T H E S E

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR DE 3ÈME CYCLE

(INFORMATIQUE)

par

Fatholah ZABETI

PROBLEMES D'EQUIVALENCE ENTRE SCHEMAS DE PROGRAMMES

Soutenue le Lundi 19 Septembre 1983 devant la Commission d'Examen

Membres du Jury	Mme F. PECAUT	Présidente
	MM. G. JACOB	Rapporteur
	G. COUSINEAU	Examinateur
	R. MICHEL	Invité
	G. BLANC	Invité

PROFESSEURS 1ère CLASSE

M.	BACCHUS Pierre	Mathématiques
M.	BEAUFILS Jean-Pierre (dét.)	Chimie
M.	BIAYS Pierre	G.A.S.
M.	BILLARD Jean	Physique
M.	BONNOT Ernest	Biologie
M.	BOUGHON Pierre	Mathématiques
M.	BOURIQUET Robert	Biologie
M.	CELET Paul	Sciences de la Terre
M.	COEURE Gérard	Mathématiques
M.	CONSTANT Eugène	I.E.E.A.
M.	CORDONNIER Vincent	I.E.E.A.
M.	DEBOURSE Jean-Pierre	S.E.S.
M.	DELATTRE Charles	Sciences de la Terre
M.	DURCHON Maurice	Biologie
M.	ESCAIG Bertrand	Physique
M.	FAURE Robert	Mathématiques
M.	FOURET René	Physique
M.	GABILLARD Robert	I.E.E.A.
M.	GRANELLE Jean-Jacques	S.E.S.
M.	GRUSON Laurent	Mathématiques
M.	GUILLAUME Jean	Biologie
M.	HECTOR Joseph	Mathématiques
M.	HEUBEL Joseph	Chimie
M.	LABLACHE COMBIER Alain	Chimie
M.	LACOSTE Louis	Biologie
M.	LANSRAUX Guy	Physique
M.	LAVEINE Jean-Pierre	Sciences de la Terre
M.	LEBRUN André	C.U.E.E.P.
M.	LEHMANN Daniel	Mathématiques
Mme	LENOBLE Jacqueline	Physique
M.	LHOMME Jean	Chimie
M.	LOMBARD Jacques	S.E.S.
M.	LOUCHEUX Claude	Chimie
M.	LUCQUIN Michel	Chimie

M.	MAILLET Pierre	S.E.S.
M.	MONTREUIL Jean	Biologie
M.	PAQUET Jacques	Sciences de la Terre
M.	PARREAU Michel	Mathématiques
M.	PROUVOST Jean	Sciences de la Terre
M.	SALMER Georges	I.E.E.A.
Mme	SCHWARTZ Marie-Hélène	Mathématiques
M.	SEGUIER Guy	I.E.E.A.
M.	STANKIEWICZ François	Sciences Economiques
M.	TILLIEU Jacques	Physique
M.	TRIDOT Gabriel	Chimie
M.	VIDAL Pierre	I.E.E.A.
M.	VIVIER Emile	Biologie
M.	WERTHEIMER Raymond	Physique
M.	ZEYTOUNIAN Radyadour	Mathématiques

PROFESSEURS 2ème CLASSE

M.	AL FAKIR Sabah	Mathématiques
M.	ANTOINE Philippe	Mathématiques
M.	BART André	Biologie
Mme	BATTIAU Yvonne	Géographie
M.	BEGUIN Paul	Mathématiques
M.	BELLET Jean	Physique
M.	BKOUCHE Rudolphe	Mathématiques
M.	BOBE Bernard	S.E.S.
M.	BODART Marcel	Biologie
M.	BOILLY Bénoni	Biologie
M.	BONNELLE Jean-Pierre	Chimie
M.	BOSQ Denis	Mathématiques
M.	BREZINSKI Claude	I.E.E.A.
M.	BRUYELLE Pierre (Chargé d'enseignement)	Géographie
M.	CAPURON Alfred	Biologie
M.	CARREZ Christian	I.E.E.A.
M.	CHAMLEY Hervé	E.U.D.I.L.

M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean-Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
Mle DACHARRY Monique	Géographie
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DEMUNTER Paul	C.U.E.E.P.
M. DE PARIS Jean-Claude	Mathématiques
M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean-Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique
M. DUEE Gérard	Sciences de la Terre
M. DYMENT Arthur	Mathématiques
M. FLAMME Jean-Marie	E.U.D.I.L.
M. FONTAINE Hubert	Physique
M. GERVAIS Michel	S.E.S.
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gabriel	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HANGAN Théodore	Mathématiques
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LAURENT François	I.E.E.A.
Mle LEGRAND Denise	Mathématiques
Mle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques

M. LEMAIRE Jean	Physique
M. LENTACKER Firmin	G.A.S.
M. LEVASSEUR Michel	I.P.A.
M. LHENAFF René	G.A.S.
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	I.E.E.A.
M. LOUAGE Francis	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
Mle MARQUET Simone	Mathématiques
M. MESSELYN Jean	Physique
M. MIGEON Michel	E.U.D.I.L.
M. MIGNOT Fulbert	Mathématiques
M. MONTEL Marc	Physique
Mme NGUYEN VAN CHI Régine	G.A.S.
M. PARSY Fernand	Mathématiques
Mle PAUPARDIN Colette	Biologie
M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L.
M. RACZY Ladislas	I.E.E.A.
M. RICHARD Alain	Biologie
M. RIETSCH François	E.U.D.I.L.
M. ROGALSKI Marc	M.P.A.
M. ROUSSEAU Jean-Paul	Biologie
M. ROY Jean-Claude	Biologie
M. SALAMA Pierre	S.E.S.
Mme SCHWARZBACH Yvette (CCP)	M.P.A.
M. SCHAMPS Joël	Physique
M. SIMON Michel	S.E.S.
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole

M.	TOULOTTE Jean-Marc	I.E.E.A.
M.	VANDORPE Bernard	E.U.D.I.L.
M.	WALLART Francis	Chimie
M.	WATERLOT Michel	Sciences de la Terre
Mme	ZINN JUSTIN Nicole	M.P.A.

CHARGES DE COURS

M.	TOP Gérard	S.E.S.
M.	ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M.	DUVEAU Jacques	S.E.S.
M.	HOFLACK Jacques	I.P..A
M.	LATOUCHE Serge	S.E.S.
M.	MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M.	OPIGEZ Philippe	S.E.S.

AVANT PROPOS

Ce travail a été réalisé au Laboratoire d'Informatique Théorique de l'Université des Sciences et Techniques de Lille I.

Il m'est agréable d'exprimer ma reconnaissance envers Madame F. PECAUT, professeur au Centre Universitaire d'Avignon pour l'honneur et le plaisir qu'elle me fait en présidant le Jury.

Je suis particulièrement reconnaissant à mon maître et professeur G. JACOB qui a accepté de diriger mes recherches ; je n'oublierai jamais la confiance qu'il m'a faite et la patience qu'il a manifestée à mon égard.

Je suis très honoré que Monsieur G. COUSINEAU, professeur à l'Université de Paris VII, ait bien voulu participer au Jury et je tiens à le remercier pour ses critiques constructives.

J'adresse aussi mes remerciements à Monsieur MICHEL, professeur au Centre Universitaire d'Avignon et à Monsieur BLANC de l'Université de Marseille II de bien vouloir m'honorer de leur participation au Jury.

Je remercie également tous les membres du Département Informatique et Mathématiques de la Faculté des Sciences d'Avignon pour leur accueil chaleureux, en particulier Madame B. ROZOY et Monsieur Y. KERGALL.

Je remercie aussi tous les membres du Laboratoire d'Informatique Théorique de l'Université de Lille I pour la gentillesse témoignée pendant la préparation de cette thèse.

Enfin, je remercie Mesdemoiselles J. GOURRET et B. FIEVET pour le soin qu'elles ont apporté à la dactylographie ainsi que Mme DEBOCK qui en a assuré le tirage.

A tous les miens.

SOMMAIRE

PREMIERE PARTIE - LES PROBLEMES D'EQUIVALENCES

...

...

DEUXIEME PARTIE - LES PROBLEMES DE COMPARAISONS

...

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

Dans cette thèse, nous avons étudié les relations entre les schémas de programmeset l'application de ces relations à la comparaison des structures des schémas de programmes.

Dans la première partie, en trois chapitres, nous étudierons six niveaux d'équivalences COUSINEAU, LIBRE, EMONDEE, SELECTIVE, IANOV et PATERSON.

Dans les deux premiers chapitres, on donne des définitions fondamentales inspirées par les travaux de COUSINEAU [11] ; [12] , JACOB [19] ; [20] ;[21] ; [22] GRIBACH [24] et KASAI [25] .

Le troisième chapitre est consacré à la définition des différents niveaux d'équivalence des schémas de programme.

i) *EQUIVALENCE DE COUSINEAU*

Ce niveau d'équivalence a été complètement étudié par COUSINEAU [11] et la plupart des théorèmes de ce chapitre sont une simple recopie du travail de cet auteur.

ii) *EQUIVALENCE DE IANOV.*

Nous avons montré que le problème d'équivalence de IANOV est décidable pour les schémas de programmes et nous avons introduit un ordre total sur les éléments de $F \cup P$ qui nous permet de mettre les blocs sélectifs sous forme canonique.

iii) *L'équivalence de PATERSON* est étudiée en deux parties :

- On a présenté une classe de schémas de programmes où le problème d'équivalence de PATERSON n'est pas décidable.

- Nous étudions trois classes de schémas de programmes où le problème d'équivalence de PATERSON est décidable dont la classe de schémas de programmes BIPARTIS présenté par JACOB [22].

Nous défissons une théorie Consistante et Complète pour ces trois classes de schémas de programmes.

iv) Enfin , nous avons étudié trois niveaux d'équivalence intermédiaires qui nous permettent de détailler le problème d'équivalence entre les schémas de programmes.

Dans la deuxième partie, nous avons appliqué ces niveaux pour comparer le structures de schémas de programmes.

Il convient de noter que l'objet de notre travail est le schéma de programmes dont l'arbre de programme associé est reconnaissable.

PREMIERE PARTIE

CHAPITRE I

PRELIMINAIRE

I.1. ARBRES FINIS ET INFINIS

I.1.1. Définitions

Etant donné un ensemble fini $W = \{a_1, \ldots, a_\ell\}$, on appelle *mot* sur W toute suite finie d'élément de W, $u = a_{i_1} \ldots a_{i_p}$.

La *Longueur* d'un mot u noté $|u|$ est la longueur de la suite, on désigne par W^* l'ensemble des mots sur W.

L'ordre préfixiel $\leq_p$ sur W^* est une relation d'ordre définie par :

Soient $u \& v \in W^*$ et $u = a_{i_1} \ldots a_{i_p}$ & $v = a_{j_1} \ldots a_{j_q}$

$u \leq_p v \iff p \leq q$ et $\forall k \leq p$, $a_{i_k} = a_{j_k}$

L'ordre lexicographique $\leq$ sur W^* est une relation d'ordre définie par :

Soient $u = a_{i_1} \ldots a_{i_p}$ et $v = a_{j_1} \ldots a_{j_q}$ dans W^*

$u \leq v \iff$ soit $u \leq_p v$

soit $\exists r \leq \inf(p, q)$ tel que $\forall k < r$ et $i_r < j_r$, $a_{i_k} = a_{j_k}$

Etant donné un entier n, on considère $[n] = \{1, 2, \ldots, n\}$ ordonné par l'ordre des entiers.

Un *domaine d'arbre de degré* n est une partie non vide D de $[n]^*$ qui possède les propriétés suivantes :

1. $\forall m \in D$, $\forall m' \in [n]^*$ $\quad m' \leq_p m \Rightarrow m' \in D$

2. $\forall m \in [n]^*$, $\forall i, j \in [n]^*$ $\quad mi \in D$ et $j \leq i \Rightarrow mj \in D$

Un *W-ARBRE de degré* n est une application partielle $\overline{A} : [n]^* \to W$ où W est un alphabet, et telle que le domaine de $\overline{A}$ est un domaine d'arbre.

L'arbre $\overline{A}$ sera dit fini ou infini suivant que le domaine de $\overline{A}$ (noté $Dom(\overline{A})$) est fini ou infini.

Soit $\overline{A}$ un arbre de degré n et $m \in Dom(\overline{A})$, on appelle *sous-arbre* de $\overline{A}$ issu du noeud m et l'on désigne par $\overline{A}\downarrow_m$ l'arbre de degré m dont la fonction est définie par $(\overline{A}\downarrow_m)(m') = \overline{A}(m\,m')$.

Il est clair que $\forall m_1, m_2 \in Dom(\overline{A})$ $(\overline{A}\downarrow_{m_1})\downarrow_{m_2} = \overline{A}\downarrow_{m_1 m_2}$.

Les éléments du domaine de $\overline{A}$ sont appelés *noeuds de* $\overline{A}$, les noeuds maximums "*les feuilles*" et le noeud minimal "*racine*" (noté ε).

EXEMPLE I.1. : W-arbre $\overline{A}$ défini par :

$$\overline{A} = \{(\varepsilon, f), (1, g), (2, h), (11, g), (12, f), (13, x)$$
$$(21, f), (111, y), (121, x), (122, y), (211, x)\}$$

$Dom(\overline{A}) = \{\varepsilon, 1, 2, 11, 12, 13, 21, 111, 121, 122, 211\}$

$W = \{f, g, h, x, y\}$

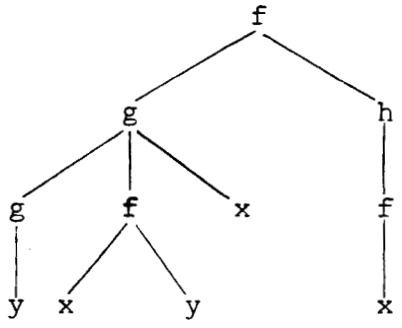
I.1.2. Représentation graphique des arbres

On peut représenter les arbres comme des graphe étiquetés dont les sommets sont les noeuds de l'arbre, chaque sommet m est étiqueté par $\overline{A}(m)$, une arête joint le sommet m au sommet mi avec $m \in [n]^*$ et $i \in [n]$.

On dessine généralement ces arbres en plaçant la racine vers le haut et en orientant implicitement les arêtes vers le bas.

De plus, les arêtes (m, m1), (m, m2),..., (m, m n) sont placées dans cet ordre de la gauche vers la droite ce qui rend implicite la désignation des sommets.

EXEMPLE I.2. : Le graphe étiqueté représentant l'arbre $\overline{A}$ de l'exemple I.1. :

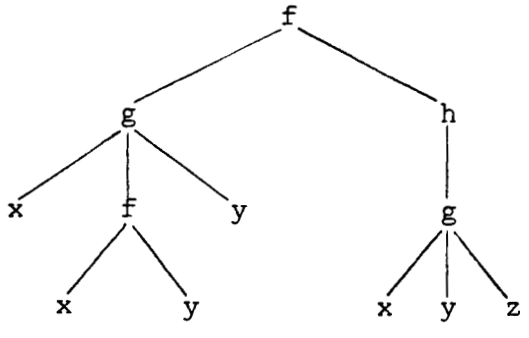


Définition : Un alphabet W est dit a-*gradué* s'il est muni d'une application $a : W \to \mathbb{N}$. $\forall x \in W$, a(x) s'appelle *l'arité de* x.

Un W-arbre $\overline{A}$ est dit a-*gradué* si W est a-gradué et

$\forall m \in Dom(\overline{A})$ Cardinal{i ∈ [n] : mi ∈ $Dom(\overline{A})$} = $a(\overline{A}(m))$.

EXEMPLE I.3. : L'arbre a-gradué $\overline{B}$ (représenté par un graphe étiqueté).



a(f) = 2
a(g) = 3
a(h) = 1

a(x) = a(y) = a(z) = 0

Dans l'exemple I.2, $\overline{A}$ n'est pas gradué.

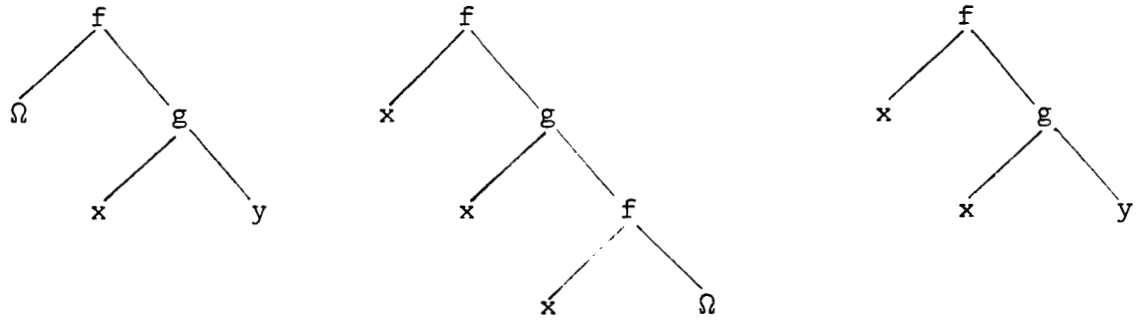
NOTATION I.1. : Etant donné un alphabet W a-gradué, $M^{\infty}(W)$ et M(W) représenteront respectivement l'ensemble des W-arbres a-gradués et l'ensemble des W-arbres a-gradués finis.

L'ordre préfixé sur $M^{\infty}(W)$ noté $\underline{[}_p$ est défini par :

$$\overline{A}_1 \underline{[}_p \overline{A}_2 \iff \mathrm{Dom}(\overline{A}_1) \subseteq \mathrm{Dom}(\overline{A}_2) \text{ et } \forall m \in \mathrm{Dom}(\overline{A}_1),\ \overline{A}_1(m) \neq \Omega \Rightarrow \overline{A}_2(m) = \overline{A}_1(m)$$

(Ω est un élément distingué de W d'arité zéro).

EXEMPLE I.4. : Considérons les trois arbres $\overline{A}_0$, $\overline{A}_1$ et $\overline{A}_2$ ci-dessous :



$\overline{A}_0$ $\overline{A}_1$ $\overline{A}_2$

$\overline{A}_0 \underline{[}_p \overline{A}_2$ mais $\overline{A}_1 \not\underline{[} \overline{A}_2$

Définitions : - Deux arbres $\overline{A}_1$ et $\overline{A}_2$ sont dits *égaux* si et seulement si ils sont superposables, autrement dit

$\overline{A}_1 = \overline{A}_2 \iff \overline{A}_1 \underline{[}_p \overline{A}_2$ & $\overline{A}_2 \underline{[} \overline{A}_1$.

- Etant donné un alphabet W a-gradué, l'ensemble des expressions bien formées sur W est le langage engendré sur W ∪ {'(', ')', ', '} par la grammaire $T = \sum_{\substack{f\in W \\ a(f)>0}} f(T,\ldots, T) + \sum_{\substack{C\in W \\ a(C)=0}} C$

A tout W-arbre *fini* $\overline{A}$, on fait correspondre *l'expression bien formée* eb($\overline{A}$) définie comme suite :

si $\overline{A}(\varepsilon) = C$ et $a(C) = 0$ alors $eb(\overline{A}) = C$

si $\overline{A}(\varepsilon) = f$ et $a(f) = i > 0$ alors

$eb(\overline{A}) = f(eb(A\downarrow_1), eb(A\downarrow_2),\ldots, eb(A\downarrow_i))$.

A tout ensemble d'arbres *finis* $\overline{A}$, on fait correspondre le langage de leur représentation : $eb(\overline{A}) = \{eb(\overline{A}) : \overline{A} \in \overline{A}\}$.

EXEMPLE I.5. : L'expression bien formée correspondant à l'arbre $\overline{A}$ de l'exemple I.3. est : $eb(\overline{A}') = f(g(x, f(x, y), y), h(g(x, x, y))$.

NOTATION I.2. : Posons $W_\Omega = W \cup \{\Omega\}$ (Ω un élément distingué).

Les ensembles $M^\infty(W_\Omega)$ et $M(W_\Omega)$ seront donc toujours munis de l'ordre préfixé.

I.1.3. Substitution

Soient deux arbres $\overline{A}$ et $\overline{A}_1$ et $m \in Dom(\overline{A})$.

On désigne par $\overline{A}[m \leftarrow \overline{A}_1]$ le résultat de la substitution de $\overline{A}_1$ au sous-arbre de racine m dans $\overline{A}$ c'est-à-dire l'arbre défini par :

- $Dom(\overline{A}[m \leftarrow \overline{A}_1]) = (Dom(\overline{A}) - m[n]^*) \cup m\ Dom(\overline{A}_1)$

- $(\overline{A}[m \leftarrow \overline{A}_1])(m') = \overline{A}(m')$ si $m' \in Dom(\overline{A}) - m[n]^*$

$= \overline{A}_1(m'')$ si $m' = m\,m''$ avec $m'' \in Dom(\overline{A}_1)$

Soient deux arbres $\overline{A}$ et $\overline{A}_1$ et M une partie préfixé de $Dom(\overline{A})$.

On désigne par $\overline{A}[M \leftarrow \overline{A}_1]$ le résultat de la substitution de A_1 aux sous-arbres de racine $m \in M$ dans $\overline{A}$ c'est-à-dire l'arbre défini par :

- $Dom(\overline{A}[M \leftarrow \overline{A}_1]) = (Dom(\overline{A}) \setminus M[n]^*) \cup M\ Dom(\overline{A}_1)$

- $(\overline{A}[M \leftarrow \overline{A}_1])(m') = \overline{A}(m')$ si $m' \in Dom(\overline{A}) - M[n]^*$

$= \overline{A}(m'')$ si $m' = m\,m''$ avec $m \in M$ & $m'' \in Dom(\overline{A}_1)$

EXEMPLE I.6. : $eb(\overline{A}) = f(g(h(x), k(x, y)), h(k(g(x, y), h(x))))$

$eb(\overline{B}_1) = g(h(x), k(x, y))$

$eb(\overline{B}_2) = h(x)$

$eb(\overline{A}_1) = f(x, h(y))$

$eb(\overline{A}_2) = k(y, x)$

$eb(\overline{A}[\overline{B}_1 \leftarrow \overline{A}_1, \overline{B}_2 \leftarrow \overline{A}_2]) = f(f(x, h(y)), h(k(g(x, y), k(y, x))))$.

I.2. MAGMAS

I.2.1. Définitions

Soit W un alphabet a-gradué, on peut distinguer les éléments de W en deux parties F et V telles que :

$$F = \{f \in W : a(f) > 0\}$$

$$V = \{f \in W : a(f) = 0\}.$$

Les éléments de V sont des variables (notées x, y, z,...).

Soit (D, ≤) un ensemble ordonné par une relation d'ordre ≤.

Une partie B de D est *dirigée* si $\forall b_1, b_2 \in B \; \exists b_3 \in B$ tel que $b_1 \leq b_3$ & $b_2 \leq b_3$.

L'ensemble ordonné (D, ≤) est *complet* si toute partie dirigée dénombrable B de D admet une plus petite borne supérieure, que l'on note sup(B).

Une fonction $f : D^n \to D$ est *croissante* si $\forall d_1,\ldots, d_n \in D$ et $\forall d'_1,\ldots, d'_n \in D \;\; d_1 \leq d'_1,\ldots, d_n \leq d'_n \Rightarrow f(d_1,\ldots, d_n) \leq f(d'_1,\ldots, d'_n)$; elle est *continue* si elle est croissante et si pour toutes parties dirigées dénombrables $B_1,\ldots, B_n$ $\sup(f(B_1,\ldots, B_n)) = f(\sup(B_1),\ldots, \sup(B_n))$.

(Si f est croissante et les B_i dirigées dénombrables, alors $f(B_1,\ldots, B_n) = \{f(b_1,\ldots, b_n) : b_1 \in B_1,\ldots, b_n \in B_n\}$ est dirigée dénombrable).

Un W-MAGMA M est un triplet $\langle D_M, F_M, E_M \rangle$ où

- D_M est un ensemble appelé domaine de M.
- F_M est un ensemble d'applications qui contient pour chaque $f \in F$ une application partielle $f_M : D_M^{a(f)} \to D_M$.
- $E_M \subseteq D_M$ contient pour chaque symbole $C \in V$ un élément $C_M \in D_M$.

Si $(D_M, \leq)$ est un ensemble ordonné ayant un plus petit élément (noté $\perp$), alors le magma M est dit *ordonné* (resp. *complet*) si les f_M sont *croissantes* (resp. *continues*).

A chacune de ces notions de magma, magma ordonné et magma complet correspond une notion de morphisme évidente.

Par exemple si M et M' sont deux magmas complets, un morphisme $h : M \to M'$ est une application continue $h : D_M \to D_{M'}$, tel que :

- $h(\perp_M) = \perp_{M'}$,
- $\forall x \in V \quad h(x_M) = x_{M'}$,
- $\forall f \in F$ et $\forall d_i \in D_M \; h(f_M(d_1,\ldots, d_n)) = f_{M'}(h(d_1),\ldots, h(d_n))$.

Les W-magmas, W-magmas ordonnés, W-magmas complets avec leurs morphismes constituent des catégories notées respectivement M(W), (MO(W) et MC(W).

Un objet M_0 d'une catégorie $\mathcal{C}$ est *universel initial* si pour tout objet $M \in \mathcal{C}$, il existe un et un seul morphisme $h : M_0 \to M$.

Nous appellerons *W-magma libre, W-magma libre ordonné* et *W-magma libre complet* les objets universels initiaux des catégories M(W), MO(W) et MC(W).

I.2.2. Les ensembles d'arbres comme magmas

Nous allons montrer que $M(W_\Omega)$ et $M^\infty(W_\Omega)$ sont munis respectivement d'une structure de magma ordonné et de magma complet.

- Montrons d'abord qu'ils ont la structure de magma :

Soient F l'ensemble des symboles d'arité non nulle, et V_Ω l'ensemble des symboles d'arité zéro.

Alors, on a l'égalité $W_\Omega = F \cup V_\Omega$, on peut donc noter $M(F, V_\Omega)$ pour $M(W_\Omega)$ et $M^\infty(F, V_\Omega)$ pour $M^\infty(W_\Omega)$.

A tout symbole $c \in V_\Omega$, on fait correspondre l'arbre $\overline{C}$ défini par :

$$\mathrm{Dom}(\overline{C}) = \{\varepsilon\}$$

$$\overline{C}(\varepsilon) = C$$

A tout symbole $f \in F$ on fait correspondre l'application (arité de $f = k$) $\overline{f} : M^\infty(F, V_\Omega)^k \to M^\infty(F, V_\Omega)$ définie par :

$$\overline{f}(\overline{A}_1, \ldots, \overline{A}_k)(\varepsilon) = f$$

$$\overline{f}(\overline{A}_1, \ldots, \overline{A}_k)(i\,m) = A_i(m) \quad \forall m \in [n]^* \text{ et } i \in [n].$$

Il est clair que la restriction de $\overline{f}$ à $M(F, V_\Omega)$ est dans $M(F, V_\Omega)$.

$M^\infty(W_\Omega)$ et $M(W_\Omega)$ sont donc munis par ces opérations *d'une structure de magma.*

- Montrons maintenant que $M^\infty(W_\Omega)$ et $M(W_\Omega)$ sont des magmas ordonnés : il faut montrer que $\overline{f}$ est croissante c'est-à-dire que :

$$\forall i\ 1 \le i \le k \quad \overline{A}_i \sqsubseteq \overline{A}'_i \Rightarrow \overline{f}(\overline{A}_1, \ldots, \overline{A}_k) \sqsubseteq \overline{f}(\overline{A}'_1, \ldots, \overline{A}'_k)$$

$\overline{f}(\overline{A}_1, \ldots, \overline{A}_k)(\varepsilon) = \overline{f}(\overline{A}'_1, \ldots, \overline{A}'_k)(\varepsilon) = f$

$\forall i \in [n]$ et $m \in [n]^*$ $\overline{f}(\overline{A}_1, \ldots, \overline{A}_k)(i\,m) = \overline{A}_i(m)$ & $\overline{f}(\overline{A}'_1, \ldots, \overline{A}'_k)(i\,m) = \overline{A}'_i(m)$.

$\overline{f}(\overline{A}_1, \ldots, \overline{A}_k)(i\,m) \neq \Omega \Rightarrow \overline{A}_i(m) \neq \Omega \Rightarrow \overline{A}'_i(m) = \overline{A}_i(m)$ par hypothèse

et donc $\overline{f}(\overline{A}_1, \ldots, \overline{A}_k) \sqsubseteq \overline{f}(\overline{A}'_1, \ldots, \overline{A}'_k)$.

- Il nous reste à montrer que l'ordre préfixé est complet sur $M^\infty(W_\Omega)$, autrement dit, il faut prouver que les $\overline{f}_i$ sont continues.

$- \overline{A}(m) = \sup(\{\overline{A}'(m : \overline{A}'(m) \in D \text{ et } m \in Dom(\overline{A}')\}) \quad \forall m \in Dom(\overline{A})$

La proposition I.2. montre que $\overline{A}$ est bien définie partant sur $\bigcup_{\overline{A}' \in D} Dom(\overline{A}')$, qu'il majore tous les éléments de D et qu'il est le plus plus majorant.

Pour montrer que $\overline{f}$ est continue, prenons $D_1, \ldots, D_n$ des parties dirigées dénombrables de $M^{\infty}(W_{\Omega})$ et $D = \overline{f}(D_1, \ldots, D_n)$.

Comme $\overline{f}$ est croissante D est une partie dirigée dénombrable de $M^{\infty}(W_{\Omega})$ et par conséquent possède un sup.

Il s'agit de montrer que : $\sup(D) = \overline{f}(\sup(D_1), \ldots, \sup(D_n))$.

Posons $\overline{A} = \sup(D)$ et $\forall i \ \overline{A}_i = \sup(D_i)$, il faut alors montrer que :

$$\overline{A} = \overline{f}(\overline{A}_1, \ldots, \overline{A}_n)$$

$$\overline{A}(\varepsilon) = [\sup(D)](\varepsilon) = [\sup(\{\overline{f}(d_1, \ldots, d_n) : d_1 \in D_1, \ldots, d_n \in D_n\})](\varepsilon) = f = \overline{f}(A_1, \ldots, A_n)(\varepsilon)$$

$$\overline{A}(i\,m) = \sup(\{d_i(m) : d_i \in D_i\}) = \overline{A}_i(m) = \overline{f}(\overline{A}_1, \ldots, \overline{A}_n)(i\,m).$$

Alors $M^{\infty}(W_{\Omega})$ *est un magma complet.*

I.2.3. Les magmas comme modèles

Considérons le W-magma $M = \langle D_M, F_M, E_M \rangle$.

Comme $E_M \subset D_M$ on peut le considérer comme ensemble de relations unaires sur W.

On vérifie facilement que M possède la structure d'une réalisation et donc qu'il peut être le modèle pour une formule ou pour un ensemble de formules (théorie).

Proposition I.1. : Si D est une partie dirigée de $M^{\infty}(W_{\Omega})$ et si $\overline{A}_1$ et $\overline{A}_2 \in D$ alors :

$$\left.\begin{array}{l} \forall m \in [n]^*,\ m \in \mathrm{Dom}(\overline{A}_1) \cap \mathrm{Dom}(\overline{A}_2) \\ \overline{A}_1(m) \neq \Omega \ \&\ \overline{A}_2(m) \neq \Omega \end{array}\right\} \Rightarrow \overline{A}_1(m) = \overline{A}_2(m)$$

DEMONSTRATION : Comme D est une partie dirigée, il existe $\overline{A}_3 \in D$ tel que $\overline{A}_1 \sqsubseteq \overline{A}_3$ & $\overline{A}_2 \sqsubseteq \overline{A}_3$

$$\left.\begin{array}{l} \overline{A}_1(m) \neq \Omega \Rightarrow \overline{A}_3(m) = \overline{A}_1(m) \\ \overline{A}_2(m) \neq \Omega \Rightarrow \overline{A}_3(m) = \overline{A}_2(m) \end{array}\right\} \Rightarrow \overline{A}_1(m) = \overline{A}_2(m)$$

Proposition I.2. : Si D et D' sont deux parties dirigées d'un ensemble ordonné complet $(E, \leq)$, $(\forall d \in D\ \exists d' \in D',\ d \leq d'$ donc $\sup(D) \leq \sup(D')$.

DEMONSTRATION : $\forall d \in D, \exists d' \in D'\ \ d \leq d' \Rightarrow \forall d \in D$

$d \leq \sup(D') \Rightarrow \sup(D) \leq \sup(D')$.

Sur W_{Ω}, on définit l'opération sup par :

$$\sup(x, y) = \begin{cases} x \text{ si } y = \Omega \\ y \text{ si } x = \Omega \\ \text{non défini sinon} \end{cases}$$

L'opération sup est associative et s'étend donc aux parties de plus de deux éléménts.

Si D est une partie dirigée de $M^{\infty}(W_{\Omega})$ et l'arbre $\overline{A}$ défini par :

$$- \mathrm{Dom}(\overline{A}) = \bigcup_{\overline{A}' \in D} \mathrm{Dom}(\overline{A}')$$

Comme nous allons voir plus loin M peut avoir certaines constantes individuelles.

I.3. SYSTEME D'EQUATIONS

I.3.1. Définition

Soient $x_1, \ldots, x_n$ des symboles d'arité zéro.

