^* la chaîne sur la bande de sortie

i un entier qui définit la position de la tête de lecture par le nombre de symboles qui existent entre l'extrémité gauche de A et celle-ci.

- Mouvement de T

Les mouvements de T sont identiques au mouvement d'une machine de Turing avec en plus à chaque pas l'adjonction d'un mot en sortie.

$$(q, A_1 \cdots A_{i-1} A_i A_{i+1} \cdots A_n, \Gamma, i) \vdash_T (p, A_1 \cdots A_{i-1} A A_{i+1} \cdots A_n, \Gamma \gamma, i+1)$$

si $\delta(q, A_i) = (p, A, D, \gamma, D') \qquad \gamma \in V_S^*$

$$(q, A_1 \cdots A_{i-1} A_i A_{i+1} \cdots A_n, \Gamma, i) \vdash_T (p, A_1 \cdots A_{i-1} A A_{i+1} \cdots A_n, \gamma \Gamma, i+1)$$

si $\delta(q, A_i) = (p, A, D, \gamma, G')$

$$(q, A_1 \cdots A_{i-1} A_i A_{i+1} \cdots A_n, \Gamma, i) \vdash_T (p, A_1 \cdots A_{i-1} A A_{i+1} \cdots A_n, \Gamma \gamma, i-1)$$

si $\delta(q, A_1) = (p, A, G, D')$

$(q, A_1, \ldots A_{i-1}\ A_i A_{i+1} \ldots A_n, \Gamma, i) \vdash_T (p, A_1 \ldots A_{i-1}\ A\ A_{i+1} \ldots A_n, \gamma\Gamma, i-1)$

si $\delta(q, A_i) = (p, A, G, \gamma, G')$

La tête de lecture du transducteur ne peut aller à gauche de la bande d'entrée et, si elle est à droite, elle remplace tous les blancs qu'elle lit.

si $\delta(q, B) = (p, A, D, \gamma, M)$ $\quad M \in \{G', D'\}$

$(q, A_1 \ldots A_n, \Gamma, n+1) \vdash_T (p, A_1 \ldots A_n A, \Gamma', n+2)$

avec $\Gamma' = \gamma\Gamma$ si $M' = D'$

$\Gamma' = \Gamma\gamma$ si $M = G'$

si $\delta(q, B) = (p, A, G\ \gamma, M)$

$(q, A_1 \ldots A_n, \Gamma, n+1) \vdash_T (p, A_1 \ldots A_n A, \Gamma', n+2)$

avec $\Gamma' = \Gamma\gamma$ si $M = D'$

$\Gamma' = \gamma\Gamma$ si $M = G'$

- Configuration d'arrêt :

Une configuration est une configuration d'arrêt $[q, \alpha, \Gamma, i]$ s'il n'existe pas de configuration de T $(q', \alpha', \Gamma', j)$ tel que

$(q, \alpha, \Gamma, i) \vdash_T (q', \alpha', \Gamma', j)$

si (q, α, Γ, i) est une configuration d'arrêt, alors

$(q, \alpha, \Gamma, i) \vdash_T [q, \alpha, \Gamma, j]$

Si deux configurations sont en relation, alors on dit que la deuxième résulte de la première par un mouvement du transducteur. Si une configuration résulte d'une autre par un nombre fini de mouvements alors les deux configurations sont dans la relation $\vdash_T^*$

- Mot transcrit par le transducteur

Soit $W \in V_E^*$

si $(q_0, W, \Lambda, 1) \vdash_T^* [q, \alpha, W', i]$ $\quad q_0 \in Q_0$

$q \in F$

$\alpha \in (V - \{B\})^*$

$W' \in V_S^*$

$i \in N$

Alors on dit que W est transcri en W' par le transducteur T.

Si W est transcri en W' par le transducteur T, on note $W \overset{*}{\underset{T}{\vdash}} W'$

- Langage transcri :

Soit $L \subset V_E^*$, alors le langage transcri T (L) est défini par :

$$T(L) = \{ u \mid (u \in V_S^*) \wedge (\exists W \in L : W \overset{*}{\underset{T}{\vdash}} u) \}$$

Exemple : Soit le transducteur défini par :

$$T = (\{0, 1, B, X, Y, Z\}, \{0, 1\}, \{0, 1\}, \{q_0, q_1, \ldots, q_7\}, \delta, \{q_0\}, \{q_5\})$$

avec δ

$\delta(q_0, 0) = (q_1, X, D, 1, D')$

$\begin{cases} \delta(q_1, 0) = (q_1, 0, D, 1, D') \\ \delta(q_1, 1) = (q_2, Y, G, 0, D') \end{cases}$

$\begin{cases} \delta(q_2, Y) = (q_2, Y, D, \wedge, D') \\ \delta(q_2, X) = (q_3, X, D, 0, G') \\ \delta(q_2, 0) = (q_4, 0, G, \wedge, D') \end{cases}$

$\delta(q_6, 0) = (q_7, X, D, 0, D')$

$\begin{cases} \delta(q_3, Y) = (q_3, Y, D, \wedge, D') \\ \delta(q_3, B) = (q_5, Y, D, \wedge, D') \end{cases}$

$\begin{cases} \delta(q_4, 0) = (q_4, 0, G, \wedge, D') \\ \delta(q_4, X) = (q_6, X, D, 0, G') \end{cases}$

$\begin{cases} \delta(q_7, 0) = (q_7, 0, D, \wedge, D') \\ \delta(q_7, 1) = (q_2, Y, G, \wedge, D') \\ \delta(q_7, Y) = (q_7, Y, D, \wedge, D') \end{cases}$

Alors $T(\{0^n 1^n \mid n \geq 1\}) = \{0^n 1^n 0^n \mid n \geq 1\}$

On a d'ailleurs ici $T(\{0, 1\}^*) = \{0^n 1^n 0^n \mid n \geq 1\}$

- Propriétés :

- L'adjonction d'une bande de sortie ne joue aucun rôle dans la marche de l'automate. Donc les propriétés d'arrêt sont les mêmes que pour la machine de Turing.

- Un langage transcri ne garde pas toujours son type (cf. exp.)

- Restriction :

On peut particulariser certains transducteurs

a) transducteur simple

Un transducteur est simple si le mouvement de sortie est constant.

b) Transducteur ordinaire

Un transducteur est ordinaire s'il est simple et si Q_0 ne contient qu'un seul élément.

c) Transducteur régulier

Un transducteur régulier est défini par :

$T_R = (V_E, V_S, Q, \delta, {}_{Q_0}, F)$

où V_E, V_S, Q, Q_0, F ont la même signification que dans le transducteur général et où δ est une application définie par

$$\delta : Q \times V_E \longrightarrow Q \times V_S^* \times \{D', G'\}$$

Le transducteur régulier lit alors la chaîne d'entrée une fois caractère par caractère, change d'état et imprime sur la sortie un mot de V_S^*. C'est un automate régulier auquel on a ajouté une bande de sortie.

d) Transducteur à pile

Un transducteur à pile est défini par :

$T_p = (V_E, V_S, V_p, Q, \delta, Q_0, F)$

où V_E, V_S, Q, Q_0, F ont la même signification que dans le transducteur général et où :

V_p : ensemble fini, vocabulaire de pile

δ une application définie par

$$Q \times V_E \times V_p \longrightarrow Q \times V_p^* \times V_S^* \times \{D', G'\}$$

Le transducteur à pile est un automate à pile dans lequel on a ajouté une bande de sortie qui n'influe pas sa marche.

e) Transducteur à tables

Un transducteur à table est défini par :

$T = (V, V_E, V_S, Q, \delta, Q_0, F, T)$

où V, V_E, V_S, Q, Q_0, F ont les mêmes significations que dans un transducteur ordinaire.

T : Un mot sur un vocabulaire fini tel que sa longueur soit exactement C (V). C (Q) (C (A) cardinal de A)

δ une application définie par

Soit f et h deux applications de Q $\longrightarrow$ N et V $\longrightarrow$ N qui, à chaque élément de Q et de V, associent un nombre inférieur au cardinal de Q ou de V (C (Q) et C (V)).

Si $T = t_1 \ldots t_n$

$$\delta : Q \times V \times t \longrightarrow Q \times (V - \{B\}) \times \{D, G\} \times V_S^* \times \{D', G\}$$
$$C(Q)\ f(Q) + h(V)$$

A chaque pas le transducteur consulte la table qui détermine son état suivant.

On peut avoir évidemment des transducteurs à tables, à pile et réguliers.

- Propriété :

Un transducteur à table est équivalent à un transducteur classique quel qu'il soit.

Il suffit de remplacer les états par les couples (q, T) $\forall$ q $\in$ Q . Les transducteurs seront utiles dans la composition ou un transducteur précédant un autre pourra alors modifier la table du suivant et, par conséquent, modifier à l'avance son comportement.

II - COMPOSITION DE TRANSDUCTEURS

Soit T un transducteur et W un mot de V_E^* ; alors à partir de W et de l'état de départ q_0, T fournit un mot μ de V_S^* s'il s'arrête dans un état final p. On note ce fait par T $(q_0, W) = (\mu, p)$.

Si T, à partir de q_0 et de W, ne fournit aucun résultat, c'est-à-dire soit qu'il ne s'arrête pas, soit qu'il s'arrête dans un état non final, on note T $(q_0, W) = (\mu, n)$.

Un transducteur étant donné, il fournit un mot μ à partir d'un mot W. Un deuxième transducteur peut alors transcrire ce mot μ en un autre mot V. C'est la composition qui est définie par :

Soit $T_0, \ldots, T_n$ n transducteurs ordonnés.

Pour tout i : $T_i = (V_i, V_{E_i}, V_{S_i}, Q_i, \delta_i, Q_{0_i}, F_i)$

On peut alors définir la composition $C_{T_1 \cdots T_n}$ des transducteurs si et seulement si

$$\left.\begin{array}{l} - \; V_{S_i} = V_{E_{i+1}} \\ - \; F_i \cap Q_{0_{i+1}} \neq \emptyset \\ - \; (Q_i - F_i) \cap Q_{0_{i+1}} = \emptyset \end{array}\right\} \quad \forall_i \quad 1 \leqslant i \leqslant n$$

Un mot W est transcri en un mot μ par le transducteur $C_{T_1 \cdots T_n}$ si et seulement si

$T_1 (q_0, W) = (W_2, P_2)$ $\quad P_2 \in Q_{0_2}$

$T_i (p_i, W_i) = (W_{i+1}, P_{i+1})$ $\quad P_{i+1} \in Q_{0_{i+1}}$ $\quad \forall i, 1 < i < n$

$T_n (P_n, W_n) = (\mu, p)$ $\quad P \in F_n$

Le transducteur ne fournit aucun résultat dans les autres cas.

a) Propriété immédiate.

1) Pour tout transducteur composé $C_{T_1 \ldots T_n}$ il existe un transducteur équivalent.

- Cette propriété est immédiate avec les propriétés suivantes des transducteurs

- Un transducteur ayant plusieurs bandes d'entrée est équivalent à un transducteur ayant une bande

- Un transducteur composé est équivalent à un transducteur ayant deux bandes en entrée

2) Composition de transducteurs et d'un automate.

Si le dernier transducteur T_n ne fournit jamais de mot, c'est-à-dire que toutes ses règles sont de la forme.

$$\delta(q, A) = (p, A', M, \wedge, M')$$

On a alors la composition de (n-1) transducteurs et d'un automate, et qui forme un automate. Cet automate accepte un mot s'il s'arrête en T_n dans un état final.

b) Ordre d'un transducteur composé

Soit $C_{T_1 \ldots T_n}$ un transducteur composé, son ordre est le nombre n de transducteur entrant dans la composition.

c) Transducteur cyclique

Un transducteur cyclique est un transducteur composé qui est tel :

$$\exists i, j : (F_i - Q_{0_{i+1}}) \wedge Q_{0_j} \neq \emptyset \quad \text{avec } i < j$$

Alors dans ce cas si l'automate i s'arrête dans un état de Q_{0_j} l'automate qui analyse le mot de la sortie de T_i est le transducteur T_j.

Un mot W est transcrit par un automate cyclique à un mot μ s'il existe une suite finie $T_1 T_{i_1} \ldots T_{i_m} T_n$ tel que :

$$T_1 (q_0, W) = (W_{i_1}, P_{i_1}) \qquad P_{i_1} \in Q_{0_{i_1}}$$

$$T_{i_j} (P_{i_j}, W_{i_j}) = (W_{i_{j+1}}, P_{i_{j+1}}) \qquad P_{i_{j+1}} \in Q_{0_{j+1}} \qquad \forall\, j\ 1 < j < m$$

$$T_n (P_{i_m}, W_{i_m}) = (\mu, P) \qquad P \in F_n$$

Un cas particulier est celui d'un transducteur cyclique d'ordre 2 $C_{T_1 T_2}$ dans lequel $T_1 = T_2$. On a dans ce cas la bande d'entrée de T_1 qui est identique à la bande de sortie de T_2 et inversement. Ce transducteur est noté S_T ; il est régulier ou à pile, si T est régulier ou à pile.

d) Propriétés

Théorème 1 :

Pour toute machine de Turing T, il existe un transducteur cyclique régulier $C_{T_1 S_T'}$ qui lui est équivalent.

Ce transducteur n'est cyclique que par S_T' qui effectue à chaque cycle un mouvement de la machine de Turing.

Démonstration :

Soit $T = (V, V_E, Q, \delta, q_0, F)$ un machine de Turing

Alors on a :

$$T_1 = (V_E, V_1, Q_1, \delta_1, q_S, Q)$$

$$T' = (V_1, V_1, Q', \delta', Q, F')$$

$$V_1 = V \cup \{\varphi, B'\}$$

$$Q' = Q \cup \{(X, q), (X, q') \mid X \in V_1 - \{\varphi\}, q \in Q\} \cup \{q', q^*, q'' \mid q \in Q\}$$

$$F' = Q \cup \{q'', q \in Q\}$$

$$Q_1 = \{q_0, q_S\}$$

δ_1 :

$$\delta_1 (q_S, X) = (q_0, \varphi X, D') \qquad \forall X \in V_E$$

$$\delta_1 (q_0, X) = (q_0, X, D') \qquad \forall X \in V_E$$

$$\delta_1 (q_S, B) = (q_0, \wedge, D')$$

T_1 recopie la chaîne d'entrée et introduit φ marqueur de la tête de lecture de la machine T.

δ' :

Cette fonction effectue à chaque fois un mouvement de T avec φ comme marqueur de la tête de lecture.

$$\delta'(q, X) = ((X, q), \wedge, D') \quad \forall X \in (V_1 - \{\varphi\}), \; q \in Q$$

$$\delta'((X, q), X_1) = ((X_1, q), X, D') \quad \forall X, X_1 \in V_1 - \{\varphi\}, \; q \in Q$$

$$\delta'((X, q), \varphi) = ((X, q'), \wedge, D')$$

$$\delta'(q, \varphi) = ((\wedge, q'), \wedge, D')$$

Avec ces quatre premières transitions, le transducteur se place devant le caractère à lire et n'a pas encore recopié le caractère à gauche.

On modifie légèrement δ en remplaçant partout B par B' car dans cet état δ' ne lit jamais B.

$$\delta'((X, q'), X_1) = (P', XX'_1 \varphi, D') \quad \forall \delta(q, X_1) = (P, X'_1, D) \in \delta$$

$$\delta'((X, q'), X_1) = (P', \varphi XX'_1, D') \quad \forall \delta(q, X_1) = (P, X'_1, G) \in \delta$$

δ' a alors effectué un pas du transducteur.

$$\delta'(q', X) = (q^*, X, D') \quad \forall X \in V_1 - \{B\}, \; q \in Q$$

$$\delta'(q', B) = (q^*, B', D')$$

$$\delta'(q^*, B) = (q, \wedge, D')$$

δ' recopie le reste de la chaîne d'entrée et se place dans un état final.

Arrêt du transducteur :

$$\delta'((X, q'), X_1) = (q'', XX_1, D') \quad \forall \delta(q', X_1) \text{ non défini}$$

$$\delta'(q'', X) = (q'', X, D) \quad \forall q'' \in Q', X \in V_1$$

Un mot étant sur la bande d'entrée de T, il est alors placé sur l'entrée d'un des transducteurs T' avec φ comme marqueur de la tête de lecture. Chaque pas de T est équivalent à un mouvement complet d'un transducteur. A la fin de chaque pas le mot de sortie est le même que celui qu'il y a sur la bande de lecture de T, avec φ comme marqueur de la position de la tête de lecture. Si T s'arrête dans un état q, un transducteur T' s'arrête dans l'état q".

Si $q \in F$, alors $q'' \in F'$ et le deuxième transducteur ne peut pas fonctionner car q" n'appartient pas aux états initiaux de T'.

Donc T' s'arrête dans le même état que T. Sur la bande de sortie de T' il y a une chaîne identique à celle qui est sur la bande d'entrée de T. Donc $C_{T_1 S_{T'}}$ s'arrête dans le même état que T et avec la même chaîne en sortie. Il a donc le même comportement.

- Transducteur simple

Un transducteur est dit simple

- s'il a un seul état initial et final
- il fournit à chaque pas un mot de longueur 1

Théorème 2 :

Un transducteur ordinaire est dit à droite si son mouvement de sortie est toujours D'.

La composition de deux transducteurs réguliers simples ordinaires à droite est équivalente à un transducteur régulier simple ordinaire à droite.

Démonstration :

On regroupe les deux fonctionnements

Soit

$T_1 = (V_E, V_{S_1}, Q_1, \delta_1, q_{0_1}, q_{0_2})$

$T_2 = (V_{S_1}, V_{S_2}, Q_2, \delta_2, q_{0_2}, q_{F_2})$

Les composants du transducteur composé $C_{T_1 T_2}$

On a alors le transducteur équivalent

$T = (V_E, V_{S_2}, Q, q_0, F_2)$

$Q = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$

$q_0 = (q_{0_1}, q_{0_2})$

δ : $\delta((q_1, q_2), X) = ((q_1', q_2'), X', D')$

si $\delta_1 (q_1, X) = (q_1', X'_1, D')$

$\delta_2 (q_2, X_1') = (q_2', X', D')$

e) Equivalence d'un transducteur composé et d'une ET

Un transducteur composé est équivalent à une ET si, étant donné une arborescence à l'entrée sous forme d'expression parenthétisée, il fournit l'expression parenthétisée équivalente à un résultat de l'application de la ET.