Un *système d'équations* est un n-uplet d'équations de la forme :

$$x_i = T_i \text{ où : } T_i \in M^{\infty}(F, \mathbb{N} \cup \{x_1, \ldots, x_n\}) \setminus \{x_1, \ldots, x_n\}).$$

Une *solution* d'un tel système est un n-uplet d'éléments de $M^{\infty}(F, \mathbb{N})$ vérifiant :

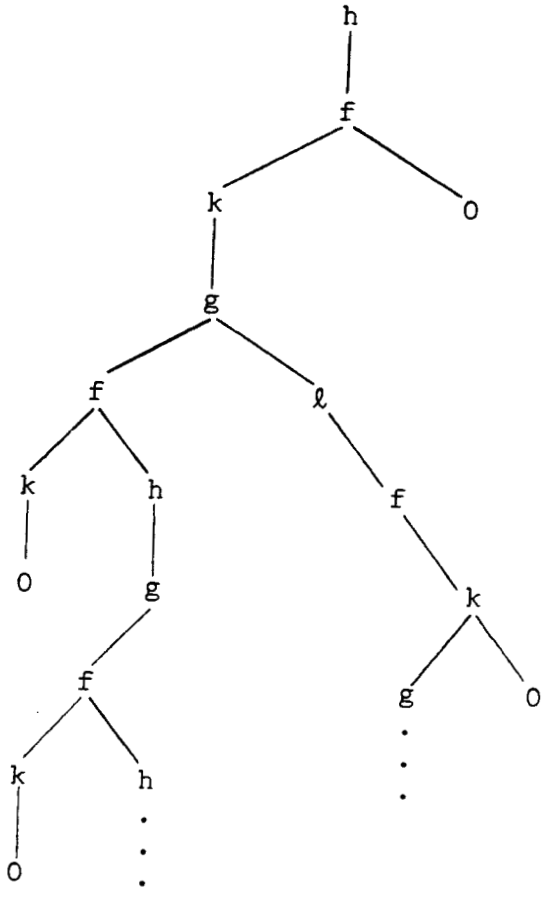
$$\forall i \quad 1 \leq i \leq n \qquad \overline{A}_i = T_i[x_i \leftarrow \overline{A}_1, \ldots, x_n \leftarrow \overline{A}_n].$$

Un système d'équations est dit *régulier* si les T_i sont finis :

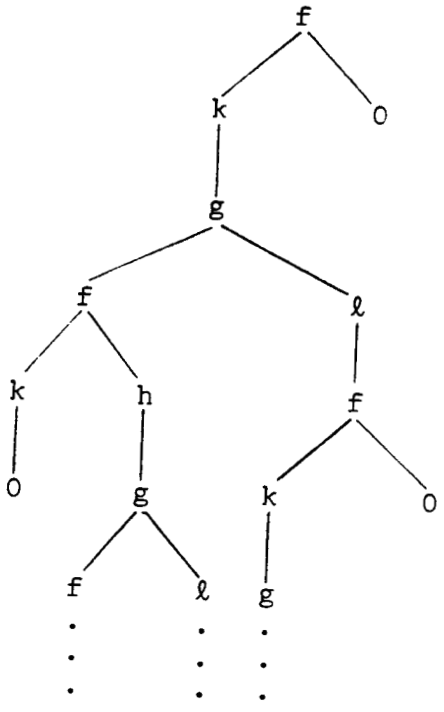
EXEMPLE I.7. :

$$S : \begin{cases} x_1 = h(x_2) \\ x_2 = f(k(x_0), 0) \\ x_3 = g(x_4, \ell(x_2)) \\ x_4 = f(k(0), h(x_3)) \end{cases}$$

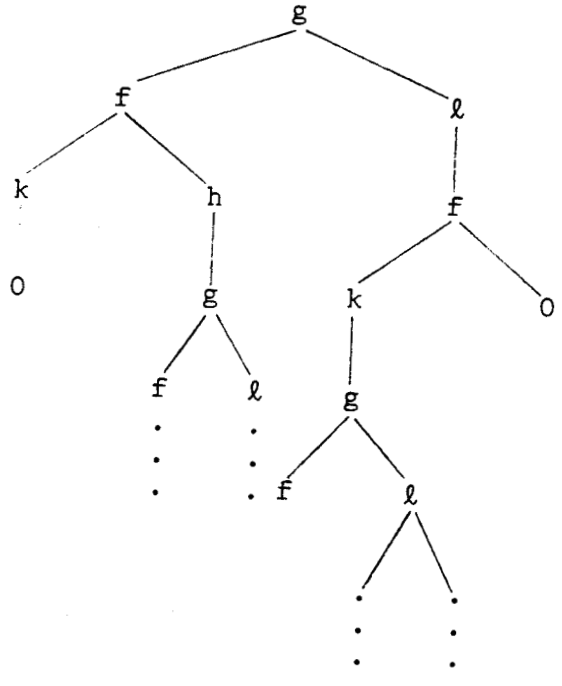
qui possède pour solution le 4-uplet $(\overline{A}_1, \overline{A}_2, \overline{A}_3, \overline{A}_4)$ avec :



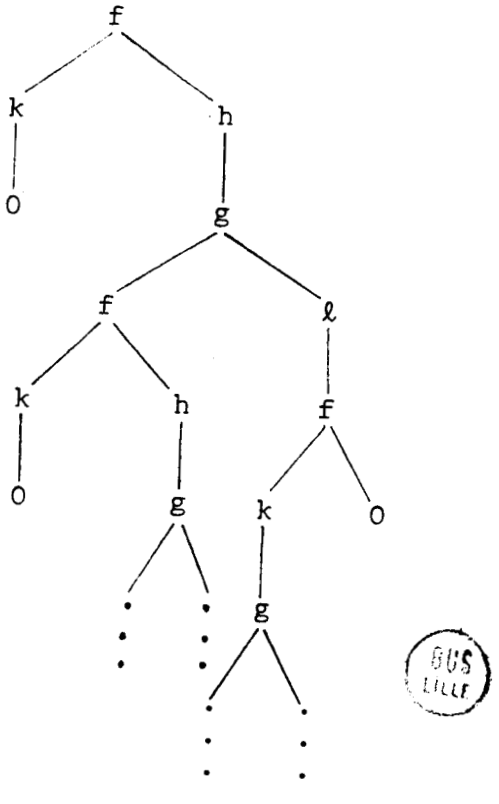
$\overline{A}_1$



$\overline{A}_2$



$\overline{A}_3$



$\overline{A}_4$

Un arbre $\overline{A} \in M^{\infty}(F, \mathbb{N}_{\Omega})$ est dit *reconnaissable* si le nombre de ses sous-arbres distincts est fini.

On peut montrer (COUSINEAU [11]) que les arbres reconnaissables sont exactement ceux qui sont solution d'un système d'équations régulier.

I.3.2. Système d'équations minimal

Soit $\overline{A}$ un arbre reconnaissable, on définit sur $Dom(\overline{A})$ l'équivalence $\underset{\overline{A}}{\equiv}$ par :

$$\forall m, m' \in Dom(\overline{A}) \quad m \underset{\overline{A}}{\equiv} m' \iff \overline{A}\downarrow_m = \overline{A}\downarrow_{m'}$$

Cette équivalence a un nombre fini de classes et dans chaque classe on choisit comme représentant l'élément minimal dans l'ordre lexicographique.

On notera $\rho(m)$ le plus petit m' tel que $\overline{A}\downarrow_m = \overline{A}\downarrow_{m'}$.

Soit $\mathbb{N} = \{m_1, m_2, \ldots, m_r\}$ l'ensemble des éléments minimaux, tel que $m_1 = \varepsilon$; on lui associe le système d'équations :

$$x_{m_i} = C \text{ si } \overline{A}(m_i) = C \in V \text{ et } a(c) = 0$$

$$= f(x_{\rho(m_{i_1})}, \ldots, x_{\rho(m_{i_s})}) \text{ si } A(m_i) = f \in F \text{ et } a(f) = s$$

On vérifie aisément que ce système d'équations possède pour solution r-uplet $(\overline{A}\downarrow_{m_1}, \overline{A}\downarrow_{m_2}, \ldots, \overline{A}\downarrow_{m_r})$.

Notons que dans ce système les membres droits (les T_i) appartiennent soit à V soit à $M^1(F, V)$.

On appelle *élémentaire* un tel système, il est clair que ce système est *minimal* au sens du nombre d'équations.

I.3.3. Equivalence de deux systèmes d'équations

Soient S et S' deux systèmes d'équations aux inconnues $x_1,\ldots, x_n$ et $y_1,\ldots, y_m$.

Pour tester l'équivalence de S et S', on réalise un arbre* dont les sommets sont étiquetés comme suite, par des couples d'inconnues supposées équivalentes :

- La racine de cet arbre est étiquetée par $x_1 \equiv y_1$
- Si un sommet est étiqueté par $x_i \equiv y_j$ et si les équations ne sont pas comparables alors la construction échoue, les deux systèmes ne sont pas équivalents.
- Si les deux équations sont du type $x_i = C$ et $y_j = C$ le sommet n'a pas de successeur.
- Si les deux équations sont de la forme :

$$x_i = f(x_{i_1},\ldots, x_{i_{a(f)}})$$

$$y_j = f(y_{j_1},\ldots, y_{j_{a(f)}})$$

le sommet a pour successeurs des sommets étiquetés par les couples (x_{i_p}, y_{j_p}) $1 \leq p \leq a(f)$ qui n'apparaissent pas encore dans l'arbre.

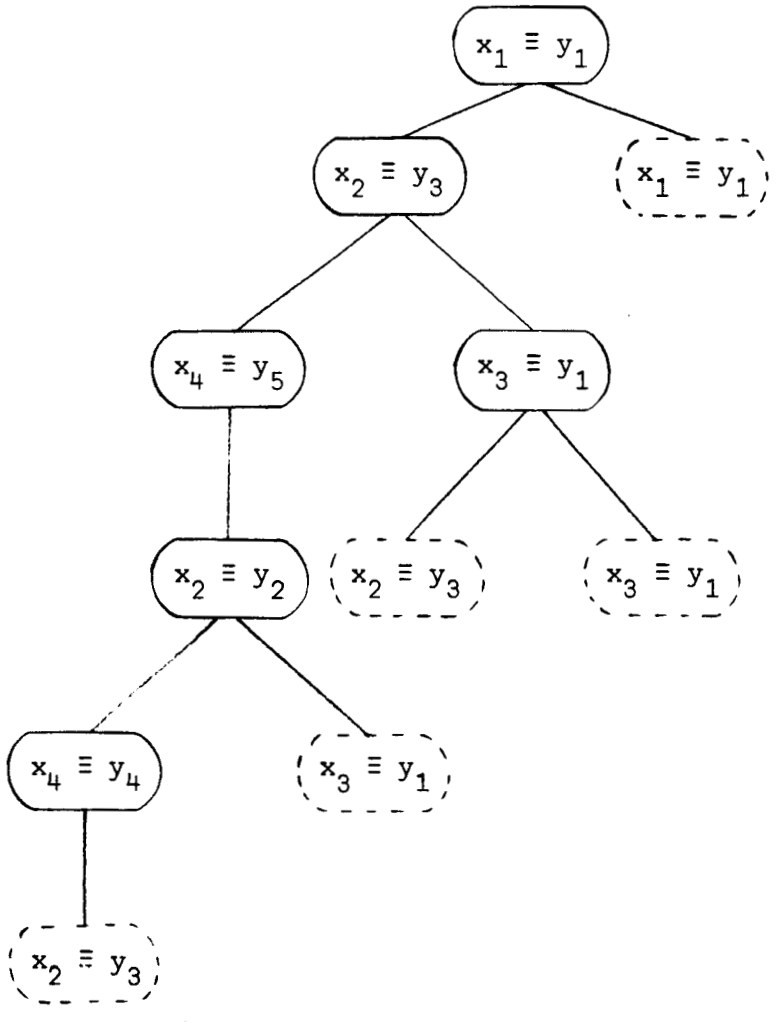
* Nous l'appelerons *l'arbre de comparaison.*

Si nous arrivons à un stade où tous les sommets sont apparus, alors les deux systèmes d'équations S et S' sont équivalents.

EXEMPLE I.8. :

$$\begin{cases} x_1 = g(x_2, x_1) \\ x_2 = f(x_4, x_3) \\ x_3 = g(x_2, x_3) \\ x_4 = h(x_2) \end{cases} \qquad \begin{cases} y_1 = g(y_3, y_1) \\ y_2 = f(y_4, y_1) \\ y_3 = f(y_5, y_1) \\ y_4 = h(y_3) \\ y_5 = h(y_2) \end{cases}$$

L'arbre de comparaison de S et S' :



Alors deux systèmes d'équations S et S' sont équivalents.

NOTATION I.3. : Une démonstration complète de la validité de cette méthode de comparaisons se trouve dans COUSINEAU [11].

I-3-4 EXPRESSIONS RATIONNELLES

DEFINITIONS : Considérons les opérations suivantes définies sur $M^{\infty}(W_{\Omega})$

- La Concaténation

$$\forall \overline{A}_1, \overline{A}_2 \in M^{\infty}(W_{\Omega}) \qquad \overline{A}_1.\overline{A}_2 = \overline{A}_1[0\backslash\overline{A}_2]$$

Cette opération est associative et on peut noter.

$$\forall \overline{A}_1,\overline{A}_2,\overline{A}_3 \in M^{\infty}(W_{\Omega}) \qquad \overline{A}_1.\overline{A}_2.\overline{A}_3 = (\overline{A}_1.\overline{A}_2).\overline{A}_3 = \overline{A}_1.(\overline{A}_2.\overline{A}_3)$$

D'autre part, nous noterons

$$a^{\circ} = \Omega$$
$$a^{1} = a$$
$$\vdots$$
$$a^{n} = a^{n-1}.a$$

- L'addition et la soustraction d'un entier

$\forall \overline{A} \in M^{\infty}(W_{\Omega})$, $\forall n \in \mathbb{N}$;

$\overline{A} + n = \overline{A}[i \in \mathbb{N} \backslash i+n]$

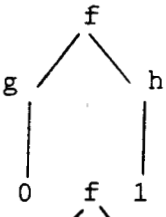
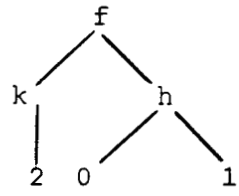
$\overline{A} - n = \overline{A}[i \geq n \in \mathbb{N} \backslash i-n]$

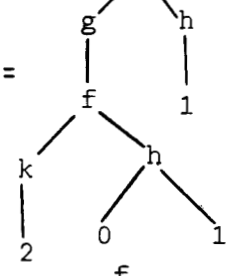
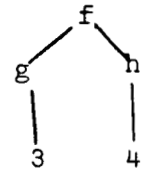
$= \Omega$ pour $i<n$.

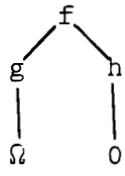
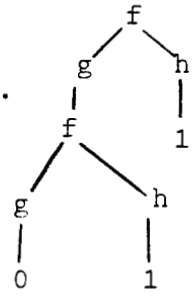
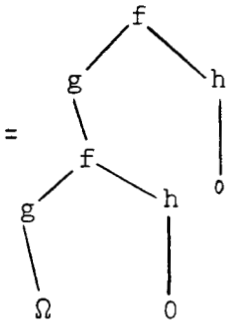
- L'opération étoile

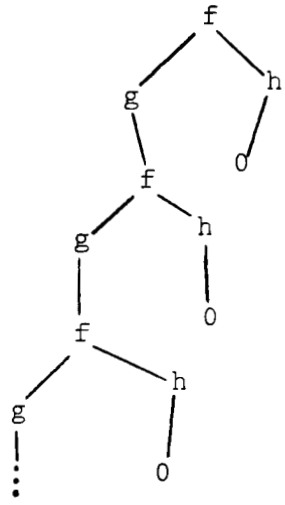
$\forall \overline{A} \in M^{\infty}(W_{\Omega})$

$A^{*} = \mathrm{Sup}(\{A^{n}-1\})$

Par exemple si $\overline{A}$ =  et $\overline{B}$ = 

$\overline{A.B.}$ =  $\overline{A}$ + 3 = 

$\overline{A}$-1 =  $\overline{A}^2$ = $\overline{A}.\overline{A}.$  A^2-1 = 

$\overline{A}^*$ =  = Sup($\{\overline{A}^n$-1$\}$)

On appelle <u>famille des arbres rationnels</u> et on note Rat($M^{\infty}(W_{\Omega})$), la plus petite famille d'arbres contenant les arbres finis et fermée par les opérations de magmas et l'étoile.

Une expression rationnelle est un terme du magma $M(F \cup \{*\}, \mathbb{N}_{\Omega})$.

A toute expression rationnelle E est associé un arbre $\overline{E}$ défini par :

$\forall\, n \in \mathbb{N} \quad \overline{n} = n$

$\forall\, f \in F$ avec ar(f) = k $\qquad \overline{f(E_1,\ldots,E_k)} = f(\overline{E}_1,\ldots,\overline{E}_k)$

$\overline{(E^*)} = (\overline{E})^*$

Sur $M(F \cup \{*\}, N_{\Omega})$, on définit l'ordre $\sqsubseteq$ et l'équivalence par :

$E_1, E_2 \in M(\ \cup\{*\}, \mathbb{N}_{\Omega})$ $\qquad E_1 \sqsubseteq E_2 \iff \overline{E}_1 \sqsubseteq_p \overline{E}_2$

$E_1 \equiv E_2 \iff \overline{E}_1 = \overline{E}_2$

Soit E une expression rationnelle, on appelle <u>sous-expression</u> tout occurrence d'ur expression dans E.

Soit G une sous-expression de E, on appelle profondeur de G dans E et on note $\sigma(G,E)$ le nombre d'étoile qui sont "au dessus" de G dans E.

Soient E une expression rationnelle et s une occurrence d'un entier i dans E. A s et E, on associe l'entier $T(s,E) = i - \sigma(s,E)$.
s est dit signe terminal de E si $T(s,E) \geqslant 0$, dans ce cas $T(s,E)$ s'appelle la valeur terminale du signe terminal s dans E.

Etant donnée une expression rationnelle E et un entier n on désigne par E+n l'expression obtenue en remplaçant dans E tous les signes terminaux i par des occurences de i+n.

Par exemple si $E = f(g(a(\underset{s_1}{1}),h(b(0),a(\underset{s_2}{1}))^*)^*,a(0))$
l'entier 1 a deux occurrences s_1 et s_2 dans E

$T(s_1,E) = 1-1 = 0$, donc s_1 est un signe terminal.
$T(s_2,E) = 1-2 = -1$, donc s_2 n'est pas un signe terminal.

$E+2 = f(f(g(a(3),h(b(0).\ a(1))^*)^*,\ a(2))$

Notation I-4

Dorénavant, nous allons considérer deux formes de substitutions :
Soient E et F deux expressions et $i \in \mathbb{N}$

1,- $E[i\backslash F]$ il s'agit de la substitution de l'expression F à toutes les occurences de l'entier i dans E.

2,- $E[i\backslash\backslash F]$ il s'agit de la substitution de F à l'élément terminal i dans E.

CHAPITRE II

SCHEMAS ET ARBRES DE PROGRAMMES

II.1. SCHEMAS DE PROGRAMMES

II.1.0. Introduction

Un schéma de programme *itératif* peut se définir de manière équivalente comme :

I - Un graphe fini orienté JACOB [19], COUSINEAU [11], MANNA [32],...

II - Une suite d'instructions (écrite au moyen d'une syntaxe voisine de celle des vrais langages de programmations) COUSINEAU [11], GREIBACH [24],...

III - Un objet ressemblant fortement à un automate fini KASAI [26], COURCELLE [9],...

Nous allons étudier ces trois définitions mais nous utiliserons principalement les définitions I et II.

II.1.1. Syntaxe (définitions de base)

A. ALPHABET :

Considérons l'alphabet W constitué des

- symboles de fonctions d'arité ≥ 0 : f_1, f_2,...
- symboles de prédicats d'arité ≥ 1 : p_1, p_2,...
- variables : x_1,..., x_k

- symboles auxiliaires : "DEB", "FIN", "( )", ν, δ (vrai, faux)
- constantes et de l'égalité

Nous noterons F l'ensemble des symboles de fonctions et P l'ensemble de symboles de prédicats et V_k l'ensemble des variables.

B. TERMES :

1. Termes de fonctions :

- Toutes les variables et les constantes sont des termes de fonctions.
- Si $f \in F$ avec $a(f) = n$ et $x_1, \ldots, x_n \in V_k$ alors $f(x_1, \ldots, x_n)$ est un terme de fonctions.

Nous noterons $TF(F, V_k)$ l'ensemble des termes de fonctions définis sur F et V_k.

2. Termes de prédicats :

- Si $p \in P$ avec $a(p) = n$ et $x_1, \ldots, x_n \in V_k$ alors $p(x_1, \ldots, x_n)$ est un terme de prédicat.

Nous noterons $TP(P, V_k)$ l'ensemble de termes de prédicats définis sur P et V_k.

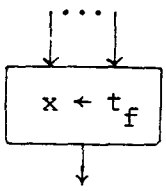
3. Termes d'entrée, de sortie et de boucle :

- Les symboles auxiliaires "DEB", "FIN", "BOU" sont respectivement les termes d'entrée, de sortie et de boucle.

C. INSTRUCTIONS :

1. Instruction d'affectation :

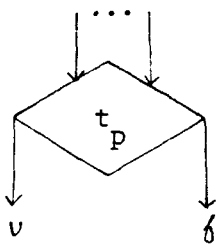
Si $t_f \in TF(F, V_k)$ et $x \in V_k$ alors :



est une instruction d'affectation.

2. Instruction de test :

Si $t_p \in TP(P, V_k)$ alors

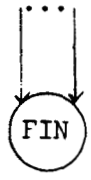


est une instruction de test.

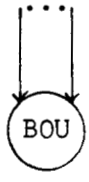
3. Instructions d'entrée, de sortie et de boucle :

- Si "DEB", "FIN", "BOU" sont respectivement les termes d'entrée, de sortie et de boucle alors :

est une instruction d'entrée



est une instruction de sortie



est une instruction de boucle

NOTATION II.1. : L'instruction d'affectation peut avoir un nombre (fini) quelconque d'entrées mais seulement une seule sortie, l'instruction de test peut avoir un nombre (fini) quelconque d'entrées mais deux sorties distinctes, l'instruction d'entrée n'a pas d'entrée et elle a une seule sortie, enfin les instructions de sortie et de boucle peuvent avoir un nombre (fini) quelconque d'entrées mais aucune sortie.

REMARQUE : Nous noterons $A(F, V_k)$ l'ensemble des affectations définies sur F et V_k, et $T(P, V_k)$ l'ensemble des tests définis sur P et V_k.

Définition : Un *W-schéma de programme itératif* S ("FLOWCHART" ou "charte") est un objet constitué par un nombre fini d'instructions des trois types ci-dessus tel que chaque "DEB" est lié à une "FIN" et chaque "FIN" est liée à au moins un "DEB", et qui vérifie les conditions suivantes :

1. S possède exactement un "DEB" et au moins une "FIN".
2. Chaque instruction est sur un chemin de "DEB" à une "FIN" ou a une "BOU".

3. Les variables sont divisées en trois ensembles : $\overline{X}$ (les variables d'entrée), $\overline{Y}$ (les variables de programme), $\overline{Z}$ (les variables de sortie) qui vérifient les conditions suivantes :

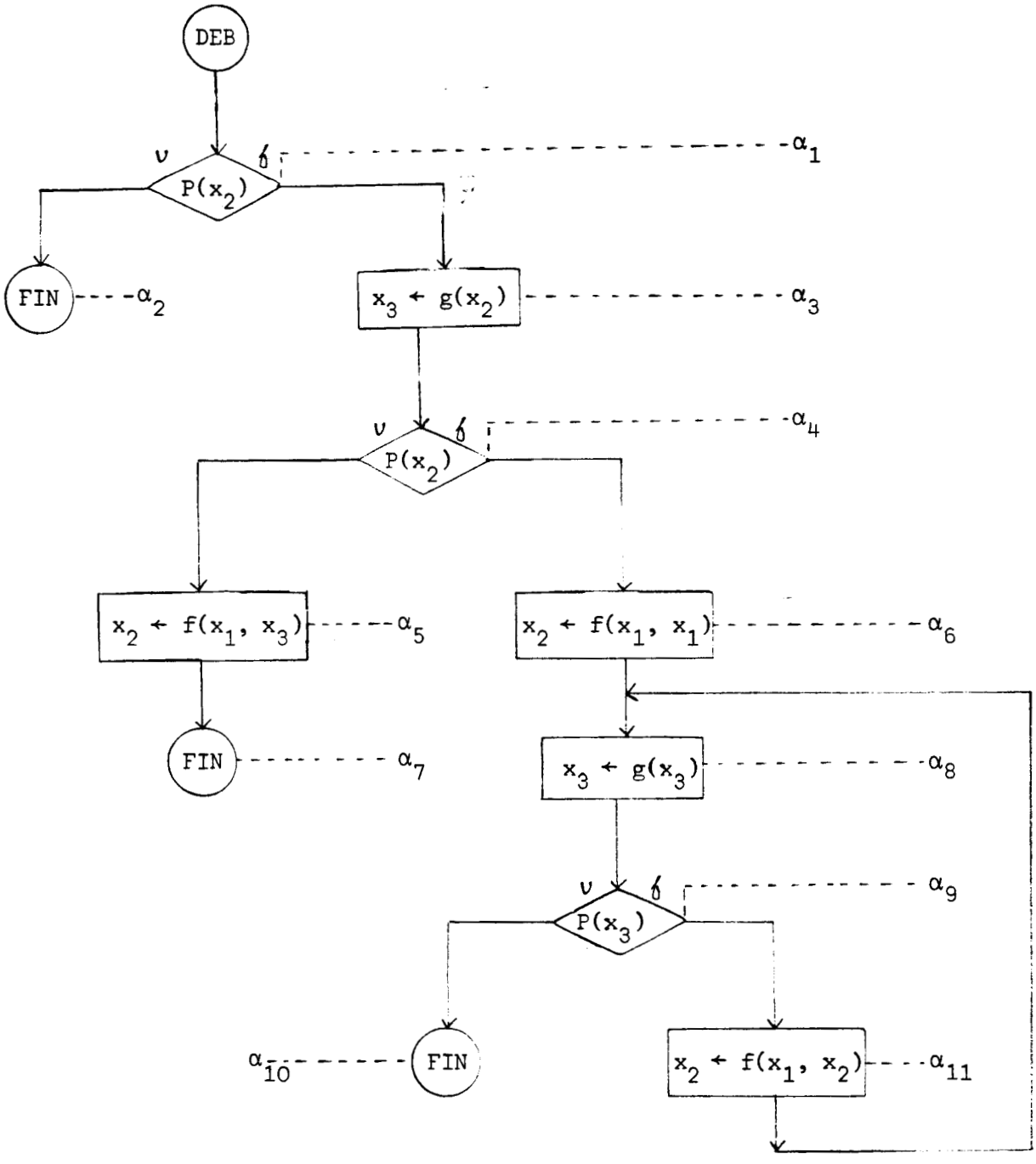
   3.1. Les éléments de $\overline{X} - \overline{Y}$ (données) n'apparaissent jamais comme éléments gauches d'une instruction d'affectation.

   3.2. Les éléments de $\overline{Z} - \overline{Y}$ (résultats) ne peuvent apparaitre comme éléments gauches d'une instruction d'affectation que pour atteindre immédiatement "FIN". Les éléments de $\overline{Z} - \overline{X} - \overline{Y}$ n'apparaissent jamais dans une instruction de test ou comme éléments droits d'une instruction d'affectation.

   3.3. $\forall x \notin \overline{X}$, chaque chemin de "DEB" à une instruction contenant x, doit avoir une instruction d'affectation qui possède x comme élément gauche.

EXEMPLE II.1. : Un schéma de programme (S) (représentation graphique) avec les variables d'entrées x_1 et x_2.

voir page suivante...



C'. NUMEROTATION DES INSTRUCTIONS :

Etant donné un ensemble M = $\{\alpha_1, \ldots, \alpha_n\}$ de noms d'instructions,

Pour une représentation linéaire des instructions, nous nommons chaque instruction par un élément de M (que nous appellons son adresse) et nous les définissons linéairement comme suite :

1. Instruction d'affectation :

Si $t_f \in TF(F, V_k)$ et $\alpha_i, \alpha_j \in M$ alors :

$\alpha_i : t_f \cdot \alpha_j$ est une instruction d'affectation.

2. Instruction de test :

Si $t_p \in TP(P, V_k)$ et $\alpha_i, \alpha_j, \alpha_k \in M$ alors :

$\alpha_i : t_p \cdot \alpha_j, \alpha_k$ est une instruction de test.

3. Instructions de sortie et de boucle

Si "FIN" et "BOU" sont respectivement des termes de sortie et de boucle et $\alpha_i, \alpha_j \in M$ alors :

α_i : "FIN" (noté 0)

α_j : "BOU" (noté ω)

sont respectivement l'instruction de sortie et de boucle.

Notons que dans cette représentation nous supposons que le calcul commence toujours par α_1 et par conséquent nous n'avons pas besoin de préciser "DEB".

Définition : Un *W-schéma de programme* S est une suite finie d'instructions des trois types ci-dessus, qui vérifie les conditions suivantes :

I - L'adresse de chaque instruction est sa position dans la suite.

II - Toutes les adresses "transférées" sont des adresses dans la suite.

III - S possède au moins une "FIN" et chaque instruction est sur un chemin qui commence par α_1.

IV - Les variables sont divisées en trois ensembles $\overline{X}$, $\overline{Y}$ et $\overline{Z}$ et vérifient les conditions 3.1, 3.2 et 3.3.

EXEMPLE II.2. : Une présentation linéaire du schéma de programme S de l'exemple II.1.

α_1 : $[p(x_2)]\ \alpha_2, \alpha_3$

α_2 : FIN

α_3 : $[x_3 \leftarrow g(x_2)] \cdot \alpha_4$

α_4 : $[p(x_2)]\ \alpha_5, \alpha_6$

α_5 : $[x_2 \leftarrow f(x_1, x_3)]\ \alpha_7$

α_6 : $[x_2 \leftarrow f(x_1, x_1)]\ \alpha_8$

α_7 : FIN

α_8 : $[x_3 \leftarrow g(x_3)]\ \alpha_9$

α_9 : $[p(x_3)]\ \alpha_{10}, \alpha_{11}$

α_{10} : FIN

α_{11} : $[x_2 \leftarrow f(x_1, x_2)]\ \alpha_8$

Définition : Un W-SCHEMA DE PROGRAMMES S de degré n est un objet :

$F = \langle W, Q, T, q_0, (S_1, \ldots, S_n)\rangle$ où :

W est un alphabet fini gradué

Q est un ensemble fini

T est une fonction qui à tout $q \in Q - \{S_1, \ldots, S_n\}$ fait correspondre est (p+1)-uplet $(f, q_1, \ldots, q_p)$ tel que $f \in W_p$ et $q_1, \ldots, q_p \in Q$

$(S_1, \ldots, S_n)$ est un n-uplet d'éléments distincts de Q.

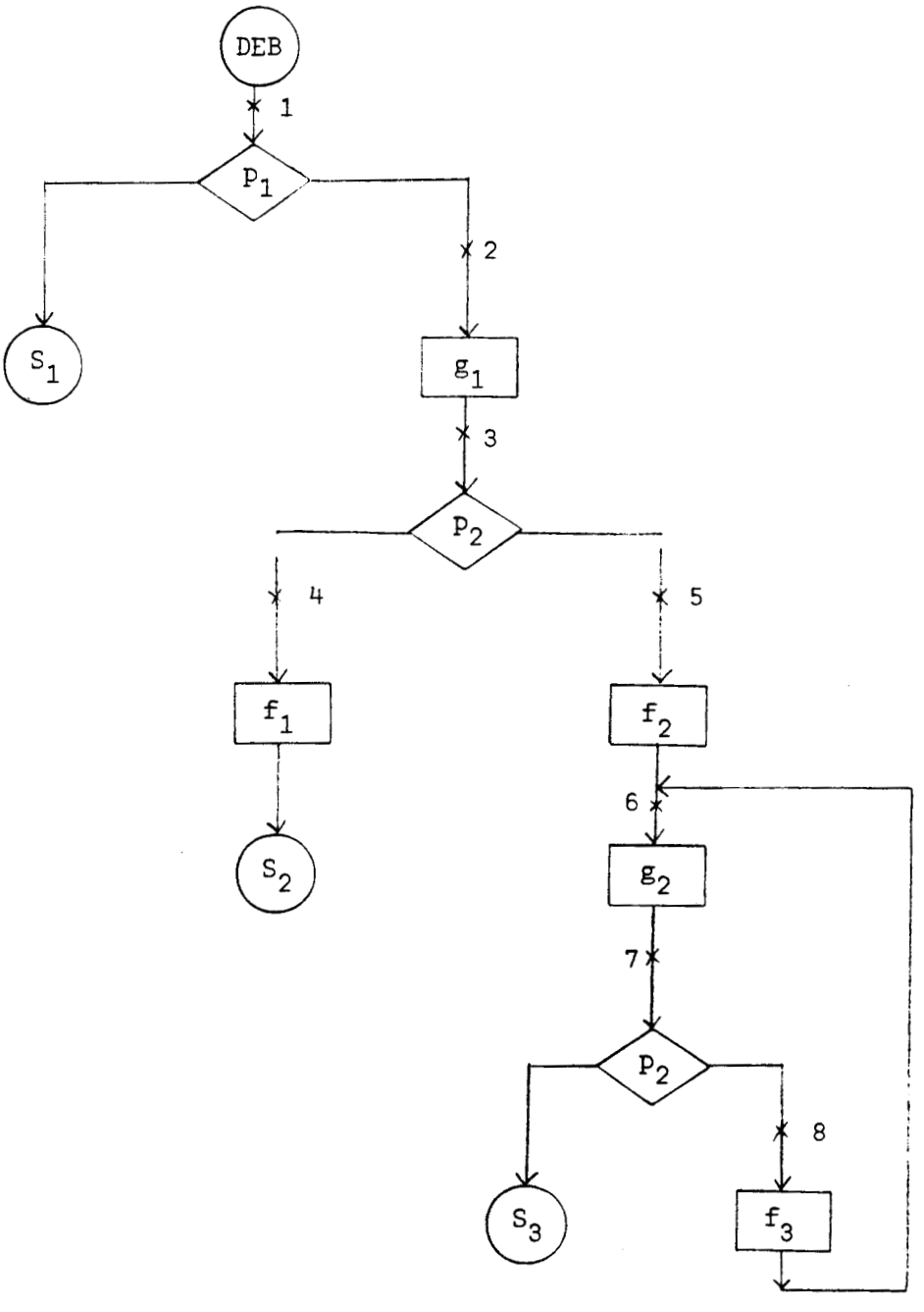
EXEMPLE II.3. : Dans l'exemple II.1. si nous prenons :

$p(x_2) = p_1$ $\quad [x_2 \leftarrow f(x_1, x_3)] = f_1$ $\quad [x_2 \leftarrow f(x_1, x_2)] = f_3$

$[x_3 \leftarrow g(x_2)] = g_1$ $\quad [x_2 \leftarrow f(x_1, x_1)] = f_2$

$p(x_3) = p_2$ $\quad [x_3 \leftarrow g(x_3)] = g_2$

et si nous étiquetons S comme suit :



Alors il peut s'écrire :

$S = \langle \{f_1, f_2, f_3, g_1, g_2, p_1, p_2 ; 1, 2, 3, 4, 5, 6, 7, 8, S_1, S_2, S_3\}, T, 1, (S_1, S_2, S_3)\rangle$ avec :

$T(1) = (p_1, S_1, 2)$

$T(2) = (g_1, 3)$

$T(3) = (p_2, 4, 5)$

$T(4) = (f_1, S_2)$

$T(5) = (f_2, 6)$

$T(6) = (g_2, 7)$

$T(7) = (p_2, S_3, 8)$

$T(8) = (f_3, 6)$

II.1.2. Sémantique (interprétation et calcul)

Une W-*INTERPRETATION* d'un W-schéma de programme est la donnée $II = (D_{II}, I)$ où :

- D_{II} est un ensemble appelé le domaine de l'interprétation II, noté Dom(II).
- A chaque symbole $f \in F$ avec $a(f) = n$, est associée une fonction totale $I(f) : D_{II}^n \to D_{II}$.
- A chaque symbole $p \in P$ avec $a(p) = n$, est associée une fonction totale I $I(p) : D_{II}^n \to \{v, f\}$ (vrai, faux).
- A chaque constante $c \in W$, est associé un élément I(c) de Dom(II).

Le couple (S, II) constitué d'un W-schéma de programme S et d'une W-interprétation est un *W-PROGRAMME*.

La fonction II se prolonge en une application $\overline{II}$ qui, à un programme, fait correspondre une fonction (partiellement définie) de D_{II}^n vers D_{II}^n appelée *interprétation du programme*.

On appelle *INITIALISATION* un point $\bar{d} = (d_1, \ldots, d_k) \in D_{II}^k$. L'initialisation $\bar{x} = (\varepsilon_1, \ldots, \varepsilon_k)$ s'appelle l'initialisation *CANONIQUE*.

CALCULS

Soient S un W-schéma de programme et II = (D_{II}, I) une W-interprétation.

Un II-*ETAT* de S est une paire $(\alpha, \bar{d})$ où $\alpha \in M$ et $\bar{d} = (d_1, \ldots, d_k) \in D_{II}^k$.

Nous écrirons $(\alpha, \bar{d}) \xrightarrow[II]{} (\alpha', \bar{d}')$ si l'on passe d'un état $(\alpha, \bar{d})$ à un état $(\alpha', \bar{d})$ en exécutant l'instruction α, c'est ce que nous appelons une *ETAPE*.

Selon le type d'instruction α deux cas sont à considérer.

1 - α est une affectation de la forme $d_i \leftarrow f(d_{j_1}, \ldots, d_{j_{a(f)}})$ alors :

$(\alpha, \bar{d}) \quad (\alpha', \bar{d}')$ et $\bar{d}' = (d_1, \ldots, d_{i-1}, f(d_{j_1}, \ldots, d_{j_{a(f)}}), d_{i+1}, \ldots, d_k)$.