III - AUTRES COMPOSITIONS

Il y a plusieurs moyens de composer deux ou plusieurs transducteurs. Dans la composition définie ci-dessus, qui est une composition simple, le premier influence le second que par le mot de sortie, moyen qui est suffisant pour avoir tous les comportements possibles mais qui est difficile à manier. En ayant recours aux transducteurs à tables, le moyen d'intéraction est bien plus évident tout en étant équivalent au premier. Quand on restreint le transducteur à une lecture unique de la bande d'entrée (transducteur à pile ou régulier) pour avoir un moyen équivalent d'intéraction, il faudrait que le transducteur ait une influence sur sa propre fonction de transition. La définition de l'intéraction par table peut être définie de la manière suivante.

Soit T_1 T_2 deux transducteurs dont le second est à table. Soit T l'ensemble des tables possibles pour T_2. Alors la fonction de transition de T_1 est définie par

$$\delta \quad Q \times V \longrightarrow Q \times (V - \{B\}) \times \{D, G\} \times V_S^* \times \{D', G'\} \times T$$

Alors la marche de T_2 dépend du résultat de T_1 (bande d'entrée de T_2) et aussi de la modification que T_1 a pu effectuer sur sa table.

Cette composition sera employée pour avoir une reconnaissance dynamique des sous-arborescences où le schéma de figure passera par T_1 et les graphes d'entrée par T_2 (ce qui est équivalent à passer par T_1 sans aucune action).

C H A P I T R E III

RECONNAISSANCE

I - RECONNAISSANCE DE SOUS-ARBORESCENCES

a) Représentation linéaire d'une sous-arborescence

Une arborescence peut être représentée linéairement par un mot de L'. Si une arborescence t est une sous-arborescence d'une arborescence T, on a alors les relations suivantes entre les mots de L' .

Soit W associé à T, W' associé à t. Il existe une suite $\mu_1 \cdots \mu_{2n+1}$ de mots de $(V \cup \{)\})^*$ tel que :

$W = \mu_1 \mu_3 \cdots \mu_{2n+1}$

$W' = \mu_2 \mu_4 \cdots \mu_{2n}$

et tel que $\mu_{2j+1} \in L' \quad \forall \quad 1 \leqslant j \leqslant n-1$

et $\mu_1 \mu_{2n+1} \in L'$

Ce qui correspond dans la composition aux relations suivantes :

si $T = C\ (T_0\ ;\ r_0 \mid C\ (t\ ;\ c_1 \cdots r_n \mid T_1, \ldots, T_n))$

alors $\mu_1 \mu_{2n+1}$ est la représentation linéaire de T_0

et μ_{2j+1} est la représentation linéaire de T_j

b) Sous-arborescence désignée

Une sous-arborescence sera désignée, c'est-à-dire reconnue, si devant chaque lettre de W' on place un symbole particulier (n'appartenant pas à $V \cup \{)\}$

c) Reconnaissance d'un sous-graphe d'état fini

Théorème : Si w est une arborescence d'état fini, il existe un transducteur ordinaire à pile d'ordre 2 qui désigne w dans un graphe quelconque W.

La désignation est inopérante (c'est-à-dire laisse W inchangé) si w n'est pas une sous-arborescence de W.

Démonstration :

Soit $A = (V_T \cup \{)\}, Q, q_0, F, \delta)$ l'automate d'état fini reconnaissant W.

Propriété : seules les entrées de la forme) peuvent produire un état final (évident sur le mot reconnu $\in L'$)

Le transducteur est défini par :

$T_1 = (V_T \cup \{)\}, V_I, V_{P_1}, Q_1, \delta_1, Q_{0_1}, F_1)$

$T_2 = (V_I, V_T \cup \{)\}, V_{P_2}, Q_2, \delta_2, Q_{0_2}, F_2)$

Avec :

$V_I = \{(x, p) \mid x \in V_T \cup \{)\}, P \in \mathcal{P}(Q)\} \cup V_T \cup \{)\}$

$V_{P_1} = \mathcal{P}(Q)$

$Q_1 = \mathcal{P}(Q) \cup \{q_T, q_n\}$

$Q_{0_1} = \{\{q_0\}\}$

$F_1 = \{q_T, q_n\}$

$V_{P_2} = \{(\}$

$Q_2 = Q \cup \{q_n, q_T \; q_F\}$

$Q_{0_2} = \{q_T, q_n\}$

$F_2 = \{q_n\}$

δ_1 $\delta_1 (P_1, a, x) = (P_{S_1}, x P_1, S_P, L')$

$\delta_1 (P_1,) , P_2) = (P'_{S_1}, \wedge, S_p, L')$

Avec :

$E_P = \{q_j \mid q_j \in Q \wedge (\exists q_i \in P_1 : x \; q_i \rightarrow q_j \in \delta, x \in V_T \cup \{)\})\}$

x : symbole étant lu dans l'application de δ_1

E_P est l'ensemble des états pouvant se déduire des états de P_1 dans l'automate A.

$P_{S_1} = \{q_0\} \cup E_P$

P'_{S_1} a deux formes suivant qu'il existe ou non dans E_P un état final de A.

Si $\exists q_i : q_i \in F \wedge q_i \in E_P$ alors $P'_{S_1} = P_2 \cup \{q_0\} \cup E_P$

Sinon $P'_{S_1} = a_T$

S_P : Dans tous les cas où l'état n'arrive pas sur q_T on a $S_P = (X, P_1)$

Dans le cas où $P'_{S_1} = q_T$ alors $S_P = (X, P_1 \cup \{q_i \mid q_i \in F \wedge q_i \in E_P\}$

$\delta_1 (q_T, x, X) = (q_T, \wedge, x, L')$ $\forall x \in V_T \cup \{)\}$

$\delta_1 (P_1, B, X) = (q_N, \wedge, \wedge, L')$ $\forall P_1 \in \mathcal{P}(Q)$

A la sortie de ce premier transducteur, on a un mot sur $(x, p)^*$ dont la projection sur $V_T \cup \{)\}$ donne l'image miroir de W. On a associé à chaque symbole les différents états possibles obtenus dans la reconnaissance de w par l'automate A.

Si l'arborescence w est une sous-arborescence de W, alors on aura obtenu un état final, donc l'état q_T, sinon on aura obtenu l'état final q_n.

La marche la plus simple de T_2 est celle où il ne peut y avoir de sous-arborescence reconnue.

$\delta_2 (q_n, x, n) = (q_n, \wedge, x, L')$ $\forall x \in V_T \cup \{)\}$

$\delta_2 (q_n, (x, P_1), \wedge) = (q_n, n, x, L')$ $\forall (x, P_i) \in V_I$

Dans ce cas, T_2 restitue l'arborescence d'entrée sans modification.

Dans le cas où T_1 est passé par un état final, on a la trace de la reconnaissance de w. Suivant les propriétés de A, T_2 peut être déterministe ou non.

T_2 ne sera pas déterministe si dans A, il existe un état q_i qui a plusieurs antécédents.

$\delta_2 (q_T, x, \wedge) = (q_T, \wedge, x, L')$ $\forall x \in V_T \cup \{)\}$

$\delta_2 (q_T, (x, p_1), \wedge) = (q_1, \wedge, dx, L')$

Avec q_1 défini par :

$q_1 \in P_1$, $xq_1 \rightarrow q_i \in \delta$, $q_i \in F \wedge q_i \in P_1$

Il peut évidemment y avoir plusieurs q_1 possibles suivant la configuration de A.

$\delta_2 (q, (x, P_1), \wedge) = (q_1, \wedge, dx, L')$

si $\exists q_1 \in P_1 : xq_1 \longrightarrow q \in \delta$ et $q_1 \neq q_0$

Sinon $\delta_2 (q, (), P_1), \wedge) = (q,),), L')$

$\delta_2 (q, (), P), x) = (q, x),), L')$ si $x \neq \wedge$

$\delta_2 (q, (a, P), x)) = (q, x, a, L')$ $\forall$ $a \in V_T$

$\delta_2 (q, (x, P), \wedge) = (q_n, \wedge, dx, L')$ si $q_0 \in P$

$\wedge x q_0 \longrightarrow q \in \delta$

Quand on a remonté tous les états jusqu'à q_0, W a été désigné par d, il suffit de sortir le reste de W.

Remarque :

Le transducteur étiquette une seule sous-arborescence w de W dans le cas où il y aurait plusieurs décompositions possibles.

d) Reconnaissance multiple

On peut désigner toutes les sous-arborescences possibles par un transducteur à pile d'ordre deux avec deux symboles de désignation: Un premier symbole désignant la sous-arborescence reconnue principale, un deuxième symbole désignant les points de l'arborescence W qui peuvent faire partie d'une autre décomposition. L'énumération de toutes les décompositions possibles s'en trouvera alors allégée par la suite, puisque l'on a désigné tous les points à considérer et eux seuls.

Pour obtenir ce résultat, il faut légèrement modifier T_1 et T_2 défini comme précédemment de la manière suivante :

T_1 continue à explorer le graphe, même après avoir reconnu une figure et donne en sortie toujours des mots de la forme (x, p).

Un symbole supplémentaire est nécessaire pour connaître la fin des sous-arborescences reconnues. Il est noté f. Si on a alors un état final de A qui apparaît, T_1 place sur sa sortie $(X, P \cup \{f\})$.

Un deuxième symbole supplémentaire désigne la fin de la figure principale f' et T_1 place alors $(X, P \cup \{f'\})$.

T_2 relit le graphe à l'envers de la même manière mais à trois cas distincts.

1) Cas où le point ne peut appartenir à aucune figure

2) Cas où il peut appartenir à une figure non principale

3) Cas où il peut appartenir à la figure principale

Ils ont, suivant le cas, la désignation appropriée, c'est-à-dire X, r'X, rX.

II - ENSEMBLE NON CONNEXE DE SOUS-ARBORESCENCES

Une arborescence W peut contenir à la fois la sous-arborescence w et une autre arborescence w' différente de w. Il existe alors deux décompositions possibles de W. La reconnaissance de w et w' dans W peut se faire simultanément et indépendemment. On peut alors avoir deux méthodes.

- La désignation se fait si W contient w et w'
- La désignation se fait si W contient w ou w'. Dans le cas de la désignation simultanée, c'est-à-dire qu'il faut reconnaître w et w' un point de la figure principale peut appartenir aux deux figures à la fois. Il sera néanmoins étiqueté qu'une seule fois sans distinction de son appartenance à w ou w'.

Pour obtenir une telle recherche, il suffit d'étudier les arborescences d'états finis.

Soient deux arborescences d'états finis et leurs automates de reconnaissances associées A et A'

$$A = (V_T \cup \{)\}\ , Q, q_0, F, \delta)$$

$$A' = (V_T \cup \{)\}\ , Q', q'_0, F', \delta')$$

Alors on obtient les automates d'états finis suivant :

a) Celui associé à W ou W' : A"

$$A'' = (V_T \cup \{)\}\ , Q \times Q' , (q_0, q'_0) , F'', \delta'')$$

Où :

$$F'' = \{(q_i, q'_j) \mid q_i \in Q, q'_j \in Q' \wedge (q_i \in F \vee q'_j \in F')\}$$

$$\delta'' : \delta''(X, (q_i, q'_j)) = (q_h, q'_k)$$

$$\text{avec } \delta(X, q_i) = q_h$$

$$\delta(X, q'_j) = q'_k$$

b) Celui associé à W et W' ne change que par l'ensemble des états finaux :

$$A''' = (V_T \cup \{)\} , Q \times Q', (q_0, q'_0), F''' , \delta''')$$

$$F''' = \{(q_i, q_j) \mid q_i \in F \wedge q_j \in F'\}$$

$$\delta''' : \delta''' (X, (q_i, q'_j)) = (q_h, q'_k)$$

$$\delta (X, q_i) = q_h$$

$$\delta (X, q'_j) = q'_k$$

Ces deux automates reconnaissent soit l'un ou l'autre des graphes, soit les deux simultanément . Si on modifie δ'' et δ''' tel que

$$\delta'' (X, (q_i, q'_j)) = (q_i, q'_k) \qquad \forall\ q_i \in F$$

$$\delta'' (X, (q_i, q'_j)) = (q_k, q'_j) \qquad \forall\ q'_j \in F'$$

alors le transducteur associé à cet automate défini ci-dessus reconnaîtra respectivement ou l'une des deux sous-arborescences, ou les deux sous-arborescences à la fois.

Cette propriétée est d'ailleurs vraie pour un nombre fini de sous-arborescences.

III - PREDICAT ASSOCIE A UNE SOUS-ARBORESCENCE

Pour définir une sous-arborescence, on peut procéder par la donnée d'une arborescence connexe, et la donnée de prédicat en chaque point de cette arborescence. On obtient alors un moyen de définir un grand nombre de sous-arborescences d'états finis. Chacune est donnée par la structure, arborescence donnée, et une valeur du prédicat associé à chaque sommet. On peut alors modifier l'automate A pour obtenir un automate d'état fini qui reconnaisse une sous-arborescence définie de cette manière. En effet, soit q_i état de l'automate quand il lit le même sommet et p le prédicat associé à ce nième sommet.

Alors si $\delta (x, q_i) \longrightarrow q_j$ on aura les définitions de nouveaux états nommés $(q_j, P_i (x))$ et

$$\delta (x, q_i) = (q_j, P_i (x))$$

L'automate A ne s'arrête pas si $P_i (x)$ est vrai.

Autrement dit $\delta(X, (q_j, P_i(x))) = (q_h, P_j(x))$

si $P_i(x)$ est vrai et si P_j est le prédicat associé à q_j

De cette manière, on ramène le calcul du prédicat sur les sommets de l'arborescence au calcul du prédicat en certains états de l'automate.

Dans l'exemple suivant, sur la recherche des schémas de figure, le prédicat est un prédicat d'étiquette. Le prédicat consiste donc à vérifier l'appartenance de l'étiquette du ième sommet à un sous-ensemble du vocabulaire fini d'étiquettes des noeuds. L'entrée de ce prédicat est la donnée de toutes les valeurs d'arguments pour lequel il est vrai. De cette manière, il suffit de considérer que seul ce nombre doit être fini et V_T peut alors être infini. Dans l'application aux manipulations de formules, le prédicat est plus compliqué. Il est d'abord un prédicat d'étiquette dont le nombre d'arguments pour lequel il est vrai est en général de 1. Mais il est aussi un prédicat plus complexe qui dépend de la nature de cette étiquette.

Enfin, dans d'autres applications possibles, analyse des langues naturelles notamment, le prédicat peut prendre une forme plus élaborée : valeur grammaticale, etc...

IV - SCHEMA DE FIGURE

Un schéma de figure est un ensemble de prédicats hiérarchisés sur un ensemble de points. Reconnaître un schéma, c'est donner l'ensemble des points d'une arborescence satisfaisant à cet ensemble de prédicat.

Cet ensemble de prédicat peut être divisé en deux groupes :

a) les prédicats de structures

Ces prédicats sont indépendants de la nature des étiquettes ou de leur présence. Ils ne font intervenir seulement que des relations de structure (avoir un descendant, un père, etc...).

b) Les prédicats d'étiquettes

Ceux-ci, par contre, sont indépendants de la structure et font intervenir la nature ou la présence de certaines étiquettes en certains points.

Cette décomposition fait ressortir les deux éléments qui composent un schéma de figure et permet de les relier aux extensions de l'arborescence d'état fini étudié plus haut.

L'ensemble des prédicats de structures définit une suite finie d'arborescence non connexe d'état fini.

C'était la première extension de l'arborescence d'état fini.

L'ensemble des prédicats d'étiquettes définit sur les arborescences des prédicats associés aux sous-arborescences, c'est la deuxième extension.

Un schéma de figure est alors une combinaison de ces deux extensions et peut être reconnu par un transducteur à pile d'ordre 2.

La donnée d'un schéma de figure se fera par la donnée d'ensembles structurés de prédicats d'étiquettes séparés par des virgules.

Chaque ensemble sera un ensemble d'arborescences de prédicats concaténées (c'est-à-dire une ramification de prédicat).

V - RECONNAISSANCE D'UN SCHEMA

Pour qu'un schéma de figure soit reconnu par un transducteur, il faut construire l'automate d'état fini A. Cet automate doit reconnaître les arborescences définies et calculer aussi la valeur du prédicat sur les états correspondants aux sommets des arborescences.

Pour reconnaître les différentes arborescences, il doit reconnaître un des groupes d'arborescences et, parmi ce groupe, l'ensemble des arborescences qui le compose, c'est-à-dire qu'il faut former un automate A qui combine les deux propriétés de l'automate reconnaissant un ensemble non connexe d'arborescences.

On aura si A_i est l'automate associé à l'arborescence W_i $\quad 1 \leqslant i \leqslant n$

$$A_i = (V_T \cup \{)\}\ ,\ Q_i,\ q_{0_i},\ F_i,\ \delta_i)$$

Alors : $A = (V_T \cup \{)\},\ \prod_{i=1}^{n} Q_i,\ (q_{0_1}, \ldots, q_{0_n}),\ F,\ \delta)$

$$\delta(X, (q_{1_{i_1}}, q_{2_{i_2}}, \ldots, q_{n_{i_n}})) = (q_{1_{j_1}}, \ldots, q_{n_{j_n}})$$

Si $\quad \delta_k (X, q_{k_{i_k}}) = q_{k_{j_k}} \qquad \forall \quad 1 \leqslant k \leqslant n$

La combinaison des deux extensions n'intervient que dans F ou pour avoir un état final, il faut simultanément une partie de $\prod_i F_i$.

Supposons pour simplifier que le schéma soit composé seulement de deux groupes :

$$F = \left\{ (q_{1_i}, \cdots q_{n_i}) \mid ((q_{1_i}, \cdots q_{j_i}) \in \prod_{k=1}^{j} F_k \vee (q_{j+1_i} \cdots q_{n_i}) \in \prod_{k=j+1}^{n} F_k) \right\}$$

L'extension au cas où il y a plusieurs groupes d'arborescences est immédiate.

On a ainsi la reconnaissance des différentes structures du schéma. Pour avoir une reconnaissance complète, il faut ajouter à cet automate, aux différents états correspondants, les prédicats d'étiquettes. Cette adjonction ne change rien à l'automate A si ce n'est la transformation de certains états.

On a ainsi l'automate d'état fini A associé au schéma de figure donné. Cet automate sert de base pour la construction du transducteur reconnaissant un schéma. Il en découle la propriété : un schéma de figure est reconnaissable par un transducteur à pile d'ordre 2.

les avantages de cette reconnaissance sont surtout la rapidité puisque pour reconnaître un schéma, il suffit de lire deux fois la chaîne d'entrée environ, (première lecture par T_1, deuxième lecture par T_2 qui lit une chaîne un peu plus longue que celle d'entrée). Le gros inconvénient est la construction de l'automate A nécessaire à la marche de T_1 T_2 qui est très pénible. La reconnaissance dynamique va palier à cet inconvénient tout en ne pénalisant pas les temps.

VI - RECONNAISSANCE DYNAMIQUE

Le problème pour la reconnaissance d'un schéma par transducteur est la construction de l'automate A. Cette construction peut se faire automatiquement. On a alors l'équivalent d'un transducteur d'ordre 3 dont les deux derniers sont à table. Le premier, construit A, et modifie les tables des deux autres. Cette construction se fait en une lecture du schéma.

a) Construction de l'automate d'état fini associé à une arborescence d'état fini. C'est la construction élémentaire d'un automate reconnaissant un W_i. Cette construction est très simple, elle est celle d'un automate d'état fini reconnaissant une seule chaîne. Le nombre des états est égal à la longueur de la chaîne plus un.

b) Construction de l'automate d'état fini reconnaissant un ensemble d'arborescences.

C'est la construction simultanée de plusieurs automates du type précédent. En fait cette construction n'est pas simultanée car les différentes arborescences arrivent les unes après les autres.

c) Construction de l'automate contenant des prédicats associés aux étiquettes.

Dans la construction de l'automate de base, l'état q_i qui correspond à la position dans l'arborescence du prédicat P_i devient un état (q_i, P_i), c'est-à-dire que A a alors deux sortes d'états, les états à prédicats et les états sans prédicats. La donnée du prédicat, précédent le symbole correspondant, est lue en même temps que la construction de l'état q_i.

d) Construction de l'automate reconnaissant un schéma. C'est la construction d'un automate regroupant les trois types de construction. Cette construction se fait par un transducteur d'état fini et ne nécessite que la lecture du schéma de figure.

La reconnaissance dynamique d'un schéma de figure se fait donc par un transducteur d'ordre 3 dont les deux derniers sont à pile et le premier d'état fini. Le premier transducteur lit le schéma et forme les tables des deux suivants puis passe sur sa bande de sortie toutes les arborescences où le schéma doit être recherché. On a donc sur l'entrée du premier transducteur la chaîne :

schéma arborescence 1 arborescence 2... arborescence n

Il faut donc en entrée avoir deux symboles particuliers désignant soit un schéma de figure, soit une arborescence. On peut alors avoir la suite, si # et ¢ sont ces signes :

¢ schéma 1 # arborescence 1 # arborescence 2... # arborescence n
¢ schéma 2 # arborescence 1.1 #...

A la sortie de T_3, on obtient la suite

arborescence 1 # ... # arborescence n # arborescence 1.1 #...

VII - EXEMPLE DE PROGRAMMATION D'UN SYSTEME DYNAMIQUE DE RECONNAISSANCE

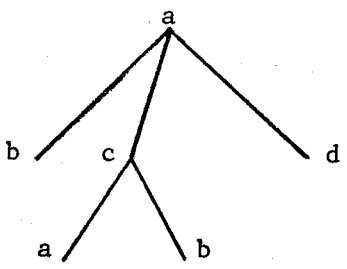
Il s'agit d'un système simulant les transducteurs de reconnaissance définis ci-dessus. Le choix adopté a été guidé par la simplicité, le seul but de ce programme étant de donner un exemple de réalisation. Une réalisation plus complexe en est faite dans l'application aux manipulations de formules. Les schémas de figure sont très simplifiés et ont les deux propriétés :

- les différents groupes d'arborescence contiennent une seule arborescence, ceci afin de pouvoir sortir sous forme de graphe la figure reconnue.

- les prédicats d'étiquettes ne concernent que la présence ou non d'une étiquette.

Pour plus de simplicité dans l'écriture, l'entrée des graphes se fait dans le langage L_1.

Exemple : le graphe



s'écrit a (b () c (a () b ()) d ())

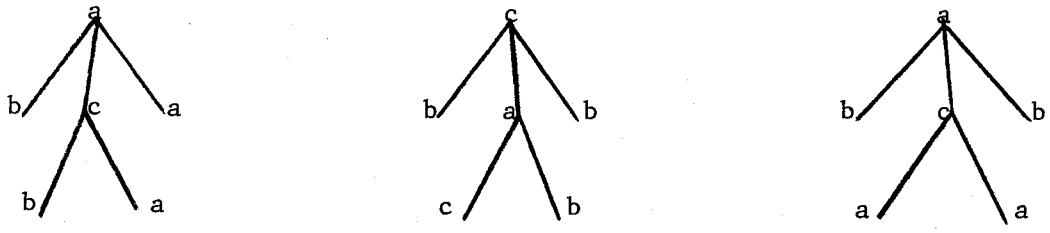
Les deux symboles qui permettent de distinguer un schéma d'une arborescence à traiter sont respectivement ' # ' pour le schéma et ' * ' pour l'arborescence. Ce symbole suit le schéma ou l'arborescence.

La donnée du schéma de figure s'effectue de la façon suivante, les différents groupes sont séparés par une virgule.

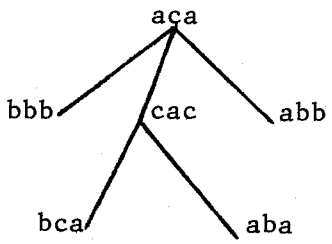
Chaque groupe qui ne comprend ici qu'une arborescence est un mot L_1 dans lequel on a remplacé chaque lettre de V_T par un mot sur V_T^*. Chaque lettre de ce mot sont les étiquettes possibles aux différents endroits de l'arborescence.

Exemple :

Soient les trois graphes de structures identiques étiquetés différemment.



Ils seront donnés par le schéma :



linéairement : aca (bbb () cac (bca () aba ()) abb ())

Ce qui est équivalent à la donnée du schéma :

ac (b () ca (bca () ab ()) ab ())

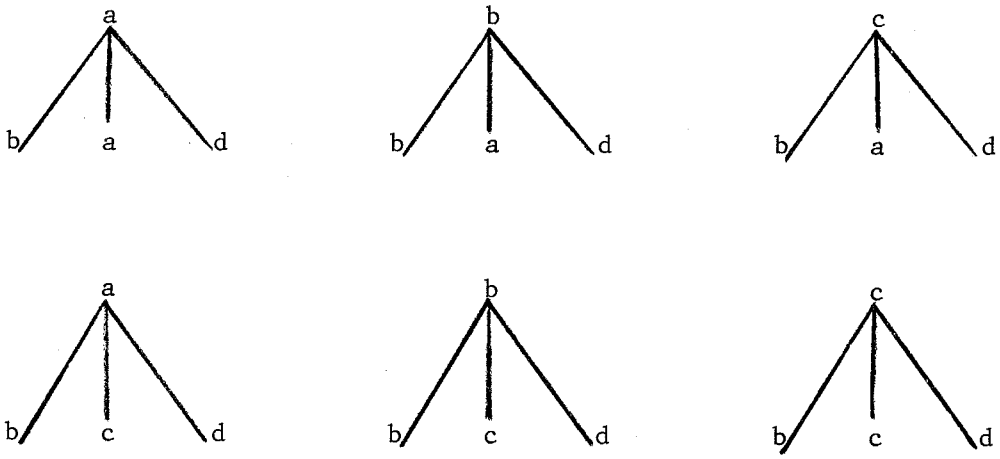
Car une lettre dans un mot devant un sommet veut dire que cette lettre est permise.

Exemple de schéma :

Soit le schéma a (bc () b ()), abc (b ()ac () d ())

Il y correspond l'ensemble possible des arborescences relatives à la première structure :

relatives à la deuxième structure :



Chaque point du graphe de sortie pourra avoir trois formes:

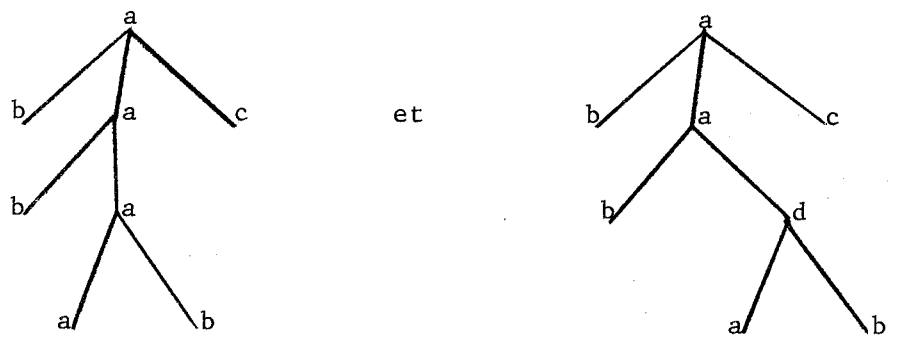
1) n'être précédé d'aucun signe si ce n'est pas un point particulier

2) être précédé du signe * s'il appartient à une figure qui n'est pas la figure principale.

3) être précédé du signe @ s'il appartient à une figure principale (il peut également appartenir à une figure non principale dans ce cas).

Après avoir sorti l'arborescence marquée, le programme développe la figure principale reconnue qui est ici toujours une arborescence. Pour cela, il suffit de sélectionner toutes les lettres qui sont précédées de @

Dans le premier schéma très simple, le deuxième graphe en entrée contient deux figures possibles qui sont



La deuxième est prise comme principale

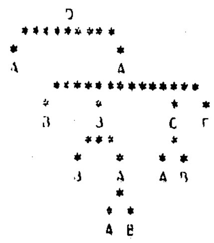
La figure choisie comme principale est celle qui est finie la première quand on lit le graphe de gauche à droite.

NOUVEAU SCHEMA DE FIGURE

AC(B()=A(B()DAB(A()B()))C()),ABCDE(FGHIJ(AB()C()DEF())GZA)()ABCD())

GRAPHE EN ENTREE

D(A()A(B()B(B()A(A()B()))C(A()B())E()))

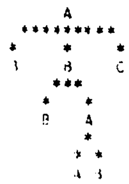


GRAPHE EN SORTIE

D(A()@A(@B()@)@B(@B(@)@A(@A(@)@B(@)@)@)@C(A()B()@)E()@))

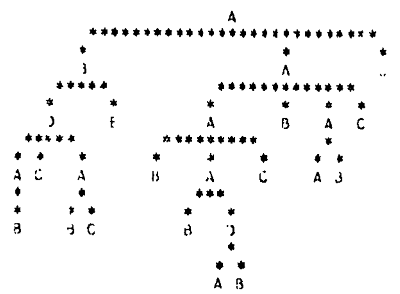
FIGURE PRINCIPALE RECONNUE

A(B()B(B()A(A()B()))C())



GRAPHE EN ENTREE

A(B(D(A(B())C()A(B()C()))E())A(A(B()A(B()F(A()B()))C())B()A(A()B())C())C())

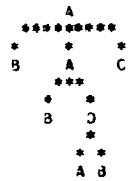


GRAPHE EN SORTIE

*A(*B(C(A(B()))C()A(B()C()))E()*)*A(@A(@B(@)@A(@B(@)@D(@A(@)@B(@)@)@)@C(@)@)*B(*)*A(*A(*)*B(*)*)C()*)*C(*)*)

FIGURE PRINCIPALE RECONNUE

A(B()A(B()C(A()B()))C())

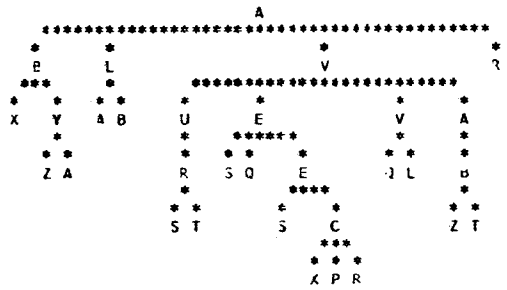


NOUVEAU SCHEMA DE FIGURE

VOIC((UN()EMA(SCH()QUI()EST(ASEZ()COMP(LEX()POUR()PROU()))))VER(QUE()LE()))),SYST(MARC()ME-C(BIEN()AVEC(CE(PRINC()TAB(BON()MAIS()))PE B()))))),MAIGR(FTOUI(ASSEZ()SIMPLE()))ILS(SUF()IT(DAV(GER()NI(MAIS(CELA()NEMA(APS()DE(PROB())POSE()))))))OIR(NI()DU()))))),CAR(CHEZ()CE ST(NATU()REL(ILR()EVIN()))MOI(SANT()AUG(ALO(JEB(ESAI()DE(FAC)N()JE(NESA()PAS()))))SI(LE()CHAZ(MAIS(DE()TOUTE()))))))),CANE(ME(FAIS( )RIEN()))DE(BLEU(BLANC()JAUNE())NIOR())NOIR(VERT()BLANC(INFRA(ROULE())ULTRA()))VIOLET(ROUTE()LESCO(LEURS()YPASSE()))))ROUGE(VVERT()BLE U()))))

GRAPHE EN ENTREE

A(B(X()Y(Z()A()))L(A()B()))V(J(R(S()T())))E(S(Q()E(S()C(X()P()R()))))V(Q()L())A(B(Z()T()))R())

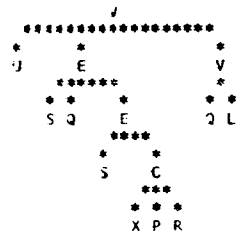


GRAPHE EN SORTIE

A(B(X( )Y(Z( )A( ))))L( )( )B( ))aV(aU(R(S( ) T( )) a) aE(aS(a) aQ(a) aE(aS(a) aC(aX(a) aP(a) aR(a) a) a) a) aV(aQ(a) aL(a) a)A(B(Z( )T( ))) a)R( ))

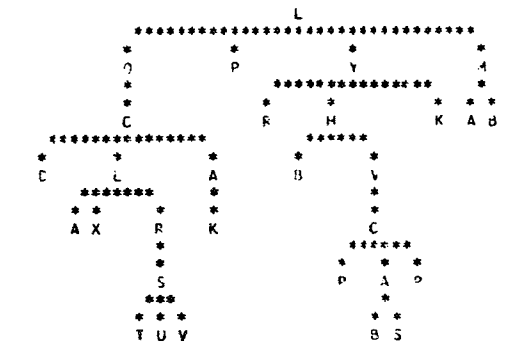
FIGURE PRINCIPALE RECONNUE

V(U( )E(S( )Q( )E(S( )C(X( )P( )R( ))))V(Q( )L( )))



GRAPHE EN ENTREE

L(Q(C(E( )L(A( )X( )R(S(T( )U( )V( ))))A(K( ))))P( )Y(R( )H(B( )V(G(P( )A(B( )S( ))P( ))))K( ))M(A( )B( )))

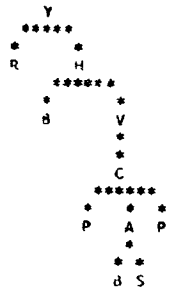


GRAPHE EN SORTIE

L()(C(E()L(A()X()R(S(T()U()V()))))A(K()))P()aY(aR(a)aH(aB(a)aV(aC(aP(a)aA(aB(a)aS(a)a)aP(a)a)a)a)K()a)M(A()B()))

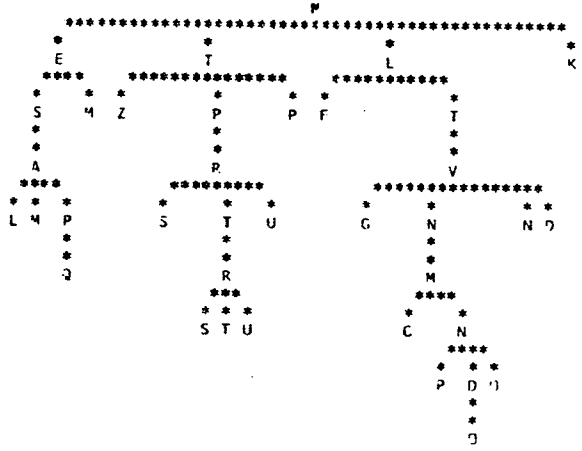
FIGURE PRINCIPALE RECONNUE

Y(R()H(B()V(C(P()A(B()S()))P()))))



GRAPHE EN ENTREE

M(E(S(A(L()M()P(Q()))))M()))T(Z()P(R(S()T(R(S()T()U()))U()))U()))P()))L(F()T(V(G()N(M(C()N(P()D(O()))O()))))(N()D()))K()))