2 - α est un test de la forme $p(d_{j_1}, \ldots, d_{j_{a(p)}})$ alors

si $I(p) = \nu$, $(\alpha, d) \rightarrow (\alpha', d)$

si $I(p) = \delta$, $(\alpha, d) \rightarrow (\alpha'', d)$

Un *CALCUL* de S et II avec l'initialisation $I = (d_1, \ldots, d_k) \in D_{II}^k$ noté Cal(S, II, $\bar{d}$) est une suite :

$$(\alpha_1, \bar{d}) \to (\alpha_2, \bar{d}_2) \to \ldots \to (\alpha_i, \bar{d}_i) \to \ldots$$

Un calcul sera dit *complet* (*réussi* dans le sens de LIVERCY [30]) si :

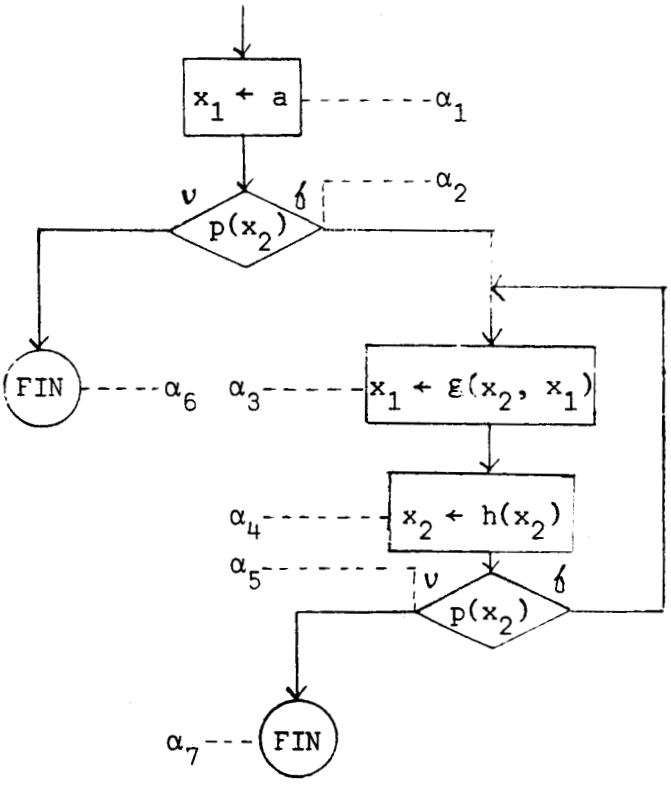
$\exists m$ tel que α_m = "FIN". Dans ce cas $\bar{d}_m$ est le *résultat du programme* que l'on note $\overline{I}(S)(\bar{d}) = \bar{d}_m$.

$\alpha_1, \ldots, \alpha_i, \ldots$ s'appelle la *SEQUENCE D'EXECUTIONS* de Cal(S, II, $\bar{d}$), elle sera *COMPLETE* si Cal(S, II, $\bar{d}$) est complet.

EXEMPLE II.4. : Soit S' le schéma de programme ci-dessous :

Si on prend $II = (D_{II}, I)$ tel que :

$$\begin{cases} D_{II} = \mathbb{N} \\ I(a) = 1 \\ I(p)(x) = (x = 0) \\ I(g)(x, y) = x * y \\ I(h)(x) = \text{si } x > 0 \text{ alors } x - 1 \\ \qquad\qquad \text{sinon } 0 \end{cases}$$



Le programme (S', II) calcule la fonction factorielle.

Si on prend $\bar{d} = (7, 2) \in \mathbb{N}^2$ on obtient le calcul complet suivant :

$$(\alpha_1, (7, 2)) \rightarrow (\alpha_2, (1, 2)) \rightarrow (\alpha_3, (1, 2)) \rightarrow (\alpha_4, (2, 2))$$
$$\rightarrow (\alpha_5, (2, 1)) \rightarrow (\alpha_3, (2, 1)) \rightarrow (\alpha_4, (2, 1)) \rightarrow (\alpha_5, (2, 0))$$
$$\rightarrow (\alpha_7, (2, 0))$$

II.1.3. Interprétation libre (HERBRAND)

Pour chaque W-schéma de programme S nous pouvons définir récursivement un langage U(S) sur l'alphabet ($F \cup V_k$ U l'ensemble des symboles auxiliaires U l'ensemble de constantes) comme suit :

1 - Si $v \in V_k$ alors $v \in U(S)$

2 - Si c est une constante alors $c \in U(S)$

3 - Si $f \in F$ et $a(f) = n$ et $t_1, \ldots, t_n \in U(S)$ alors $f(t_1, \ldots, t_n) \in U(S)$

4 - U(S) est le plus petit langage qui vérifie les conditions 1, 2 et 3.

Une interprétation $II^* = (D_{II}, I^*)$ de S est une *interprétation LIBRE* (interprétation de HERBRAND) de S si :

1 - $D_{II^*} = U(S)$

2 - $\forall x \in V_k \quad I^*(x) = x$ et $\forall c$ une constante $I^*(c) = c$

3 - si $f \in F$ et $a(f) = n \geq 1$ alors : $I^*(f)$ est la fonction totale de $(U(S))^n$ dans $(U(S))$ définie par :

$$I^*(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n) \quad \forall t_1, \ldots, t_n \in U(S)$$

4 - Si $p \in P$ et $a(p) = n$ alors $I^*(p)$ est une fonction totale de la forme :

$$I^*(p) : (U(S))^n \rightarrow \{v, f\}$$

Dans cette interprétation l'initialisation est toujours l'initialisation *canonique* $\bar{x} = (\varepsilon_1, \ldots, \varepsilon_k)$ et pour cela on peut dire *interprétation et initialisation libres.*

THEOREME II.1. : Soient S un schéma de programme, $II = (D_{II}, I)$ une interprétation et $\bar{d} = (d_1, \ldots, d_k) \in D^k$ une initialisation.

Si $\bar{\alpha} = \alpha_1, \ldots, \alpha_i$ est une séquence d'exécutions pour Cal(S, II, $\bar{d}$) alors il existe une interprétation libre (HERBRAND).

$II^* = (U(S), I^*)$ telle que $\bar{\alpha}$ est une séquence d'exécutions pour Cal(S, II^*, $\bar{x}$).

Dans le cas particulier où $\bar{\alpha} = \alpha_1, \ldots, \alpha_m$ est une séquence d'exécutions complète pour Cal(S, II, $\bar{d}$), $\bar{\alpha}$ sera une séquence d'exécutions complète pour Cal(S, II^*, $\bar{x}$) et $I(S)(\bar{d}) = I(I^*(S)(\bar{x}))(\bar{d})$.

PREUVE : Considérons une forme linéaire de S.

Nous allons construire simultanément une interprétation libre $II^*(U(S), I^*)$ et le calcul associé Cal(S, II^*, $\bar{x}$).

Sur la séquence d'exécution $\bar{\alpha}$, pour chaque symbole de test p d'arité n nous constituons deux ensembles distincts $Q(\alpha, p, \nu)$ et $Q(\alpha, p, \delta)$ (appelons-les ensembles de Q) de n-uplets d'éléments de U(S) tels que :

$$\forall \bar{t} \in (U(S))^n \quad \begin{cases} \text{Si } I^*(p)(\bar{t}) = \text{vrai alors } \bar{t} \in Q(\bar{\alpha}, p, \nu) \\ \text{Si } I^*(p)(\bar{t}) = \text{faux alors } \bar{t} \in Q(\bar{\alpha}, p, \delta) \end{cases}$$

La construction des ensembles de Q est de la forme suivante :

1 - Dans $\bar{\alpha}$, selon la valeur de chaque variable dans un point donné nous remplaçons la valeur de la fonction par le nom de cette fonction i.e. f(x) au lieu de $I(f)(\bar{d})$.

2 - Toutes les fois que α_i est un test de la forme :
$p(u_1,\ldots, u_n)\ \gamma, \eta : \gamma \neq \eta$ avec $a(p) = n$ et $u_1,\ldots, u_n \in V_k$
et $\bar{t} = (t_1,\ldots, t_n)$ avec $\forall\ 1 \leq i \leq n\ \ t_i \in U(S)$ le nom de la fonction en ce point, deux cas sont à considérer :
Si $\alpha_{i+1} = \gamma$ plaçons $\bar{t}$ dans $Q(\bar{\alpha}, p, \nu)$ et si $\alpha_{i+1} = \eta$, plaçons $\bar{t}$ dans $Q(\bar{\alpha}, p, \delta)$.

3 - La valeur des ensembles de Q ne change que dans le cas 2.

4 - A la fin s'il existe un $\bar{t}$ qui n'était placé ni dans $Q(\bar{\alpha}, p, \nu)$ ni dans $Q(\alpha, p, \delta)$ plaçons-le dans l'un des deux arbitrairement.

Or, nous allons expliquer en détail la procédure pour obtenir II^* et $Cal(S, II^*, \bar{x})$.

Supposons que α_i et α_i^* soient respectivement les dernières instructions qui ont été exécutées au stade i de $Cal(S, II, \bar{d})$ et $Cal(S, II^*, \bar{x})$.

Pour chaque variable u, supposons que val(u, i) soit la valeur de u après l'exécution du stade i - 1 et juste avant l'exécution du stade i, et soit $val^*(u, i)$ la valeur correspondante dans $Cal(S, II^*, \bar{x})$.

Nous montrons par récurrence sur i que $\forall i \quad \alpha_i = \alpha_i^*$ et pour chaque variable u, si u est définie au stade i, $val(u, i) = I(val^*(u, i))(\bar{d})$.

Autrement dit, si $\bar{t} = (t_1, \ldots, t_n)$ est dans l'ensemble $Q(\bar{\alpha}, p, r)$ et $r \in \{v, f\}$ au stade i pendant l'exécution du test $p(u_1, \ldots, u_n)$ alors :

$$r = I^*(p)(\bar{t}) = I(p)(I(\bar{t})(\bar{d})) = I(p)(val(u_1, i), \ldots, val(u_n, i)).$$

Alors dans chaque stade nous aurons les relations désirées et si le calcul se termine, nous aurons le même résultat pour ces deux interprétations.

Ces relations entre $\bar{\alpha}$ et $\bar{\alpha}^*$, $val(u, i)$ et $val^*(u, i)$, les ensembles de Q et entre $val(u, i)$ et $val^*(u, i)$ sont satisfaites si les calculs sont complets ou pas, autrement dit les ensembles de Q et l'interprétation II^* existent toujours et ils sont bien définis. Néanmoins nous allons définir un algorithme et nous construirons les ensembles de Q quand $\bar{\alpha}$ est complète.

Nous procédons par récurrence, comme suit :

- $\alpha_1 = \alpha_1^*$ = "DEB" et pour chaque initialisation canonique $\bar{x}$, $val^*(\bar{x}, 1) = \bar{x}$ et $I(val(\bar{x}))(\bar{d}) = \bar{d} = val(\bar{d}, 1)$ et tous les ensembles de Q sont vides.

- Supposons que les relations désirées soient satisfaites jusqu'à l'exécution de α_i de $Cal(S, II, \bar{d})$ et α_i^* de $Cal(S, II^*, \bar{x})$ trois cas sont à considérer :

  1 - α_i est une instruction d'affectation de la forme :

  $y \leftarrow f(u_1, \ldots, u_n)$

  alors $val^*(y, i+1) = f(val^*(u_1, i), \ldots, val^*(u_n, i))$

  et $val(v, i+1) = val^*(v, i)$ pour $v \neq y$

par la définition du calcul et comme les ensembles de Q restent inchangés alors :

$$\begin{aligned} val(y, i+1) &= I(f)(val(u_1, i), \ldots, val(u_n, i)) \\ &= I(f(I(val^*(u_1, i))(\bar{d}), \ldots, \\ &\quad I(val^*(u_n, i))(\bar{d})) \text{ par l'hypothèse} \\ &= I(f(val^*(u_1, i), \ldots, val^*(u_n, i))(\bar{d}) \\ &= I(val^*(y, i+1))(\bar{d}). \end{aligned}$$

Nous avons les relations désirées pour la variable y et pour toute variable $v \neq y$ nous n'avons aucun changement.

1' - α_i est une affectation de la forme : $y \leftarrow u$

pour toute variable u, on a $val^*(y, i+1) = val^*(u, i)$ et $val^*(v, i+1) = val^*(v, i)$ pour tout $v \neq y$

$\alpha^*_{i+1} = \alpha_{i+1}$ et les ensembles de Q restent inchangés.

Il est clair que les relations désirées sont satisfaites.

1" - α_i est une affectation de la forme : $y \leftarrow c$ (c une constante)

$val^*(y, i+1) = c$ et $val^*(v, i+1) = val(v, i)$ pour tout $v \neq y$ et les ensembles de Q restent inchangés.

Du fait que $val(y, i+1) = I(c) = I(c)(\bar{d}) = I(val^*(y, i+1))(\bar{d})$ nous avons les relations désirées pour y.

Il est évident que $\alpha^*_{i+1} = \alpha_{i+1}$ et nous maintenons les relations désirées pour tout $v \neq y$.

2 - α_i est un test de la forme

$p(u_1, \ldots, u_n)\ \gamma, \eta \cdot \gamma \neq \eta$

deux cas sont à considérer :

2' - α_{i+1} est une instruction avec l'adresse γ, alors nous plaçons le n-uplet $(val^*(u_1, i),..., val^*(u_n, i))$ dans $Q(\bar{\alpha}, p, \nu)$ et définissons pour II^* :

$I^*(p)(val^*(u_1, i),..., val^*(u_n, i)) = \nu$

il est évident que dans ce cas $\alpha^*_{i+1} = \gamma = \alpha_{i+1}$ et du fait que $\alpha_{i+1} = \gamma$ on a :

$\nu = I(p)(val(u_1, i),..., val(u_n, i))$

$= I(p)(I(val^*(u_1, i)(\bar{d}),..., val^*(u_n, i))(\bar{d}))$ par l'hypothèse

$= I(p)(I(val^*(u_1, i),..., val^*(u_n, i))(\bar{d})$

Si on prend $\bar{t} = (val^*(u_1, 1),..., val^*(u_n, i))$ nous avons les relations désirées.

$\forall u\ val^*(u, i+1) = val^*(u, i)$ et les relations entre $val(u, i+1)$ et $val^*(u, i+1)$ sont toujours satisfaites.

2" α_{i+1} est une instruction avec l'adresse η, alors nous plaçons le n-uplet $(val^*(u_1, i),..., val^*(u_n, i))$ dans $Q(\bar{\alpha}, p, \nu)$ et définissons pour II^* :

$II^*(p)(val^*(u_1, i),..., val^*(u_n, i) = f$

Il est évident que $\alpha^*_{i+1} = \eta = \alpha_{i+1}$ et de la même façon que 2' nous avons les relations désirées.

3 - α_i est une instruction de la forme : $P(u_1,..., u_n)\ \gamma, \gamma$

dans ce cas nous n'avons aucun changement dans les ensembles de Q et toutes les valeurs de variables restent inchangées, et pour toute interprétation $\alpha^*_{i+1} = \alpha_{i+1} = \gamma$.

4 - α_i est une instruction "FIN" alors α et $\bar{\alpha}^*$ sont des séquences d'exécutions complètes et pour toute interprétation le calcul doit se terminer en ce point et les valeurs de variables restent inchangées

Dans le cas particulier où les variables de sortie sont $z_1, \ldots, z_n$ alors $I(S)(\bar{d}) = (val(z_1, i), \ldots, val(z_n, i))$

$= (I(val^*(z_1, i)(\bar{d}), \ldots, I(val^*(z_n, i)(\bar{d}))$

$= I(I^*(S)(\bar{x}))(\bar{d})$

Intuitivement si le cas 4 existe, nous avons un calcul complet $\bar{\alpha}$ et les valeurs $I(S)(\bar{d})$ et $I^*(S)(\bar{x})$ convergent pour le calcul complet $\bar{\alpha} = \bar{\alpha}^*$ et on a les relations désirées i.e. $I(S)(\bar{d}) = I(I^*(S)(\bar{x}))(\bar{d})$.

Soient p un test d'arité n et $\bar{t} \in (U(S))^n$.

$\bar{t} \in Q(\bar{\alpha}, p, \nu) \cup Q(\bar{\alpha}, p, \not\nu)$ si et seulement si $\exists$ un stade i de $\bar{\alpha}$ tel que $\bar{t}$ a été placé dans $Q(\bar{\alpha}, p, \nu) \cup Q(\bar{\alpha}, p, \not\nu)$. A ce stade, (cas 2) où il n'était placé ni dans $Q(\bar{\alpha}, p, \nu)$ ni dans $Q(\bar{\alpha}, p, \not\nu)$ par toute application du cas 2, alors il a été placé dans l'un des deux ensembles arbitrairement.

Alors, d'après cette convention les ensembles de Q sont toujours bien définis mais le placement de $\bar{t}$ dans $Q(\bar{\alpha}, p, \nu)$ ou $Q(\bar{\alpha}, p, \not\nu)$ est algorithmique.

Il nous reste à montrer que II^* est une interprétation légitime i.e. pour tous les $p \in P$, la définition de $I^*(p)$ est consistante.

Autrement dit, on doit montrer que $I^*(p)(\bar{t})$ a une seule valeur dans $\{\nu, \not\nu\}$ i.e. $Q(\bar{\alpha}, p, \nu) \cap Q(\bar{\alpha}, p, \not\nu) = \emptyset$.

Supposons que $\bar{t}$ soit placé dans $Q(\bar{\alpha}, p, \nu)$ au stade i, alors il existe en ce point un test $\alpha_i : p(u_1,\ldots, u_n)\ \gamma, \eta$ tel que $\gamma = \alpha_{i+1}$ et $val^*(u_k, i) = t_k \quad 1 \leq k \leq n$ et $\bar{t} = (t_1,\ldots, t_n)$ alors en ce point nous avons :

$$\nu = I^*(p)(\bar{t}) = I(p)(I(\bar{t})(\bar{d})) = I(p)(val(u_1, i),\ldots, val(u_n, i))$$

et $val(u_k, i) = I(val^*(u_k, i)) = t_k \quad 1 \leq k \leq n$

Or, supposons qu'il existe un stade j tel que $\bar{t}$ soit placé dans $Q(\alpha, p, \delta)$ alors il existe à ce stade un test $\alpha_j = p(v_1,\ldots, v_n)\ \gamma', \eta'$ et $\alpha_{j+1} = \eta'$.

On a $val(u_k, j) = t_k \quad 1 \leq k \leq n$ et par construction nous avons :

$$\begin{aligned}
\delta &= I^*(p)(\bar{t}) = I(p)(I(\bar{t})(\bar{d})) = I(p)(val(v_1, j),\ldots, val(v_n, j)) \\
&= I(p)(I(val^*(v_1, j))(\bar{d}),\ldots, I(val^*(v_n, j))(\bar{d}) \\
&= I(p)(I(t_1)(\bar{d}),\ldots, I(t_n)(\bar{d})) \\
&= I(p)(I(val^*(u_1, i)(\bar{d}),\ldots, I(val^*(u_n, i)(\bar{d})) \\
&= I(p)(val(u_1, i),\ldots, val(u_n, i)) \\
&= \nu
\end{aligned}$$

qui est une contradiction car II est une interprétation légitime.

Alors $Q(\bar{\alpha}, p, \nu) \cap Q(\bar{\alpha}, p, \delta) = \emptyset$ et $I^* = (U(S), I^*)$ est une interprétation libre et légitime avec les propriétés désirées.

II.1.4. Propriété de décidabilité

- On dit qu'une propriété est *décidable* sur un ensemble W, s'il existe un algorithme (décrit par un programme) qui, pour chaque

élément $\omega \in W$ répond au bout d'un temps fini par "oui" si la propriété est vraie pour ω, par "non" si la propriété est fausse pour ω.

- On dit qu'une propriété est *semi-décidable* sur W s'il existe un algorithme qui pour chaque élément $\omega \in W$, répond au bout d'un temps fini, par "oui" si la propriété est vraie pour ω et calcule éventuellement indéfiniment (i.e. ne donne pas de résultat en un temps fini) si la propriété est fausse pour ω.

- Soit $\mathcal{G}$ une classe de schémas de programmes et $II = (D_{II}, I)$ une interprétation. On dit que *le problème d'équivalence est décidable* pour $\mathcal{G}$, si $\forall S_1, S_2 \in \mathcal{G}$, il existe un algorithme qui répond au bout d'un temps fini par "oui" si $I(\overline{S}_1) = I(\overline{S}_2)$ et par "non" si $I(\overline{S}_1) \neq I(\overline{S}_2)$. ($\overline{S}_1$ et $\overline{S}_2$ sont respectivement les arbres de programme associés à S_1 et S_2).

EXEMPLE II.5. : - Le problème d'équivalence est décidable pour la classe des schémas de IANOV (Chapitre III.2).

- Le problème d'équivalence est indécidable pour la classe de schémas de programmes en général (Chapitre III.3).

II.2. ARBRES DE PROGRAMMES

II.2.1. Définition

Montrons d'abord que les schémas de programmes sont les systèmes d'équations rationnelles i.e. à chaque schéma de programme correspond un système d'équations rationnelles.

Etant donné un schéma de programme S (forme linéaire) :

- A chaque instruction d'affectation $\alpha_i : t_f \cdot \alpha_j$, on associe l'équation $\alpha_i = t_f(\alpha_j)$.
- A chaque instruction de test $\alpha_i : t_p \cdot \alpha_j, \alpha_n$ on associe l'équation $\alpha_i = t_p(\alpha_j, \alpha_k)$.
- A chaque instruction de sortie α_i : "FIN" on associe l'équation $\alpha_i = 0$.
- A chaque instruction de boucle α_i = "BOU" on associe l'équation $\alpha_i = \omega$.

Alors à S est associé un *système d'équations* $\hat{S}$ constitué des équations correspondant aux instructions de S.

Ce système d'équations est rationnel car le nombre des instructions est fini.

La solution de ce système d'équations est un arbre reconnaissable $\overline{S}$ que nous appelerons *l'arbre de programme associé à* S.

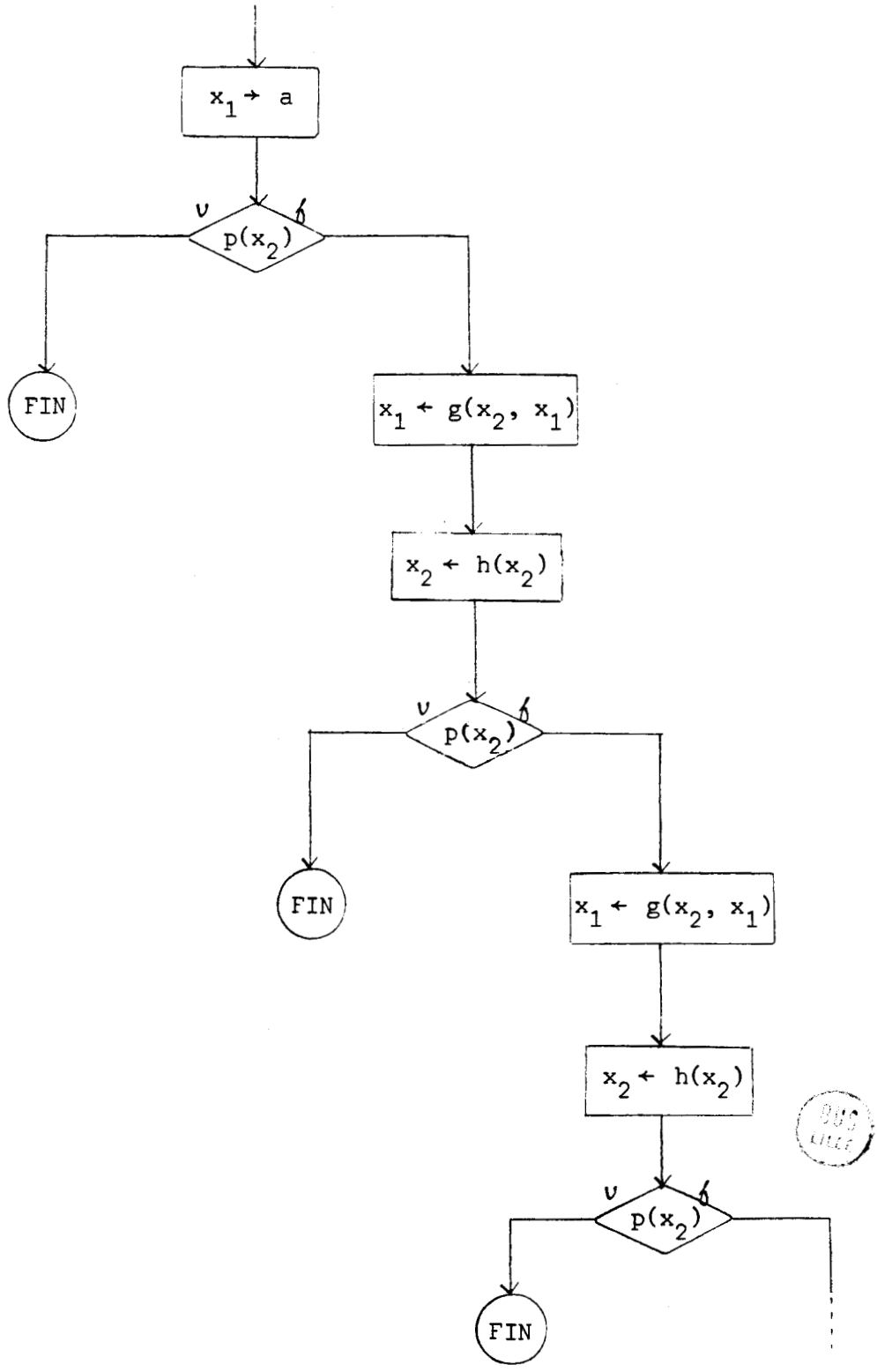
Par conséquent, à tout schéma de programme S est associé un arbre de programme reconnaissable $\overline{S}$, solution du système d'équations correspondant, ce qui va nous permettre de définir la sémantique d'un schéma de programme par la sémantique de l'arbre associé.

Cette façon de faire a l'avantage suivant :

Toute transformation préservant l'arbre associé à un schéma préservera la fonction calculée par voie de conséquence, sans qu'il soit nécessaire d'en faire une démonstration.

Notons que si S est un schéma de programme fini alors $\overline{S} = S$.

EXEMPLE II.6. : L'arbre $\overline{S}'$ associe à l'arbre S' de l'exemple II.4. est :



II.2.2. Interprétation et calcul pour un arbre de programme

De la même façon que pour un schéma de programme, nous définissons l'interprétation pour un arbre de programme.

En ce qui concerne le calcul, nous nommerons chaque instruction par l'élément correspondant du domaine d'arbre.

EXEMPLE II.7. : Dans l'arbre $\overline{S}$, de l'exemple II.8, si nous choisissons l'interprétation et l'initialisation de l'exemple II.4, le calcul associé est :

$(1, (7, 2)) \rightarrow (11, (1, 2)) \rightarrow (112, (1, 2)) \rightarrow (1121, (2, 2)) \rightarrow (11211, (2, 1))$
$(112112, (2, 1)) \rightarrow (1121121, (2, 1)) \rightarrow (11211211, (2, 0)) \rightarrow$
$(112112111, (2, 0))$.

Nous définissons l'équivalence sur l'ensemble des arbres de programmes comme suit :

$\forall \overline{A}$ et $\overline{A}'$ deux arbres de programmes et II = (D, I) une interprétation

$$\overline{A} \equiv \overline{A}' \Longleftrightarrow I(\overline{A}) = I(\overline{A}').$$

Note : $\overline{A} \equiv \overline{A}' \Longleftrightarrow \forall$II une interprétation $I(\overline{A}) = I(\overline{A}')$

II.3. SCHEMAS ET ARBRES DE IANOV

II.3.1. Définitions

Un schéma de IANOV est un schéma de programme dans lequel l'ensemble des variables V_k est réduit à une seule variable $\bar{x}$ et par conséquent tous les symboles de fonctions $f \in F$ et tous les symboles de prédicats $p \in P$ sont monadiques.

Ces schémas de programme simulent assez bien le comportement d'une machine, à condition de considérer $\bar{x}$ comme un vecteur d'état qui contient à chaque instant tous les éléments variables de la machine.

A tout W-schéma de programme qui possède $\{p_1,\ldots, p_m\}$ comme ensemble de symboles de prédicats et $\{f_1,\ldots, f_n\}$ comme ensemble de symboles de fonctions, est associé un $[(W - V_k) \cup \{\bar{x}\}]$-schéma de IANOV comme suit :

Soit $\{p'_1,\ldots, p'_m\}$ un ensemble de m symboles de prédicats monadiques et $\{f'_1,\ldots, f'_n\}$ un ensemble de n symboles de fonctions monadiques.

- A chaque p_i $\quad 1 \leq i \leq m$ on associe un $p'_i(\bar{x})$.
- A chaque t_i $\quad 1 \leq i \leq n$ on associe un $\bar{x} \leftarrow f'_i(\bar{x})$.
- Les symboles de sortie et de boucle restent inchangés.

Nous appelerons "*squelette de* S" le schéma de IANOV associé à S et nous le noterons sq(S).

A toute interprétation II = (D, I) de S, on associe l'interprétation $\hat{II} = (D, \hat{I})$ de sq(S) où :

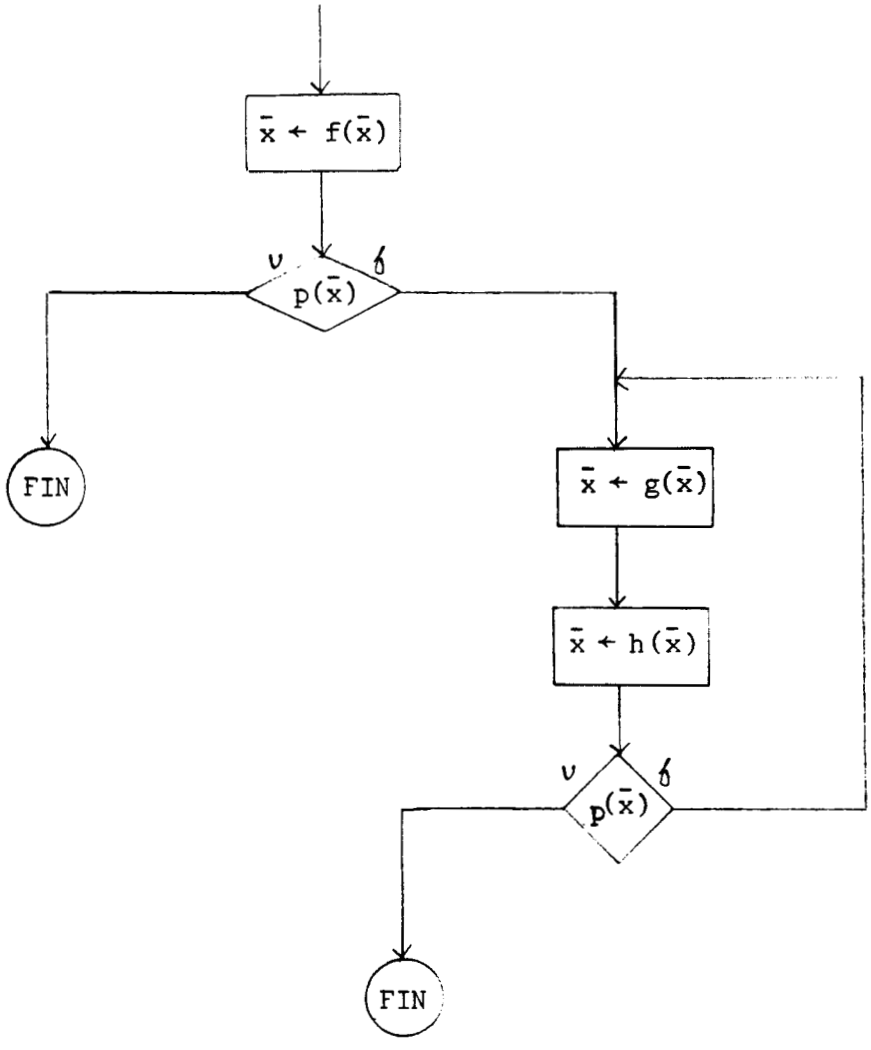
- si $\alpha_i = p(d_{j_1}, \ldots, d_{j_{a(p)}})$ alors $\alpha_i' \in p'(x_1, \ldots, x_k) = p(x_{j_1}, \ldots, x_{j_{a(p)}})$

- Si $\alpha_i : d_j \leftarrow f(d_{j_1}, \ldots, d_{j_{a(f)}})$ alors

$\alpha_i' : f(x_1, \ldots, x_k) = f(x_1, \ldots, x_{j-1}, f'(d_{j_1}, \ldots, d_{j_{a(f)}}), x_{j+1}, \ldots, x_k)$

EXEMPLE II.8. : Le schéma de IANOV sq(S) associé au schéma S de l'exemple II.4.

est :



Et l'interprétation associée : $\hat{I} = (D, \hat{I})$ où :

$$\begin{cases} \hat{I}(a)(x_1, x_2) = (1, x_2) \\ \hat{I}(g)(x_1, x_2) = (x_1 * x_2, x_2 - 1) \\ \hat{I}(q)(x_1, x_2) = (x_2 = 0) \end{cases}$$

NOTATION II.2. : Comme un schéma de IANOV (resp. un arbre de IANOV) ne possède qu'une seule variable $\bar{x}$, on peut noter sans ambiguïté f au lieu de $\bar{x} \leftarrow f(\bar{x})$ et p au lieu de $p(\bar{x})$. Donc nous pouvons considérer les arbres de IANOV (resp. les schémas de IANOV) comme des éléments de $M^{\infty}(P \cup F \cup \{\text{"BOU"}, \text{"FIN"}\})$

II.3.2. Schémas et arbres de IANOV libres

Etant donné un arbre de programme A, nous appelerons *branche de* A br(A) toute suite $m_0, m_1, \ldots, m_p, \ldots$ des éléments de domaine de A véfifiant :

$$m_0 = \varepsilon \text{ et } \forall i \quad m_{i+1} = m_1 \text{ ou } m_2.$$

Nous noterons Br(A) l'ensemble des branches de A (une branche finie est caractérisée par son élément maximal).

Soit A un arbre de programme et II = (D, I) une interprétation de A.

Nous noterons br(A, II, $\bar{d}$) la branche $u_0, u_1, \ldots, u_p, \ldots$ suivie lors du calcul cal(A, II, $\bar{d}$).

Enfin si A est un arbre de programme et J une classe de A-interprétation, nous noterons $Br(A, J) = \{br(A, I), \bar{d} : II \in J, d \in D_{II}\}$.

Un arbre de programme A est dit *libre* si Br(A, J) = Br(A) i.e. tout chemin de "DEB" à "FIN" est un chemin du calcul.

Un arbre de programme A est dit *émondé* si :

$$\forall m \in Dom(A) \quad A\downarrow_m \in M^{\infty}(P \cup F \cup \{\text{"BOU"}\}) \Longrightarrow A\downarrow_m = \text{"BOU"}.$$

Un schéma de programme est dit *libre* (resp. émondé) si l'arbre de programme associé est libre (resp. émondé).

Définition : - Etant donné un arbre $A \in M(P, \{x_1, \ldots, x_n\})$ libre et émondé, nous dirons que $p \in P$ couvre A si $\forall m \in Dom(A)$ tel que $A(m) \in \{x_1, \ldots, x_n\} \cup \{FIN\}$, $\exists m' \leq_p m$ tel que $A(m') = p$.

Nous noterons PC(A) l'ensemble des prédicats qui couvrent A.

Définitions : - Etant donné un arbre de IANOV A, un bloc sélectif de A est un couple (m_0, M) où : $m_0 \in Dom(A)$, $M \subseteq \{1, 2\}^*$ tel que :

- $\forall m \in M, \forall m' \in \{1, 2\}^* \quad m' \leq_p m \Longrightarrow m' \in M$
- $\forall m \in M, A(m_0\ m) \in P$
- $\forall m \in M, \forall u \in \{1, 2\} \quad mu \notin M \Longrightarrow A(m_0\ mu) \notin P$

Pour tout arbre de IANOV A, si $\exists m_0 \in Dom(A)$ tel que $A(m_0) \in P$ alors il existe un unique $M \subseteq \{1, 2\}^*$ tel que (m_0, M) est un bloc sélectif de A. Nous le noterons $bl(m_0)$.

- Un symbole de prédicat $p \in P$ couvre un bloc sélectif (m_0, M) d'un arbre A si $\forall m \in \{1, 2\}^*$ $m \notin M$ et $A(m_0\ m) \in F \cup \{FIN\}$ $\Longrightarrow \exists m' \in M$, $m' \leq_p m$ et $A(m_0\ m') \in P$.

II.3.3. Arbres (schémas) de IANOV canoniques (standards)

Etant donné un ordre total < des éléments de P, un arbre de IANOV est dit *CANONIQUE* vis-à-vis de cet ordre si :

1 - A est libre

2 - A est émondé

3 - $\forall m \in Dom(A)$, $\forall i \in \{1, 2\}$

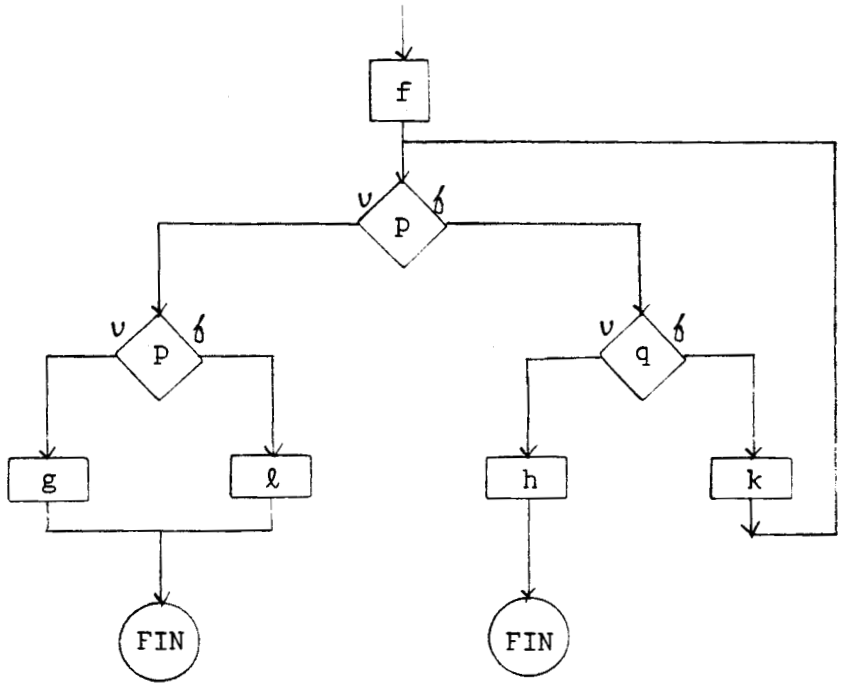
$mi \in Dom(A)$, $A(m) = p \in P$ et $A(mi) = p' \in P \Longrightarrow$

ou bien p' ne couvre pas bl(m)

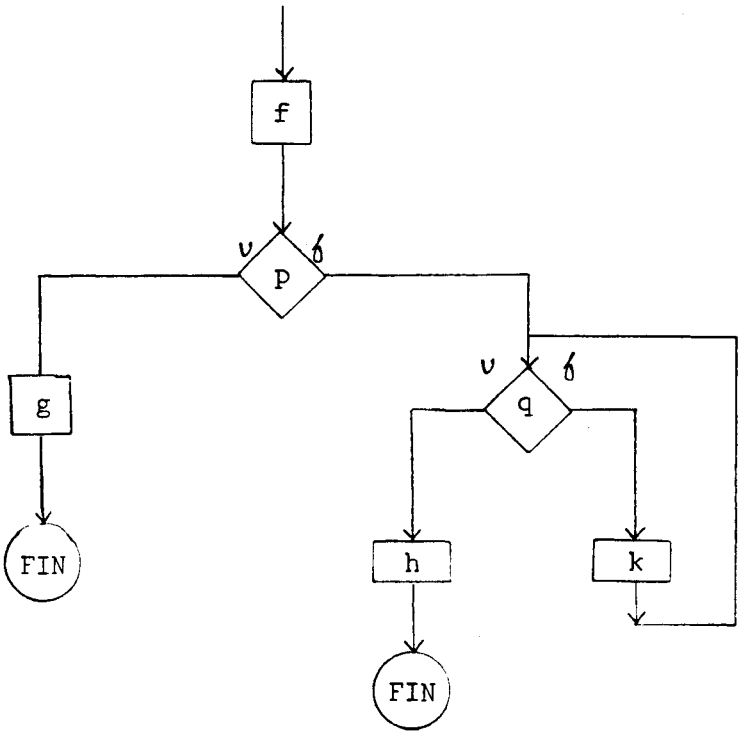
ou bien $p' < p$.

Un schéma de IANOV est dit *STANDARD* si l'arbre de programme associé est canonique.

EXEMPLE II.9. : Un schéma de IANOV S non libre :

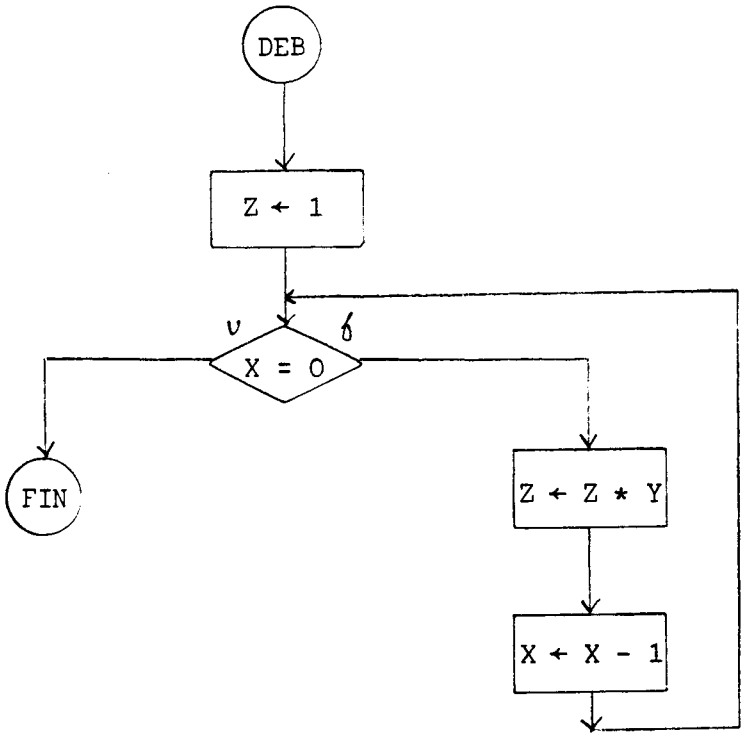


SCHEMA S



$\ell i(S)$

EXEMPLE II.10. :



Programme P

Pour ce programme, si les valeurs initiales des variables X, Y et Z sont des entiers x, y et z positifs ou nuls, leurs valeurs terminales sont 0, y, y^x.

Si on prend II = ($\mathcal{D}_{II}$, I)

tel que :

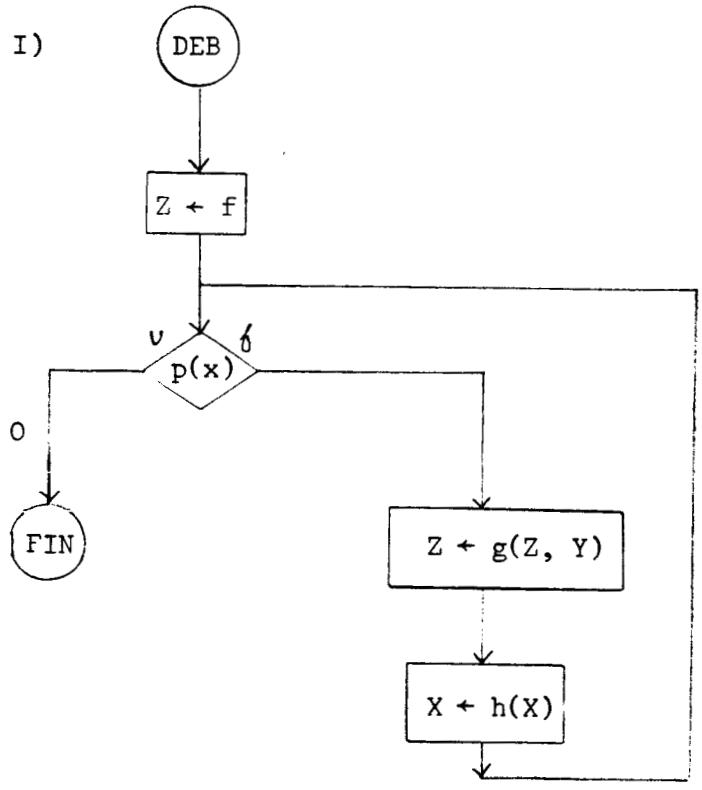
$\mathcal{D}_{II} = \mathbb{N}$

I(f) = 1

I(g)(Z, Y) = Z * Y

I(h)(X) = X - 1

I(p)(X) = vrai si X = 0



Alors (S_p, II) = P

Schéma de programme S_p

Si on prend II' = ($\mathcal{D}_{II'}$, I')

tel que :

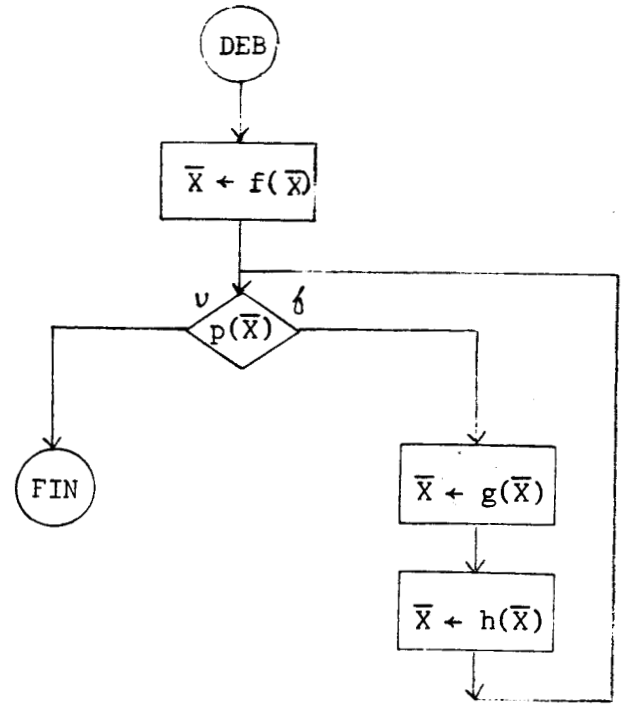
$\mathcal{D}_{II'} = \mathbb{N} \times \mathbb{N} \times \mathbb{N}$

I'(f(X, Y, Z) = (X, Y, 1)

I'(q)(X, Y, Z) = (X, Y, Z * Y)

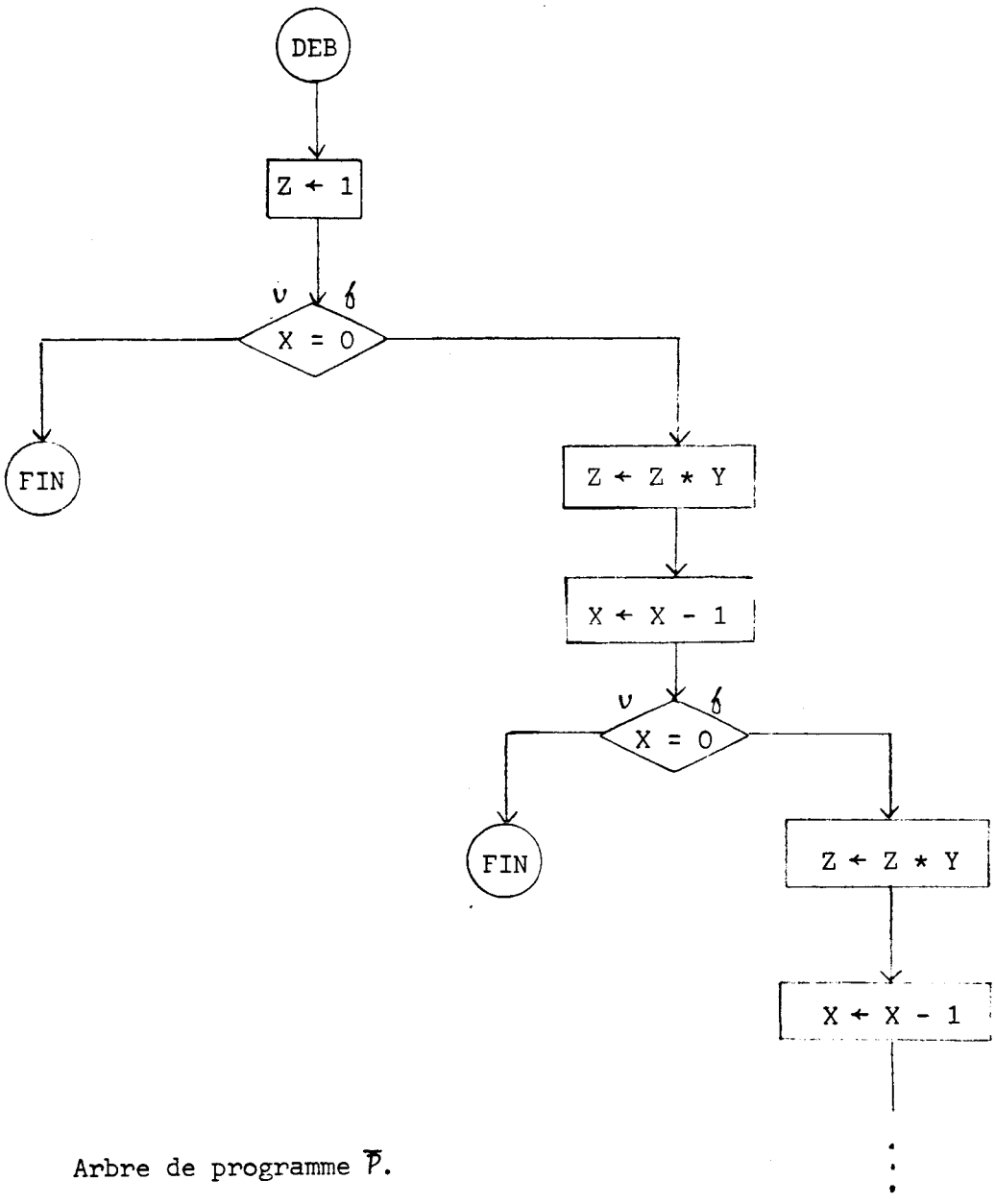
I'(h)(X, Y, Z) = (X - 1, Y, Z)

I'(p)(X, Y, Z) = (vrai si X = 0)

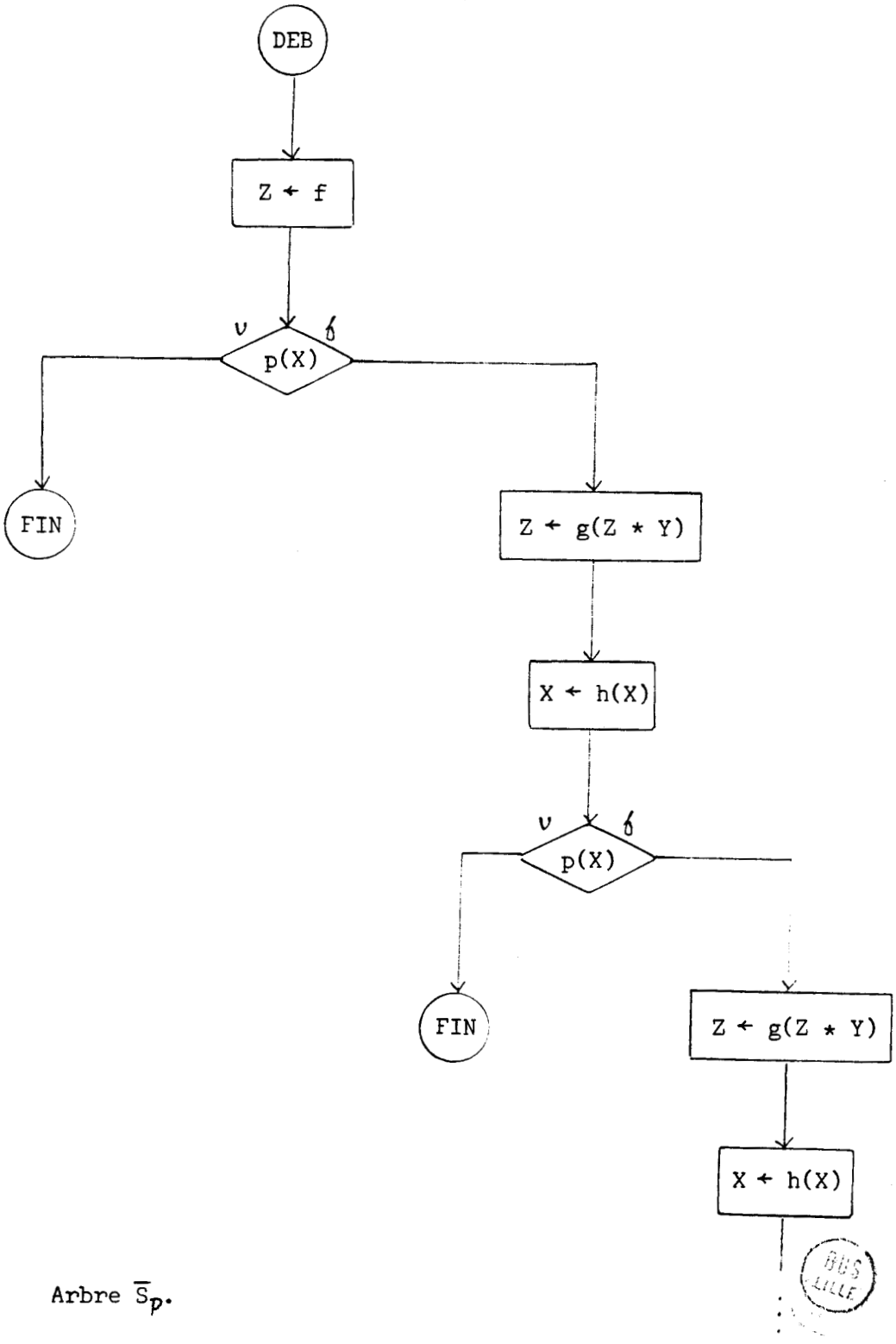


Alors (sq(p), II') = P

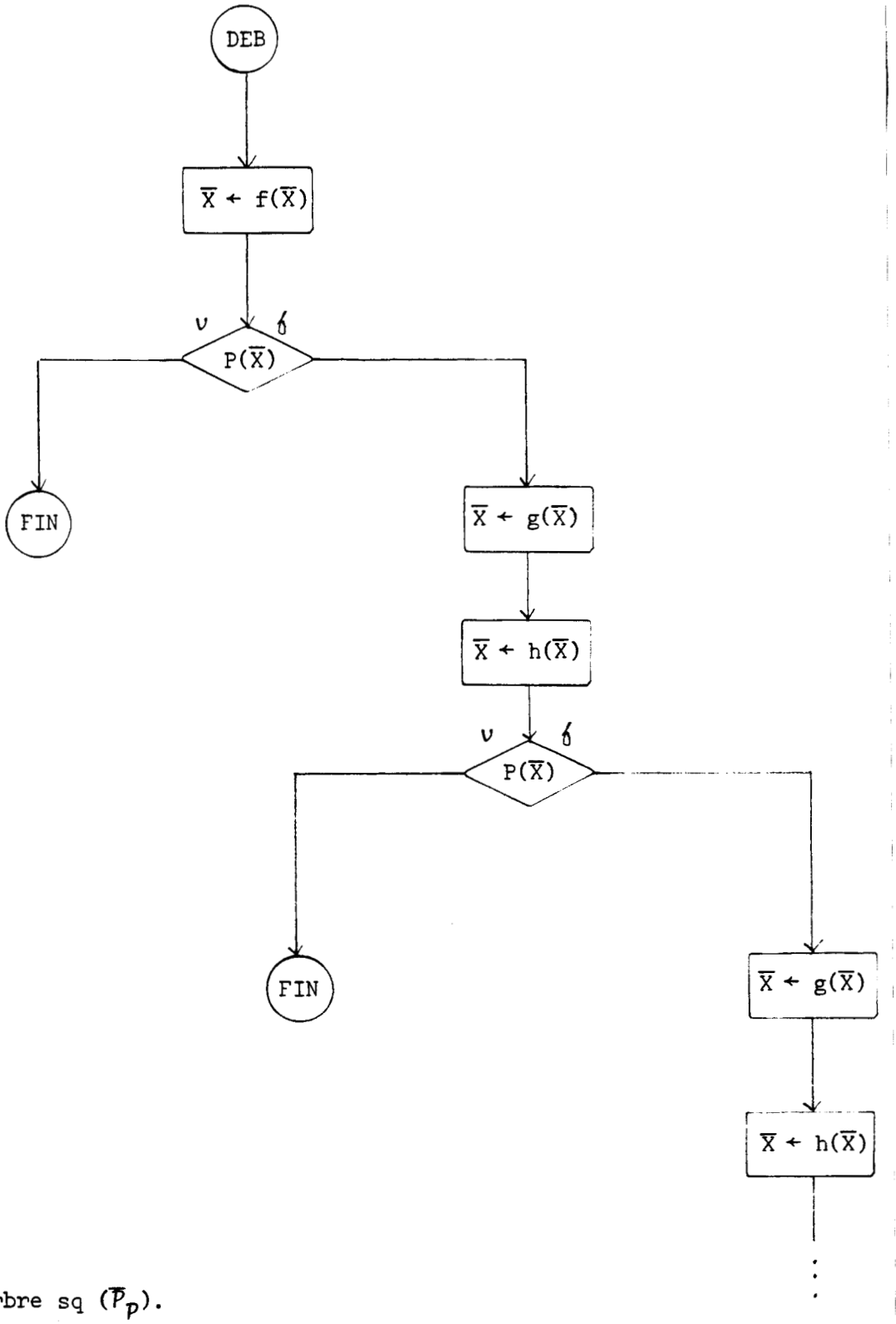
sq(P)



Arbre de programme $\bar{P}$.



Arbre $\overline{S}_p$.



Arbre sq $(\overline{P}_p)$.

CHAPITRE III

LES RELATIONS ENTRE LES SCHEMAS DE PROGRAMMES

EQUIVALENCE DE COUSINEAU

III-0 INTRODUCTION

Le problème d'équivalence entre les schémas de programmes a été étudié par de nombreux auteurs JACOB [19] [20] COUSINEAU [11], KOSARAJU [29], KASAI [26], MANNA [32],...

Ici, nous étudierons certaines relations entre les schémas de programme *itératifs* en essayant d'accumuler tous les résultats obtenus à propos de ces relations.

Considérons les relations :

1 - $\simeq$ (Paterson-Totale)
PT

2 - $\simeq$ (Ianov-Totale)
IT

3 - $\simeq$ (Cousineau-Totale)
CT

4 - $\equiv$ (Paterson-Partielle)
PP

5 - $\equiv$ (Ianov-Partielle)
IP

6 - $\equiv$ (Cousineau-Partielle)
CP

définies comme suit :

Soient S et S' deux schémas de programmes et $\overline{S}$ & $\overline{S}'$ les arbres de programmes associés.

1) $S \underset{PT}{\simeq} S'$ pour toute interprétation I = (D, I), si $I(\overline{S})$ et $I(\overline{S}')$ sont définies alors elles sont égales.

2) $S \underset{IT}{\simeq} S'$ pour toute interprétation I = (D, I), si $I(\overline{S})$ et $I(\overline{S}')$ sont définies, alors $sq(\overline{S}) \equiv sq(\overline{S}')$.

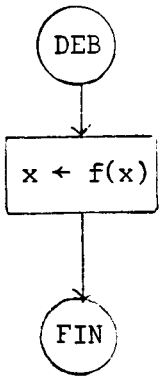
3) $S \underset{CT}{\simeq} S'$ pour toute interprétation I = (D, I), si $I(\overline{S})$ et $I(\overline{S}')$ sont définies, alors $\overline{S} = \overline{S}'$.

1') $S \underset{PP}{\equiv} S'$ pour toute interprétation I = (D, I), $I(\overline{S}) = I(\overline{S}')$.

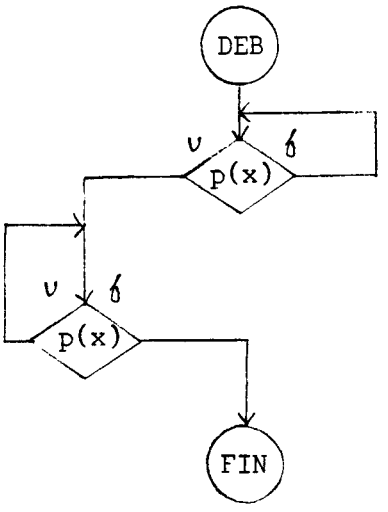
2') $S \underset{IP}{\equiv} S'$ pour toute interprétation I = (D, I), $sq(\overline{S}) \equiv sq(\overline{S}')$.

3') $S \underset{CP}{\equiv} S'$ pour toute interprétation I = (D, I), $\overline{S} = \overline{S}'$.

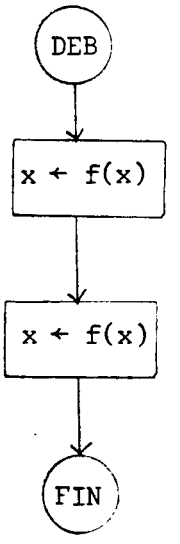
Notons que les relations $\underset{PT}{\simeq}$, $\underset{IT}{\simeq}$, $\underset{CT}{\simeq}$, ne sont pas des relations d'équivalence car elles ne sont pas transitives, comme sur les trois schémas de programme suivants :



S_1



S_2



S_3

On a : $S_1 \underset{CT}{\simeq} S_2$ & $S_2 \underset{CT}{\simeq} S_3$ mais $S_1 \underset{CT}{\not\simeq} S_3$.

Nous nous limiterons donc à travailler sur les relations d'équivalence, c'est-à-dire $\equiv_{PP}$, $\equiv_{IP}$ et $\equiv_{CP}$.

Pour faciliter, nous les appelerons simplement équivalences de PATERSON, IANOV et COUSINEAU, et nous les noterons respectivement $\equiv_{(P)}$, $\equiv_{(I)}$ et $\equiv_{(C)}$.

Il est évident qu'on peut obtenir, à partir des définitions, le résultat suivant :

$$\begin{array}{ccccc} C_{PT} & \subset & C_{IT} & \subset & C_{CT} \\ \cap & & \cap & & \cap \\ C_{PP} & \subset & C_{IP} & \subset & C_{CP} \end{array}$$

où chaque C représente la classe des relations indiquées.

III.1. EQUIVALENCE DE COUSINEAU

III.1.1. Définitions

Soient S et S' deux schémas de programmes, comme nous l'avons vu dans le chapitre précédent, à S et S' correspondent deux *systèmes d'équations élémentaires* $\hat{S}$ et $\hat{S}'$.

Ces deux systèmes d'équations élémentaires ont pour solutions deux *arbres reconnaissables* $\overline{S}$ et $\overline{S}'$ qui sont les arbres de programmes associés à S et S'.

On peut également résoudre deux systèmes d'équations $\hat{S}$ et $\hat{S}'$ pour obtenir deux *expressions rationnelles* $\tilde{S}$ et $\tilde{S}'$ qui représentent $\overline{S}$ et $\overline{S}'$.

Notons qu'il existe une infinité de façon de résoudre un même système d'équations qui aboutissent à une infinité d'expressions rationnelles différentes, mais équivalentes.

Nous disons que S est *COUSINEAU-équivalent* à S' et nous le notons $S \underset{(c)}{\equiv} S'$ si et seulement si $\overline{S} = \overline{S}'$.

D'après les définitions, on peut également dire que :

$$S \underset{(c)}{\equiv} S' \Longleftrightarrow \overline{S} = \overline{S}' \Longleftrightarrow \hat{S} \equiv \hat{S}' \Longleftrightarrow \tilde{S} \equiv \tilde{S}' \Longleftrightarrow Tr(S) = Tr(S')$$

III.1.2. Théorie consistante et complète pour l'équivalence de COUSINEAU T_c

La théorie que nous allons étudier a été proposée par COUSINEAU [11] pour l'équivalence des expressions rationnelles et par définition elle sera applicable pour l'équivalence de COUSINEAU.

La théorie T_c = {A1, A2, A3, R1, R2}, tel que :

$\vdash$ A1 : $E \equiv E$

$\vdash$ A2 : $E \equiv (E + 1)^*$

$\vdash$ A3 : $E \equiv E[0\backslash\backslash E, i\backslash\backslash i - 1]$

$\vdash$ R1 : substitution

$$\left.\begin{array}{l} E_1 \equiv E_2 \\ E \equiv H \\ E_2' \text{ occurrence de } E_2 \text{ dans } E \end{array}\right\} \; H \equiv E[E_2' \backslash\backslash E_1]$$

⊢ R2 : résolution

$$\left.\begin{array}{l} X \equiv E \\ X \text{ élément terminal de } E \\ \text{ayant dans } E \text{ les occurrences } \vec{X} = X^1, \ldots, X^p \end{array}\right\} \; X \equiv E[\vec{X} \backslash\backslash 0,\ s \backslash\backslash s+1]$$

où E, E_1, E_2, F, F_1, X, H sont des expressions rationnelles, (s désigne les éléments terminaux entiers de E qui ne figurent pas dans X).

Avant de prouver la consistance et la complétude de cette théorie pour l'équivalence des expressions rationnelles, nous allons établir quelques lemmes :

Lemme III.1. : Pour toute expression rationnelle E et pour tout entier n, on a : $\overline{E+n} = \overline{E} + n$

PREUVE : On démontre le résultat plus fort :

E une expression rationnelle, n, p ∈ N

$$\overline{E[i \geq p \backslash\backslash i+n]} = (\overline{E})[i \geq p \backslash i+1]$$

On fait une récurrence sur ρ(E) complexité de E

1 - $\rho(E) = 0$ soit $E = \Omega$ ⎫ $\overline{E[i \geq p \backslash\backslash i+n]} = \overline{E}$

ou : $E = i < p$ ⎭ $\overline{E}[i \geq p \backslash i+n] = \overline{E}$

soit $E = i \geq p$ $\overline{E[i \geq p \backslash i+n]} = \overline{(i+n)}$

par la définition

d'autre part $(\overline{E})[i \geq p \backslash i+n] = (\bar{i})[i \geq p \backslash i+n] = \overline{(i+n)}$

2 - $E = f(E_1, \ldots, E_{a(f)})$

$$\overline{E[i \geq p \backslash\backslash i+n]} = \overline{f(E_1, \ldots, E_{a(f)})\,[i \geq p \backslash\backslash i+n]}$$

$$= f(\overline{E_1[i \geq p \backslash i+n]}, \ldots, \overline{E_{a(f)}[i \geq p \backslash i+n]}$$

$$= f(\overline{E}_1[i \geq p \backslash i+n], \ldots, \overline{E}_{a(f)}[i \geq p \backslash i+n])$$

par hypothèse de récurrence

$$= f(\overline{E}_1, \ldots, \overline{E}_{a(f)})[i \geq p \backslash i+n]$$

$$= \overline{E}[i \geq p \backslash i+n]$$

3 - $E = E_1^*$

d'une part : $\overline{E[i \geq p \backslash\backslash i+n]} = \overline{(E_1[i \geq p+1 \backslash i+n]^*)}$ par définition

$= \overline{(E_1[i \geq p+1 \backslash i+n])^*}$ par définition

$= ((\overline{E}_1[i \geq p+1 \backslash i+n])^*$ par hypothèse de récurrence

d'autre part : $(\overline{E})[i \geq p \backslash i+n] = (\overline{E}_1)^*[i \geq p \backslash i+n]$

alors il faut établir que : $((\overline{E}_1)[i \geq p+1 \backslash i+n])^* = (\overline{E}_1)^*[i \geq p \backslash i+n]$

cela résulte du fait que pour tout arbre A :

$$A[i \geq p+1 \backslash i+n]^* = A^*[i \geq p \backslash i+n]$$

En effet, posons $dom(A) = D_0 \cup D_1 \cup D_1' \cup D_2$ avec

$$m \in D_0 \iff A(m) = 0$$

$$m \in D_1 \iff A(m) \in N \backslash \{0\} \text{ et } A(m) < p+1$$

$m \in D_1' \iff A(m) \in N \setminus 0$ et $A(m) \geq p+1$

$m \in D_2 \iff A(m) \notin N$

$Dom(A[i \geq p+1 \setminus i+n]) = Dom(A)$

$A[i \geq p+1 \setminus i+n](m) = A(m)$ si $m \in D_0 \cup D_1 \cup D_2$

$= A(m)+n$ si $m \in D_1'$

$Dom(A^*[i \geq p \setminus i+n]) = Dom(A^*) = D_0^*(D_1 \cup D_1' \cup D_2)$

$Dom(A[i > p+1 \setminus i+n]^*) = D_0^*(D_1 \cup D_1' \cup D_2)$

$\Rightarrow$ $\underline{Dom(A[i \geq p+1 \setminus i+n]^*) = Dom(A^*[i \geq p \setminus i+n])}$

$A^*[i \geq p \setminus i+n](m) = A(m'')$ si $m = m'm''$, $m' \in D_0^*$ et $m'' \in D_2$

$= A(m'')-1$ si $m = m'm''$, $m' \in D_0^*$ et $m'' \in (D_1 \cup D_1')$

avec $A(m'')-1 < p$ i.e. $A(m'') < p+1$ ou $m'' \in D_1$

$= A(m'')-1+n$ si $m = m'm''$, $m' \in D_0^*$ et $m'' \in (D_1 \cup D_1')$

avec $A(m'')-1 \geq p$ i.e. $A(m'') \geq p+1$ ou $m'' \in D_1'$

$A[i \geq p+1 \setminus i+n]^*(m) = A[i \geq p+1 \setminus i+n](m'') =$

$A(m'')$ si $m = m'm''$, $m' \in D_0^*$ et $m'' \in D_2$

$= A[i \geq p+1 \setminus i+n](m'')-1 = A(m'')-1$ si $m'' \in D_1$

$= A[i \geq p+1 \setminus i+n](m'')-1 = A(m'')+n-1$ si $m'' \in D_1'$

Lemme III.2. : (point fixe). Soit $(K, \leq, \perp)$ un ensemble ordonné complet et $\perp$ son plus petit élément.

Soit $f : K \to K$ une fonction continue. Elle admet un plus petit point fixe u et on écrit $u = \sup_{n \geq 1}(\{f^n(\perp)\})$

PREUVE : On a $\perp \leq f(\perp)$ et puisque f est croissante $f(\perp) \leq f^2(\perp) \leq \ldots \leq \ldots \leq f^n(\perp) \leq \ldots$

L'élément $u = \mathrm{Sup}(\{ f^n(\perp)\})$ est donc bien défini, puisque f est
$n \geq 0$
continue, alors $f(u) = f(\mathrm{Sup}(\{f^n(\perp)\})) = \mathrm{Sup}(\{f^{n+1}(\perp)\}) = \mathrm{Sup}(\{f^n(\perp)\})$.
$n \geq 0$ $n \geq 0$ $n \geq 1$

Pour montrer que u est le plus petit point fixe supposons qu'il existe un point fixe u' alors $\perp \leq u' \Rightarrow f(\perp) \leq f(u') = u' \ldots \Rightarrow$

$\forall n\ f^n(\perp) \leq u' \Rightarrow \mathrm{sup}(\{f^n(\perp)\}) \leq u' \Rightarrow u < u'$.
$n \geq 0$

Lemme III.3. : Pour tout arbre $\overline{A} \neq 0$. $\overline{A}^*$ est la solution unique de l'équation :

$$X = \overline{A}[0 \backslash X,\ i \geq 1 \backslash i - 1]$$

PREUVE : Considérons la fonction $\eta : M^\infty(W_\Omega) \to M^\infty(W_\Omega)$ définie comme : $\eta(X) = \overline{A}[0 \backslash X,\ i \geq 1 \backslash i - 1]$, elle est continue sur l'ensemble (magma) complet $M^\infty(W_\Omega)$, alors elle admet un plus petit point fixe (Lemme III.2.) qui est $\mathrm{Sup}(\{\eta^n(\Omega)\})$. Il nous reste à montrer que $\eta^n(\Omega) = \overline{A}^n - 1$.