GRAPHE EN SORTIE

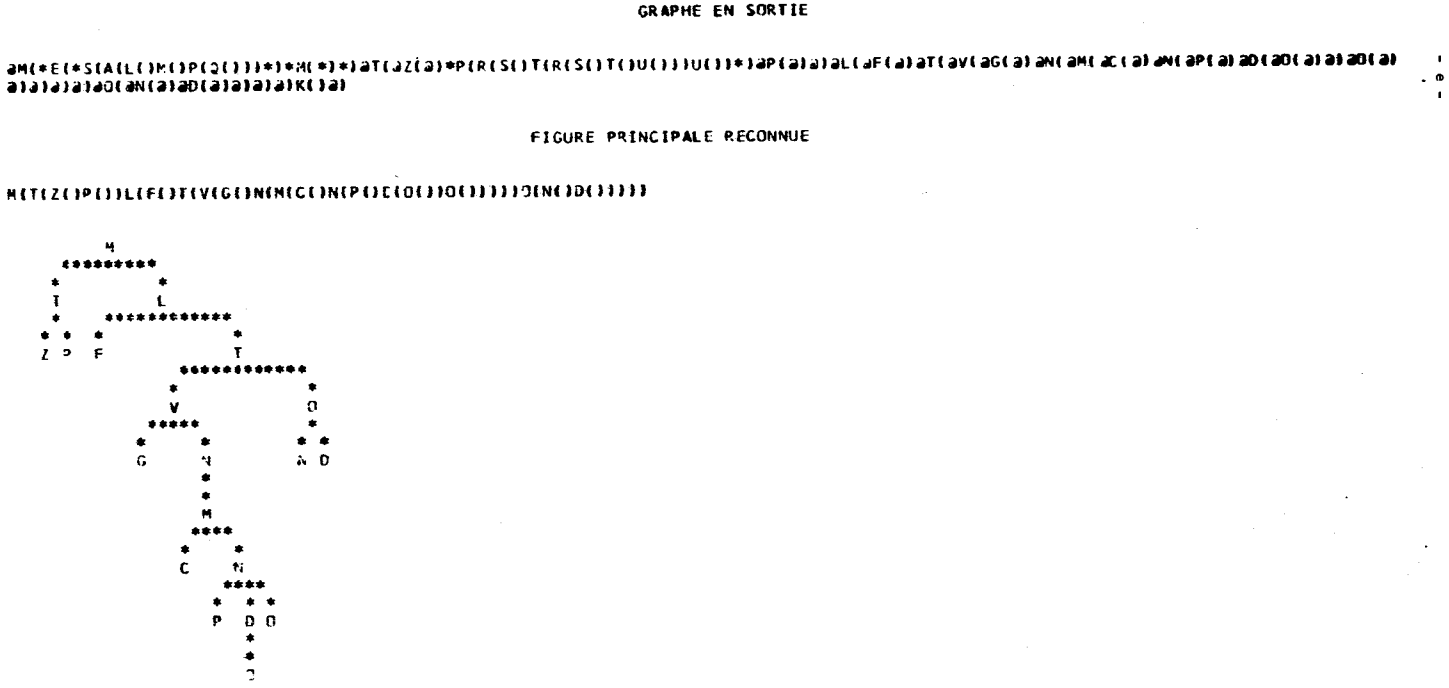
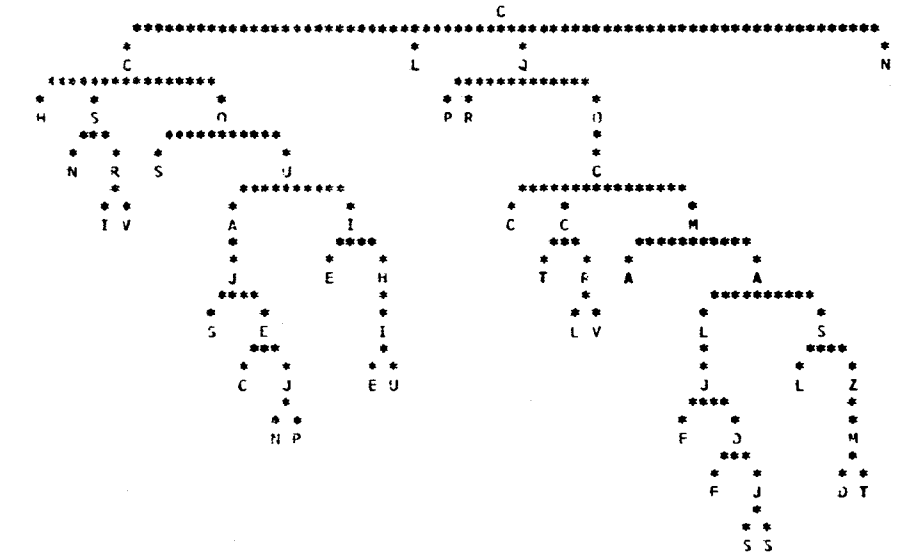


FIGURE PRINCIPALE RECONNUE

GRAPHE EN ENTREE



GRAPHE EN SORTIE

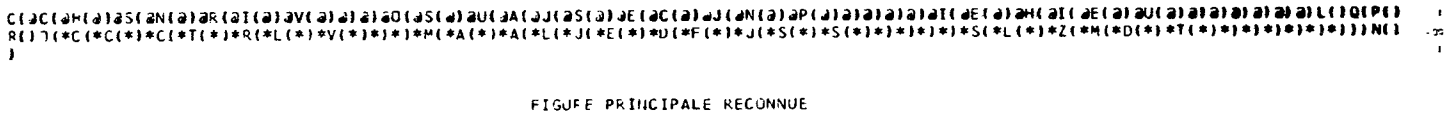
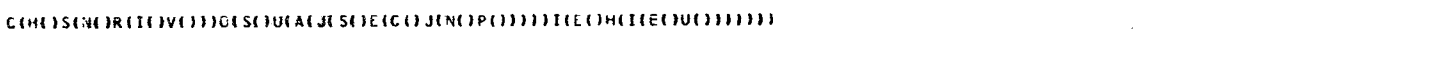
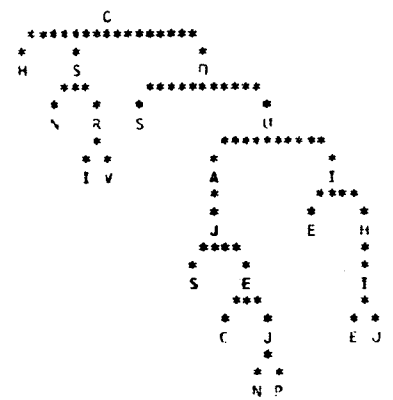


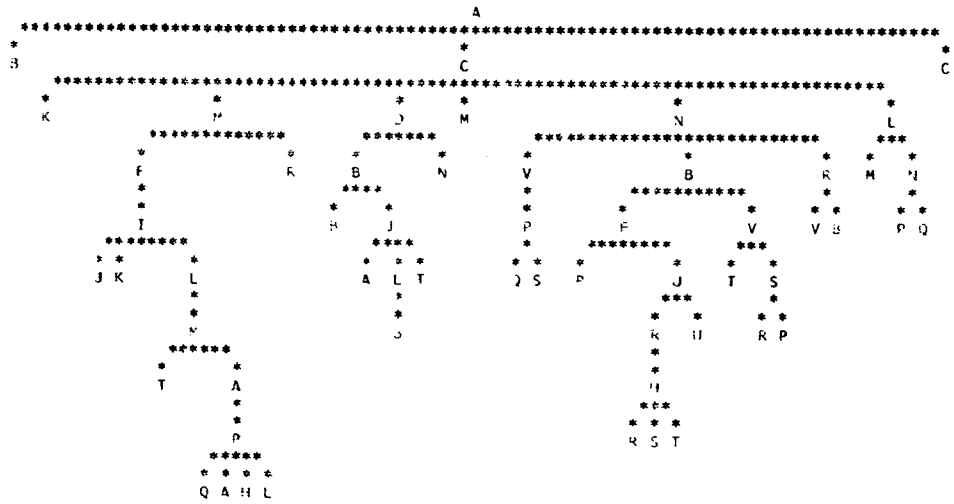
FIGURE PRINCIPALE RECONNUE





GRAPHE EN ENTREE

A(B( )C(K( )M(F( )I( )J( )K( )L(M( )T( )A(P(Q( )A( )H( )L( ))))))R( ))D(B(B( )J(A( )L(B( ))T( ))))N( ))M( )N(V(P(Q( )S( ))))B(F(R( )U(R(U(R( )S( )T( ))))U( ))))V(
T( )S(R( )P( ))))P(V( )H( ))L(M( )N(P( )Q( ))))C( ))

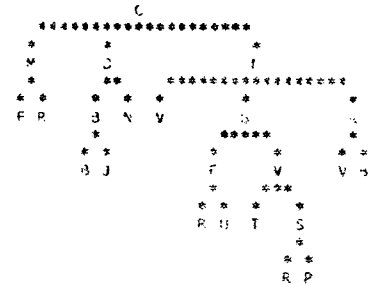


GRAPHE EN SORTIE

A(B( )C(K( )M(F( )I( )J( )K( )L(M( )T( )A(P(Q( )A( )H( )L( ))))))R( )D(B(B( )J(A( )L(B( ))T( ))))N( )M( )N(V(P(Q( )S( ))))B(F(R( )U(R(U(R( )S( )T( ))))U( ))))V(
T( )S(R( )P( ))))P(V( )H( ))L(M( )N(P( )Q( ))))C( ))

RETOUR PRIMAIRES RECURSIF

C(M(F( )R( ))D(B(B( )J( ))N( ))N(V( )R(F(R( ) J( ))V(T( )S(R( )P( ))))P(V( )S( ))))

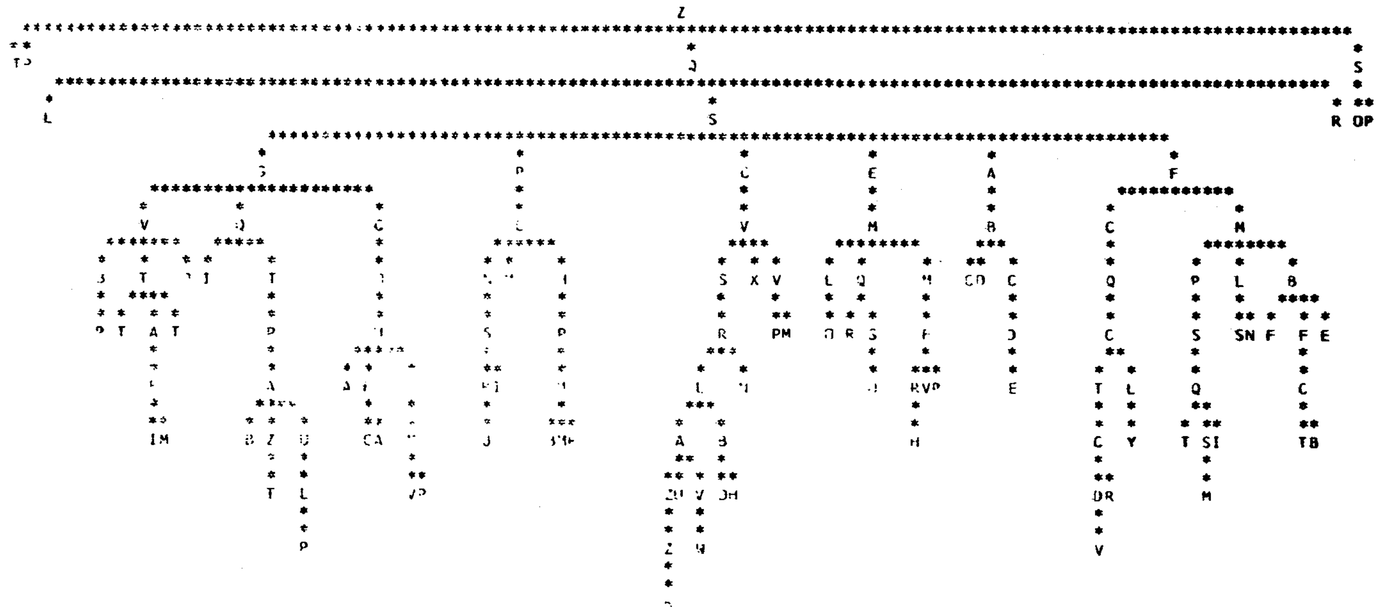


NOUVEAU SCHEMA DE FIGURE

```
SCHEMA(FIGURE(AVEC(BEACOUP(DEPUI())ETAUS(PEUT()BEAUC(DEP SI(BILATE()JAMIS()))ETRE()}CEST3()}QUEV(AIS()METRE(POURT()))CESTNO(VOILAU
(UNCHOIX(AFIRE()DIFICILE(ONY(JAFIV())3IEN2(MENA(VEQUP()PEUT)I4A())))))PREMIER(SCHEMA(CONSA(AFLSPI(MIREL(MOIJE())MAIS()))ANABI()CH
ETE(LAPEL(HAPUCE(BICHE()AMJI()MIREL()))))SCHENMA(AVECTP(DESCAN(VERTIG(VESLE(BASFO(JUGRA()PHELA()))NEUSE()))DEUXIM()AVINTLE(TRAPE(
)SANM()))}S32TROISEM(SCHEMA(PALUS(ORIGINEA()}QUEL(PR [illegible] ULSC()))MAISL(FAJTH(FAISF(UNSH()AVEC()NIMPOR()))))FINDEG(CESTLA(Q
UECA(COMPLIQ2E(TOUTE(CARMENS(DEQUCIS(AVECS()}TRINQ()}) SCRV(ILYAPS()))))MOIJENA(PAEURQ(ASCFOMPL(POURQUI(CESTYFA()PASMYT(CESTIMPL(
)}ILYNA())))}ILEU(SANM()NIP())BRANCHE(FINALE()AFIRE(PE CAUNTP()OUB()))ETDIFC()))))
```

GRAPHE EN ENTREE

```
Z(T()P()Q(L()S(G(V(B(P()}T(T()A(E(I()M()}}T()}O()}Q(I()T(P(A(B()Z(T()}U(L(P()}}}}}}C(O(U(A()F(O()A()}B(M(V()P()}}}}}}P(C(N(S(R(J()
}I()}}M()H(P(M(B()N()F()}}}}}C(V(S(R(L(A(Z(Z(P()}}U()V(W()}}B(O()H()}}N()}}X()V(P()M()}}}E(M(L(O()}Q(R()S(O()}}M(F(R(H()}V()P()}}}
}A(B(C()O()C(D(E()}}}}F(C(Q(C(T(C(D(V()}F()}}L(Y()}}}}M(P(S(Q(T()S(M()}I()}}}L(S()N()}B(F()F(C(T()B()}E()}}}}R()S(O()P()}}
```

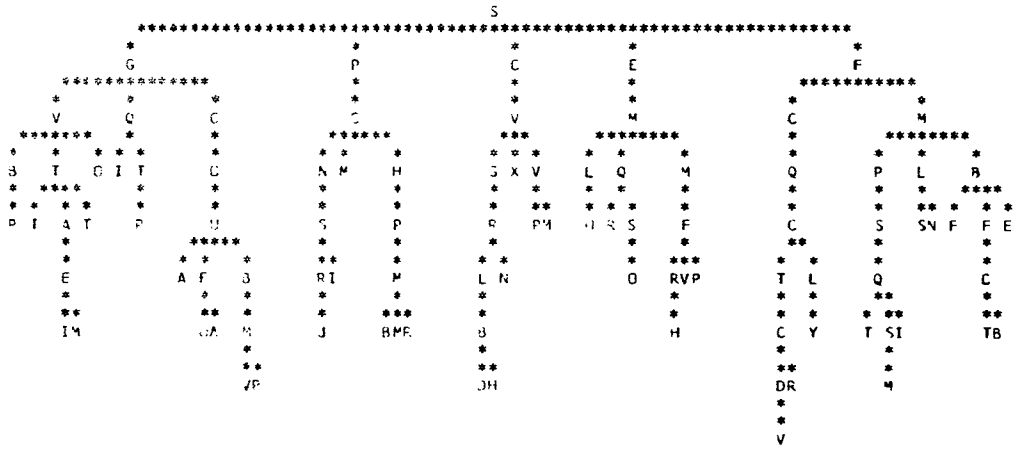


GRAPHE EN SORTIE

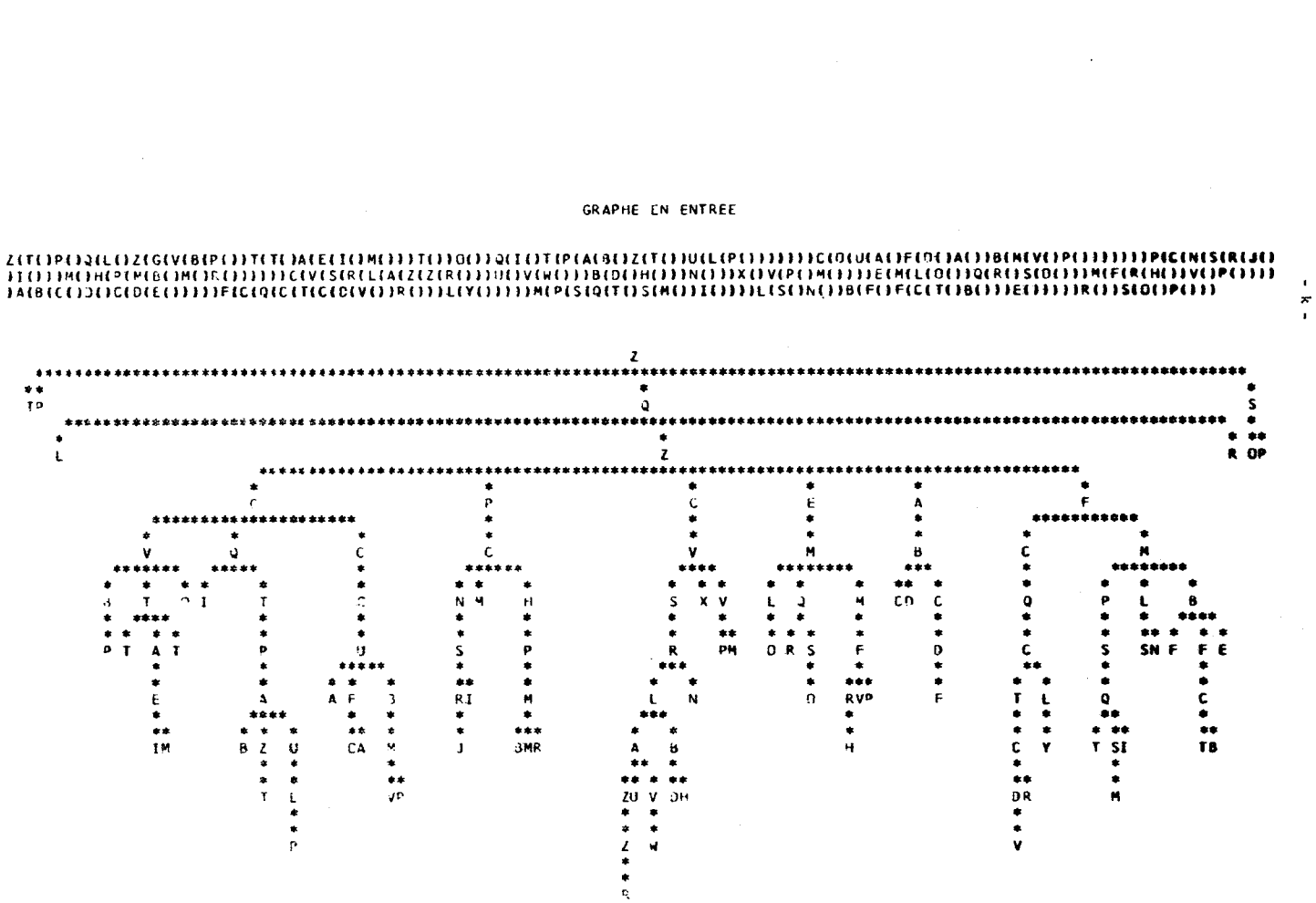
```
Z(T()P()}(L()aS(aG(aV(aB(aP(a)a)aT(aT(a)aA(aE(aI(a)aM(a)a)a)aT(a)a)aO(a)a)aQ(aI(a)aT(aP(A(B()Z(T()}U(L(P()}}}a)a)a)aC(aO(aU(aA(a)a
F(aO(a)aA(a)a)aB(aM(aV(a)aP(a)a)a)a)a)a)aP(aC(aN(aS(aR(aJ(a)a)aI(a)a)a)aM(a)aH(aP(aM(aB(a)aM(a)aR(a)a)a)a)a)a)aC(aV(aS(aR(aL(A(Z
(Z(P()}}U()V(W()}}aB(aO(a)aH(a)a)a)aN(a)a)a)aX(a)aV(aP(a)aM(a)a)a)a)aE(aM(aL(aO(a)a)aQ(aR(a)aS(aO(a)a)a)aM(aF(aR(aH(a)a)aV(a)aP(a)
a)a)a)a)A(B(C()O()C(D(E()}}}}aF(aC(aQ(aC(aT(aC(aD(aV(a)a)aF(a)a)a)aL(aY(a)a)a)a)a)aM(aP(aS(aQ(aT(a)aS(aM(a)a)aI(a)a)a)a)aL(aS(a)aN
(a)a)aB(aF(a)aF(aC(aT(a)aB(a)a)a)aE(a)a)a)a)a)R()S(O()P()}}
```