On démontre par récurrence sur n

Pour $n = 0$, $\eta^0(\Omega) = \overline{A}^0 - 1 = \Omega$

Supposons que le lemme soit vrai pour tout entier $\leq n$.

$\eta^{n+1}(\Omega) = \overline{A}[0 \backslash \eta^n(\Omega),\ i \geq 1 \backslash i - 1]$ par définition

$= \overline{A}[0 \backslash \overline{A}^n - 1,\ i \geq 1 \backslash i - 1]$ par hypothèse de récurrence

$= \overline{A}[0 \backslash \overline{A}^n] - 1$

$= [\overline{A} \cdot \overline{A}^n] - 1$

$= \overline{A}^{n+1} - 1$

Lemme III.4. : Si E, F et G sont des expressions rationnelles et F' une occurrence de F dans E

$$F \equiv G \Rightarrow E \equiv E[F'\backslash G]$$

PREUVE : On fait une récurrence sur la complexité de l'expression $H = E[F\backslash\Omega]$

i - Si $\rho(H) = 0$ alors $E = F$, $E[F'\backslash G] = G$ d'où le résultat.

ii - Supposons les propriétés prouvées pour $\rho(H) < n$ et soit $\rho(H) = n$:

- ou bien $E = f(E_1, \ldots, E_{a(f)})$ et F' dans E_i

$$E[F'\backslash G] = f(E_1, \ldots, E_{i-1}, E_i[F\backslash G], E_{i+1}, \ldots, E_{a(f)})$$

$F \equiv G \Rightarrow E_i \equiv E_i[F'\backslash G]$ par hypothèse de récurrence

$\Rightarrow E \equiv E[F'\backslash G]$ car les fonctions de f sont croissantes

- ou bien $E = E_1^*$ et F' dans E_1

$$E[F'\backslash G] \equiv E_1[F'\backslash G]^*$$

$F \equiv G \Rightarrow E_1 \equiv E_1[F'\backslash G]$ par hypothèse de récurrence

$E \equiv E[F'\backslash G]$ car l'étoile est *croissante*.

Lemme III.5. : Si E est une expression rationnelle et $\vec{T} = T^{(1)}, \ldots, T^{(n)}$ des occurrences d'un élément terminal T dans E,

$$\overline{E} = \overline{(E[\vec{T}\backslash\backslash\theta(0)])\ [\theta(0)\backslash\overline{T}]}$$

(θ est un symbole unaire qui représente certaines occurrences d'éléments terminaux, il sera toujours supposé ne pas appartenir à F).

PREUVE : On fait une récurrence sur la complexité de E :

1) $\rho(E) = 0$ on ne peut avoir que $T = E$

alors $(E[T^n\backslash\theta(0)]) = \theta(0)$ et

$(E[T^1\backslash\backslash\theta(0)])\ [\theta(0)\backslash\overline{T}] = (\{\theta(0)\})\ [\theta(0)\backslash\overline{T}] = \overline{T} = \overline{E}$.

2) Supposons le résultat démontré pour toute expression de complexité inférieure à m et soit E une expression rationnelle de complexité m.

Deux cas sont possibles :

$\underline{1^{er}\ cas}$: $E = f(E_1,\ldots, E_{a(f)=p})$

ou bien $T = E$ (cas trivial)

ou bien les occurrences des éléments terminaux T se répartissent entre les sous-expressions $E_1,\ldots, E_p$. Nous noterons $T_1,\ldots, T_p$ les éléments de $\overline{T}$ qui se trouvent dans $E_1,\ldots, E_p$.

$$E[\vec{T}\backslash\backslash\theta(0)] = f(E_1[\vec{T}_1\backslash\backslash\theta(0)],\ldots, E_1[\vec{T}_p\backslash\backslash\theta(0)])$$

$$\overline{(E[\vec{T}\backslash\backslash\theta(0)])\ [\theta(0)\backslash\overline{T}]} = \overline{(f(E_1[\vec{T}_1\backslash\backslash\theta(0)\ ,\ldots\ E_p[\vec{T}_p\backslash\backslash\theta(0)]))\ [\theta(0)\backslash\overline{T}]}$$

$$= \overline{f(E_1[\vec{T}_1\backslash\backslash\theta(0)\ ,\ldots\ E_p[\vec{T}_p\backslash\backslash\theta(0)])\ [\theta(0)\backslash\vec{T}]}$$

$$= f(E_1[\vec{T}_1\backslash\backslash\theta(0)])\ [\theta(0)\backslash\overline{T}],\ldots\ E_p[\vec{T}_p\backslash\backslash\theta(0)])\ [\theta(0)\backslash\backslash T])$$

$$= f(\overline{E}_1,\ldots, \overline{E}_p)$$ par hypothèse de récurrence

$$= \overline{E}$$

$\underline{2^{ème}\ cas}$: $E = E_1^*$

ou bien $T = E$ (cas trivial)

ou bien $T + 1$ est élément terminal de E_1 les occurrences $\vec{T}' = T^{(1)'},\ldots, T^{(n)'}$ correspondant à $T = T^{(1)},\ldots, T^{(n)}$.

D'une part, par hypothèse de récurrence, nous avons :

$$\overline{E}_1 = \overline{(E_1[\vec{T}'\backslash\backslash\theta(0)])\ [\theta(0)\backslash\overline{T+1}]} = \overline{(E_1[\vec{T}'\backslash\backslash\theta(0)])\ [\theta(0)\backslash\overline{T}+1]}$$

En vertu du Lemme III.1.

$$\overline{E} = \overline{(E_1^*)} = (E_1[\vec{T}'\backslash\backslash\theta(0)])\ [\theta(0)\backslash\overline{T}+1])^*$$

D'autre part, nous avons :

$$(E[\vec{T}\backslash\backslash\theta(0)]) = E_1[\vec{T}'\backslash\backslash\theta(1)]^*$$

Par conséquent :

$$(E[\vec{T}\backslash\backslash\theta(0)])\ [\theta(0)\backslash\overline{T}] = E_1[\vec{T}'\backslash\backslash\theta(1)])^*\ [\theta(0)\backslash\backslash\overline{T}]$$

Il nous faut établir que :

$$((E_1[\vec{T}'\backslash\backslash\theta(0)])\ [\theta(0)\backslash\backslash\overline{T}+1] = (E_1[\overline{T}'\backslash\backslash\theta(1)])^*[\theta(0)\backslash\overline{T}]$$

Supposons : $H = E_1[\vec{T}'\backslash\backslash\theta(0)]$ et $H' = E_1[\vec{T}'\backslash\backslash\theta(1)]$,

alors $H' = H[\theta(0)\backslash\backslash\theta(1)]$

On a donc : $Dom(\overline{H}) = Dom(\overline{H}')$

$$\overline{H}(m) = \theta \Rightarrow H'(m) = \theta,\ \overline{H}(m1) = 0 \text{ et } \overline{H}'(m1) = 1$$

Il faut prouver :

$$(\overline{H}[\theta(0)\backslash\overline{T}+1])^* = (\overline{H}')^*\ [\theta(0)\backslash\overline{T}]$$

D'une part posons :

$$Dom(\overline{H}) = Dom(\overline{H}') = D_0 \cup D_0' \cup D_1 \cup D_2 \text{ avec}$$

$$\overline{H}(m) = \overline{H}'(m) = 0 \iff m \in D_0$$

$$\overline{H}(m) = \overline{H}'(m) = \theta,\ \overline{H}(m1) = 0,\ \overline{H}'(m1) = 1 \iff m \in D_0'$$

$$\overline{H}(m) = \overline{H}'(m) \in \mathbb{N}\backslash\{0\} \iff m \in D_1$$

$$\overline{H}(m) = \overline{H}'(m) \in \mathbb{N} \iff m \in D_2$$

D'autre part, posons : $D = \mathrm{Dom}(\overline{T})$

$$\mathrm{Dom}(\overline{H}[\theta(0)\backslash\overline{T}+1]) = D_0 \cup D_0' D \cup D_1 \cup D_2$$

$$\mathrm{Dom}(\overline{H}[\theta(0)\backslash\overline{T}+1]^*) = D_0^*(D_0' D \cup D_1 \cup D_2)$$

$$\mathrm{Dom}(\overline{H}'^*) = D_0^*(D_0' \cup D_1 \cup D_2)$$

$$\mathrm{Dom}(\overline{H}'^*[\theta(0)\backslash\overline{T}]) = D_0^*(D_0' D \cup D_1 \cup D_2)$$

- $\overline{H}[\theta(0)\backslash\overline{T}+1]^*(m) = \overline{H}[\theta(0)\backslash\overline{T}+1](m'') = \overline{H}(m'')$ si $m = m'm''$ et $m' \in D_0^*$, $m'' \in D_2$

$$\overline{H}'^*[\theta(0)\backslash\overline{T}](m) = \overline{H}'(m'') = \overline{H}(m'')$$

- $\overline{H}[\theta(0)\backslash\overline{T}+1]^*(m) = \overline{H}[\theta(0)\backslash\overline{T}+1](m'') - 1$ si $m = m'm''$ et $m' \in D_0^*$, $m'' \in D_1$

$$\overline{H}'^*[\theta(0)\backslash\overline{T}](m) = \overline{H}'(m'') - 1 = \overline{H}(m'') - 1$$

- $\overline{H}[\theta(0)\backslash\overline{T}+1]^*(m) = \overline{H}[\theta(0)\backslash\overline{T}+1](m_1'') = (\overline{T}+1)(m_2'') = \overline{T}(m_2'')$ si $m = m'm_1''m_2''$ et $m' \in D_0^*$, $m_1'' \in D_0'$, $m_2'' \in D$ $\overline{T}(m_2'') \notin N$

$$\overline{H}'^*[\theta(0)\backslash\overline{T}](m) = \overline{H}'(m'') = \overline{T}(m_2'')$$

- $\overline{H}[\theta(0)\backslash\overline{T}+1]^*(m) = \overline{H}[\theta(0)\backslash\overline{T}+1](m'') - 1 = (\overline{T}+1)(m_2'') - 1 = \overline{T}(m_2'')$ si $m = m'm_1''m_2''$ et $m' \in D_0^*$, $m_1'' \in D_0'$, $m_2'' \in D$ $\overline{T}(m_2'') \in N$

$$\overline{H}'^*[\theta(0)\backslash\overline{T}](m) = \overline{H}'(m'') = \overline{T}(m_2'')$$

Lemme III.6. : Si $\vec{X} = X^{(1)},\ldots, X^{(p)}$ sont des occurrences d'un élément terminal X dans une expression rationnelle E et si $X \equiv E$ et $(X \neq E)$ alors :

$$X = E[\vec{X}\backslash\backslash 0,\ s\backslash s+1]^*$$

(s désigne les éléments terminaux de E qui ne figurent pas dans X).

PREUVE : Posons $H = E[\vec{X}\backslash\backslash 0,\ s\backslash s+1]$

On va montrer que $X = H^*$

Nous nous servirons notamment du résultat du Lemme III.5. c'est-à-dire $\overline{X} = \overline{E} = (E[\vec{X}\backslash\backslash\theta(0)])\ [\theta(0)\backslash\overline{X}]$

Posons $K = E[\vec{X}\backslash\backslash\theta(0)]$ et $Dom(K) = D_0 \cup D_0^1 \cup D_1 \cup D_2$ tous disjoints deux à deux tel que :

$$\overline{K}(m) = 0 \Longleftrightarrow m \in D_0$$
$$\overline{K}(m) = \theta,\ K(m1) = 0 \Longleftrightarrow m \in D_0{}^1$$
$$\overline{K}(m) \in \mathbb{N}\backslash\{0\} \Longleftrightarrow m \in D_1$$
$$K(m) \notin N \Longleftrightarrow m \in D_2$$

$Dom(H) = D_0 \cup D_1 \cup D_2$ avec :

$$\overline{H}(m) = 0 \Longleftrightarrow m \in D_0$$
$$\overline{H}(m) = \overline{K}(m) + 1 \Longleftrightarrow m \in D_1$$
$$\overline{H}(m) = \overline{K}(m) \Longleftrightarrow m \in D_2$$

Alors $Dom(\overline{H}^*) = D_0^*(D_1 \cup D_2)$, $Dom(\overline{X}) = D_0\ Dom(\overline{X})D_1 \cup D_2$

Comme $X \neq E$, le mot vide $\notin D_0$ et par conséquent

$$Dom(\overline{X}) = D_0^*(D_1 \cup D_2) = Dom(\overline{H}^*)$$

On a $\overline{X}(m) = \overline{K}(m'')$ si $m = m'm''$, $m' \in D_0^*$, $m'' \in D_1 \cup D_2$

Donc $\overline{H}^*(m) = \overline{H}(m'') = \overline{K}(m'') = \overline{X}(m'')$ si $m = m'm''$, $m' \in D_0^*$, $m'' \in D_2$

$\overline{H}^*(m) = \overline{H}(m'') - 1 = \overline{K}(m'') = \overline{X}(m'')$ si $m = m'm''$, $m' \in D_0$, $m'' \in D_1$.

III.1.3. La preuve de la consistance de T_c

Considérons le magma $M^\infty(W_\Omega) = \langle M^\infty(W_\Omega), F_M, C_M \rangle$.

Nous allons montrer que ce magma peut être un modèle pour T_c c'est-à-dire que toutes les formule de T_c sont valides dans $M^\infty(W_\Omega)$

A1 : $E \equiv E$ car $\overline{E} = \overline{E}$ par définition

A2 : $E \equiv (E+1)^*$, $\overline{E} = (\overline{E}+1)^*$

$= \overline{(E+1)^*}$ Lemme III.1.

$= (\overline{E+1})^*$ par définition

A3 : $E^* = E[0\backslash\backslash E^*, i \geq 1 \backslash i-1]$

$\overline{E}^* = \overline{E}[0\backslash\backslash \overline{E}^*, i \geq 1 \backslash\backslash i-1]$ Lemme III.3.

R1 : $E_1 \equiv E_2 \Rightarrow E_2 \equiv E_1$ puisque $\equiv$ est une relation d'équivalence. De ce fait que E_2' est une occurrence de E_2 dans E, on obtient par le Lemme III.4, $E \equiv E[E_2' \backslash E_1]$

$E \equiv H \Rightarrow H \equiv E$ et $E \equiv E[E_2' \backslash E_1] \Rightarrow H \equiv E[E_2' \backslash E_1]$ par transitivité de $\equiv$

R2 : est un résultat du Lemme III.6.

III.1.4. La preuve de la complétude de T_c

Il faut montrer que $\forall$ a une formule ou bien $a \in T_c$

ou bien $\sim a \in T_c$

D'abord nous allons montrer que les règles symétrie et transitivité peuvent être considérer comme deux éléments de T_c.

- *symétrie* $\vdash F \equiv G$ alors $\vdash G \equiv F$?

$\vdash F \equiv G$

$\vdash G \equiv G$

G' occurrence de G dans G

$\Big\}$ (R1) $\Longrightarrow \vdash G \equiv G[G'\backslash F]$

$\Longrightarrow \vdash G \equiv F$

- *transitivité* $\vdash F \equiv G$ et $\vdash G \equiv H$ alors $\vdash F \equiv H$?

$\vdash F \equiv G$

$\vdash G \equiv H$

G' occurrence de G dans G

$\Big\}$ (R1) $\Longrightarrow \vdash H \equiv G[G'\backslash F]$ symétrie

$\Longrightarrow \vdash H \equiv F \Longrightarrow F \equiv H$

Dorénavant nous considérons T_c = {A1, A2, A3, R1, R2, R3 (symétrie), R4 (transitivité)}.

Lemme III.7. : Soient 2n formules $E_1, \ldots, E_n, F_1, \ldots, F_n$ et n membres droits d'équations rationnelles $T_1(X_1, \ldots, X_n), \ldots, T_n(X_1, \ldots, X_n)$.

Si $\forall i \quad 1 \leq i \leq n \quad \vdash (E_i, F_i) = T_i((E_1, F_1), \ldots, (E_n, F_n))$.

Alors $\forall i \quad 1 \leq i \leq n \quad \vdash E_i \equiv F_i$.

PREUVE : On fait une récurrence sur n.

1 - n = 1 On a $\vdash E = T(E) \xRightarrow{(R2)} \vdash E = T(0)[i\backslash\backslash i+1]^*$

de $\vdash F = T(F) \xRightarrow{(R2)} \vdash F = T(0)[i\backslash\backslash i+1]^*$

$\xRightarrow{(R3)} \vdash T(0)[i\backslash\backslash i+1]^* = F$

et par (R4) et (A1) on obtient $\vdash E \equiv F$.

2 - Supposons que le résultat établi pour tout entier $\leq n$ et soit un système de n équations.

On a : $E_n = T_n(E_1,\ldots, E_n) \xRightarrow{(R2)} E_n = T_n(E_1,\ldots, E_{n-1}, 0)\ [i\backslash\backslash i+1]^*$

$F_n = T_n(F_1,\ldots, F_n) \xRightarrow{(R2)} F_n = T_n(F_1,\ldots, F_{n-1}, 0)\ [i\backslash\backslash i+1]^*$

Par (R1) nous avons : $\forall i \quad 1 \leq i \leq n-1$

$\vdash (E_i, F_i) = T_i((E_1, F_1),\ldots, (E_{n-1}, F_{n-1}), (T_n(E_1,\ldots, E_{n-1}, 0)\ [i\backslash\backslash i+1]^*,$

$T_n(F_1,\ldots, F_{n-1}, 0)\ [i\backslash\backslash i+1]^*))$.

Par hypothèse de récurrence $\forall i \quad 1 \leq i \leq n-1$, on a : $\vdash E_i \equiv F_i$

et par (R1) , (R3) et (A1) $\vdash E_n = F_n$.

C.Q.F.D.

Lemme III.8. : Toute expression rationnelle est élémentairement décomposable

PREUVE : Nous allons démontrer par récurrence le résultat plus fort :

Etant donnée une expression rationnelle E et des expressions rationnelles $G_1,\ldots, G_p$, il existe des expressions rationnelles $E_1,\ldots, E_n$ telles que :

$$\alpha \begin{cases} \text{i)} \quad \vdash E = E_1 \\ \text{ii)} \ \forall i \ \ 1 \le i \le n \text{ ou bien } \exists n \in \mathbb{N}_\Omega, \vdash E_i = n \\ \qquad\qquad \text{ou bien } \exists j \in [p], \vdash E_j = G_j \\ \qquad\qquad \text{ou bien } \exists f \in F, \exists i_1, \ldots, i_{a(f)}, \vdash E_i = f(E_{i_1}, \ldots, E_{i_{a(f)}}) \end{cases}$$

On fait une récurrence sur la complexite de E sur magma $M^\infty(F \cup \{*\}, \mathbb{N}_\Omega \cup G)$ où $G = \{G_1, \ldots, G_p\}$ que nous noterons $\rho_G(E)$.

1 - Si $\rho_G(E) = 0$, on a soit $E = n \in \mathbb{N}_\Omega$, soit $E = G \in G$ dans ces cas nous prenons $E = E_1$ et nous avons $\vdash E = E_1$ ou bien $\vdash E_1 = n \in \mathbb{N}_\Omega$ ou bien $\vdash E_1 = G \in G$

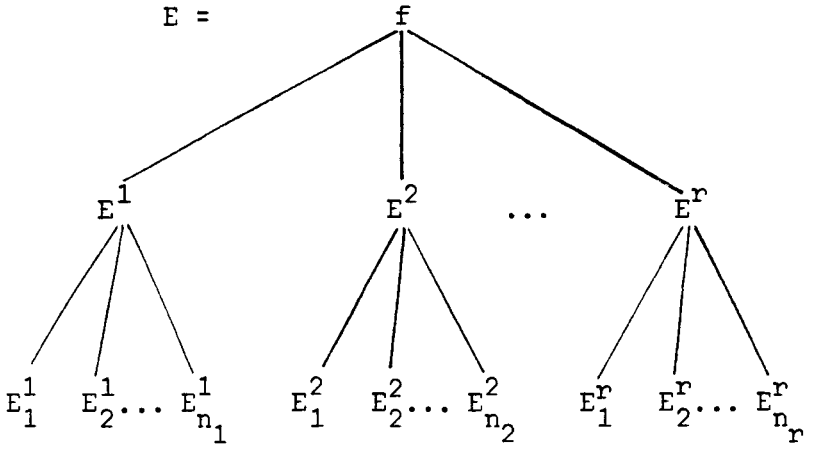
2 - Supposons que pour toute expression rationnelle E telle que $\rho_G(E) < q$ la condition (α) soit satisfaite et soit une expression rationnelle E telle que $\rho_G(E) = q$.

Deux cas sont à considérer

- $E = f(E^1, \ldots, E^r)$ par hypothèse de récurrence pour chaque expression E^i, il existe des expressions $E^i_1, \ldots, E^i_{n_i}$ telles que :

I) $\vdash E^i = E^i_1$

II) $\forall j \ \ 1 \le j \le n_i$ ou bien $\exists n \in N_\Omega, \vdash E^i_j = n$
ou bien $\exists G_j \in G, \vdash E^i_j = G_j$
ou bien $\exists f \in F, \exists j_1, \ldots, j_{a(f)} \in [n_i]$
$\vdash E^i_j = f(E^i_{j_1}, \ldots, E^i_{j_{a(f)}})$



Si on prend $E_1 = f(E_1^1, \ldots, E_1^r)$ alors les expressions rationnelles $E_1, E_1^1, \ldots, E_{n_1}^1, E_1^2, \ldots, E_{n_2}^2, \ldots, E_1^r, \ldots, E_{n_r}^r$ satisfont la condition α vis-à-vis de G.

$$E = H^*$$

Soit $H' = H[0\backslash\backslash E,\ i \geq 1\backslash\backslash i - 1]$

Nous avons $\rho_{G\cup\{E\}}(H') < q$ par conséquent il existe des expressions $H_1', \ldots, H_n'$ telles que :

I) $\vdash H' = H_1'$

II) $\forall i, 1 \leq i \leq n$ ou bien $\exists n \in \mathbb{N}_\Omega \vdash H_i' = n$

ou bien $\exists G_1 \in G \vdash H_i' = G_s$

ou bien $\exists f \in F$ et $\exists i_1, \ldots, i_{a(f)} \in [n]$ tels que :

$$\vdash H_i' = f(H_{i_1}', \ldots, H_{i_{a(f)}}').$$

(A3)

Nous prendrons alors $E_1 = H_1' \Rightarrow \vdash E = H'$ et par hypothèse $\vdash H' = H_1'$ et nous avons donc $\vdash E = H_1'$.

Par ailleurs, si nous remplaçons les expressions H_i' telles que $\vdash H_i' = E$ par H_1' nous obtenons un ensemble de formules $\{H_1', \ldots, H_n'\}\backslash\{H_i' : H_i' = E\}$ qui vérifie la condition α vis-à-vis de G.

C.Q.F.D.

Théorème III.1. : La théorie T = {A1, A2, A3, R1, R2, R3, R4} est complète pour l'équivalence des expressions rationnelles.

PREUVE : Si deux expressions rationnelles E et F sont équivalentes, d'après les lemmes III.7. et III.8, on peut trouver à partir de leur décomposition élémentaire des expressions $E'_1,\ldots, E'_m$ et $F'_1,\ldots, F'_m$ telles que :

$$(\beta)\begin{cases} \text{i) } \vdash E = E'_1 \text{ et } F = F'_1 \\ \text{ii) } \forall i \quad 1 \leq i \leq m \text{ ou bien } \exists n \in \mathbb{N} \vdash (E'_i, F'_i) = n \\ \qquad\qquad \text{ou bien } \exists f \in F, \exists i_1,\ldots, i_{a(f)} \in [m] \text{ tels que :} \\ \qquad\qquad\qquad \exists (E'_i, F'_i) = f((E'_1, F'_1),\ldots, (F'_{a(f)}, F'_{a(f)}) \end{cases}$$

On peut déduire par le Lemme III.6 que $\vdash E'_1 = F'_1$ et donc que $\vdash E = F$.

Pour trouver les expressions $E'_1,\ldots, E'_m$ et $F'_1,\ldots, F'_m$ vérifiant β à partir de deux expressions rationnelles équivalentes E et F c'est-à-dire vérifier que si elles sont dans la théorie T_c ou non ; on considère deux décompositions élémentaires $E_1,\ldots, E_n$ et $F_1,\ldots, F_p$ de E et F. Lemme III.7. $\Rightarrow E \equiv E_1$ et $F \equiv F_1$ sont valides alors $\vdash E = E_1$ et $\vdash F = F_1$; on a $F \equiv G \Rightarrow F_1 \equiv G_1$.
Du fait que $E_1,\ldots, E_n$ constitue une décomposition élémentaire de E nous avons :

$$\text{ou bien } \exists q \in \mathbb{N} \quad \vdash E_1 = q$$
$$\text{ou bien } \exists f \in F, \exists i_1,\ldots, i_{a(f)} \in [n] \text{ tels que}$$
$$\vdash E_1 = f(E_{i_1},\ldots, E_{i_{a(f)}})$$

Du fait que $F_1,\ldots, F_p$ constitue une décomposition élémentaire de F nous avons :

$$\text{ou bien } \exists q \in \mathbb{N} \;, \vdash F_1 = q$$
$$\text{ou bien } \exists f' \in F, \exists j_1,\ldots, j_{a(f)} \in [p] \text{ tels que}$$
$$\vdash F_1 = f'(F_{j_1},\ldots, F_{j_{a(f)}})$$

Comme $E_1 \equiv F_1$ deux cas sont possibles :

I) ou bien $\exists q \in \mathbb{N}_\Omega$, $\vdash E_1 = q$ et $\vdash F_1 = q$

II) ou bien $\exists f \in F$, $\exists i_1, \ldots, i_{a(f)} \in [n]$, $\exists j_1, \ldots, j_{a(f)} \in [p]$

tels que : $\vdash (E_1, F_1) = f((E_{i_1}, F_{j_1}), \ldots, E_{i_{a(f)}}, F_{j_{a(f)}}))$

Dans le cas I, les expressions E, E_1, F et F_1 satisfont à la condition (β).

Dans le cas II, on a :

$$E_1 = f(E_{i_1}, \ldots, E_{i_{a(f)}}) \text{ et } F_1 = f(F_{j_1}, \ldots, F_{i_{a(f)}})$$

De la consistance de la théorie on déduit $\forall k \quad 1 \leq k \leq a(f), E_{i_k} = F_{j_k}$

On peut faire cette construction à propos des expressions équivalentes E_{i_k} et F_{i_k} $\forall k \in [a(f)]$. On obtient un nombre fini de couples d'expressions équivalentes (E_i, F_j). Pour chaque couple ainsi obtenu, nous posons $E_i^j = E_i$ et $F_i^j = F_j$. Les ensembles d'expressions rationnelles $\{E_i^j\}$ et $\{F_i^j\}$ vérifient la condition (β).

C.Q.F.D.

EXEMPLE III.1. : Soit à démontrer $\vdash E_1 \equiv F_1$ où

$$E_1 = h(f(k(g(f(k(2), h(0)) \ell(1))^*), 1)^*)$$

$$F_1 = h(f(k(g(f(k(1), h(0)), \ell(f(k(0), 1)))^*), 0))$$

Décomposition élémentaire de E_1

$E_1 = h(E_2)$

$E_2 = f(E_3, E_2'')$

$E_2'' = 0$

$E_3 = k(E_4)$

$E_4 = g(E_5, E_8)$

$E_5 = f(E_6, E_7)$

$E_6 = k(E_6')$

$E_6' = 0$

$E_7 = h(E_4)$

$E_8 = \ell(E_2)$

Décomposition élémentaire de F_1

$\vdash F_1 = h(F_2)$

$\vdash F_2 = f(F_3, F_2'')$

$\vdash F_2'' = 0$

$\vdash F_3 = k(F_4)$

$\vdash F_4 = g(F_5, F_8)$

$\vdash F_5 = f(F_6, F_7)$

$\vdash F_6 = k(F_6')$

$\vdash F_6' = 0$

$\vdash F_7 = h(F_4)$

$\vdash F_8 = \ell(F_9)$

$\vdash F_9 = f(f_{10}, F_9'')$

$\vdash F_9'' = 0$

$\vdash F_{10} = k(F_4)$

On pose $E_2 = E_2$, $F_2^9 = F_2$, $E_3^{10} = E_3$, $F_3^{10} = F_3$, $E_2''^9 = F_2''^9 = 0$

$\vdash (E_1, F_1) = h(E_2, F_2)$

$\vdash (E_2, F_2) = f((E_3, F_3), (E_2'', F_2''))$

$\vdash (E_2'', F_2'') = 0$

$\vdash (E_3, F_3) = k(E_4, F_4)$

$\vdash (E_4, F_4) = g((E_5, F_5),(E_8, F_8))$

$\vdash (E_5, F_5) = f((E_6, F_6), (E_7, F_7))$

$\vdash (E_6, F_6) = k(E_6', F_6')$

$\vdash (E_6', F_6') = 0$

$\vdash (E_7, F_7) = h(E_4, F_4)$

$\vdash (E_8, F_8) = \ell(E_2^9, F_2^9)$

$$- (E_2^9, F_2^9) = f((E_3^{10}, F_3^{10}), (E_2''^9, F_2''^9))$$
$$- (E_2''^9, F_2''^9) = 0$$
$$- (E_3^{10}, F_3^{10}) = k(E_4, F_4)$$

Alors - $E_1 \equiv F_1$ est une formule de T.

On peut schématiser la preuve de la complétude de T_c par :

schéma de programme S

$\Downarrow$ en utilisant uniquement les éléments de T_c

expression rationnelle associée $\tilde{S}$

$\Downarrow$ par définition

arbre de programme associé $\bar{S}$

EXEMPLE III.2. : Considérons les deux schémas de programmes S et S' ci-dessous :

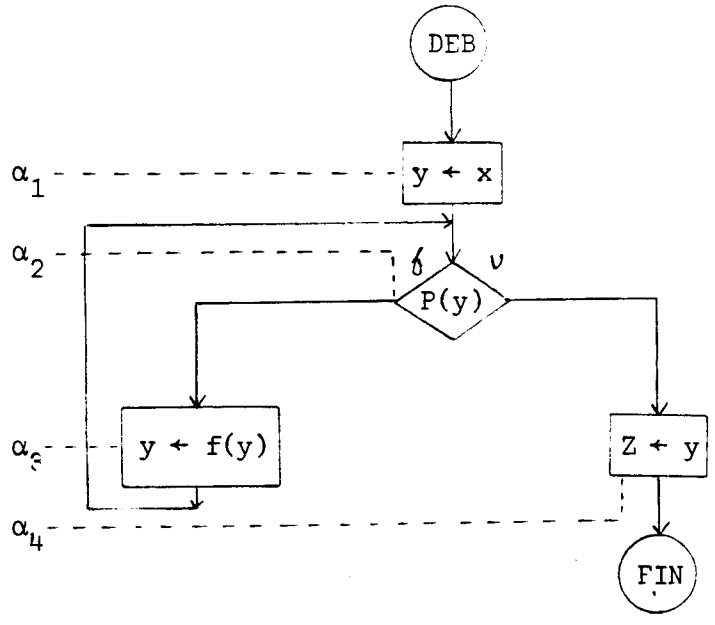


Schéma S

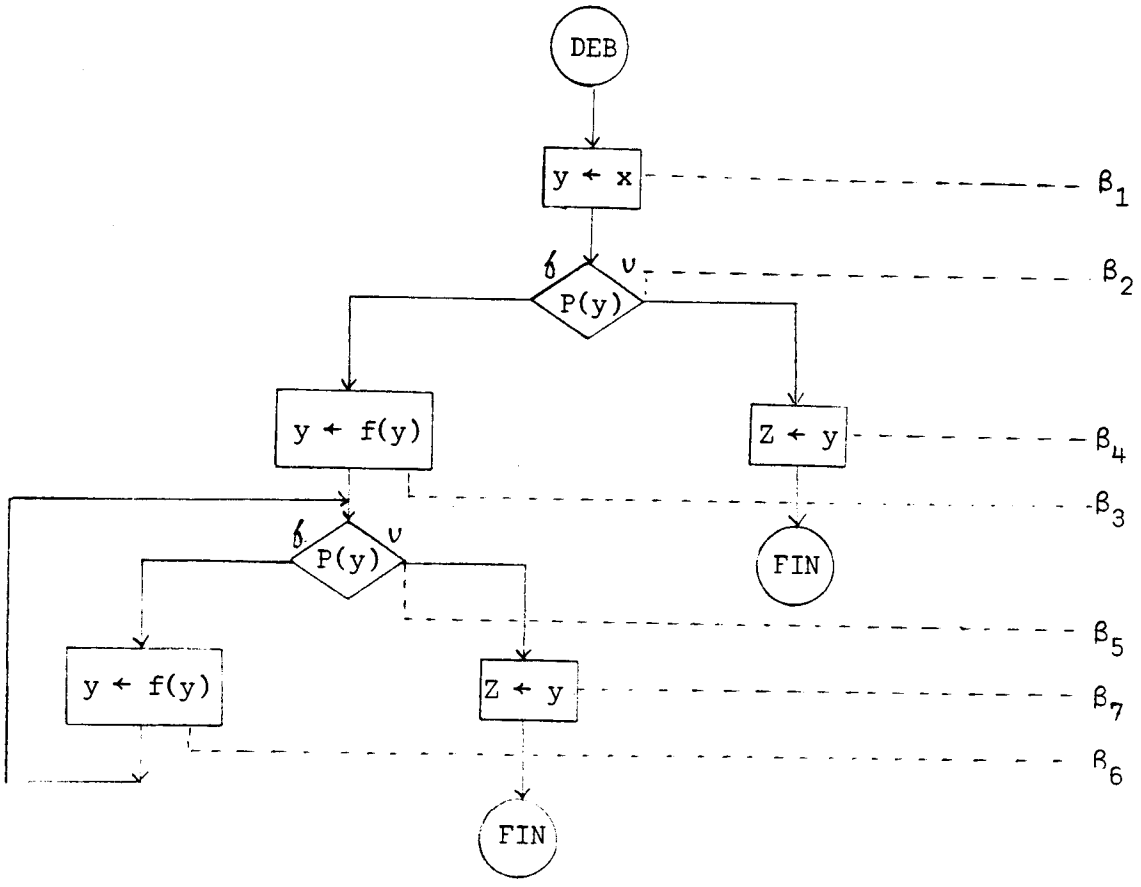


Schéma S'

Supposons :

$y \leftarrow x = f_1$

$P(y) = P_1$

$y \leftarrow f(y) = f_2$

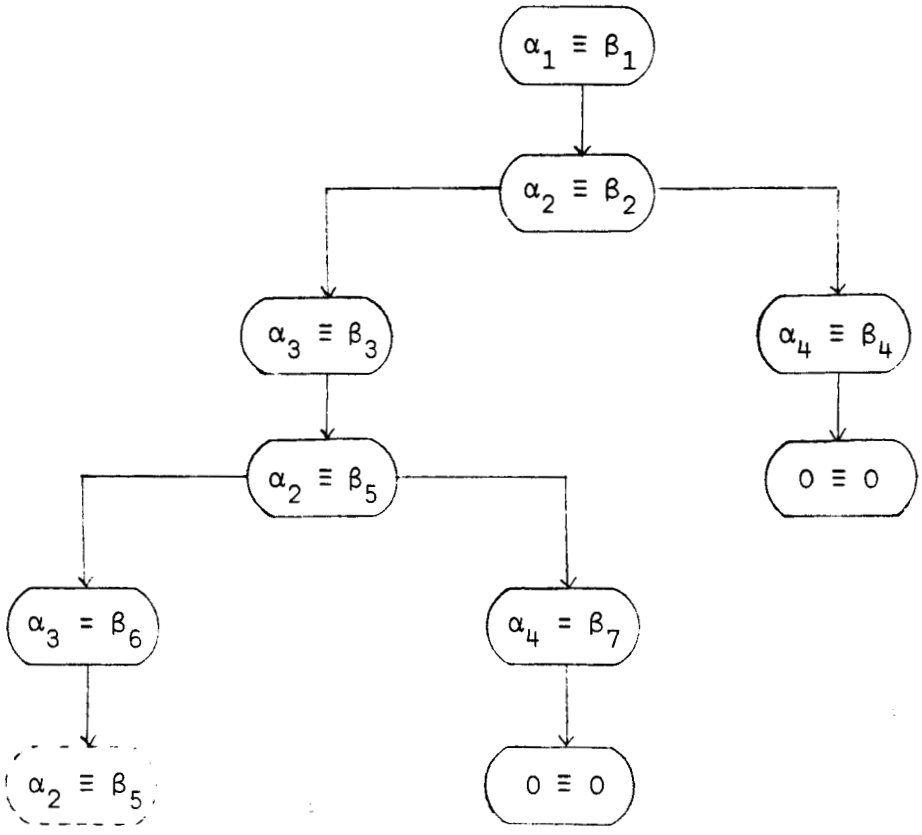
$Z \leftarrow y = f_3$

Alors :

$$\hat{S} = \begin{cases} \alpha_1 = f_1(\alpha_2) \\ \alpha_2 = P_1(\alpha_3, \alpha_4) \\ \alpha_3 = f_2(\alpha_2) \\ \alpha_4 = f_3(0) \end{cases}$$

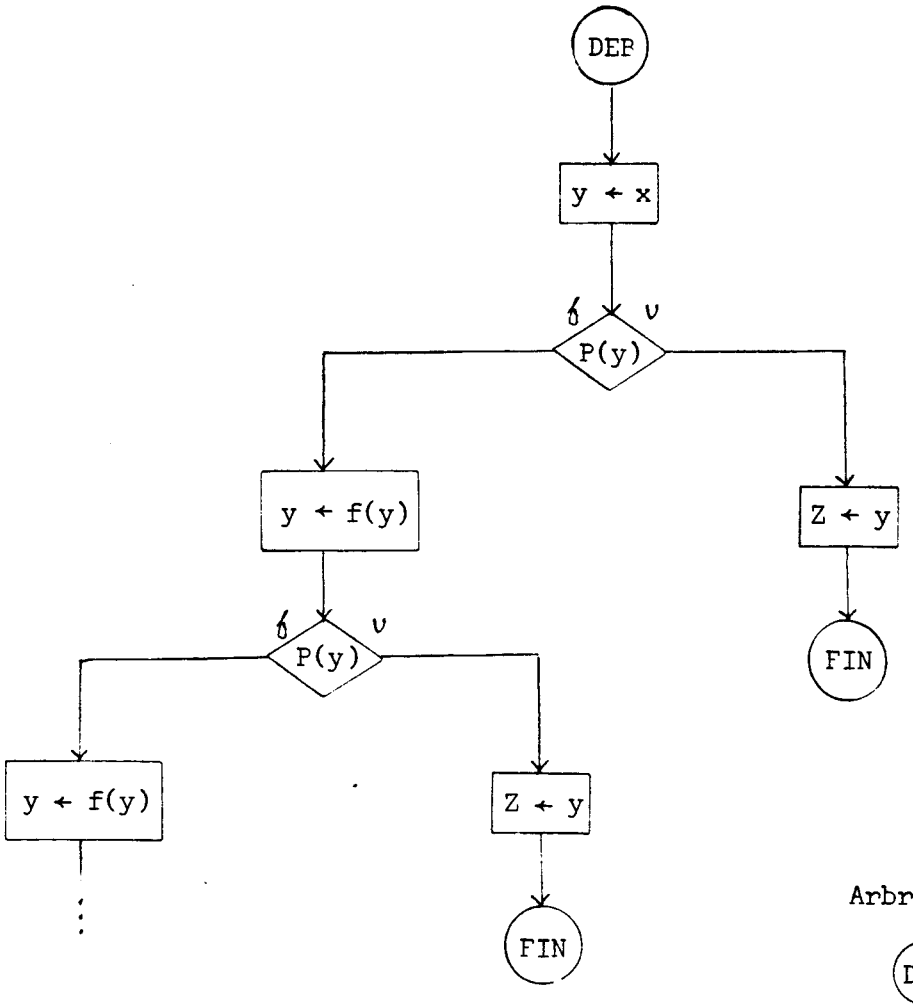
$$\hat{S}' = \begin{cases} \beta_1 = f_1(\beta_2) \\ \beta_2 = P_1(\beta_3, \beta_4) \\ \beta_3 = f_2(\beta_5) \\ \beta_4 = f_3(0) \\ \beta_5 = P_1(\beta_6, \beta_7) \\ \beta_6 = f_2(\beta_5) \\ \beta_7 = f_3(0) \end{cases}$$

L'arbre de comparaison de $\hat{S}$ et $\hat{S}'$ est :

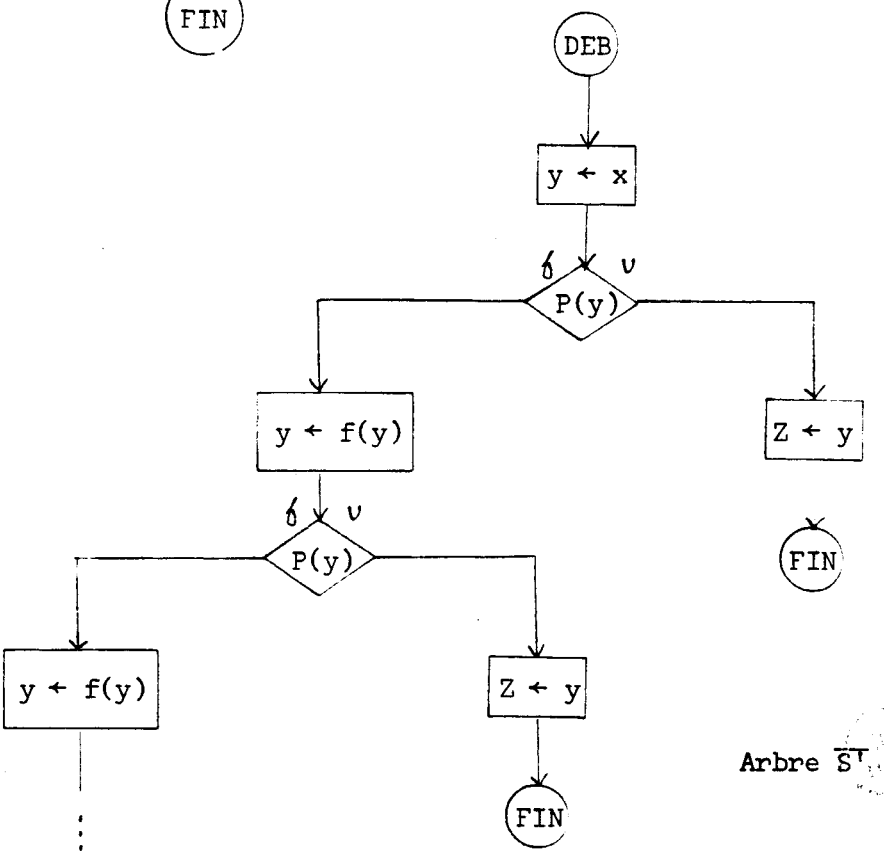


Alors S et S' sont COUSINEAU-équivalents.

On peut également vérifier l'identité de leurs arbres associés $\overline{S}$ et $\overline{S}'$.



Arbre $\overline{S}$



Arbre $\overline{S}'$

$$\tilde{S} = f_1(P_1(f_2(1), f_3(0))) = \tilde{S}' = f_1(P_1(f_2(P_1(f_2(1), f_3(0))), f_3(0)).$$

EQUIVALENCE DE IANOV

III.2. EQUIVALENCE DE IANOV

III.2.1. Définition

Soient S et S' deux schémas de programmes et $\overline{S}$ et $\overline{S}'$ leurs arbres de programmes.

On dit que S est IANOV-EQUIVALENT à S' et on note $S \underset{(I)}{\equiv} S'$ si et seulement si $sq(\overline{S}) \equiv sq(\overline{S}')$.

Définition : *Un graphe de transition* T sur l'alphabet W est un graphe direct et fini tel que chaque flèche est étiquetée par un mot $\omega \in W^*$ (ω peut être le mot vide).

Il existe au moins un sommet étiqueté par le signe '-' appelé *sommet initial* et un ensemble (possible vide) de "sommets" étiquetés par le signe '+' et appelés sommets finals.

Notons qu'un sommet peut être à la fois initial et final.

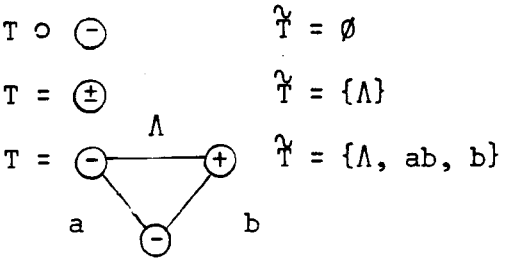
Définition : Un *ω-chemin* ($\omega \in W^*$) de sommet i à sommet j dans T est un chemin fini de i à j tel que la concaténation des étiquettes sur la longueur de ce chemin est ω.

Un mot $\omega \in W^*$ est dit acceptable par un graphe de transition T s'il existe un ω-chemin d'un initial "sommet" à un final "sommet".

Le mot vide Λ est acceptable par le graphe de transition T s'il existe dans T un sommet qui est à la fois le sommet initial et final, autrement dit s'il existe un Λ-chemin d'un sommet initial à un sommet final.

Nous noterons $\tilde{T}$ l'ensemble de mots acceptables par un graphe de transition T. Il est clair que chaque automate fini est un graphe de transition mais la réciproque est fausse.

EXEMPLE III.3. :



$\tilde{T} = \emptyset$

$\tilde{T} = \{\Lambda\}$

$\tilde{T} = \{\Lambda, ab, b\}$

Théorème III.2. (MOORE) : Il existe un algorithme pour déterminer si deux automates finis A et A' sont équivalents ou non (c'est-à-dire $\tilde{A} = \tilde{A}'$ ou non).

PREUVE : Supposons que A et A' sont définis sur W = {a, b} (pour faciliter nous avons choisi w = {a, b}).

Renommons d'abord les "sommets" de A et A' sachant qu'ils sont distincts.

Supposons x et x' les "sommets" initials de A et A' respectivement.

Pour décider si A et A' sont équivalents ou non, nous construirons un tableau de comparaisons avec trois colonnes dans lesquelles les éléments du tableau sont des paires de sommets (v, v') tel que v est un sommet de A et v' un sommet de A'. Nous choisissons (x, x') comme le premier élément du tableau situé à la tête de la première colonne.

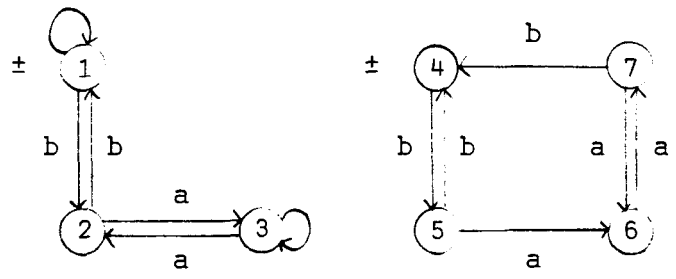
En général pour chaque paire (v, v') située dans la première colonne nous plaçons (v_a, v'_a) dans la deuxième colonne où a est la flèche entre v et v_a dans A et entre v' et v'_a dans A'. Ainsi que dans la troisième colonne nous plaçons la paire de "sommets" (v_b, v'_b) obtenue à partir de (v, v') par la flèche b.

Si (v_a, v'_a) et (v_b, v'_b) n'ont pas d'occurrence dans la première colonne alors nous les plaçons dans la première colonne comme premier élément de la ligne suivante et répétons la procédure.

Dans le courant de la procédure si nous trouvons une paire (u, u') dans le tableau tel que u soit le sommet final de A et u' le sommet final de A' ou réciproquement, nous arretons la procédure. A et A' ne sont pas équivalents. Sinon, si la procédure se termine et s'il n'existe aucune paire de sommets dans les colonnes 2 et 3 qui ne figurent pas dans la colonne 1, alors A et A' sont équivalents.

Notons que cette procédure doit toujours se terminer car, d'une part, le nombre de sommets de A et A' sont finis, d'autre part, toutes les paires de sommets de da colonne 1 sont distinctes.

EXEMPLE III.4. : Testons l'équivalence entre deux automates finis A et A' ci-dessous :



A A'

Le tableau de comparaison correspond à A et A' :

(v, v')	(v_a, v'_a)	(v_b, v'_b)
(1, 4)	(1, 4)	(2, 5)
(2, 5)	(3, 6)	(1, 4)
(3, 6)	(2, 7)	(3, 6)
(2, 7)	(3, 6)	(1, 4)

A est équivalent à A' car dans les colonnes 2 et 3, il n'existe aucun élément qui n'est pas figuré dans la colonne 1.

NOTATION III.1. : KELENE a prouvé que pour chaque graphe de transition T, il existe un automate fini A tel que $\tilde{T} = \tilde{A}$.

Par conséquent, d'après le Théorème III.2, on peut dire que *le problème d'équivalence pour les graphes de transition est décidable.*

Proposition III.1. : Il est décidable qu'un schéma (arbre) de IANOV soit libre ou pas.

DEMONSTRATION : Elle résulte du fait qu'un schéma (arbre) de IANOV n'est pas libre si et seulement si il a deux tests identiques sans aucune action entre les deux.

Théorème III.3. : Le problème d'équivalence pour les schémas de IANOV est décidable.

PREUVE : Soit S un schéma de IANOV libre avec l'instruction d'entrée "DEB" notée A_0, et les instructions d'assignations $A_1,\ldots, A_k$. (Nous pouvons associer à chaque schéma de IANOV un schéma de IANOV libre équivalent (Théorème III.4.)).

Du fait que chaque schéma de IANOV peut être transformé facilement en un schéma d'IANOV équivalent avec seulement une seule "FIN" alors nous pouvons supposer que S possède une seule "FIN" et que nous la noterons A_{k+1}. Supposons aussi que W_S possède n symboles de fonctions $f_1,\ldots, f_n$ et m symboles de prédicats $p_1,\ldots, p_m$.

Nous allons montrer que S possède la structure d'un graphe de transition G_S.

a - Les "sommets" sont :

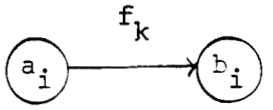
- b_0 représente l'instruction d'entrée
- a_i et b_i $1 \leq i \leq k$ représentent les instructions d'assignations
- a_{k+1} représente l'instruction "FIN"

Le sommet b_0 est le seul sommet initial (étiqueté par '-') et le vecteur a_{k+1} est le seul sommet final (étiqueté par '+'). (Notons que nous négligeons l'instruction "BOU" dans S).

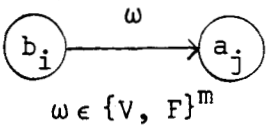
b - L'alphabet de G_S consiste de n symboles de fonctions $f_1,\ldots, f_n$ et 2^m mot de $\{F, V\}^m$.

Intuitivement chaque mot de $\{F, V\}^m$ (de longueur m) indique la possibilité vraie ou fausse de la valeur de m symboles de prédicats $p_1,\ldots, p_m$ pour une variable y donnée.

c - Les flèches dans G_S sont construites comme suit :



si A_i est l'instruction $y \leftarrow f_k(y)$



si dans S il existe un chemin de A_i à A_j et il n'a pas l'instruction assignation dans laquelle l'ensemble des valeurs du test sur ce chemin indiqué par ω n'est pas consistant.

Or, pour deux schémas de IANOV S et S' donnés.

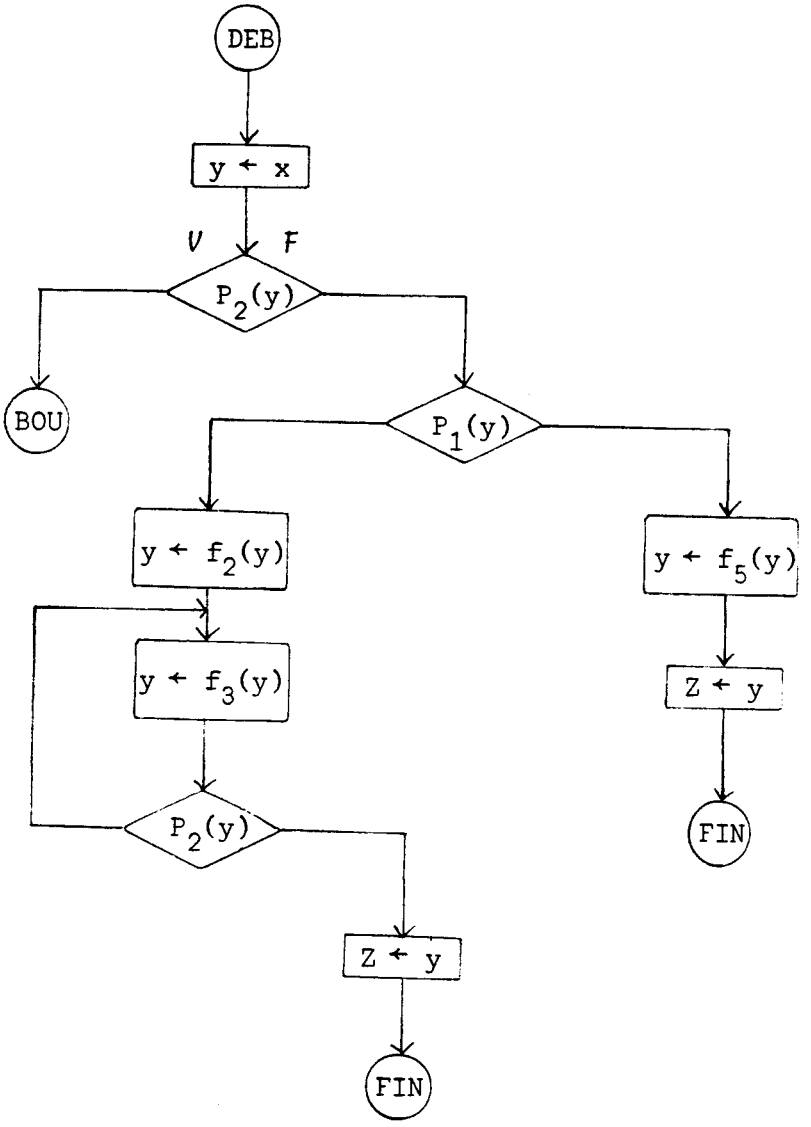
D'abord transformons-les en deux schémas de IANOV libres équivalents (Théorème III.4.) puis construisons les graphes de transitions associés G_S et $G_{S'}$.

Du fait que le problème d'équivalence est décidable pour les graphes de transitions (NOTATION III.1.), ce problème sera aussi décidable pour les schémas de IANOV.

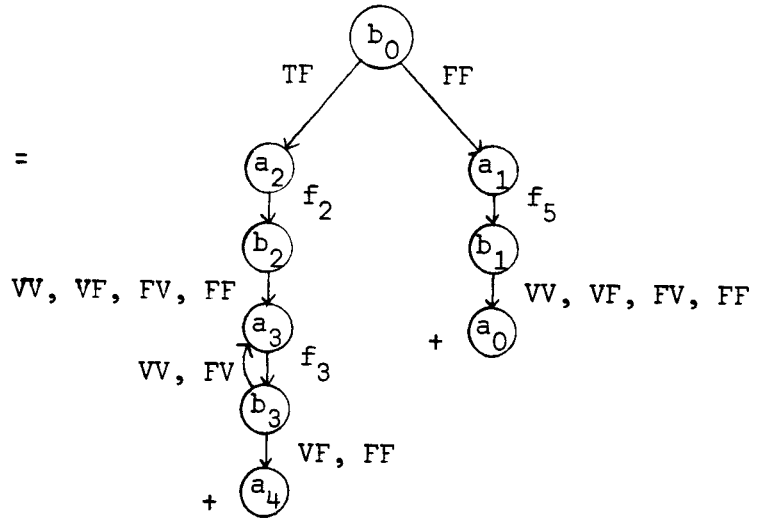
EXEMPLE III.5. :

Voir page suivante ...

S =



G_S =



III.2.2. Théorie consistante et complète pour l'équivalence de IANOV

La théorie que nous allons étudier pour l'équivalence de IANOV est :

$T_1 = \{A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, R1, R2, R3\}$ où :

A1 : $\vdash S = S$

A2 : $\vdash S = (S + 1)^*$

A3 : $\vdash S^* = S[0\backslash\backslash S^*,\ i\backslash\backslash i + 1]$

A4 : $\vdash S_1 \cdot S_2 = S_1[0\backslash\backslash S_2]$

A5 : $\vdash$ si p alors si p alors S_1 sinon S_2 si sinon S_3 fsi =
si p alors S_1 sinon S_3 fsi

A6 : $\vdash$ si p alors S_1 sinon si p alors S_2 sinon S_3 fsi =
si p alors S_1 sinon S_3 fsi

A7 : $\vdash$ si p_1 alors si p_2 alors S_1 sinon S_2 fsi sinon si p_2 alors S_3
sinon S_4 fsi fsi p_2 alors si p_1 alors S_1 sinon S_3 fsi sinon
si p_1 alors S_2 sinon S_1 fsi fsi

A8 : $\vdash f \cdot BOU = BOU$

A9 : $\vdash$ si p alors BOU sinon BOU fsi = BOU

A10 : $\vdash S^* = BOU$ si S est propre

R1 : $\vdash S_1 = S_2$
$\vdash S = S'$
s'occurrence de l'élément terminal S_2 dans S
$\Longrightarrow \vdash S' = S[S'_2\backslash\backslash S_1]$

R2 : - $S' = S$

s' élément terminal de S ayant dans S

les occurrences $S' = S'^1, \ldots, S'^p$

$\Longrightarrow S' = S[S'\backslash\backslash O,\ i\backslash\backslash i+1]^*$

R3 : - $S_1 = S_2$

S_2' occurrence direct de S_2 dans S_1

$\Longrightarrow - S_1 = S_1[S_2'\backslash BOU]$

III.2.3. Preuve de la consistance de T_I

Considérons le magma $M_I^\infty(W_\Omega) = \langle M_I(W_\Omega),\ F_{M_I}^*,\ C_{M_I}^* \rangle$, l'ensemble d'arbres de IANOV, (pour faciliter nous le noterons simplement M_I^∞) avec équivalence. Nous allons montrer que M_I^∞ est un modèle pour T_I i.e. tous les éléments de T_I sont valides dans M_I^∞.

- Les formules A1 à A4 sont valides dans T_C (voir page 70) alors elles sont également valides dans M_I^∞.

- A5 : Posons $\overline{S}$ = si p alors si p alors $\overline{S}_1$ sinon $\overline{S}_2$ fsi

  $\overline{S}'$ = si p alors S_1 sinon $\overline{S}_3$ fsi

  et soit II = (D, I) une interprétation et d une initialisation

  si $I(p)(d) = \omega$ alors $I(\overline{S}) = I(\overline{D}') = \omega^k$

  si $I(p)(d) = V$ alors $I(\overline{S}) = I(\overline{S}_1) = I(\overline{S}')$

  si $I(p)(d) = F$ alors $I(\overline{S}) = I(\overline{S}_2) = I(\overline{S}')$

- A6 : le même raisonnement que A5

- A7 : Posons $\overline{S}$ = si p_1 alors si p_2 sinon S_2 fsi sinon si p_2 alors S_3 sinon S_4 fsi fsi

$\overline{S}'$ = si p_2 alors si p_1 alors S_1 sinon S_3 fsi sinon
si p_2 alors S_2 sinon S_4 fsi fsi

Soit II = (D, I) une interprétation et d une initialisation

si $I(p_1)(d) = I(p_2)(d) = \omega$ alors $I(\overline{S}) = \omega^k$

si $I(p_1)(d) = I(p_2)(d) = V$ alors $I(\overline{S}) = I(\overline{S}_1) = I(\overline{S}')$

si $I(p_1)(d) = I(p_2)(d) = F$ alors $I(\overline{S}) = I(\overline{S}_4) = I(\overline{S}')$

si $I(p_1)(d) = V$ et $I(p_2(d) = F$ alors $I(\overline{S}) = I(\overline{S}_2) = I(\overline{S}')$

si $I(p_1)(d) = F$ et $I(p_2)(d) = V$ alors $I(\overline{S}) = I(\overline{S}_6) = I(\overline{S}')$

- A8 : Posons $\overline{S} = f \cdot BOU$ et $\overline{S} = BOU$ et soit II = (D, I) une interprétation et d une initialisation

  $I(\overline{S})(d) = I(\overline{S}')(d) = \omega^k$

- A9 : Même chose que A8

- A10 : Si $\overline{S}$ est un arbre de IANOV propre, $\overline{S}^*$ ne contient aucune occurrence de zéro par conséquent $\forall II = (D, I)$ et $\forall d$, $I(\overline{S}^*)(d) = \omega^k$

- R1 et R2 : Les règles R1 et R2 préservent la validité pour l'équivalence de COUSINEAU par conséquent elles préservent également la validité pour l'équivalence de IANOV

- R3 : Si $\overline{S}$ est un arbre de IANOV et $m \in Dom(S)$ vérifie :

  1 - $\overline{S}/m = \overline{S}$ et

  2 - $\forall m' < m$ $\overline{S}(m') \in P$

  alors $\overline{S} \equiv \overline{S}[m \rightarrow BOU]$

III.2.4. Preuve de la complétude de T_I

Nous allons prouver la complétude de $T_I = \{A1, \ldots, A10, R1, R2, R3, R4, R5\}$ pour l'équivalence de IANOV. R4 (symétrie) et R5 (transitivité) appartiennent à T_C alors on peut les considérer comme deux éléments de T_I.

D'abord nous établissons quelques résultats :

Lemme III.9. : Si A est une partie dirigée d'un ensemble d'arbres.

sup(A) est libre <==> ∀A ∈ A est libre.

PREUVE : (=>) Supposons qu'il existe un A ∈ A tel que A n'est pas libre ,cela implique qu'il existe une branche br de A telle que T(br) est contradictoire, mais cette branche est également une branche de sup(A), donc sup(A), n'est pas libre.

(<=) Suppososn que sup(A) est non libre alors il existe une branche br' de sup(A) telle que T(br') est contradictoire.
Alors il existe un A' ∈ A telle que br' est une branche de A' et donc A' n'est pas libre.

Théorème III.4. : A tout W-arbre de IANOV A correspond un arbre de IANOV libre équivalent.

PREUVE : - A est fini. A tout W-arbre de IANOV A et à tout ensemble de tests consistants T, on associe l'arbre de IANOV libre défini par :

$\ell i(A, T)$ = si A(ε) = FIN alors FIN
sinon si A(ε) = BOU alors BOU
sinon si A(ε) = f ∈ F alors A(ε) ($\ell i(p\downarrow_1, \emptyset)$)
sinon si A(ε) = p ∈ P alors si ni p ni $\bar{p}$ n'appartiennent pas à T alors A(ε) ($\ell i(A\downarrow_1, T \cup \{p\})$, $\ell i(A\downarrow_2, T \cup \{\bar{p}\})$)
sinon si p ∈ T alors $\ell i(A\downarrow_1, T)$
sinon si $\bar{p}$ ∈ T alors $\ell i(A\downarrow_2, T)$

- A est infini.

Choisissons pour A une suite d'approximats $A_1,\ldots, A_i,\ldots$

D'après le Lemme III.9, $\sup(\{\ell i(A_i)\})$ est un arbre de programmes libre équivalent à A.

Il nous faut démontrer que $A' = \ell i(A)$ est reconnaissable.

A' est un arbre libre alors $\forall m \in Dom(A')$, $A'\downarrow_m \in M^{\infty}(P, \{FIN, BOU\}) \Rightarrow$ $A'\downarrow_m$ est fini car P est un ensemble fini.

Par conséquent, A' est reconnaissable s'il possède un nombre fini de sous-arbres $A'\downarrow_m$ tel que $A'^{(m)}$ est une affectation.

L'algorithme qui fait passer de A à i(A) associe à tout $m \in Dom(\ell i(A))$ tel que $\ell i(A)(m) \in F$ un $t(m) \in Dom(A)$ tel que $A(t(m)) \in F$ et $A\downarrow_m = \ell i(A\downarrow_{t(m)}, \emptyset) = \ell i(A\downarrow_{t(m)})$.

Soit k le nombre de sous-arbres $A\downarrow_m$ de A tel que $A(m) \in F$ et supposons qu'il existe k + 1 sous-arbres distincts $\ell i(A)\downarrow_{m_1},\ldots, \ell i(A)\downarrow_{m_{k+1}}$ de $\ell i(A)$ tels que $\ell i(A)(m_1),\ldots, \ell i(A)(m_{k+1}) \in F$.

Soient $t(m_1),\ldots, t(m_{k+1})$ les éléments de Dom(A) associés à $m_1,\ldots, m_{k+1}$. Comme A ne contient que k sous-arbre $A\downarrow_m$ tel que $A(m) \in F$ alors il existe u, $j \in [k+1]$ et $i \neq j$ tel que $A\downarrow_{t(mi)} = A(t\downarrow_{mj})$.

Mais alors $\ell i(A\downarrow_{mi}) = \ell i(A\downarrow_{t(mi)}) = \ell i(A\downarrow_{t(mj)}) = \ell i(A\downarrow_{mj})$.

Par conséquent, $\ell i(A)$ ne possède que k sous-arbres distincts C'est-à -dire qu'il est reconnaissable.

NOTATION III.2. : Il convient de noter que si A est un arbre de programmes reconnaissable, $\ell i(A)$ n'est pas reconnaissable, en général, autrement dit il est indécidable qu'un arbre de programmes donné soit libre ou non.

REMARQUE : Il est facile à vérifier que si nous pouvons obtenir un arbre de IANOV canonique Ca(A) à partir d'un arbre de IANOV libre et réduit $A \in M(P, \{BOU, FIN\})$, nous pourrons obtenir un arbre de IANOV canonique Ca(A) pour tout arbre de IANOV libre et réduit $A \in M(F \cup P, \{BOU, FIN\})$ car d'une part la définition de bloc sélectif concerne uniquement les tests et d'autre part, si $A = f(A_i)$ avec A_i libre et fini alors $Ca(A) = f(Ca(A_i))$.

Théorème III.5. : Pour tout arbre de IANOV A il existe un arbre canonique équivalent (noté Ca(A)).

PREUVE : La définition d'un arbre canonique comporte trois conditions que nous pouvons satisfaire l'une après l'autre :

1 - On a déjà montré (Théorème III.4.) qu'en utilisant les formules A5 et A6 nous pouvons obtenir un arbre de IANOV libre $\ell i(A)$ pour tout arbre de IANOV A.

2 - En utilisant les formules A8, A9 et R3, on peut obtenir un arbre de IANOV réduit et libre pour tout arbre de IANOV libre.

3 - Cette condition est locale aux blocs sélectifs et que dans un arbre de IANOV les blocs sélectifs sont finis et en utilisant la formule A8, nous pouvons mettre tous ces blocs sélectifs en formes canoniques.

ALGORITHME : Pour obtenir un arbre de IANOV canonique à partir d'un arbre de IANOV libre et émondé.

- A est fini

Si $A \in \{x_1, \ldots, x_n\} \cup \{FIN, BOU\}$ alors $Ca(A) = A$

sinon soit p le plus petit prédicat couvrant A

si $O^p(A) = p(A_1, A_2)$ alors $Ca(A) = p(Ca(A_1), Ca(A_2))$

si A_1 et $A_2 \notin M(P, \{BOU\}$

$= p(BOU, Ca(A_2))$

si $A_1 \in M(P, \{BOU\})$

$= p(Ca(A_1), BOU)$

si $A_2 \in M(P, \{BOU\})$

$= BOU$ si A_1 et $A_2 \in M(P, \{BOU\})$

où pour chaque plus petit gradient p couvrant A, $O^p(A) = p(A', A'')$ défini comme :

si $A = BOU$ $\quad O^p(A) = (BOU, BOU)$

si $A = p(A_1, A_2)$ $\quad O^p(A) = p(A_1, A_2)$

si $A = p'(A_1, A_2)$ et $p' \neq p$, p couvre A_1 et A_2.

Soit $O^p(A_1) = p(A_1', A_1'')$ et $O^p(A_2) = p(A_2', A_2'')$

$O^p(A) = p(p'(A_1', A_2'), p'(A_1'', A_2''))$.

Donc, nous sommes à présent capables de construire un arbre IANOV canonique fini pour tout arbre de IANOV libre et réduit fini, en allant de la racine vers les feuilles, et en mettant tous les blocs sélectifs en formes canoniques.

- A est infini

Dans ce cas nous choisissons pour A un ensemble A_i d'approximats tels que : $\forall i, \forall m \in Dom(A_i)$ $A_i(m) = BOU \Rightarrow$ ou bien $A(m) = BOU$ ou bien il existe un $m' \leq_p m$ tel que $m \equiv m'$.

Pour chaque A_i nous pouvons obtenir $Ca(A_i)$, en utilisant l'algorithme ci-dessus. $sup(\{Ca(A_i)\})$ est un arbre de IANOV canonique équivalent à A en vertu du Lemme III.9. ($sup(A)$ est canonique $\Longleftrightarrow \forall A \in A$ est canonique).

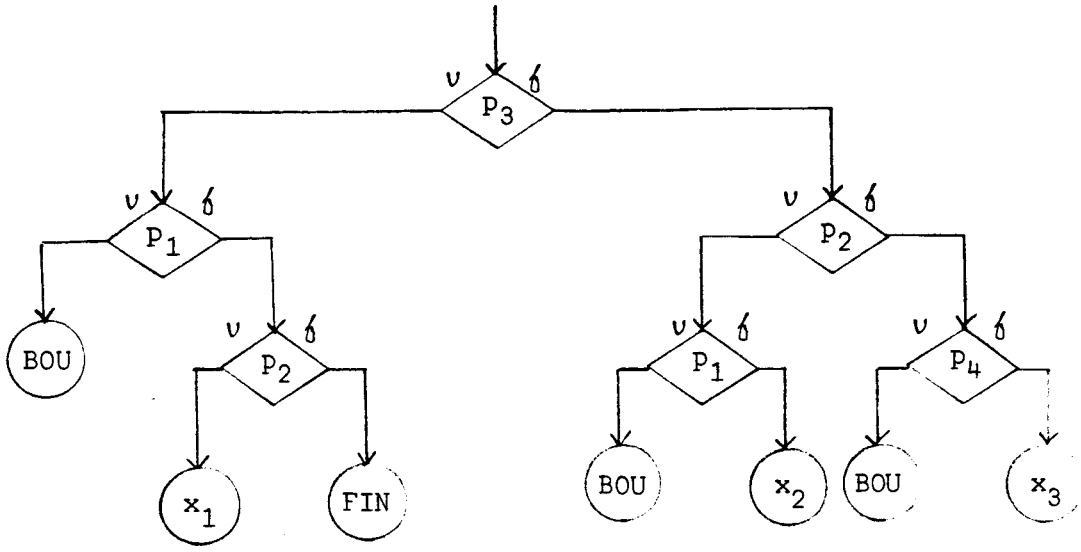
Il nous faut démontrer que Ca(A) est reconnaissable.

- Premièrement : On sait déjà par le Théorème III.4, que $\ell i(A)$ est reconnaissable.

- Deuxièmement : L'opération de réduction qui remplace par BOU les sous-arbres appartenant à $M^{\infty}(P \cup F, \{BOU\})$ conserve le caractère reconnaissable.