FIGURE PRINCIPALE RECONNUE

S(G(V(B(P()))T(T()A(E(I()M()))T()))C()Q(I()T(P()))C(D(U(A()F(J()A()))B(M(V()P()))))))P(C(N(S(R(J()))I()))M()H(P(M(B()M()R()))))C(V(S(R(L(E()J()H()))A()))X()V(P()M()))E(M(L(C()))Q(R()S(M()))M(F(R(H()))V()P()))))F(C(Q(C(T(C(D(V()))R()))L(Y()))))M(P(S(Q(T()S(M()))I()))L(S()N()))B(F()F(C(T()B()))E()))))



GRAPHE EN ENTREE



GRAPHE EN SORTIE

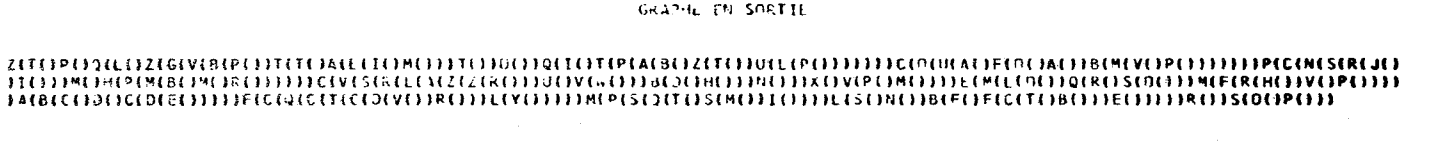
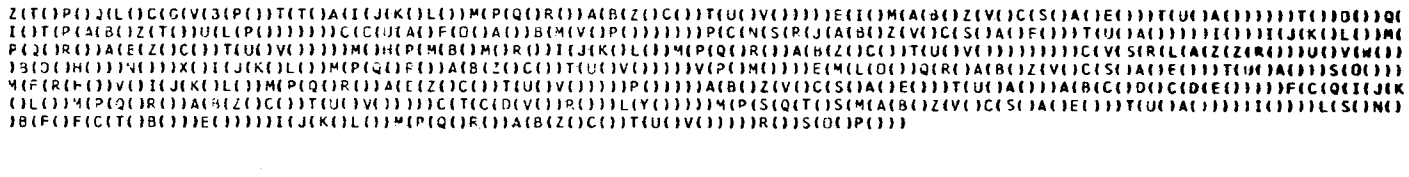


FIGURE PRINCIPALE RECONNUE

GRAPHE EN ENTREE



GRAPHE EN SORTIE

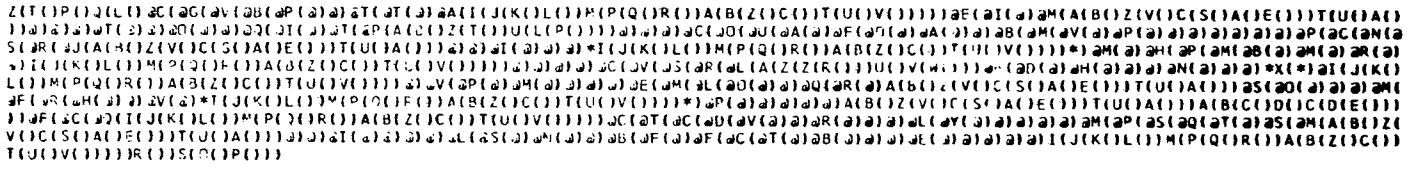
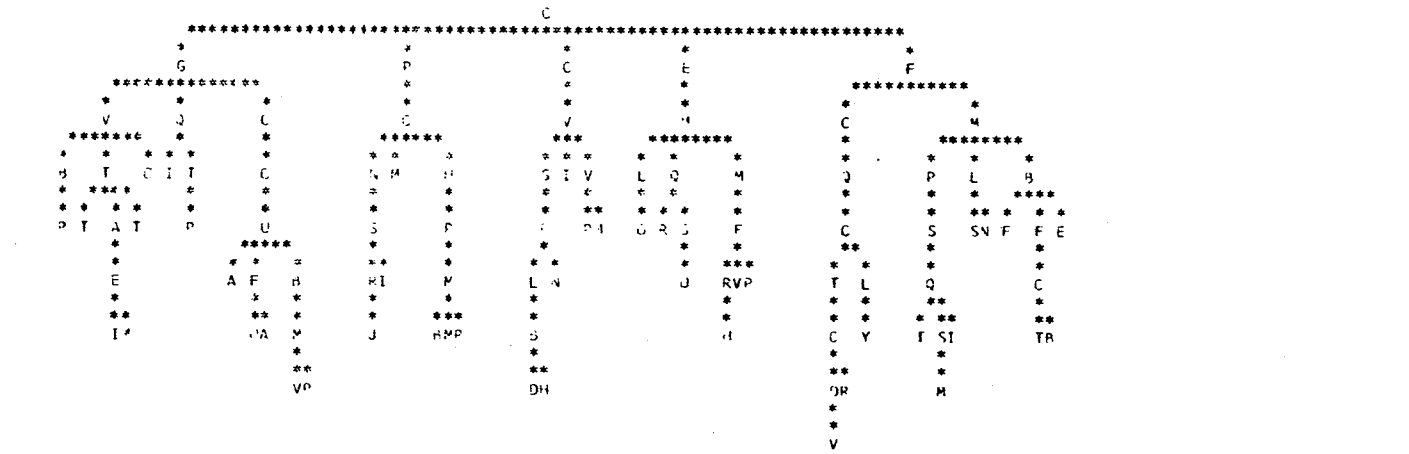


FIGURE PRINCIPALE RECONNUE



C H A P I T R E IV

TRANSFORMATION

Pour avoir un transducteur équivalent à une grammaire transformationnelle il faut être en mesure de simuler n'importe quelle transformation élémentaire par un transducteur. Le transducteur composé des transducteurs simulant toutes les transformations élémentaires sera alors équivalent à une grammaire transformationnelle. Si on veut appliquer cette grammaire de façon cyclique, c'est-à-dire effectuer une autre transformation, quand elle est possible, après un certain nombre de transformations, le transducteur équivalent devra être cyclique. On sait que dans ce cas, il sera toujours équivalent à une machine de Turing et donc l'arrêt ne peut être décidable. Le même problème se pose d'ailleurs pour toute grammaire transformationnelle où on ne fait pas de restriction sur l'application des règles. Ces restrictions seront alors équivalentes dans le cas présent, à des restrictions sur les conditions de "recyclage" d'une arborescence sur l'entrée du transducteur.

Pour pouvoir construire un transducteur qui soit équivalent à une transformation élémentaire il faut être capable de :

a) Reconnaître une figure

b) Transformer cette figure suivant la ET

La reconnaissance de la figure a été étudiée plus haut. Elle est effectuée par un transducteur à pile d'ordre 2. Pour pouvoir faire la transformation il suffit d'être capable d'effectuer les trois transformations élémentaires spéciales définies par Gladky et Mel'chuk.

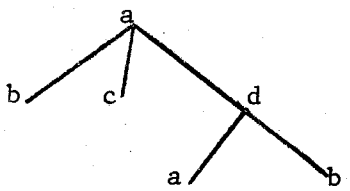
Par composition d'un nombre fini de ces transducteurs, simulant ces transformations spéciales, on aura un transducteur effectuant la transformation élémentaire désirée. Cette transformation élémentaire se s'appliquera que dans le cas où une figure a été reconnue. Dans ce cas, il s'agit de la première arborescence t_1 de la transformation. Dans les autres cas, les transducteurs équivalents seront sans effet sur l'arborescence puisque la transformation ne peut être appliquée. Si cette transformation est applicable seul le dernier transducteur effectuant la dernière transformation élémentaire spéciale enlèvera dans le graphe la désignation des sommets. Les autres transducteurs effectuent seulement leur transformation propre en gardant la désignation de la figure pour que le suivant puisse jouer son rôle.

I - SIMULATION DES TRANSFORMATIONS ELEMENTAIRES SPECIALES

Il y a trois transformations élémentaires spéciales : l'absorption d'un noeud, l'éclatement d'un noeud, le transfert d'un noeud. La dernière de ces transformations se divise en deux : un transfert vers le haut et un transfert vers le bas.

Pour l'application de ces transformations spéciales on suppose que seuls les points désignés sont nécessaires à la transformation. Les points de la figure reconnue sont dans toutes les transformations à effectuer désignés par un symbole d et les points où l'on doit appliquer la transformation spéciale désignés par sd. Chaque transformation spéciale enlève ce symbole supplémentaire s. Un point est désigné par un symbole devant l'étiquette de son sommet et devant la parenthèse correspondante.

Exemple : soit l'arborescence :



en représentation linéaire

a(b()c()d(a()b()))

et soit la sous-arborescence :

elle sera désignée en représentation linéaire par le symbole @ de la façon suivante :

@a(b()@c(@)@d(a()b()@)@)

Si on veut sur-désigner la sous-arborescence a on aura alors la représen-
c

tation suivante (sur-désignation par le symbole *)

* @a(b()*@c(*@)@d(a()b()@)*@)

c) Absorption d'un noeud

L'absorption d'un noeud est la transformation élémentaire qui à deux points en fait correspondre un seul.

en représentation linéaire on a :

a(b())⟶a()

Il suffit donc d'éliminer b()

Cette transformation est très simple et ne nécessite qu'un transducteur d'état fini extrêmement simple. Il lui suffit de recopier sur sa bande de sortie le graphe d'entrée sauf pour le point b. Pour ce faire, il est au départ dans l'état q_0 et recopie tous les symboles jusqu'à la première marque

$$\delta(X, q_0) = (q_0, X, D') \quad \forall\ X \neq *$$

A la première étoile, il recopie les symboles suivants (pas l'étoile) jusqu'à la deuxième. Il est passé alors dans l'état q_1

$$\delta(q_0, *) = (q_1, \Lambda, D')$$

$$\delta(q_1, X) = (q_1, X, D') \quad \forall \quad X \neq *$$

Dans ce dernier état q_1, il efface le premier symbole derrière la deuxième étoile

$$\delta(q_1, *) = (q_2, \Lambda, D')$$

$$\delta(q_2, X) = (q_3, \Lambda, D')$$

Il recopie ensuite jusqu'à l'étoile suivante qui marque le symbole de fermeture correspondant à b.

$$\delta(q_3, X) = (q_3, X, D') \quad \forall \quad X \neq *$$

$$\delta(q_3, *) = (q_4, \Lambda, D')$$

$$\delta(q_4, X) = (q_5, \Lambda, D')$$

Il recopie ensuite le reste du graphe en effaçant l'étoile restante qui marque la fermeture correspondant à a.

$$\delta(q_5, X) = (q_5, X, D') \quad \forall \quad X \neq *$$

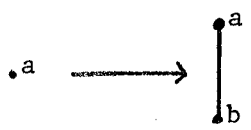
$$\delta(q_5, *) = (q_5, \Lambda, D')$$

L'état q_5 étant l'état final de ce transducteur.

Dans le cas d'une double désignation, il faudrait en tenir compte car le symbole qui suit une étoile est d'abord un élément de désignation. Cet automate simule l'absorption d'un noeud dans L'. Pour l'avoir dans L_1 il faudrait que l'état q_2 efface deux symboles et non un seul comme c'est le cas ici. Ce ne sont que de petites modifications dues aux différents langages employés pour représenter linéairement une arborescence.

a) Eclatement d'un noeud

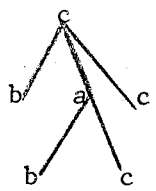
L'éclatement d'un noeud est la transformation élémentaire qui a un point en fait correspondre deux.



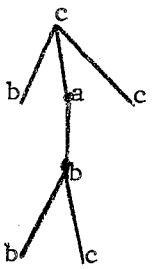
$$f(a) = a \text{ ou } f(a) = b$$

Il y a deux éclatements possibles suivant la fonction f. Ces deux éclatements sont les mêmes dans le cas où a est une feuille. Autrement on a deux cas différents :

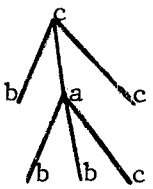
Exemple :



Eclatement de a



f(a) = b



f(a) = a

Ce qui correspond en représentation linéaire aux transformations suivantes :

Dans le premier cas :

c(b()a(b()c())c()) ⟶ c(b()a(b(b()c()))c())

Dans le second cas :

c(b()a(b()c())c())) ⟶ c(b()a(b()b()c())c())

Comme on le voit sur cet exemple, seule change la position du symbole de fermeture correspondant à b.

Dans le premier cas il est situé juste avant celui de a.
Dans le second cas il suit b ; b est alors une feuille.

Pour la simulation par un transducteur il suffit dans le second cas d'introduire b() juste derrière a(ce qui se fait sans problème par un transducteur d'état fini.

$$\delta(q_0, X) = (q_0, X, D') \quad \forall \; X \neq *$$

$$\delta(q_0, *) = (q_1, \Lambda, D')$$

Insertion : on se place dans L' où il n'y a qu'un seul symbole pour un sommet

$$\delta(q_1, a) = (q_2, a\ b), D')$$

$$\delta(q_2, X) = (q_2, X, D') \quad \forall \; X \neq *$$

$$\delta(q_2, *) = (q_2, \Lambda, D')$$

L'état final de ce transducteur est q_2, il recopie jusqu'à la fin le mot d'entrée.

Pour la simulation du premier cas, il faut faire précéder le signe de fermeture correspondant à a par un signe de fermeture qui correspondra alors à b et introduire b derrière a. Il suffit encore d'un transducteur d'état fini :

$$\delta(q_0, X) = (q_0, X, D') \quad \forall \; X \neq *$$

$$\delta(q_0, *) = (q_1, \wedge, D')$$

$$\delta(q_1, a) = (q_2, a\ b, D')$$

On se place encore dans L'.

Il reste à recopier toute la sous-arborescence ayant pour racine a.

$$\delta(q_2, X) = (q_2, X, D') \quad \forall \; X \neq *$$

$$\delta(q_2, *) = (q_3, \wedge, D')$$

$$\delta(q_3,)) = (q_4,)), D')$$

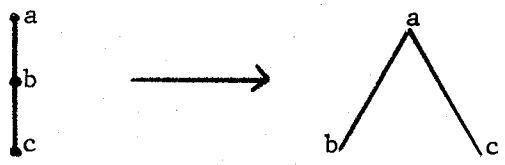
On a alors introduit un nouveau signe de fermeture qui correspond à b. L'état final de cet automate est q_4 et il ne reste plus qu'à recopier le reste du mot d'entrée :

$$\delta(q_4, X) = (q_4, X, D') \quad \forall \; X$$

b) <u>Transfert d'un noeud</u>

C'est la plus complexe des trois transformations élémentaires spéciales. Elle est divisée en deux catégories :

1 - Transfert vers le haut



en représentation linéaire

$$a(b(c())) \longrightarrow a(b()c())$$

Si C est une feuille elle peut être simulée par un transducteur d'état fini ; dans le cas général le problème est plus complexe.
On a :

$$a(w_1\ b(w_2\ c(w_3)\ w_4)\ w_5) \longrightarrow a(w_1\ b(w_2\ w_4)\ c(w_3)\ w_5)$$

Il faut donc transposer tous les descendants de c au-delà de w_4.
Cette opération peut être faite naturellement par une pile mais on aura alors $\tilde{w}_3$.

D'où une deuxième opération de pile nécessaire pour obtenir correctement w_3.

C'est pourquoi cette transformation peut être simulée par un transducteur à pile d'ordre 2 dans le cas d'un transducteur ordinaire.
Dans le cas d'un transducteur à pile il peut être seulement d'ordre 1.
C'est ce dernier transducteur que l'on emploiera dans ce cas.

La figure étant au milieu d'une arborescence on a, en fait, la décomposition plus générale

$$w_1\ a(w_2\ b(w_3\ c(w_4)\ w_5)\ w_6)\ w_7 \longrightarrow w_1\ a(w_2\ b(w_3\ w_5)\ c(w_4)\ w_6)\ w_7$$

Le transducteur est alors défini par :

Empilement de w_1 :

$$\delta(q_0, x, X) = (q_0, xX, \Lambda, D') \quad \forall\ x \neq *$$

$$\delta(q_0, *, X) = (q_1, X, \Lambda, D')$$

Empilement de $a(w_2$:

$$\delta(q_1, x, X) = (q_1, x\ X, \Lambda, D') \quad \forall\ x \neq *$$

$$\delta(q_1, *, X) = (q_2, X, \Lambda, D')$$

Empilement de $b(w_3$:

$$\delta(q_2, x, X) = (q_2, x\ X, \Lambda, D') \quad \forall\ x \neq *$$

$$\delta(q_2, *, X) = (q_3, X, \Lambda, D')$$

Sortie de $c(w_4)$:

$$\delta(q_3, x, X) = (q_3, X, x, D') \quad \forall\ x \neq *$$

$$\delta(q_3, *, X) = (q_4, X, \Lambda, D')$$

$$\delta(q_4,), X) = (q_5, X,), D')$$

Empilement de $w_5)$:

$$\delta(q_5, x, X) = (q_5, x\ X, \Lambda, D') \quad \forall\ x \neq *$$

$$\delta(q_5, *, X) = (q_6, X, \Lambda, D')$$

$$\delta(q_6,), X) = (q_7,)\ X, \Lambda, D')$$

Sortie de $w_6)w_7$:

$$\delta(q_7, x, X) = (q_7, X, x, D') \quad \forall\ x \neq *$$

$$\delta(q_7, *, X) = (q_7, X, \Lambda, D')$$

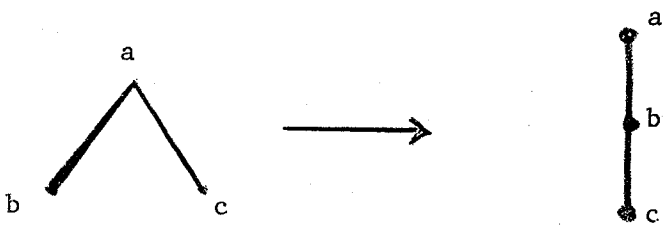
$$\delta(q_7, B, X) = (q_8, X, \Lambda, D')$$

Sortie de la pile devant le mot formé

$$\delta(q_8, \xi, xX) = (q_8, X, x, G') \qquad \forall\, x$$

L'état final est naturellement q_8

2 - Transfert vers le bas



en représentation linéaire

$$a(b()c()) \rightarrow a(b(c()))$$

Cette transformation n'est pas plus simple que la précédente, car on ne peut se contenter de reporter le signe de fermeture correspondant à b derrière c.

Exemple :

$$a(b()w_1c()) \rightarrow a(b(w_1\ c()))$$

w_1 qui avait pour racine a avant la transformation a maintenant pour racine b. La transformation ne doit pas affecter les différents descendants respectifs. D'une façon complète, on a la transformation

$$w_1\ a(w_2\ b(w_3)\ w_4\ c(w_5)\ w_6)\ w_7 \rightarrow w_1\ a(w_2\ b(w_3\ c(w_5))\ w_4\ w_6)\ w_7$$

On a encore un transfert de sous-arborescence à effectuer qui peut être pris en charge comme précédemment par un transducteur à pile ordinaire d'ordre deux.

On donnera le transducteur à pile d'ordre 1 qui simule cette transformation élémentaire spéciale.

Le transducteur est défini par :

Empiler w_1

$$\delta(q_0, x, X) = (q_0, x\ X, \Lambda, D') \quad \forall\, x \neq *$$

$$\delta(q_0, *, X) = (q_1, X, \Lambda, D').$$

Empiler $a(w_2$

$$\delta(q_1, x, X) = (q_1, xX, \wedge, D') \quad \forall\, x \neq *$$

$$\delta(q_1, *, X) = (q_2, X, \wedge, D')$$

Empiler $b(w_3$

$$\delta(q_2, x, X) = (q_2, xX, \wedge, D') \quad \forall\, x \neq *$$

$$\delta(q_2, *, X) = (q_3, X, \wedge, D')$$

Effacement de) et sortie de w_4

$$\delta(q_3,), X) = (q_4, X, \wedge, D')$$

$$\delta(q_4, x, X) = (q_4, X, x, D') \quad \forall\, x \neq *$$

$$\delta(q_4, *, X) = (q_5, X, \wedge, D')$$

Empilement de $c(w_5)$

$$\delta(q_5, x, X) = (q_5, xX, \wedge, D') \quad \forall\, x \neq *$$

$$\delta(q_5, *, X) = (q_6, X, \wedge, D')$$

$$\delta(q_6,), X) = (q_7,)) X, \wedge, D')$$

On rattrape la) perdue dans l'état q_3

Sortie de $w_6)$ w_7 :

$$\delta(q_7, x, X) = (q_7, X, x, D') \quad \forall\, x \neq *$$

$$\delta(q_7, *, X) = (q_7, X, \wedge, D')$$

$$\delta(q_7, B, X) = (q_8, X, \wedge, D')$$

Sortie de la pile devant le mot déjà sorti

$$\delta(q_8, \varepsilon, xX) = (q_8, X, x, G')$$

L'état final de ce transducteur est naturellement q_8

2 - SIMULATION D'UNE TRANSFORMATION ELEMENTAIRE

Par des transducteurs à pile au plus d'ordre 2, on peut simuler une transformation élémentaire spéciale. D'après le th 1 de Gladky et Mel'cuk, il suffit d'un nombre fini de transformations élémentaires spéciales pour simuler une transformation élémentaire quelconque. On en déduit la propriété.

Pour toute transformation élémentaire, il existe un transducteur à pile d'ordre fini qui la simule.

Remarque : Les trois premiers transducteurs sont utilisés pour reconnaître la sous-arborescence t_1, à laquelle doit s'appliquer la transformation élémentaire.

3 - TRANSFORMATION DYNAMIQUE

Les transducteurs qui simulent ces trois transformations élémentaires spéciales ne dépendent pas de la figure d'entrée mais de sa position. Pour qu'une de ces trois transformations puisse être dynamique, on a seulement besoin d'avoir une recherche de sous-arborescence dynamique. Problème qui est déjà étudié et résolu plus haut. Il suffit donc de donner :

a) le type de la transformation

b) la figure où elle doit s'appliquer.

Pour que la transformation puisse être dynamique, un transducteur d'état fini peut vérifier si le type de structure donnée correspond à la transformation à appliquer. On peut aisément supposer que cette vérification est inutile.

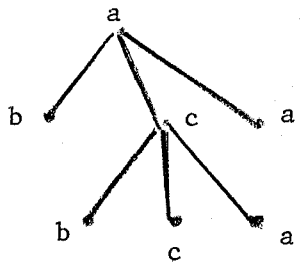
4 - AUTRES TRANSFORMATIONS

Il est assez pénible de décomposer chaque transformation à effectuer en une suite de transformations élémentaires spéciales. On peut se donner une série de transformations synthétiques. Ajoutées aux transformations de base elles n'augmentent pas la puissance de la grammaire transformationnelle mais en facilitent son emploi.

a) Suppression d'une figure :

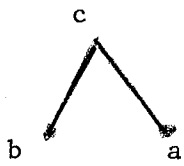
Cette transformation élémentaire est une transformation qui supprime une figure dans l'arborescence donnée.

Exemple - Soit l'arborescence :



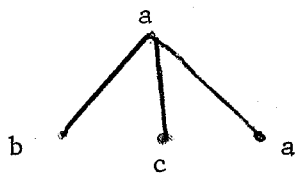
en représentation linéaire
a(b()c(b()c()a())a())

et soit la figure à supprimer :



c(b()a())

Après application de la transformation, on obtient l'arborescence :



a(b()c()a())

Cette transformation est très simple à réaliser. Il suffit d'enlever tous les points désignés dans la représentation linéaire.

Cette transformation est équivalente à une suite d'absorption de point si la racine de l'arborescence d'entrée n'est pas désignée.

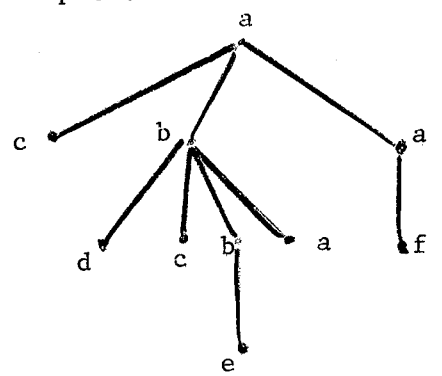
Dans ce dernier cas, on pourrait obtenir, après application de cette transformation, un graphe non connexe. C'est pourquoi on définit plutôt la transformation suivante.

b) Suppression simple

Cette transformation diffère de la précédente par le fait que :

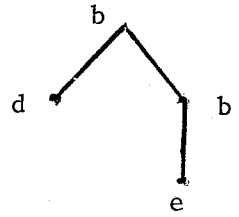
- elle ne traite que des sous-arborescences
- la racine de la sous-arborescence désignée n'est pas supprimée.

Exemple :



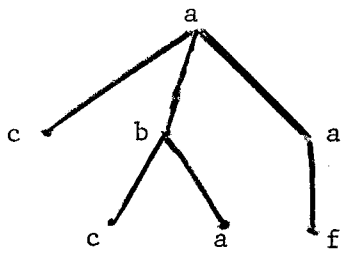
en représentation linéaire
a(c()b(d()c()b(e())a())a(f()))

et soit la sous-arborescence :



b(d()b(e())

Après la transformation, on obtient le graphe



a(c()b(c()a())a(f()))

Cette transformation est équivalente à une suite d'absorption de point, quel que soit la position de la figure.

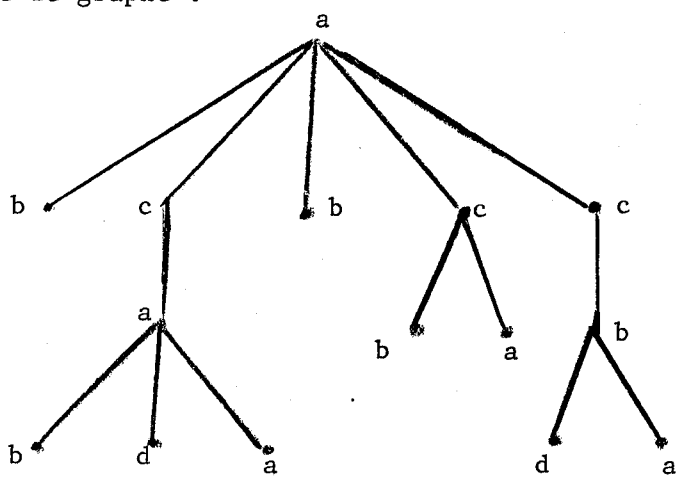
c) Suppression multiple

La recherche d'un schéma de figure dans un graphe fournit plusieurs sous-arborescences n'ayant pas de liens apparents entre elles.

La suppression d'une telle figure s'opère de la même manière que dans le cas d'une sous-arborescence en traitant les sous-arborescences reconnues une par une.

Cette suppression s'effectue toujours en commençant par la sous-arborescence la plus profonde et remonte jusqu'à la racine. La sous-arborescence la plus profonde est celle dont la racine est la plus profonde.

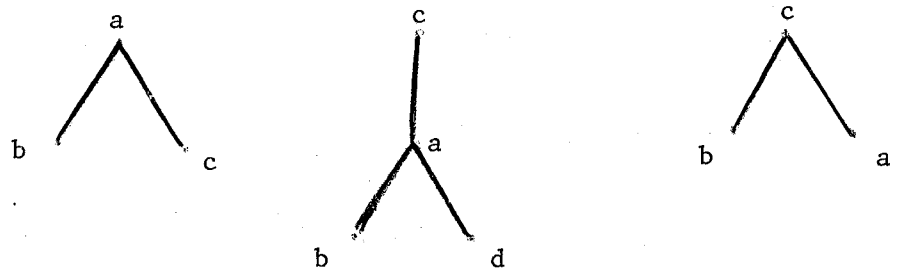
Exemple - Soit le graphe :



en représentation linéaire

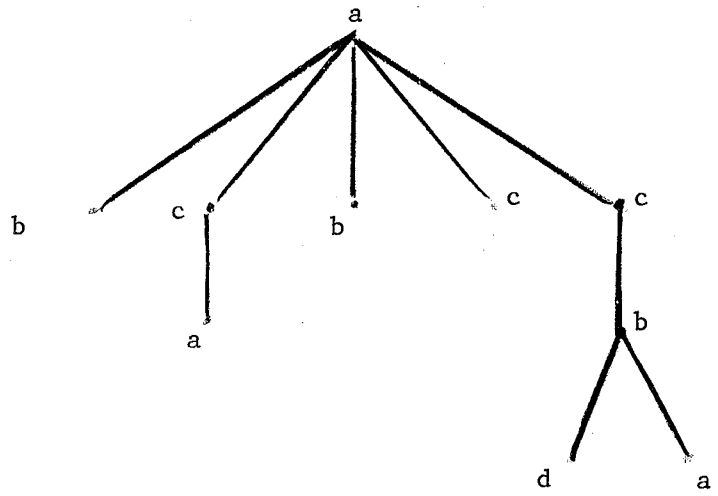
a(b()c(a(b()d()a()))b()c(b()a())c(b(d()a())))

et soit le schéma reconnu



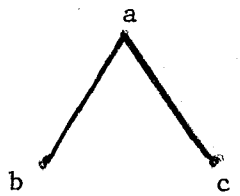
Les deux dernières sous-arborescences sont à la même hauteur, donc l'ordre de suppression est indifférent.

On obtient le graphe

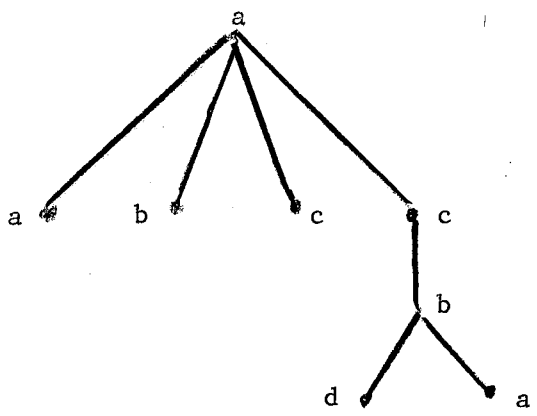


a(b()c(a())b()c()c(b(d()a())))

Il reste encore la figure



qui, après suppression, donne le graphe



a(a()b()c()c(b(d()a())))

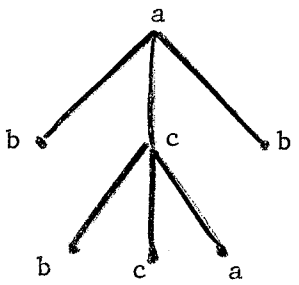
Dans le cas où les sous-arborescences se recoupent, les points ne sont supprimés qu'une seule fois.

Cette suppression est encore équivalente à une suite d'absorption de point mais elle est plus complète par la définition des points à supprimer.

d) Substitution

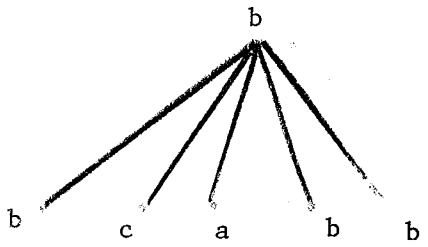
Cette transformation élémentaire supprime une figure et la remplace par une sous-arborescence donnée.

Exemple : soit l'arborescence :



a(b()c(b()c()a())b())

échange de a(b()c(c()a())) par b(b()c()a())
conduit à l'arborescence :



On a d'abord une suppression de la figure puis une adjonction de la nouvelle sous-arborescence. La racine de cette sous-arborescence est la plus haute racine des sous-arborescences de la figure reconnue.

Dans ce cas, cet échange ne conduit jamais à un ensemble non connexe d'arborescence si le graphe à ajouter n'est pas vide.

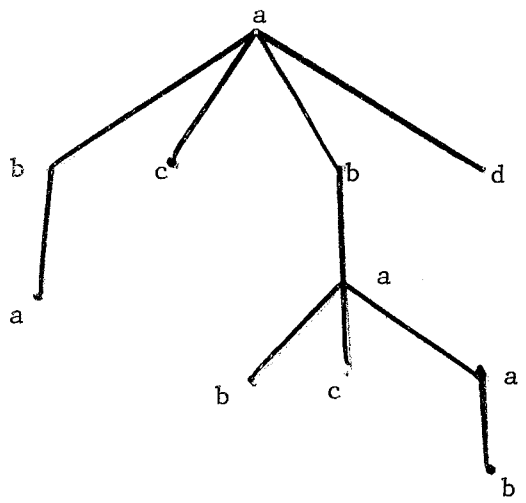
Si le graphe à ajouter est vide, on a alors la transformation équivalente à la suppression.

Cette transformation élémentaire est équivalente à une suite d'absorption et d'éclatement de noeuds.

e) Substitution simple

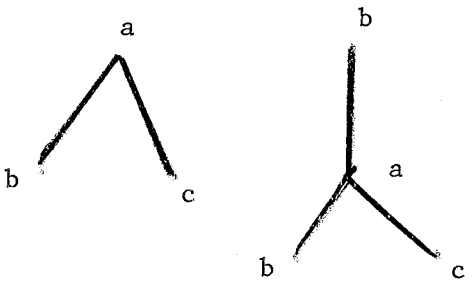
Cette transformation est la transposée de la suppression simple. La substitution se fait en gardant la racine de la sous-arborescence reconnue. Elle place l'arborescence à substituer sous cette racine. Elle est toujours équivalente à une suite d'absorptions et d'éclatement de noeud. On peut avoir une substitution multiple comme dans le cas de la suppression multiple. Dans ce cas, la sous-arborescence à introduire aura pour racine la plus haute racine de la figure reconnue.

Exemple - Soit l'arborescence :



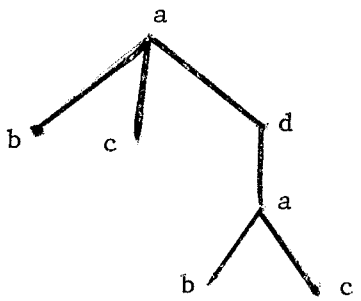
a(b(a())c()b(a(b()c()a(b())))d())

l'échange de



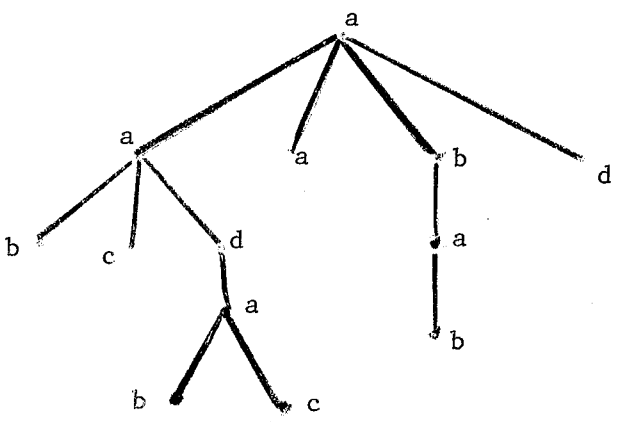
a(b()c())b(a(b()c()))

par



a(b()c()d(a(b()c())))

conduit à l'arborescence



a(a(b()c()d(a(b()c())))a()b(a(b()))d())

f) Autres transformations

Toutes ces transformations synthétiques ne font pas intervenir la transformation élémentaire spéciale "transfert d'un noeud".

Cette transformation est très puissante et ne peut être synthétisée que dans des cas particuliers. Citons par exemple le cas des manipulations de formules où elle pourrait intervenir pour faire la transformation $X/(Y/Z) \rightarrow (XZ)/Y$, X Y Z étant des expressions algébriques quelconques.

C H A P I T R E V

UNE APPLICATION :

MANIPULATION DE FORMULE ALGEBRIQUE SUR ORDINATEUR

1 - INTRODUCTION

L'utilisation d'un ordinateur pour le calcul algébrique donne une série de résultats numériques. Elle ne permet pas généralement la manipultation des formules fournissant ce résultat autrement que par le changement de programme. Pour remédier à cet inconvénient, des systèmes de calcul formel ont été étudiés. Tous ces systèmes ne manipulent les formules que comme des entités.

Ils ne peuvent sélectionner des sous-formules ni les remplacer par d'autres sous-formules. L'utilisation des grammaires transformationnelles définies ici permet cette manipulation. La représentation interne d'une formule algébrique sera alors une arborescence, comme dans la plupart des systèmes. Une sous-formule sera une sous-arborescence. Désigner une formule reviendra à rechercher une figure et l'échange s'effectuera par la transformation élémentaire de même nom.