- Troisièmement : Le passage à la forme canonique est strictement local aux blocs sélectifs et deux blocs maximums identiques A'_1 et A'_2 sont transformés en deux blocs canoniques identiques $Ca(A'_1)$ et $Ca(A'_2)$.
Par conséquent, deux sous-arbres $A/m_1 = A/m_2$ tels que $A(m_1, A(m_2) \in F$ sont transformés en deux sous-arbre canoniques identiques $Ca(A\downarrow_{m_1})$ et $Ca(A\downarrow_{m_2})$.

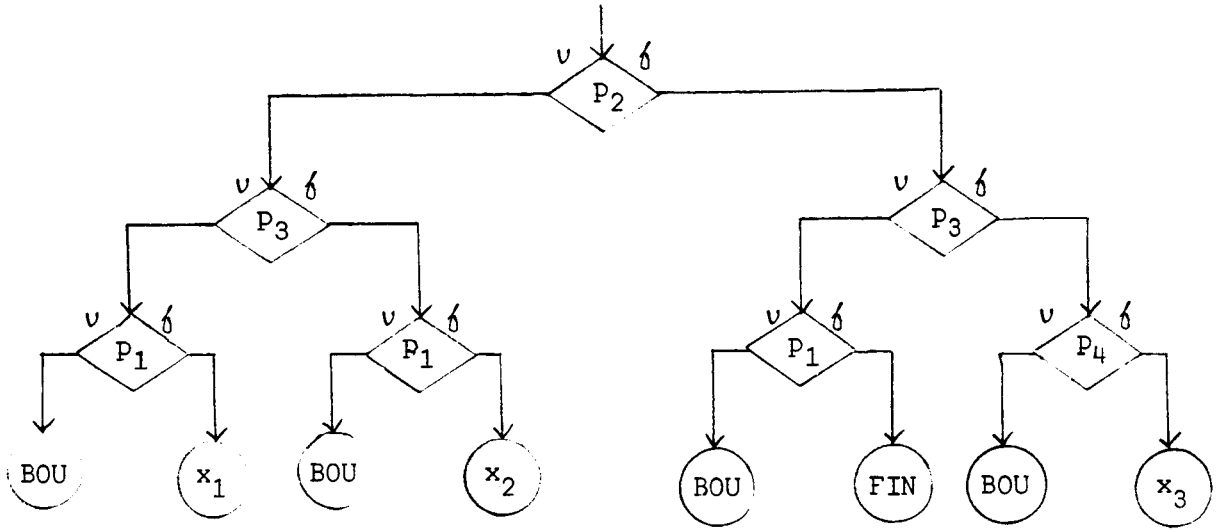
EXEMPLE III.6. :



Arbre de IANOV, libre et réduit A.

A n'est pas canonique car $A(\varepsilon) = p_3$ et $A(2) = p_2$ et $p_2, p_3 \in P \subset (A)$.

Le plus petit prédicat couvrant A est p_2 et nous avons :

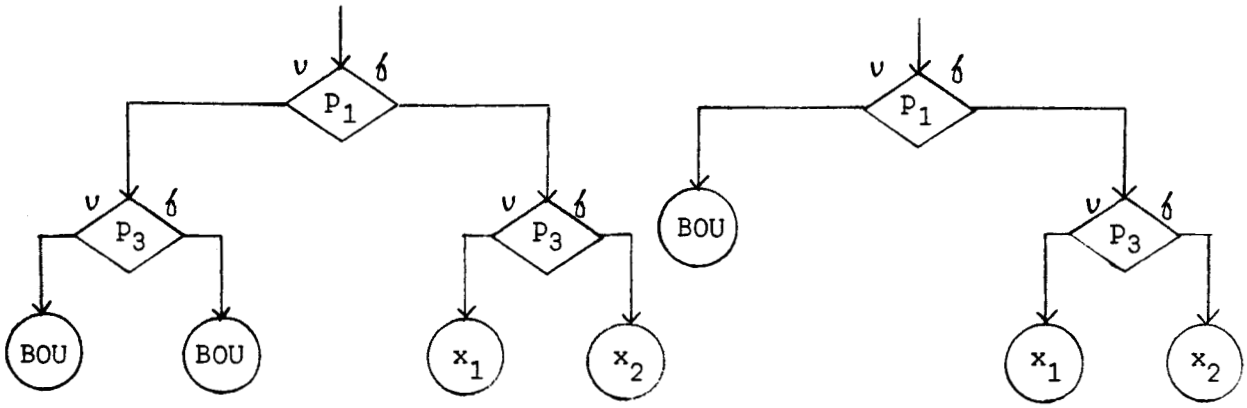


$O^{p_2}(A)$

Posons $O^{p_2}(A) = p_2(A_1, A_2)$, nous avons $Ca(A) = p_2(Ca(A_1), Ca(A_2))$.

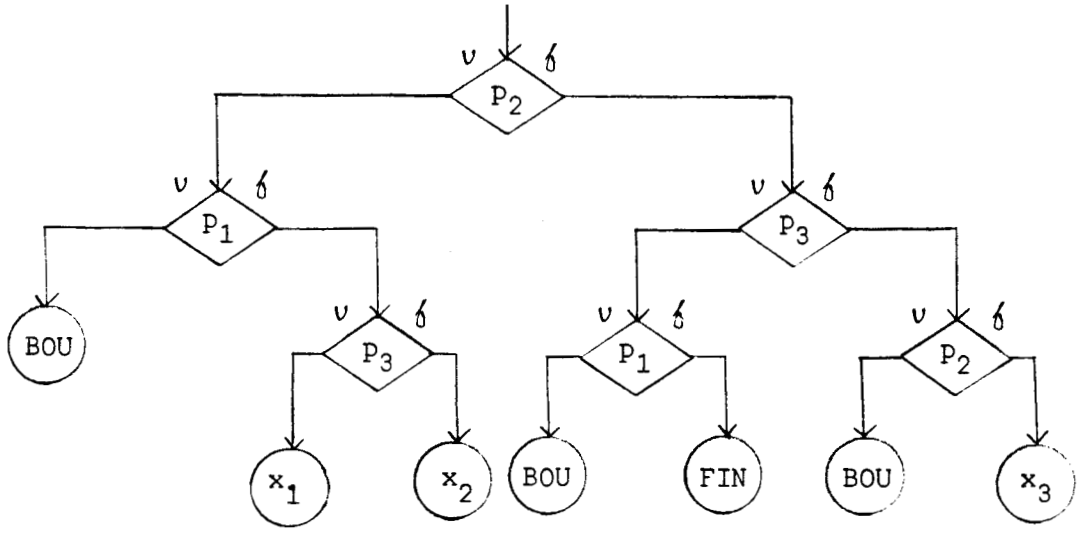
A_1 n'est pas canonique car $A_1(\varepsilon) = p_3$, $A_1(1) = p_1$ et $p_1, p_3 \in p \subset (A_1)$.

Le plus petit prédicat couvrant A_1 est p_1 et nous avons :



$o^{P_1}(A_1)$ $\qquad$ $Ca(A_1)$

A_2 est canonique et nous avons donc finalement :



Ca(A)

Théorème III.6. : Deux arbres de IANOV canoniques équivalents sont égaux.

PREUVE : Soient A et A' deux arbres de IANOV canoniques équivalents. Supposons qu'ils ne sont pas égaux et soit m un élément minimal dans l'ordre préfixiel tel que $A(m) \neq A'(m)$, quatre cas sont à considérer :

1 - A(m) = BOU et A'(m) ≠ BOU

Il existe alors un m' tel que A'(m m') = FIN et comme A' est libre, alors il existe une interprétation libre II pour laquelle les calculs de A et A' pour II atteignent tous les deux le stade (m, t) pour quelque t ∈ TF(F, V) et le calcul de A' pour II atteint ensuite l'étape (m m', t') pour quelque t' ∈ TF(F, V).

On a I(A) = t' et I(A) = ω et les deux arbres A et A' ne sont pas équivalents.

2 - A(m) ∈ F et A'(m) = p ∈ P

Il existe un m' tel que A(m m') = FIN, alors il existe une interprétation II = (D_{II}, I) telle que les calculs de A et A' pour II atteignent tous les deux le stade (m, t) pour quelque t ∈ TF(F, V) ; p(t) = ω et il existe un t' ∈ TF(F, V) tel que le calcul de A pour II atteint ensuite le stade (m m', t').

On a I(A) = t' et I(A') = ω, les deux arbres ne sont pas équivalents.

3 - A(m) = FIN et A(m') ≠ FIN

Il existe une interprétation libre II pour laquelle les deux calculs atteignent le stade (m, t) et I(A) = t et I(A') = ω.

4 - A(m) = p ∈ P et A'(m) = p' ∈ P avec p < p'

p ne couvre pas le bloc m dans A' car sinon ou bien A' n'est pas canonique ou bien A n'est pas libre (possède deux tests p sans affectation intermédiaire).

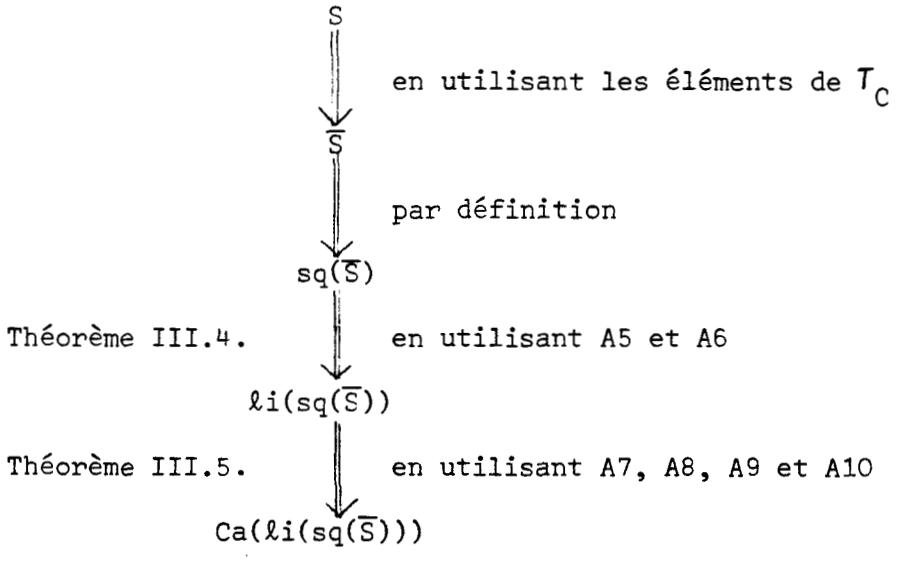
Alors il existe une interprétation libre telle que les calculs de A et A' pour II atteignent un stade (m, t) tel que I(p)(t) = ω et que le calcul de A' pour II atteint le stade (m m', t') avec

$A'(m\,m') = FIN$.

On a alors $I(A) = \omega$ et $I(A') = t'$ i.e. A et A' ne sont pas équivalents.

La preuve de la complétude de T_I se fait en passant par les étapes suivantes :

Soit S un schéma de programmes et $\overline{S}$ l'arbre de programmes associé :



Si nous avons un autre schéma de programmes S', IANOV-équivalent à S en passant par les étapes ci-dessus, on peut obtenir $Ca(\ell i(sq(\overline{S}')))$ qui est identique à $Ca(1i(sq(\overline{S})))$. (En vertu du Théorème III.6.).

EQUIVALENCE DE PATERSON

III.3. EQUIVALENCE DE PATERSON

III.3.1. Introduction

Nous allons étudier l'équivalence de PATERSON en deux parties, premièrement nous montrerons que le problème d'équivalence de PATERSON (que nous appelèrons simplement problème d'équivalence) pour les schémas (arbres) de programmes n'est pas décidable en présentant une classe de schémas de programmes dont le problème d'équivalence n'est pas décidable.

Ensuite, nous étudierons certaines classes de schémas et d'arbres de programmes dans lesquelles le problème d'équivalence est décidable pour ces classes.

III.3.2. Une classe dont le problème d'équivalence n'est pas décidable

Théorème III.7. : [32]. Soit $\mathcal{S}$ l'ensemble de toutes les classes de schémas de programmes et $\mathcal{S}_0$ l'ensemble de tous les schémas de programmes dont le problème d'équivalence est décidable.

$$\mathcal{S}_D \subsetneq \mathcal{S}$$

PREUVE : ($\Rightarrow$). Il est évident que $\forall C \in \mathcal{S}_D \Rightarrow C \in \mathcal{S}$

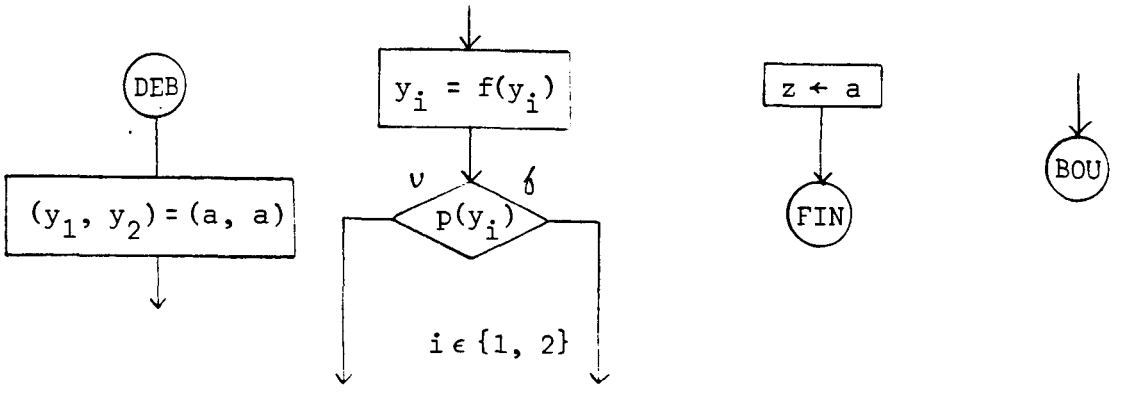
($\Leftarrow$). Il suffit de trouver une classe de schémas de programmes telle que le problème d'équivalence ne soit pas décidable pour cette classe.

Considérons la classe C_1 telle que :

$$\forall S \in C_1.$$

- W_S consiste d'une seule constante indécidable a, d'un seul symbole de fonction f et d'un seul symbole de prédicat p, de deux variables de programmes y_1, y_2, d'une seule variable de sortie z et d'aucune variable d'entrée.

- Les instructions de S sont de la forme suivante :



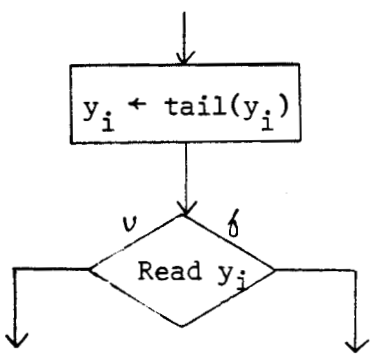
Montrons que le problème d'équivalence pour C_1 est indécidable.

PATERSON [31] a montré que le "non acceptence problème" pour les automates finis à deux enregistrements sur $W = \{\upsilon, \delta\}$ n'est pas décidable.

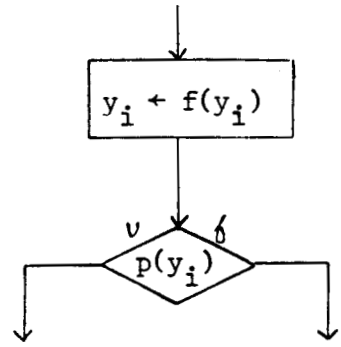
Après ce résultat nous complétons la preuve du théorème par quelques lemmes.

Lemme III.10. : Elle se fait en démontrant que le "non acceptence problème" pour les automates finis a deux enregistrements est traduisible en problème de divergence pour les schémas de C_1 (pour toute interprétation).

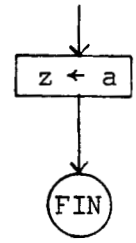
En effet pour chaque automate fini à deux enregistrements A sur $\{\upsilon, \delta\}$ nous pouvons construire un schéma S_A de C_1 de la façon suivante :



est remplacé par



est remplacé par

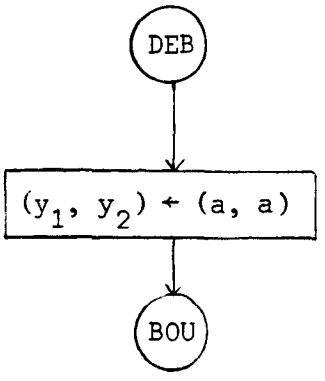


est remplacé par

ACCEPT (A) = ∅ si et seulement si (S_A, II) diverge pour toute II car pour toute interprétation libre (HERBRAND) II^* de S, chaque élément de $U(S_A)$ peut être présenté comme un mot de $\{\upsilon, \delta\}^*$ dont la ième lettre de ce mot indique la vraie ou fausse valeur de $p(f^{(i)}(a))$.

Lemme III.11. : Le problème d'équivalence pour la classe C_1 n'est pas semi-décidable.

PREUVE : En effet, chaque schéma de programmes diverge s'il est PATERSON-équivalent au schéma :



dans C_1

On voit que si le problème d'équivalence est décidable pour ce schéma de programmes alors le problème de divergence doit être également décidable et d'après le lemme III.12, le problème d'équivalence pour C_1 n'est pas semi-décidable.

Alors le problème d'équivalence pour la classe C_1 n'est pas décidable

III.3.3. Les classes dont le problème d'équivalence est décidable

Nous allons étudier trois classes de schémas de programmes (trois éléments de $\mathcal{S}_S$), ainsi qu'une théorie consistante et complète pour ces classes.

Avant d'aborder cette étude nous définissons un ordre total sur les éléments de $F \cup P$.

Etant donné un ensemble $v = \{x_1, \ldots, x_i, \ldots\}$ dénombrable et soit $<$ un ordre total des éléments de v.

Nous montrerons $\{x_{i_1}, \ldots, x_{i_{t=a(p)}}\}$ l'ensemble de tous les éléments de P qui possèdent exactement les variables $x_{i_1}, \ldots, x_{i_t}$ et

$\{x_i, \{x_{i_1}, \ldots, x_{i_{t'=a(f)}}\}\}$ l'ensemble de tous les éléments de F qui possèdent exactement les variables $x_i, x_{i_1}, \ldots, x_{i_{t'}}$.

L'ORDRE $\leq$

Soient $x_{i_1}, \ldots, x_{i_k} \in v_k$ et supposons $x_{i_1} < \ldots < x_{i_k}$.

Nous définissons l'ordre $\leq$ sur $F \cup P$ comme

$$[x_{i_1}] \leq [x_{i_2}] \leq \ldots \leq [x_{i_k}] \leq [x_{i_1}, x_{i_2}] \leq \ldots \leq [x_{i_{k-1}}, x_{i_k}]$$

$$\leq [x_{i_1}, x_{i_2}, x_{i_3}] \leq \ldots \leq [x_{i_1}, \ldots, x_{i_{k-1}}] \leq [x_{i_1}, \ldots, x_{i_k}]$$

Lemme III.12. : $\forall u = [x_i, x_{i_1}, \ldots, x_{i_t}] \in F$ et $\forall u' = [x_j, x_{j_i}, \ldots, x_{j_{t'}}] \in F$

u commute avec u' $\Longleftrightarrow$ $i \neq j$ et $x_i \notin \{x_j, \ldots, x_{j_{t'}}\}$

et $x_j \notin \{x_i, \ldots, x_{i_t}\}$

Lemme III.13. : $\forall u = [x_i, x_{i_1}, \ldots, x_{i_t}] \in F$ et $\forall u' = [x_j, \ldots, x_{j_{t'}}] \in P$

u commute avec u' $\Longleftrightarrow$ $i \neq j$ et $x_i \notin \{x_j, \ldots, x_{j_{t'}}\}$

III.3.4. Théorie consistante et complète pour les éléments de $\mathcal{F}_D$

La théorie que nous allons étudier pour l'équivalence de PATERSON est :

$T_p = T_I$ {A11, A12} où :

A11 : Soient u et u' deux éléments de F, u et u' sont commutables alors :

$i \cdot v = v \cdot u$ (Lemme III.12).

A12 : Soient u ϵ F et v ϵ P, u et v sont commutables alors

si v alors u sinon u fsi = u • v (Lemme III.13).

III.3.5. Les arbres des programmes finis $M(W_\Omega)$

Considérons le magma $M(W_\Omega)$ des arbres de programmes finis (voir page 8). Nous allons montrer que $M(W_\Omega)$ est un élément de $\mathcal{J}_D$.

PREUVE DE LA CONSISTANCE DE T_p POUR $M(W_\Omega)$

Il faut montrer que ce magma est un modèle pour T_p, autrement dit il faut montrer que tous les éléments de T_p sont valides dans $M(W_\Omega)$.

Les formules A1 à A10 sont valides dans $M_I(W_\Omega)$ (elles prouvent l'équivalence de IANOV (voir page 91) alors elles sont également valides dans $M(W_\Omega)$.

A11 : Lemme III.12.

A12 : Lemme III.13.

Les règles R1 à R5 préservent l'équivalence de IANOV par conséquent elles préservent l'équivalence de PATERSON.

PREUVE DE LA COMPLETUDE DE T_p POUR $M(W_\Omega)$

Lemme III.14. : A tout arbre de programmes fini A correspond un arbre libre équivalent.

PREUVE : A tout W-arbre de programmes fini A et à toute donnée $\bar{d} \in TF(F, V)^k$ et à tout ensemble de test consistant T, nous associons l'arbre libre $\ell i(A)$ défini récursivement par :

$\ell i(A, \bar{d}, T)$ = si $A(\varepsilon) = FIN$ alors FIN

sinon si $A(\varepsilon) = BOU$ alors BOU

sinon si $A(\varepsilon) = x_i \leftarrow f(x_{i_1}, \ldots, x_{i_{a(f)}})$

alors $A(\varepsilon)(\ell i(A\downarrow_1, (d_1, \ldots, d_{i-1}, f(x_1, \ldots, x_{i_{a(p)}}), d_{i+1}, \ldots, d_k), T)$

sinon si $A(\varepsilon) = p(x_{i_1}, \ldots, x_{i_{a(p)}})$

alors si ni $p(d_{i_1}, \ldots, d_{i_{a(p)}})$ ni $(\bar{p}(d_{i_1}, \ldots, d_{i_{a(p)}})$ n'appartient pas à T

alors $A(\varepsilon)(\ell i(A\downarrow_1, \bar{d}, T \cup \{p(d_{i_1}, \ldots, d_{i_{a(p)}}\}).$ $\ell i(A\downarrow_2, \bar{d}, T \cup \{\bar{p}(d_{i_1}, \ldots, d_{i_{a(p)}})\})$

sinon si $p(x_{i_1}, \ldots, x_{i_{a(p)}}) \in T$ alors $\ell i(A\downarrow_1, \bar{d}, T)$

sinon si $\bar{p}(x_{i_1}, \ldots, x_{i_{a(p)}}) \in T$ alors $\ell i(A\downarrow_2, \bar{d}, T)$

Il est évident que $\ell i(A)$ est fini et naturellement reconnaissable.

Corollaire : A tout arbre de programmes fini correspond un arbre de programmes fini émondé équivalent.

En effet par les formules A8, A9 et la règle R3, on peut obtenir un arbre fini et émondé à partir d'un arbre fini, en préservant les propriétés de finitude et de reconnaissabilité.

Définition : Un arbre de programme A est dit P-CANONIQUE si :

1 - A est canonique

2 - $\forall m \in Dom(A)$ et $\forall x \in \{1, 2\}$

$A(m) = u$ et $A(mr) = u'$

ou bien $u < u'$

ou bien en appliquant A11 et A12, u ne commute pas avec u'.

Théorème III.8. : Deux arbres de programmes finis P-canoniques équivalents sont égaux.

PREUVE : Soient A et A' deux arbres finis P-canoniques équivalents. Supposons qu'ils ne sont pas égaux et soit m un élément minimal dans l'ordre préfixiel, tel que $A(m) \neq A'(m)$. Cinq cas sont à considérer :

1 - $A(m) = FIN$ et $A'(m) \neq FIN$

Il existe une interprétation libre II^* pour laquelle les deux calculs atteignent le stade (m, t) tel que $I^*(A) = t$ et $I^*(A') = \omega$ alors A et A' ne sont pas équivalents.

2 - $A(m) = BOU$ et $A'(m) \neq BOU$

Il existe alors un m' tel que $A'(mm') = FIN$ et comme A est libre alors il existe une interprétation libre II^* pour laquelle les calculs de A et A' pour II^* atteignent tous deux le stade (m, t) pour quelque $t \in TF(F, V)$ et le calcul de A' pour II^* atteint ensuite l'étape (mm', t') pour quelque $t' \in TF(F, V)$.

On a $I^*(A) = t'$ et $I^*(A') = \omega$ et les deux arbres A et A' ne sont pas équivalents.

3 - $A(m) \in F$ et $A'(m) \in P$

Alors il existe un m' tel que $A(m\,m') = FIN$, par conséquent il existe une interprétation libre II^* telle que les calculs de A et A' pour II^* atteignent tous les deux le stade (m, t) pour quelque $t \in TF(F, V)$. Il existe un $t' \in TF(F, V)$ tel que le calcul de A pour II^* atteint ensuite le stade $(m\,m', t')$ qui n'est pas le cas pour A' i.e. on a $I^*(A) = t'$ et $I^*(A') = \omega$ car $p(t) = \omega$.

4 - $A(m) = p \in P$ et $A'(m) = p' \in P$ avec $p \sqsubset p'$

p ne couvre pas le bloc m dans A' car sinon ou bien A' n'est pas canonique ou bien A n'est pas libre (il possède deux tests identiques p sans affectation intermédiaire).

Alors il existe une interprétation libre II^* telle que les calculs de A et A' pour II^* atteignent un stade (m, t) tel que $I^*(p)(t) = \omega$ et que le calcul de A' pour II^* atteint un stade $(m\,m', t')$ avec $A'(m\,m') = FIN$. On a alors $I^*(A) = \omega$ et $I^*(A') = t'$.

5 - $A(m) = u \in F$ et $A'(m) = u' \in F$ avec $u \sqsubset u'$

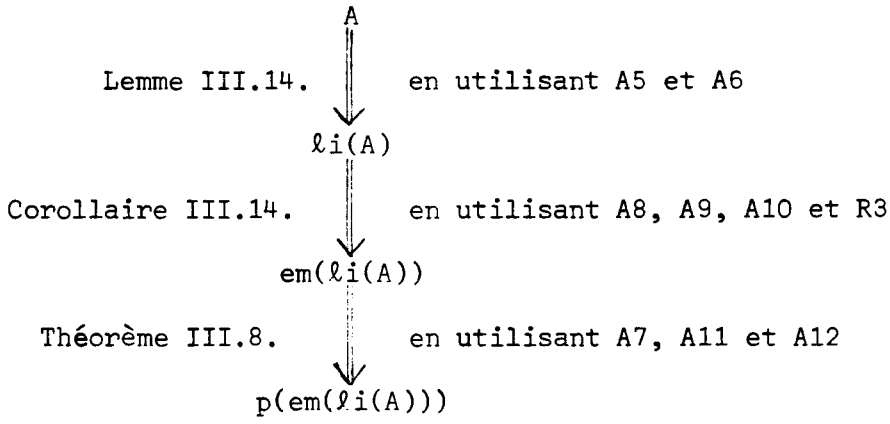
u ne permute pas avec u' dans A' car sinon A' n'est pas P-canonique.

Il existe une interprétation libre II^* telle que les calculs de A et A' pour II^* atteignent un stade (m, t) pour quelque $t \in TF(F, V)$ et ensuite il existe un m' tel que le calcul de A' atteint le stade $(m\,m', t')$ tel que $A'(m\,m') = FIN$ tandis que $A(m\,m') = \omega$.

i.e. les deux arbres A et A' ne peuvent pas être équivalents.

La preuve de la complétude de T_p pour $M(W_\Omega)$ se fait en passant par les étapes suivantes :

Etant donné un arbre de programmes fini A



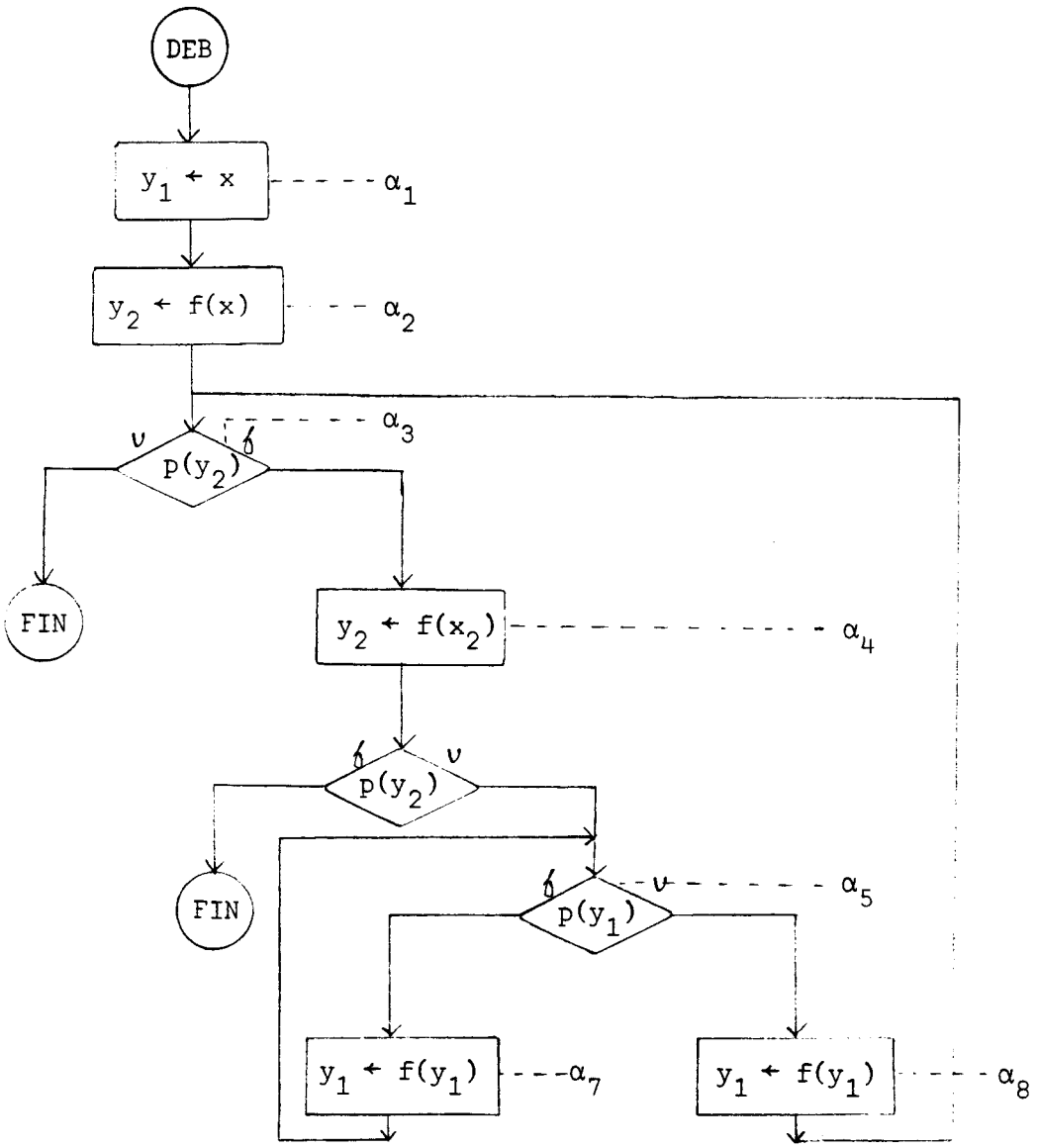
Alors s'il existe un autre arbre de programmes fini A' PATERSON équivalent à A en passant par les étapes, ci-dessus, et en utilisant uniquement les éléments de T_p, on obtient un arbre fini p(em(ℓi(A'))) qui d'après le Théorème III.8. est identique à p(em(ℓi(A))).

III-3-6. Les schémas de programmes qui s'arrêtent toujours

A tout schéma de programmes S *qui s'arrête toujours* correspond un arbre de programmes fini $\overline{S}$ COUSINEAU équivalent.

En effet, chaque schéma de programmes S qui s'arrête toujours possède un nombre fini de séquences d'exécutions complètes et à chaque séquence d'exécution complète est associée une branche finie de $\overline{S}$.