Il est inutile de définir la suppression puisqu'elle peut se faire en échangeant une figure par une arborescence vide. Pour des besoins de programmation, le système de manipulation aura deux commandes pour effectuer un échange :

1) localisation de la sous-formule
2) échange par une autre formule.

L'échange peut se faire de deux façons :

- en gardant la racine
- sans garder la racine.

Le deuxième échange permet notamment de supprimer des sous-formules entières.

Le premier supprime des sous-formules également. Il est nécessaire pour garder l'opérateur principal.

Exemple - Soit la formule :

A. X * 2 + B. X + C

Le point représente le produit et l'étoile l'opération puissance.

Si on échange A.X * 2 + C par D, on obtiendra le graphe correspondant à B. X + D dans le premier cas (en gardant la racine) qui est la forme désirée. Dans le deuxième cas, on obtiendra un graphe non connexe B. X et D ne correspondant à aucune formule algébrique.

Alors que si on remplace B. X * 2 + C par une arborescence vide, on obtiendrait :

- + (B.X) dans le premier cas : graphe qui n'est pas satisfaisant algébriquement car l'opérateur + doit avoir deux opérandes.

- B.X dans le second cas : opération effectivement recherchée.

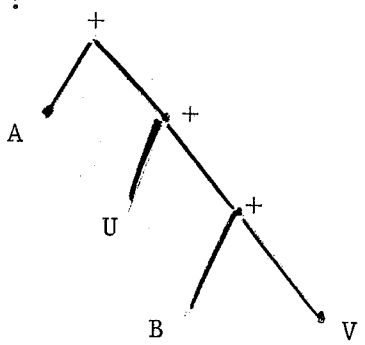
La décision de l'emploi de la commande appropriée est laissée au soin de l'utilisateur. Une méthode pour supprimer cet inconvénient est de toujours effectuer la substitution en gardant la racine et dans un deuxième temps absorber tous les opérateurs binaires qui n'ont qu'une opérande. Enfin, une grammaire transformationnelle de simplification peut être construite grâce à cette reconnaissance.

2 - REPRESENTATION INTERNE D'UNE EXPRESSION

L'expression algébrique doit être mise sous forme arborescente pour pouvoir être traitée. La notation post-fixée est équivalente à une représentation arborescente. On peut même dire que c'est une représentation linéaire de l'arborescence. Dans ce cas, les opérateurs binaires, comme + ou . , ont toujours deux opérandes, ce qui fait mal ressortir la propriété d'associativité de ces opérateurs. Dans une expression mise sous forme post-fixée ou sous forme arborescente, la recherche de la sous-formule + U V ne peut s'effectuer par une arborescence d'état fini car le lien entre U et V peut avoir une profondeur quelconque.

Exemple :

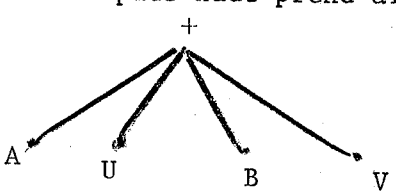
Soit A + U + B + V. On aura :



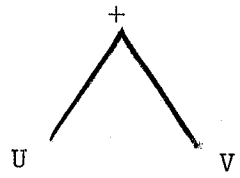
Il est évident qu'à la place de V, on peut avoir une somme et le graphe peut "s'étendre" en profondeur.

Pour cette raison, un opérateur + sera un opérateur ayant un nombre quelconque d'opérandes, avec un minimum de deux.

L'expression donnée plus haut prend alors la forme :



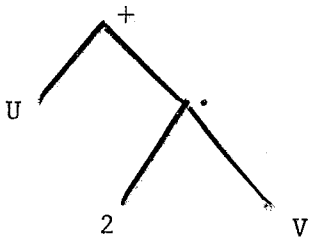
Et reconnaître l'expression U + V revient alors à rechercher la sous-arborescence



Une expression algébrique est composée de variables et de nombres. Les nombres, suivant où ils sont placés, jouent des rôles différents.

Exemple - Soit l'expression U + 2.V

Cette expression a la forme arborescente :



De par la nature de 2.V, la recherche de U + V dans ce cas doit pouvoir être obtenue. On a évidemment

$$U + 2V = U + V + V$$

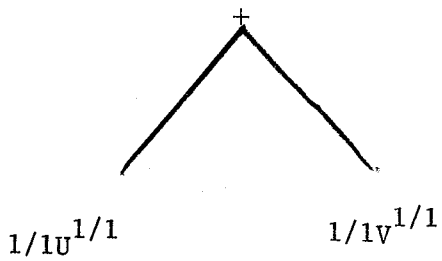
Le chiffre 2 ici est particulier car c'est un coefficient. Pour cette raison, à chaque variable de la formule, on associera en représentation interne

1) Un coefficient de la forme p/q

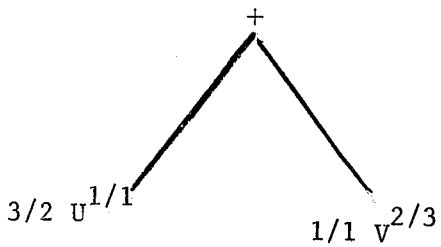
2) Un exposant de la forme r/s

Alors, on a les représentations suivantes :

$U + V$



et $\frac{3}{2} U + V^{2/3}$

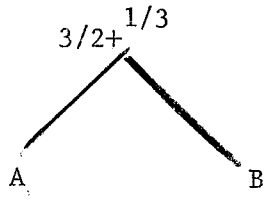


Une variable sous forme interne ne diffère des opérateurs que par le fait d'être toujours une feuille.

Un opérateur sera donc affecté également d'un coefficient et d'un exposant.

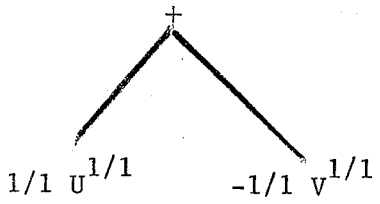
exp: $\frac{3}{2}\ (A+B)^{1/3}$

aura la forme :

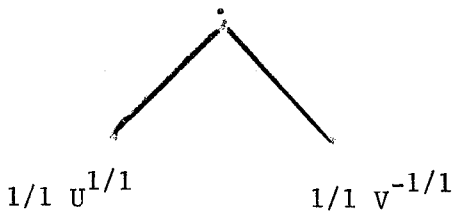


Le coefficient et l'exposant peuvent être négatifs. Il ressort de cette représentation que les opérateurs - et / sont inutiles.
On aura :

U - V →



U / V →



Représentation effective.

Chaque opérateur ou variable aura un code (un octet). Il peut donc y avoir au plus 256 opérateurs et variables différentes dans une formule. Si ce nombre devenait insuffisant, on pourrait, moyennant une petite perte de place, prendre deux octets pour coder un opérateur. On aurait alors 65 536 opérateurs et variables possibles.

L'exposant d'un opérateur occupe un mot (1/2 mot pour r et 1/2 pour s). On peut ainsi avoir un exposant variant entre $\pm$ 32 768 et $\pm \frac{1}{32768}$.

Le coefficient occupe un double mot (un mot pour p et un mot pour q). On peut alors avoir un coefficient variant entre $\pm$ 2 millions environ.

Exemple : la représentation interne de + est :

01 000 1 000 1 0000000 1 0000000 1 01 en hexadécimal.

Les arborescences sont représentées linéairement dans le langage L' donc à chaque opérateur ou variable est associé un symbole de fermeture qui est FE en hexadécimal.

Il y a encore trois valeurs spéciales possibles de l'octet qui ne représente pas un opérateur. Ce sont FC et FD employés dans la désignation des sous-formules.

Ces différentes contraintes réduisent à 252 le nombre d'opérateurs et de variables possibles.

3 - LES COMMANDES

Les commandes des systèmes commençant par D permettent d'avoir des Dumps des représentations internes des expressions.

D1 donne le Dump de l'expression principale, celle qui est traitée et conservée

D2 donne le Dump de l'expression secondaire, qui est par exemple la sous-expression à rechercher.

Dans l'exemple donné, on a fait apparaître ces dumps qui montrent bien la représentation interne.

Le passage d'une expression de la forme externe à la forme interne et réciproquement se fait par deux programmes qui sont basés sur les principes d'analyse d'une expression dans les différents langages évolués. Ils font intervenir une priorité d'opérateurs et deux piles : pile opérateur et pile opérande. On a alors une forme arborescente de l'expression. On obtient la forme définitive par une suite d'absorption des opérateurs convenables.

a) La commande d'entrée : e

Une expression est introduite dans le système et considérée comme expression principale par la commande "entrée" :e.

b) La commande de localisation : 1

Cette commande permet de reconnaître une sous-formule à l'intérieur de la formule principale. La sous-formule à localiser est entrée sur la ligne suivante de la commande 1 et la réponse de cette commande est :

- soit une ligne blanche si la sous-formule n'appartient pas à la formule principale

- soit cette sous-formule si elle a été localisée.

Cette commande met en jeu deux programmes. Le premier programme correspondant à l'initialisation de la table pour les automates de recherches. Cette initialisation se fait en lisant la sous-formule donnée.

La recherche proprement dite est effectuée par un transducteur à pile d'ordre deux qui travaille sur l'arborescence de l'expression principale.

Cette commande se termine en plaçant la sous-formule reconnue sur l'emplacement réservé aux formules non principales et en imprimant cette formule.

Pour pouvoir échanger deux sous-formules, il faut naturellement avoir localisé la première sous-formule au préalable.

c) La commande d'énumération des sous-formules : a

Dans une formule donnée, il peut exister plusieurs fois une même sous-formule. La commande de localisation va localiser une de ces occurrences. Il faut avoir un moyen d'énumérer toutes les occurrences d'une sous-formule pour s'arrêter sur celle désirée. Cette énumération est assurée par la commande a. Pour pouvoir obtenir l'occurrence suivante, il faut effacer provisoirement une variable localisée. Une sous-formule étant localisée, elle peut contenir des éléments d'une autre occurrence de la sous-formule. Si on effaçait provisoirement la sous-formule entière, il ne serait plus possible d'atteindre ces autres occurrences.

Cette commande fournit comme réponse :

Une ligne blanche si une nouvelle occurrence n'a pu être trouvée.

La sous-formule si on a pu localiser une nouvelle occurrence de cette sous-formule.

Exemple - Soit la formule :

A + X + A + X + X

et soit à reconnaître la sous-formule A + X en désirant les occurrences de A et X précédées du signe @ dans la formule écrite de la façon suivante :

A + X +@ A + X +@ X

La première commande de localisation fournit A + X et a localisé

@A + @X + A + X + X

Si on demande alors la commande a, on a deux possibilités :

l'effacement de A fournit la formule suivante :

A + X + @A + @X + X

l'effacement de X fournit la formule suivante :

@A + X + A + @X + X

Pour le but à atteindre dans ce cas, la première méthode est plus rapide, il suffit de recommencer une commande a en effaçant X. On obtient alors : A + X + @A + X + @X

Si on avait effacé de nouveau A, il n'y aurait plus d'occurrence possible et la réponse serait une ligne blanche.

Dans le deuxième cas, on pourrait avoir les commandes suivantes :
"a" en effaçant X, puis "a" en effaçant A

"a" en effaçant A, puis "a" en effaçant X.

Ces deux suites de commandes fournissent le même résultat, mais n'énumèrent pas les mêmes occurrences des sous-formules possibles. Il y a cinq sous-formules possibles dans ce cas. Une fois que la suite des commandes "a" est terminée, la figure principale est restituée avec pour sous-formule désignée la dernière sous-formule localisée par la dernière commande "a". Dans le cas où cette dernière commande ne fournit aucun résultat, il n'y a aucune désignation même s'il y avait des occurrences de la sous-formule localisées auparavant.

d) Les commandes d'échanges de sous-formules : c et cs

Une sous-formule étant localisée, la manipulation de la formule principale devient effective si on peut :

- supprimer cette sous-formule
- la remplacer par une autre sous-formule.

Cette manipulation est rendue possible par ces deux dernières commandes.

L'une diffère de l'autre par le fait que C détruit complètement la sous-formule avant de la remplacer par la nouvelle. CS ne détruit pas la racine de la sous-formule désignée.

Dans le premier cas (c), la racine de la sous-formule à introduire est à la même position que la racine de la sous-formule désignée et détruite.

Dans le second cas (cs), la racine de la sous-formule à introduire a pour racine la racine de la sous-formule désignée qui n'est pas détruite.

Intérêt :

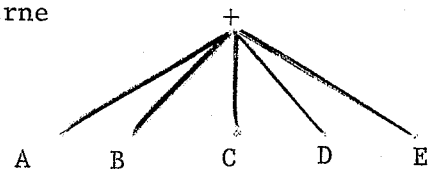
- si on veut supprimer complètement une somme ou un produit de termes, il faut employer c.

si on veut par contre supprimer une partie d'une somme ou d'un produit, il faut employer cs pour pouvoir garder comme racine la somme ou le produit pour les opérandes restantes.

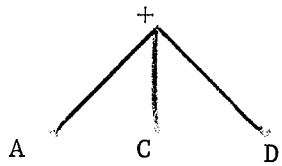
Exemple :

1) Soit la formule A + B + C + D + E

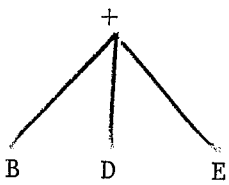
en représentation interne



Si on veut supprimer la sous-formule A + C + D



Il restera comme formule principale :

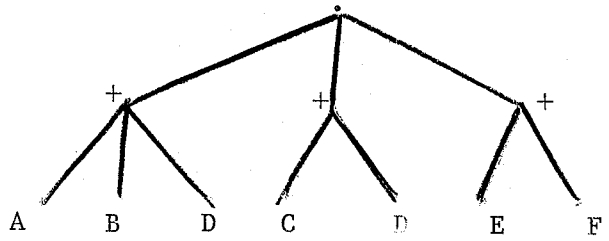
α) 

si on emploie la commande cs

β) B D E graphe non connexe n'ayant plus la signification d'une formule si on emploie c.

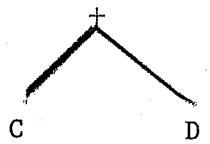
2) Soit la formule (A + B + D).(C + D).(E + F)

en représentation interne

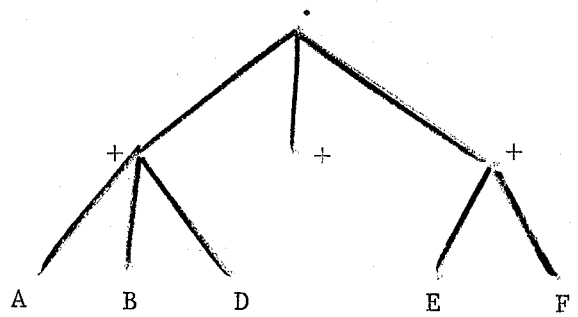


Si on veut supprimer la formule C + D

Il restera comme formule principale :

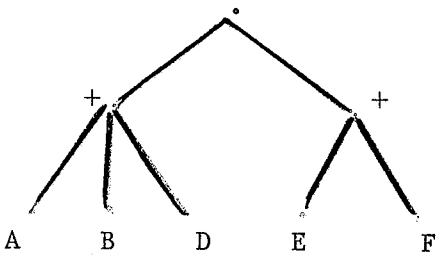


α)



Si on emploie cs. On a alors un graphe qui n'a pas de signification algébrique. (Un opérateur + n'a plus d'opérande).

β)



Si on emploie c, formule désirée effectivement.

Le problème se pose comme on le voit pour les opérateurs + et . seulement. Pour les autres opérateurs ou opérandes, il faut tout remplacer y compris la racine pour avoir une signification algébrique. On pourrait remédier à cet inconvénient par l'emploi de la règle systématique : absorption des opérateurs + ou . qui ont moins de deux opérandes. Pour les manipulations faisant intervenir un + ou . en racine de la sous-formule, on aurait alors toujours cs et dans les autres cas c.

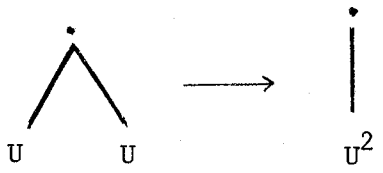
c) La commande imprime : I

Cette commande imprime la formule principale.

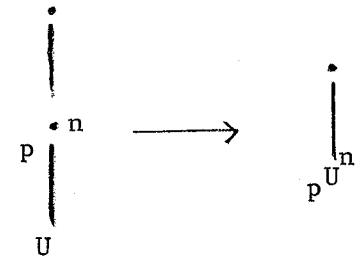
d) La commande de simplification S

Cette commande simplifie la formule principale avec les règles habituelles de l'algèbre. C'est en fait une grammaire transformationnelle qui effectue une suite de reconnaissances et de remplacements successifs. La grammaire s'arrête lorsqu'il n'y a plus de partie gauche de règle qui puisse être reconnue, donc lorsque la formule est complètement simplifiée.

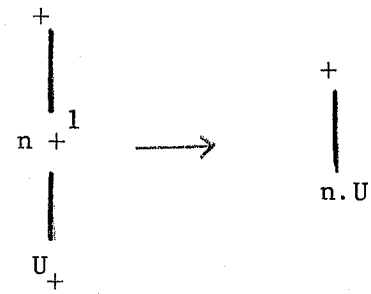
Les règles de cette grammaire sont, U étant une arborescence quelconque :

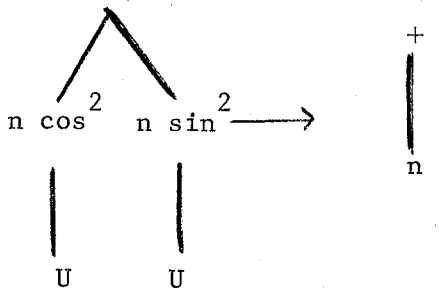


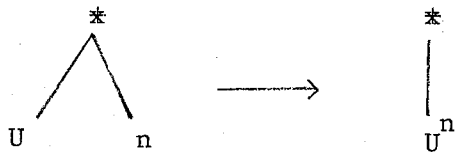
si U n'a pas pour racine un produit



si U n'a pas pour racine un plus à la puissance un.







Autres règles :

- tout opérateur binaire qui n'a plus qu'un descendant est absorbé. Son coefficient et sa puissance sont alors multipliés au coefficient et à la puissance de son unique descendant.

- règles de simplification des coefficients (règles sur les étiquettes). Le coefficient et la puissance qui sont de la forme p/q sont simplifiés par le PGCD du numérateur et du dénominateur.

- règles de simplification des coefficients sous les opérateurs + et . :

* sous le + : son coefficient est multiplié par le PGCD de tous ses descendants qui sont alors simplifiés

* sous le . : a) son coefficient est multiplié par le produit des coefficients de ses descendants qui sont alors positionnés à 1.

b) sa puissance est multipliée par le PGCD des puissances des descendants qui sont alors simplifiées.

Remarque - Chaque règle est applicable si U est une sous-arborescence complète de l'arborescence principale, c'est-à-dire que pour tout point X de la formule principale, si X est un descendant de la racine de U alors X est dans U.

4 - REMARQUES GENERALES SUR LE SYSTEME

Comme on le voit dans l'exemple qui suit,ce système n'est pas conçu pour permettre une utilisation courante.

Les commandes ont été simplifiées au maximum et pour effectuer une manipulation, il faut souvent en effectuer plusieurs. L'entrée des formules se fait toujours sur la ligne suivante de la commande, ceci afin de ne pas compliquer l'analyse de la chaîne d'entrée.

Le but essentiel de ce système est de montrer une application possible de ce mode de transduction d'arborescence et d'en donner une démonstration effective.

L'adaptation des commandes pour en former un outil plus souple est un problème de programmation qui n'influence pas le mode de fonctionnement. De même, une liaison avec les systèmes de calcul formel existant permettrait d'étendre ceux-ci. Enfin, la recherche de sous-structure donnée permet d'effectuer plusieurs types de calculs difficiles jusqu'à présent comme l'intégration de formules.

5 - REPONSE DU SYSTEME

Les entrées du système peuvent se faire en caractère majuscule ou minuscule, les réponses sont toujours en caractères majuscules,aussi dans l'exemple de fonctionnement suivant, on a employé les conventions :

- les entrées sont en minuscules
- les réponses sont en majuscules.

Réponse aux différentes commandes :

e : expression entrée et considérée comme principale jusqu'à la prochaine commande "e" où elle sera détruite

l : expression localisée si elle a été trouvée sinon une ligne blanche

a : même chose que pour l

i : impression de la formule considérée à ce moment comme principale

c : la nouvelle formule principale une fois la transformation effectuée

cs : même chose qu'en c

Dl : le dump de la formule principale

S : formule simplifiée

D2 : le dump d'une formule secondaire qui est :

- soit la formule à localiser si D2 suit une commande l ;
- soit la formule à inclure si D2 suit une commande c ;
- soit la formule à effacer si D2 suit une commande a.

Pour les Dumps, les codes sont :

01 : + Plus	06 : Sinus	0A : a
02 : Produit	07 : Cosinus	21 : x
03 : ** Puissance	08 : Tangente	22 : y
04 : e exponentiell	09 : Cotangente	0B : b
05 : LOG Logarithme		23 : Z

Les autres codes (de variables nouvelles notamment) sont construits par le programme.

```
CP/67 II.1b a votre service

l chauche
ENTER PASSWORD:
********
READY AT 09.59.31 ON 01/24/71
CP
ipl cms
CMS..VERSION 24 nov 70


09.59.44 GLOBAL M SYSLIB CHAMACRO
09.59.47 GLOBAL T SYSLIB CHAPROG1
R; T=0.06/0.22 09,59.47

load nouva
R; T=0.23/0.69 10.00.04

start phar
EXECUTION BEGINS...
e                    entrée d'une expression
a.x*2+b.x+c|
A.X*2+B.X+C
l                    localisation d'une sous formule
a.x*2|
A.X*2
c                    changement d'une sous-formule
a.x+b.x*2+c|
A.X+B.X*2+C+B.X+C
l
b.x*2+c|
B.X*2+C
a                    énumération de toutes les occurences de la sous-formule
c|
B.X*2+C
a
c|
                     il y a plus d'occurence de la sous-formule. toutes ont été énumérées
l                    impression de la formule principale => réinitialisation de la formule
A.X+B.X*2+C+B.X+C                                          pour les occurences de B.X*2+C
l
b.x*2+c|
B.X*2+C
a
c|
B.X*2+C
cs                   changement simple (avec conservation de la racine)
log(a.x+b.x)|
LOG(A.X+B.X)+A.X+B.X+C
l
a.x+b.x|
A.X+B.X
a
a|
A.X+B.X
cs
sin(a.x+b.x+tg(a.x+b.x-c))|
SIN(A.X+B.X+TG(A.X+B.X-C))+LOG(A.X+B.X)+C

l
a.x+b.x-c|
A.X+B.X-C
c
a.x*2+b.x|
SIN(A.X+B.X+TG(A.X*2+B.X))+LOG(A.X+B.X)+C
l
a.x+b.x|
A.X+B.X
a
a|
A.X+B.X
c
x|
SIN(A.X+B.X+TG(A.X*2+B.X))+LOG(X)+C
l
tg(a.x*2+b.x)|
TG(A.X*2+B.X)
c                    changement par un graphe vide => suppression
|
SIN(A.X+B.X)+LOG(X)+C
l
sin(a.x+b.x)+log(x)|
SIN(A.X+B.X)+LOG(X)
cs
a.x*2+b.x|
A.X*2+B.X+C
d1                   dump de la formule principale
DUMP DU GRAPHE D'UNE EXPRESSION
 0100010001000000010000000101 0200010001000000010000000102 0A000100010000000100000010A
 FE 2100020001000000010000000121 FE FE 0200010001000000010000000102 0B000100010000000100000010B
 FE 2100010001000000010000000121 FE FE 0C000100010000000100000010C FE FE
```

```
e
log(a.x*2+y*z-tg(sin(a.x+b)/cos(a.x-b)))|
LOG(A.X*2+Y*Z-TG(SIN(A.X+B).(1/COS(A.X-B))))
d1
DUMP DU GRAPHE D'UNE EXPRESSION
 0500010001000000010000000105 0100010001000000010000000101 0200010001000000010000000102
 0A0001000100000001000000010A FE 2100020001000000010000000121 FE FE 0300010001000000010000000103
 2200010001000000010000000122 FE 2300010001000000010000000123 FE FE 0800010001FFFFFFFF0000000108
 0200010001000000010000000102 0600010001000000010000000106 0100010001000000010000000101
 0200010001000000010000000102 0A0001000100000001000000010A FE 2100010001000000010000000121
 FE FE 0B0001000100000001000000010B FE FE FE 07FFFF0001000000010000000107
 0100010001000000010000000101 0200010001000000010000000102 0A0001000100000001000000010A
 FE 2100010001000000010000000121 FE FE 0B00010001FFFFFFFF0000000010B FE FE
 FE FE FE FE FE
l
a.x*2+y*z|
A.X*2+Y*Z
d2          dump de la formule secondaire : ici celle qui a été localisée
DUMP DU GRAPHE D'UNE EXPRESSION
 0100010001000000010000000101 0200010001000000010000000102 0A0001000100000001000000010A
 FE 2100020001000000010000000121 FE FE 0300010001000000010000000103 2200010001000000010000000122
 FE 2300010001000000010000000123 FE FE FE
e
a.x*3+b.x*2+c.x+d|
A.X*3+B.X*2+C.X+D
l
a.x*3+c.x|
A.X*3+C.X



d1          dump montrant la désignation d'une sous formule chaque point désigné est précédé de FD
DUMP DU GRAPHE D'UNE EXPRESSION
 FD 0100010001000000010000000101 FD 0200010001000000010000000102 FD 0A0001000100000001000000010A
 FD FE FD 2100030001000000010000000121 FD FE FD FE 0200010001000000010000000102
 0B0001000100000001000000010B FE 2100020001000000010000000121 FE FE FD 0200010001000000010000000102
 FD 0C0001000100000001000000010C FD FE FD 2100010001000000010000000121 FD
 FE FD FE 0D0001000100000001000000010D FE FD FE
a
c|

d1          dump montrant l'effacement provisoire de C (valeur entière 0C.) par un symbole spécial FA
DUMP DU GRAPHE D'UNE EXPRESSION
 0100010001000000010000000101 0200010001000000010000000102 0A0001000100000001000000010A
 FE 2100030001000000010000000121 FE FE 0200010001000000010000000102 0B1000100000001000000010B
 FE 2100020001000000010000000121 FE FE 0200010001000000010000000102 FA0001000100000001000000001FA
 FE 2100010001000000010000000121 FE FE 0D0001000100000001000000010D FE FE

l           commande d'impression qui a pour effet dans ce cas de remplacer C à la formule
A.X*3+B.X*2+C.X+D
d1          dump montrant C remplacé
DUMP DU GRAPHE D'UNE EXPRESSION
 0100010001000000010000000101 0200010001000000010000000102 0A0001000100000001000000010A
 FE 2100030001000000010000000121 FE FE 0200010001000000010000000102 0B0001000100000001000000010B
 FE 2100020001000000010000000121 FE FE 0200010001000000010000000102 0C0001000100000001000000010C
 FE 2100010001000000010000000121 FE FE 0D0001000100000001000000010D FE FE

l           arrêt du programme
R; T=0.69/1.77 10.13.50

CP
log
CONNECT= 00.14.27  VIRTCPU= 000.00.98  TOTCPU= 000.02.85
LOGOUT AT 10.13.58 ON 01/24/71
```

```
CP/67 II.1b a votre service

1 chauche
ENTER PASSWORD:
********
READY AT 08.23.10 ON 02/25/71
CP
ipl cms
CMS..VERSION 2.1a (fev 71)


08.23.29 GLOBAL M SYSLIB CHAMACRO
08.23.31 GLOBAL T SYSLIB CHAPROG1
R; T=0.06/0.28 08.23.32

load nouva
R; T=0.25/1.3 4 08.23.55

start phar
EXECUTION BEGINS...
e
2x+3y-4z+5t+(3/2).zt*4+6r*3|
2.X+3.Y-4.Z+5.T+(3/2).ZT*4+6R*3
1        reconnaissance de x+r qui ne sera pas trouver car il y a r*3
x+r|

1    reconnaissance de x+r*3 qui seara effictivement trouver
x+r*3|
2.X+6R*3
1  on considere le cas ou le coefficient est fractonnaire
x+(3/2).zt*4|
2.X+(3/2).ZT*4
1   il faut toujours dans ce cas specifier le coefficient
x+zt*4|

e   ces regles de reconnaissance sont valable n'importe ou
log(x*2+3y*3)|
LOG(X*2+3Y*3)
1   reconnaissance de x+y , non reconnue car on a x*2+y*3
x+y|

1  x*2+y*3 par contre est toujours reconnue
x*2+y*3|
X*2+3Y*3
1   on aura pas log(x*2+y*3) mais log(x*2+3y*3)
log(x*2+y*3)|

1
log(x*2+3y*3)|
LOG(X*2+3Y*3)
|
R; T=0.22/2.77 08.33.09

CP
log
CONNECT= 00.10.06  VIRTCPU= 000.00.54  TOTCPU= 000.04.49
LOGOUT AT 08.33.16 ON 02/25/71
```

```
CP-67 V3.L0 a votre service

l chauche
ENTER PASSWORD:
********
READY AT 23.06.32 ON 03/19/71
CP
i cms
CMS..VERSION 2.1a (fev 71)


23.06.45 GLOBAL M SYSLIB CHAMACRO
23.06.48 GLOBAL T SYSLIB CHAPROG1
R; T=0.04/0.17 23.06.48

load nouva
R; T=0.26/0.83 23.07.50

start phar
EXECUTION BEGINS...
e
x+x|
X+X
s
2X
e
x.x|
X.X
s
X*2
e
(x+y).(x+y).(x+y)|
(X+Y).(X+Y).(X+Y)
s
(X+Y)*3
e
log(sin(x+y)+sin(x+y)+tg(x*2).tg(x*2).c0s(x)¢
log(sin(x+y)+sin(x+y)+tg(x*2).tg(x*2).cos(x))+log(2sin(x+y)+tg(x*2)*2.cos(x))|
LOG(SIN(X+Y)+SIN(X+Y)+TG(X*2).TG(X*2).COS(X))+LOG(2.SIN(X+Y)+TG(X*2)*2.COS(X))
s
2.LOG(2.SIN(X+Y)+TG(X*2)*2.COS(X))
e
log(sin(x+y*2)+tg(x).tg(x))+log(sin(x+y*2)+tg(x)*2)|
LOG(SIN(X+Y*2)+TG(X).TG(X))+LOG(SIN(X+Y*2)+TG(X)*2)
s
2.LOG(SIN(X+Y*2)+TG(X)*2)
e
a.x*2+b.x+c|
A.X*2+B.X+C
s
A.X*2+B.X+C
l
a.x*2+c|
A.X*2+C
cs
b.x+d|
B.X+D+B.X
s
2.B.X+D
|
R; T=0.81/1.66 23.13.56

CP
log
CONNECT= 00.07.32  VIRTCPU= 000.31.12  TOTCPU= 000.02.84
LOGOUT AT 23.14.04 ON 03/19/71
```

```
CP-67 V3.L0 a votre service

l chauche
ENTER PASSWORD:
********
READY AT 10.19.42 ON 03/23/71
CP
i cms
CMS..VERSION 2.1a (fev 71)

load nouva
10.20.09 GLOBAL M SYSLIB CHAMACRO
10.20.12 GLOBAL T SYSLIB CHAPROG1
R; T=0.32/1.36 10.20.47

start phar
EXECUTION BEGINS...
e
x-x|
X-X
s
0
e
x/x|
X/X
s
1
e
2x+6y+4z|
2X+6Y+4Z
s
2.(X+3Y+2Z)
e
sin(x)*2+cos(x)*2|
SIN(X)*2+COS(X)*2
s
1
e
3cos(x)*2+3sin(x)*2|
3.COS(X)*2+3.SIN(X)*2
s
3
e
sin(x)*2+3cos(x)*2|
SIN(X)*2+3.COS(X)*2
s
SIN(X)*2+3.COS(X)*2
e
sin(2x+6y++4z)*2+cos(2(xc
sin(2x+6y+4z).sin(2.(x+3y+2z))+cos(2x+6y+4z)*2|
SIN(2X+6Y+4Z).SIN(2.(X+3Y+2Z))+COS(2X+6Y+4Z)*2
s
1
e
sin(log(a.x*2+tg(x)+cos(x)*2+b.x+sin(x)*2)-log(x))*2+cos(log(a.x*2+tg(x)+1+b.x)-log(x))*2|
SIN(LOG(A.X*2+TG(X)+COS(X)*2+B.X+SIN(X)*2)-1.LOG(X))*2+COS(LOG(A.X*2+TG(X)+1+B.X)-1.LOG(X))*2
s
1
|
R; T=1.01/2.08 10.26.48

CP
log
CONNECT= 00.07.17  VIRTCPU= 000.01.33  TOTCPU= 000.03.02
LOGOUT AT 10.26.58 ON 03/23/71
```

C O N C L U S I O N

L'intérêt de ce mode de définition et de traitement des arborescences est l'apparition du moyen de reconnaissance des schémas de figures. Il permet de définir un ensemble d'arborescences de manière synthétiques. C'est aussi la manipulation de sous-arborescences indépendamment de l'arborescence principale. Les applications possibles sont :

- évidemment tous les traitements d'arborescence dans l'étude des langues naturelles ou des langages de programmation.

- la manipulation de fichiers structurés où les modifications ne doivent s'appliquer que sur des parties de fichiers.

- enfin comme dans l'exemple donné, un moyen plus grand de manipuler des formules algébriques. Ce qui permet d'augmenter les possibilités du calcul formel sur ordinateur.

B I B L I O G R A P H I E

[1] Gladky et Mel'cuk

Tree grammars (Δ grammars)

International conference on computational Linguistics.
Classification AL 32 14 Septembre 1969 Sanga-Säby Sweden

[2] Cinsburg et Rose

Préservation of Languages by transducers

Information and control 9 (1966) 153-176

[3] Hopcroft et Uhlman

Formal languages and their relation to automata

Addison-Wesley 1969

[4] Kuntzmann

Cours : Théorie des graphes

Grenoble 1968

[5] Vauquois

Cours : automates et grammaires formelles

Grenoble 1968

Sur la définition des grammaires transformationnelles et la manipulation d'arborescences.

[6] Barbault et Descles

Etude structurelle et catégorielle du système de transition avec application à la science du calcul et à la linguistique mathématique

Thèse Paris Janvier 1970.

[7] Barry K. Rosen

Tree-manipulating systems and Chunch-Roser Theorems

Haivard University 02 138

[8] Brainerd W.S.

Tree generating systems and tree automata

Ph. Dissertation Purdue University June 1967.

[9] Brainerd W.S.

The minimalization of tree automata

Information and Control 13 (1968) 484-491

[10] Brainerd W.S.

Tree generating regular systems

Information and Control 14 (1969) 217-231

[11] Eilenberg et Wright

Automata in général algebras

Information and Control 11 (1967) 217-231

[12] Pair et Quere

Définition et études des bilangages réguliers

Information and Control 13 (1968) 563-593

[13] Quere

Etude des ramifications et des bilangages

Thèse Nancy Juillet 1969

[14] Rounds W

Tree, transducers, and transformations

Ph D thesis - Standford Unice 1968

[15] Rounds W.

Context free grammars on trees

ACM Symp. on theory of computing (1969) 143-148

[16] Teatcher J.W.

Characteriging derivation tree of context-free grammars through a generalization of finite automata theory

J. Comp. Sys. Sci Vol 1 n° 4 (Dec 1967) 317-322

[17] Teatcher et Wright

Generalized finite automata theory with an application to a decision problem of second-order logic

Mathematical Syst. theory Vol 2 n° 1 (1968) 57-81

[18] Teatcher

Transformation and translation from the point of view of generalized finite automata theory

ACM Synp. on theory of computing (1969) 129-142

[19] Veillon Veyrunes Vauquois

Un métalangage de grammaires transformationnelles

Document CETA Janvier 1967

Sur le calcul formel.

[20] Engelman C.

Mathlab : A program for on-line machine assistance in symbolic computations

Proc. FJCC Nov. 1965

[21] Siret Y.

Contribution and calcul formel sur ordinateur

Thèse Grenoble Juillet 1970

Sur le Programmation.

[22] IBM System/360 Principles of operation

[23] IBM System/360 Assembler Programmer's guide

[24] IBM System/360 Assembler language

[25] CP/CMS User's guide.

VU

Grenoble, le

Le Président de la Thèse

VU

Grenoble, le

Le Doyen de la Faculté des Sciences

VU, et permis d'imprimer
Le Recteur de l'Académie de GRENOBLE