Par exemple, si on prend le schéma de programmes suivant :



Les séquences d'exécutions possibles sont :

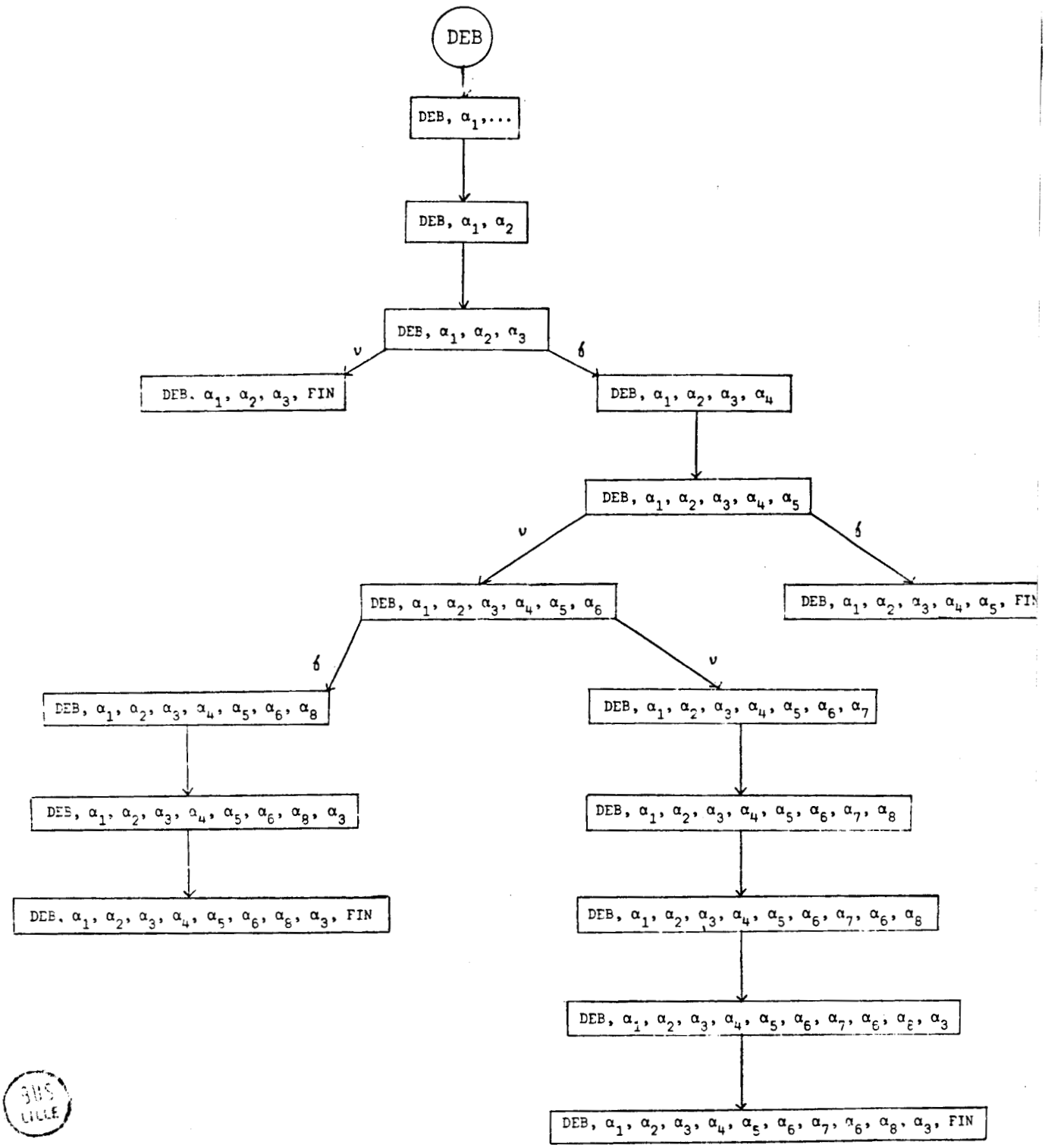
(DEB, α_1, α_2, α_3, FIN)

(DEB, α_1, α_2, α_3, α_4, α_5, FIN)

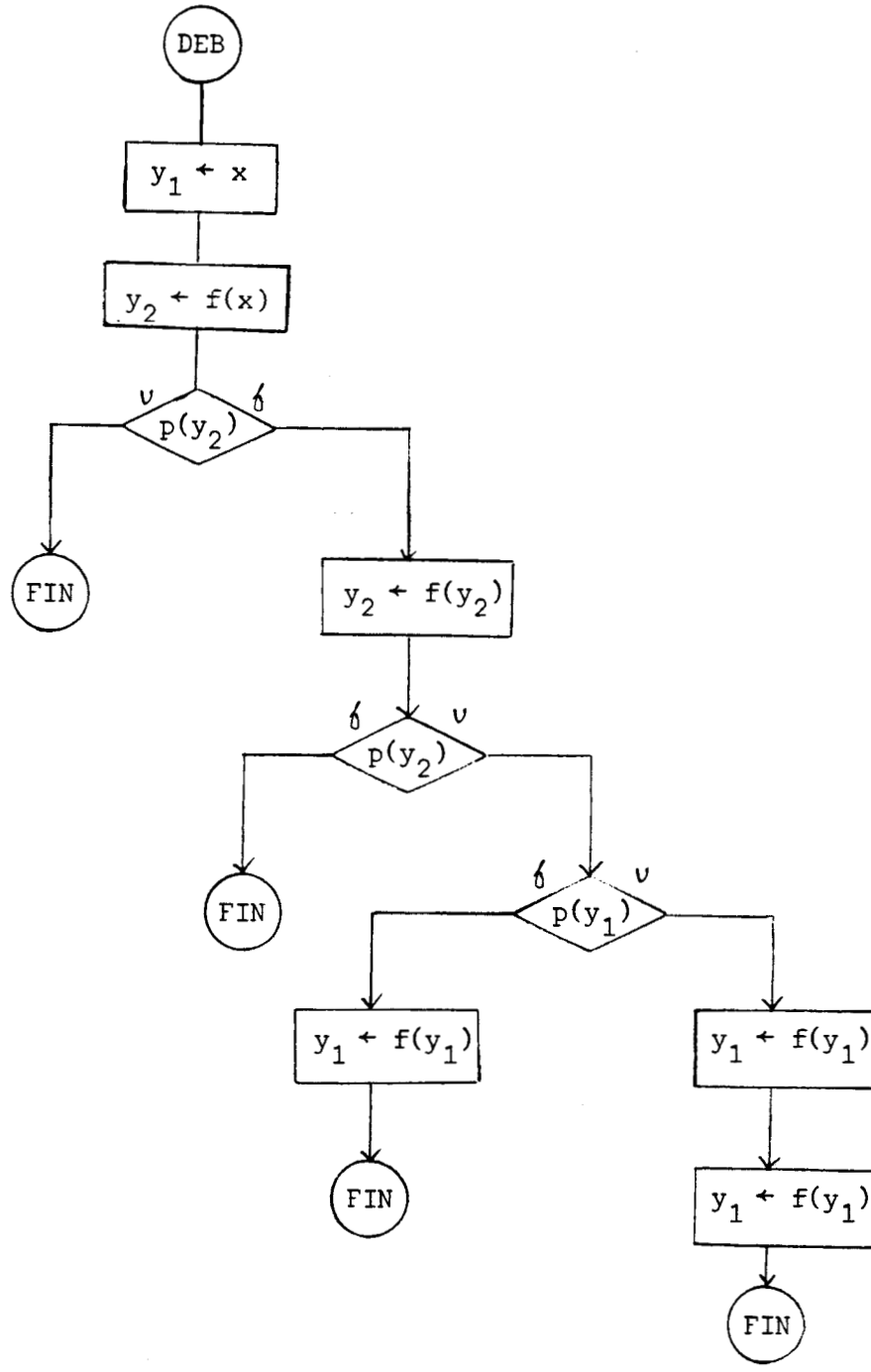
(DEB, α_1, α_2, α_3, α_4, α_5, α_6, α_8, α_3, FIN)

(DEB, α_1, α_2, α_3, α_4, α_5, α_6, α_7, α_6, α_8, α_3, FIN)

Avant d'associer un arbre de programmes fini $\overline{S}$ nous pouvons passer par un arbre d'exécution correspondant aux séquences d'exécutions suivantes :



et finalement l'arbre de programmes fini associé à S est :



L'arbre $\overline{S}_{(p)} \equiv S$

Alors la preuve de la complétude et de la consistance de T_p pour $\int_J$ est la même que pour les arbres de programmes finis.

III-3-7___SCHEMAS_BIPARTIS

Un schéma BIPARTI est un schéma de programme dans lequel l'ensemble des variables est réduit à deux variables et tous les symboles de fonctions et tous le symboles de prédicats sont monodiques.

Cette classe de schémas de programmes a été proposée par JACOB [22], et nous allons étudier la décidabilité d'équivalence de PATERSON pour cette classe.

BIRD [5], a présenté une classe de schémas de programmes et il a prouvé la décidabilité d'équivalence de PATERSON pour cette classe.

La classe étudiée par cet auteur ressemble a celle présentée par LUCKHAM PARK et PATERSON [32, page 264] mais au lieu d'avoir une seule variable d'entrée a on a deux variables d'entrée a et b i.e. en remplaçant

$$(y_1,y_2) \leftarrow (a,a) \qquad \text{par} \qquad (y_1,y_2) \leftarrow (a,b)$$

On peut appliquer cette preuve pour les schémas BIPARTIS . Néanmoins, nou allons étudier quelques résultats pour adapter cette preuve à notre méthode de tra vail.

COROLLAIRE

Il est décidable qu'un schéma biparti soit libre ou pas.

En effet, un schéma biparti n'est pas libre si et seulement s'il a deux tests identiques d'une variable sans aucune action de cette variable entre les deux.

LEMME III.15

Pour tout arbre biparti S, il existe un arbre biparti libre PATERSON équivalent.

PREUVE

α) S est fini

A tout arbre biparti S et à tout ensemble de tests non Consistant T_s et T

est associé un arbre biparti libre comme suit :

$li(S,T_s,T_t)$ = si $S(\varepsilon)$ = FIN alors FIN
sinon si $S(\varepsilon) = BOU$ alors BOU
sinon si $S(\varepsilon) = f(s), f \in F$ alors $S(\varepsilon)(li(S\downarrow_1),\phi,T_t)$
sinon si $S(\varepsilon) = f(t), f \in F$ alors $S(\varepsilon)(li(S\downarrow_1),T_s,\phi)$
sinon si $S(\varepsilon) = P(s)$ et $p \in P$ alors si ni P(s) ni $\overline{P}(s)$

n'appartient pas à T_s alors $S(\varepsilon)((li(S\downarrow_1),T_s \cup \{\overline{P}(s)\},T_t),$
$(li(S\downarrow_2),T_s \cup \{P(s)\},T_t))$

sinon si $P(s) \in T_s$ alors $li((S\downarrow_2),T_s,T_t)$
sinon si $\overline{P}(s) \in T_s$ alors $li((S\downarrow_1),T_s,T_t)$

sinon si $S(\varepsilon) = P(t)$, $P \in P$ alors si ni P(t) ni $\overline{P}(t)$ n'appartient pas à T alors $S(\varepsilon)((li(S\downarrow_1), T_s, T_t \cup \{\overline{P}(t)\}),(li(S\downarrow_2),T_s,T_t \cup \{P(t)\}))$
sinon si $P(t) \in T_t$ alors $li((S\downarrow_2), T_s, T_t)$
sinon si $\overline{P}(t) \in T_t$ alors $li((S\downarrow_1), T_s, T_t)$

β) S est infini

Choisissons pour S une suite d'approximants $S_1,\ldots, S_i, \ldots$
$Sup(\{li(S_i)\})$ est un arbre biparti libre équivalent à S (en vertu du lemme III-9).

THEOREME III.9

Pour tout arbre BIPARTI S, si S est reconnaissable alors li(S) est aussi reconnaissable.

PREUVE

$\forall m \in Dom(li(S))$, le nombre de sous-arbres $li(S)\downarrow_m \in M^{\infty}(P,\{FIN,BOU\})$ est fini car S est reconnaissable et le nombre de tests dans li(S) $\leq$ au nombre de tests dans S.
Par conséquence li(S) est reconnaissable s'il possède un nombre fini de sous-arbres $li(S)\downarrow_m$ tel que $li(S)(m) \in F$.

...

L'algorithme qui fait passer de S à li(S) associe à tout $m \in Dom(li(S))$ tel que $li(S)(m) \in F$ un $r(m) \in Dom(S)$ tel que $S(r(m)) \in F$ et $li(S)\downarrow_m = li(S\downarrow_{r(m)})$

Supposons que S possède k sous-arbres distincts $S\downarrow_{m_1}, \ldots, S\downarrow_{m_k}$ de S tel que $S(m_i) \in F$ $1 \leqslant i \leqslant k$ et soit k+1 sous-arbres distincts $li(S)\downarrow_{m_1}, \ldots li(S)\downarrow_{m_{k+1}}$ de li(S) tel que $li(S)(mi) \in F$ $1 \leqslant i \leqslant k+1$.

Soient $r(m_1), \ldots, r(m_{k+1})$ les éléments de Dom(S) associés a $m_1, \ldots, m_{k+1}$. Comme S ne contient que k sous-arbres distincts $S\downarrow_m$ tel que $S(m) \in F$ alors il existe $i,j \in [k+1]$ et $i \neq j$ tel que $S\downarrow_{r(mi)} = S\downarrow_{r(mj)}$

D'autre part, nous avons $li(S)\downarrow_{m_i} = li(S\downarrow_{r(mj)}) = li(S)\downarrow_{mj}$

Par conséquent, li(S) ne possède que k sous-arbres distincts.

Exemple III.7

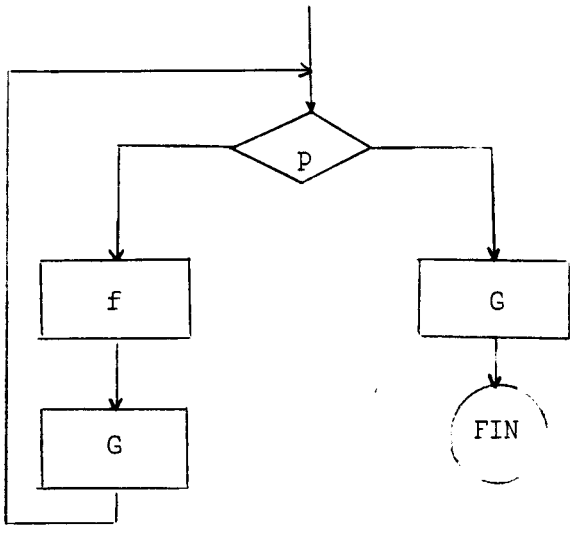


Schéma S

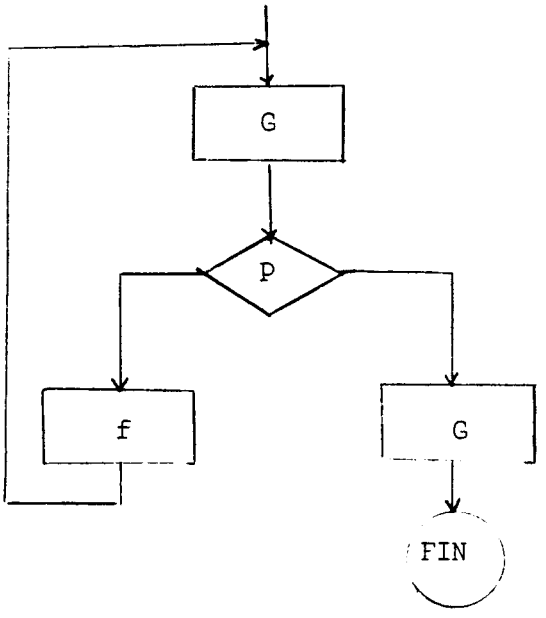
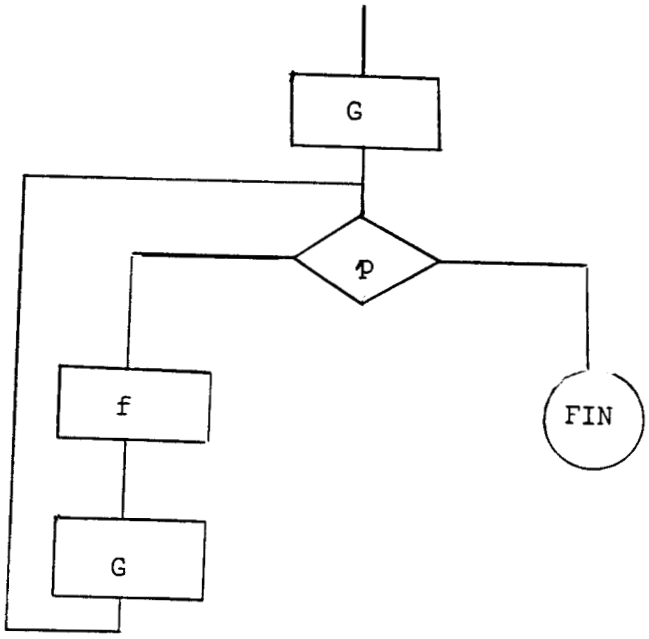


Schéma S_1



On a : $R(SR)^* = (RS)^*R$.

$G(pfG)^* = (Gpf)^*G \Longrightarrow_{p} S \equiv S'$

Schéma S'

Exemple III.8

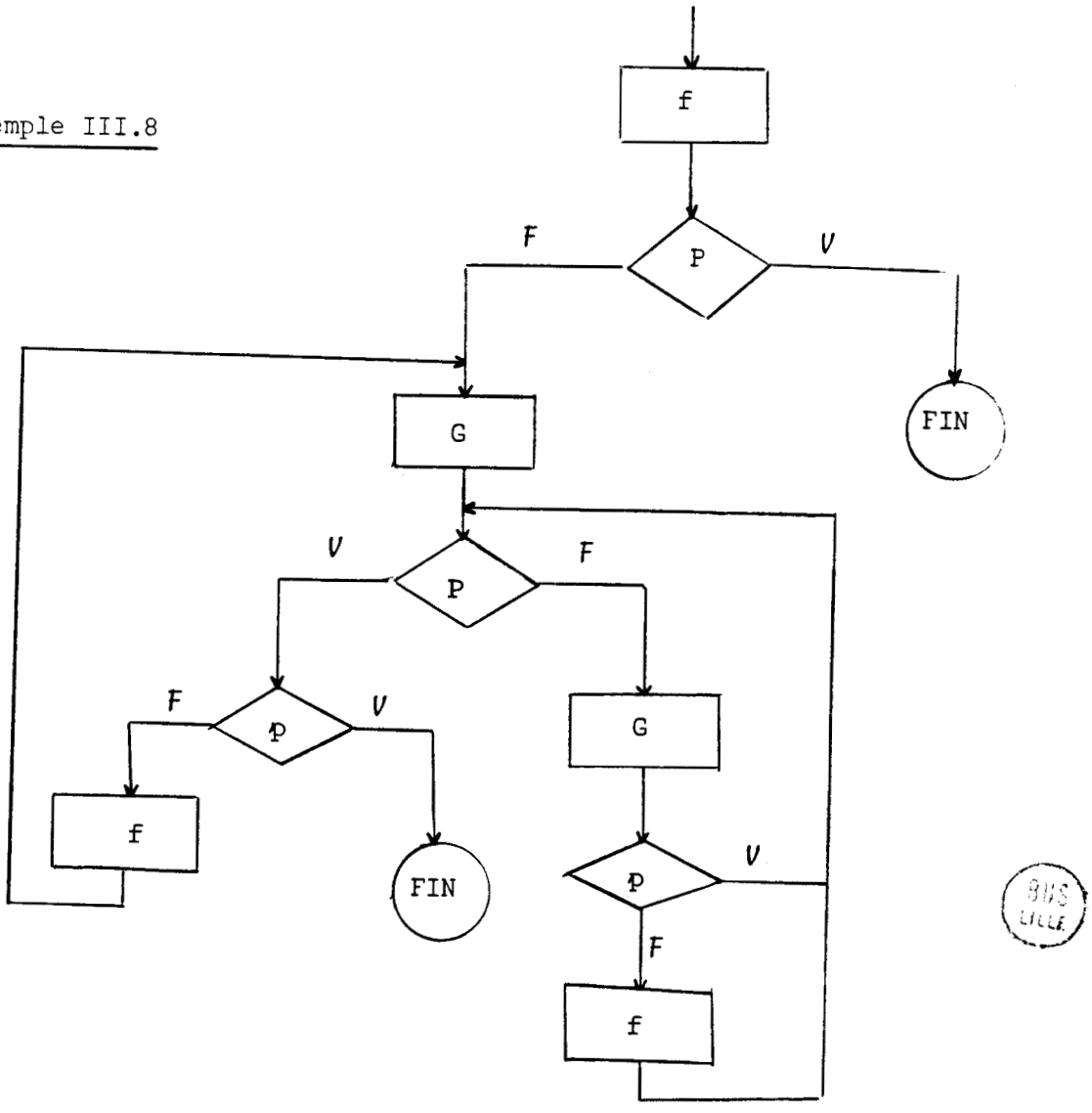


Schéma S.

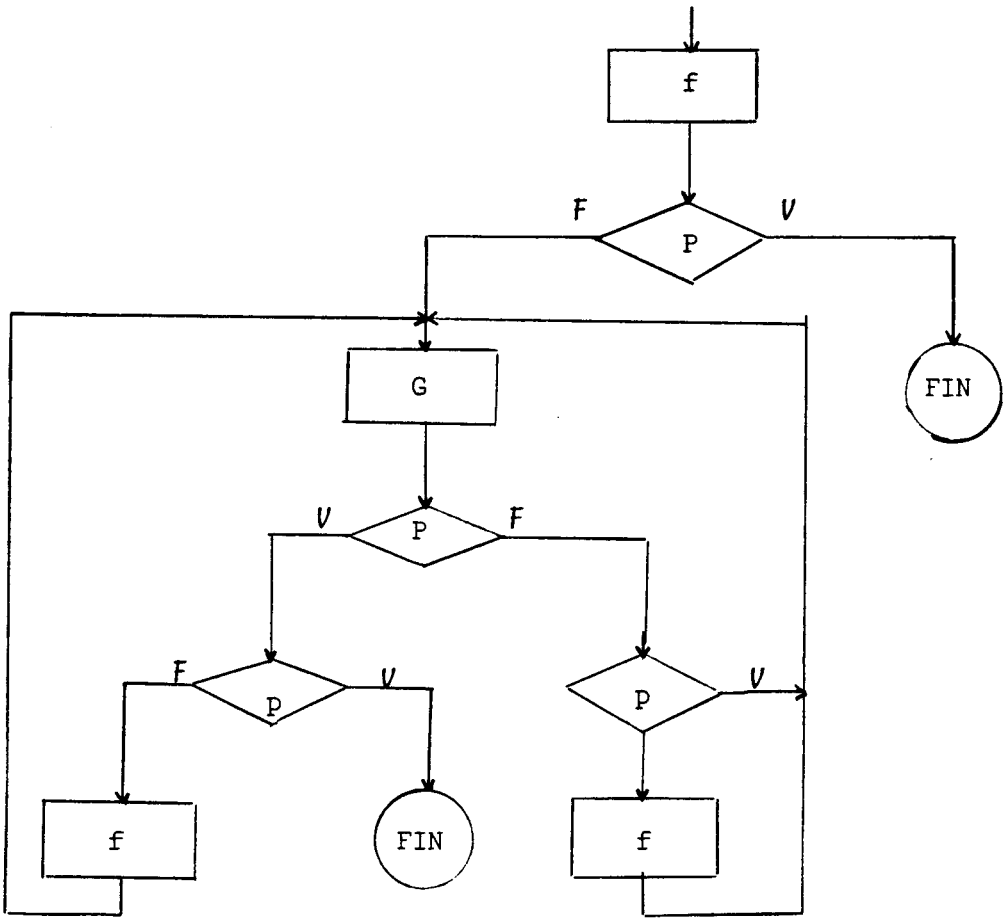


Schéma S_1

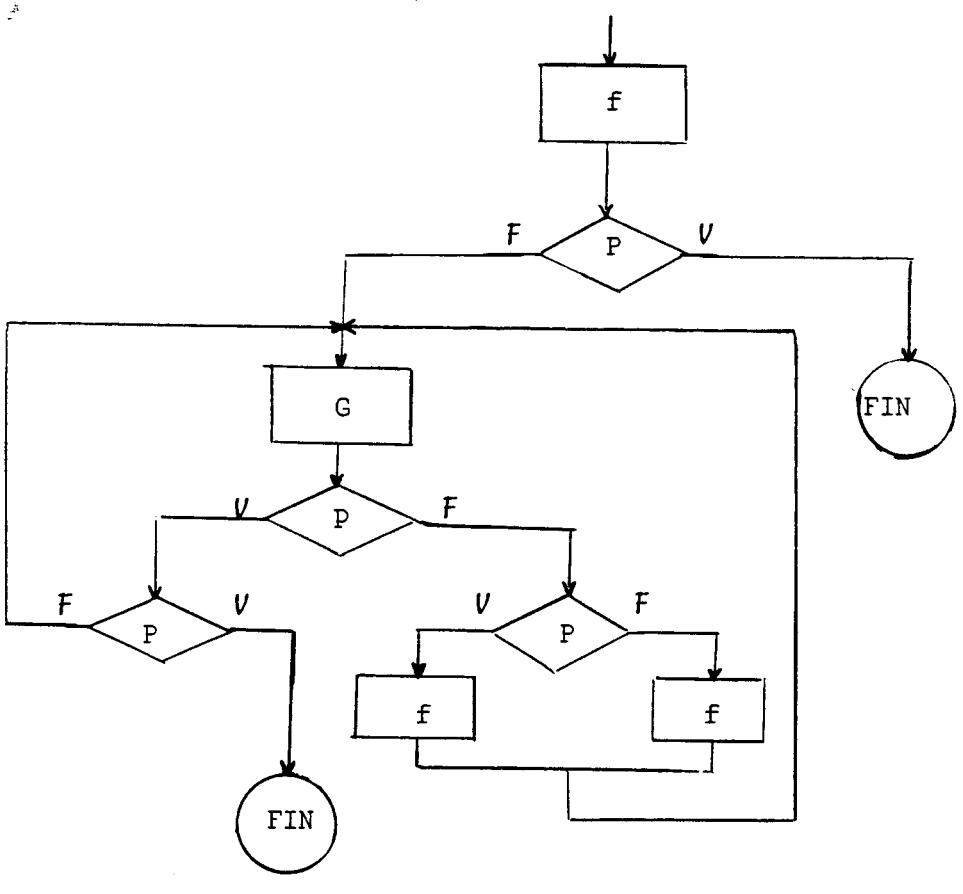


Schéma S_2

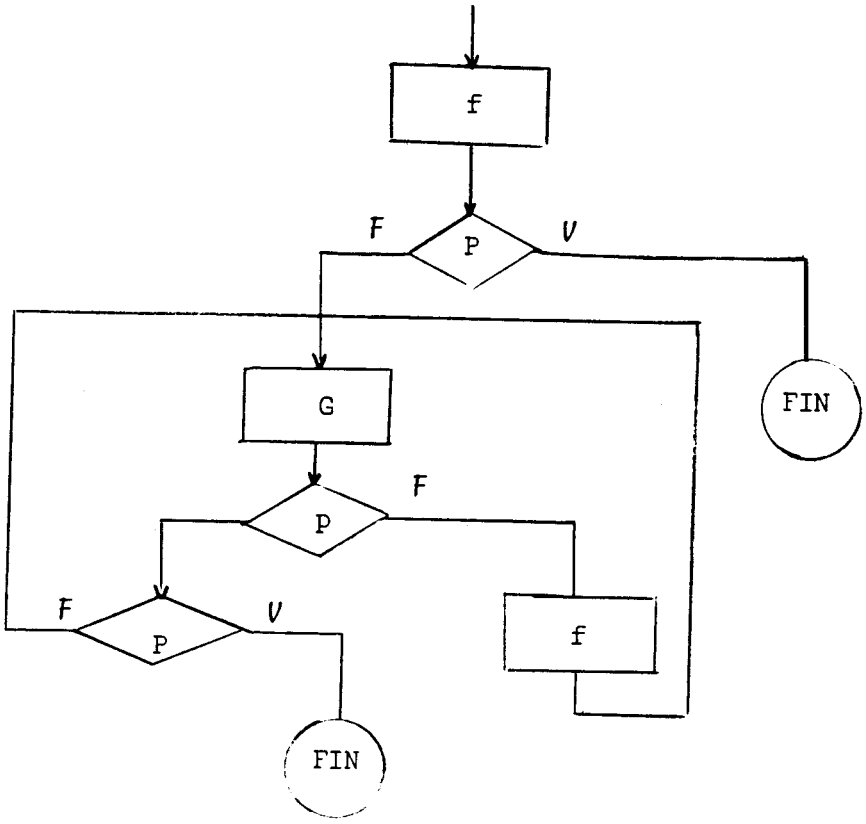


Schéma S'

$$S \xRightarrow{A12} S_1 \xRightarrow{A7} S_2 \xRightarrow{A9} S'$$

Alors $S \underset{P}{\equiv} S'$

EQUIVALENCES INTERMEDIAIRES

III-4 EQUIVALENCES INTERMEDIAIRES

III-4-1 EQUIVALENCE LIBRE

Définition : Soient A et A' deux arbres de programmes ; li(A) et li(A') les arbres de programmes libres associés.
Nous disons que A est librement équivalent à A' si et seulement si li(A)= li(A').

Notons : cette équivalence $\underset{L}{\equiv}$

Définition :

Soit $u = (q_1, \ell_1, q_2)(q_2, \ell_2, q_3)\ldots(q_k, \ell_k, q_{k+1})$ un chemin de q_1 à q_{k+1} tel que $\ell_1 = P(x_{i_1}, \ldots x_{i_n})$ et $\ell_k = P(x_{j_1}, \ldots, x_{j_n})$. Nous disons que $Z = \ell_2 \ldots \ell_{k-1}$ est un passage neutre si et seulement si $Z(x_{j_1}, \ldots x_{j_n}) = (x_{i_1}, \ldots x_{i_n})$

THEORIE CONSISTANTE ET COMPLETE POUR EQUIVALENCE LIBRE T_L

$T_L = T_C$ {A5, A6, R^1, R^2} où

R^1 : ⊢ si P alors Z; si P alors S_1 sinon S_2 fsi sinon S_3 fsi
= si P alors Z; S_1 sinon S_3 fsi.

R^2 : ⊢ si P alors S_1 sinon Z; si P alors S_2 sinon S_3 fsi fa=
si P alors S_1 sinon Z S_3 fsi

PREUVE DE LA CONSISTANCE DE T_L POUR L'EQUIVALENCE LIBRE

R^1 : La même preuve que pour l'axiome A5 (Page 91), en effet le passage neutre d'une branche ne modifie pas la consistance de l'ensemble des tests de cette branche.

R^2 : Le même raisonnement que R^1.

PREUVE DE LA COMPLETUDE DE T_L POUR EQUIVALENCE LIBRE

Considérons l'algorithme de la page 91 qui nous permet de passer d'un arbre de programme non libre à l'arbre de programme libre associé.

Pour effectuer cet algorithme nous n'utilisons que les règles R^1 & R^2 alors. Cette théorie est Complète pour cette équivalence.

On peut schématiser la preuve de la Complétude de T_L par :

Schéma de programme A

⇓ en utilisant les éléments de

arbre de programme associé $\overline{A}$

⇓ en utilisant les éléments de $T_L \setminus T_C$

$li(\overline{A})$

III.4.2. EQUIVALENCE EMONDEE

Définitions : Soient A et A' deux arbres de programmes, em(A) et em(A') les arbres de programmes émondés associés.
Nous disons que A est équivalent à A' pour équivalence émondée si et seulement si em(A) = em(A').

THEORIE CONSISTANTE ET COMPLETE POUR EQUIVALENCE EMONDEE T_E

$T_E = T_C \cup T\{A8, A9, A10\}$ (voir page 92).

La preuve de la Consistance et de la Complétude de cette théorie pour l'équivalence émondée a été démontrée au cours de l'étude d'équivalence de IANOV.

On peut schématiser la preuve de la Complétude de T_E par :

..

Etant donné un schéma de programme A

$\Downarrow$ en utilisant les éléments de T_C

arbre de programme associé $\overline{A}$

$\Downarrow$ en utilisant les élements de $T_E - T_C$

em(A)

III-4-3 EQUIVALENCE SELECTIVE

Définition : Un arbre de programme est dit "à blocs sélectifs ordonnés" si tous les blocs sélectifs apparaissant dans cet arbre sont ordonnés vis à vis de l'ordre présenté en page 107.

Définition : Soient A et A' deux arbres de programmes, se(A) et se(A') les arbres "à blocs sélectifs ordonnés" associés.

Nous disons que A est équivalent à A' vis à vis de l'équivalence sélective si et seulement si se(A) = se(A').

THEORIE CONSISTANTE ET COMPLETE POUR EQUIVALENCE SELECTIVE T_S

$T_S = T_C \cup \{A7\}$ (voir page 91)

La Consistance de cette théorie a été démontrée au cours de l'étude d'équivalence de IANOV et la preuve de la Complétude est évidente, en effet, en appliquant uniquement l'axiome A7 nous pouvons ordonner tous les blocs sélectifs d'un arbre de programme.

Etant donné un schéma de programme A

$\Downarrow$ en utilisant les éléments de T_C

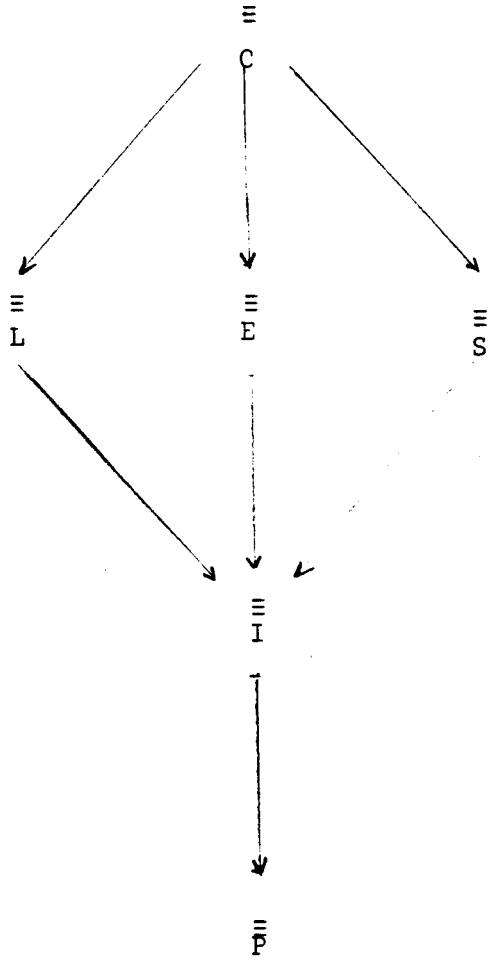
arbre de programme associé $\overline{A}$

$\Downarrow$ en utilisant A7

se(A)

...

On peut présenter les liens entre ses niveaux d'équivalences déjà étudiés comme suit :



DEUXIEME PARTIE

CHAPITRE IV

PRESENTATION DE QUELQUES STRUCTURES DE SCHEMAS DE PROGRAMMES

CHAPITRE IV

PRESENTATION DE QUELQUES STRUCTURES DE SCHEMAS DE PROGRAMMES

IV-1 D-SCHEMAS OU "GO TO-LESS PROGRAMS" OU "WHILE PROGRAMS"

La structure que l'on va présenter a été proposée par DIJKSTRA. Nous la notons D, elle a été introduite comme un outil permettant une simplification dans la construction de programmes.

En général, un schéma admet un point d'entrée et un point de sortie, on voudrait, ici, que tout "morceau" de schéma vérifie cette propriété (unicité du point d'entrée et du point de sortie) ; ceci correspond à une analyse "structurée" ; par ailleurs, les preuves de programmes ainsi conçues peuvent être plus systématiques.

La classe de D-schémas est la plus petite classe définie comme ci-dessous :

i chaque action est un D-schéma
ii si F_D et F'_D sont deux D-schémas et P est un test alors :

ii-1 séquence (F_D, F'_D)
ii-2 si p alors F_D sinon F'_D fsi.
ii-3 tant que P faire F_D fait.

sont les D-schémas.

Autrement dit, les D-schémas construits sur :

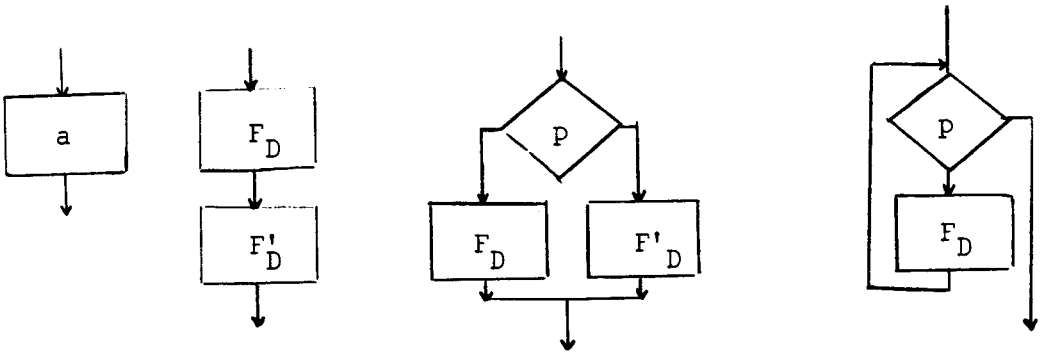
$F \cup P \cup \overline{P} \cup \{$;, si, alors, sinon, fsi, tant que, faire, fait$\}$.

...

Le langage L_D de D-schémas est défini par :

$L_D = F \cup L_D;L_D \cup$ si$(P \cup \bar{P})$ alors L_D sinon L_D fsi $\cup$

tant que $P \cup \bar{P}$ faire L_D fait.

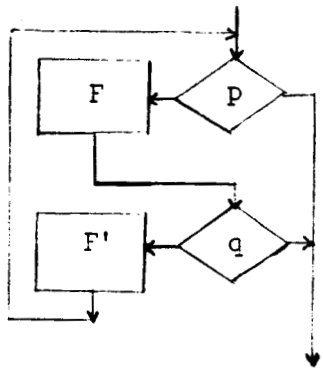


i ii-1 ii-2 ii-3

FRAGMENTS D'ORGANIGRAMMES ASSOCIES AUX D-SCHEMAS

IV-2 BJ_n SCHEMAS

BOHM & JACOPINI introduisent essentiellement à titre d'exemple contraire de D-schémas la construction :



faire P ⟶ F ; q ⟶ F' fait.

Plus généralement pour chaque n ⩾ 1, la classe de BJ_n-schémas est la plus petite classe définie ci-dessous :

i chaque action est un BJ_n-schéma

ii si F_B et F'_B sont deux BJ_n-schémas et p est un test.

ii-1 séquence $(F_B;F'_B)$ est un BJ_n-schéma.

ii-2 si p alors F_B sinon F'_B fsi est un BJ_n-schéma.

iii pour i < n, si F_{B_1} , $F_{B_2},\ldots,F_{B_i}$ sont des BJ_n-schémas et $p_1,p_2,\ldots,p_i$ sont les tests : faire $p_1 \longrightarrow F_{B_1}$; $p_2 \longrightarrow F_{B_2}$;...; $p_i \longrightarrow F_{B_i}$ fait est un BJ_n-schéma.

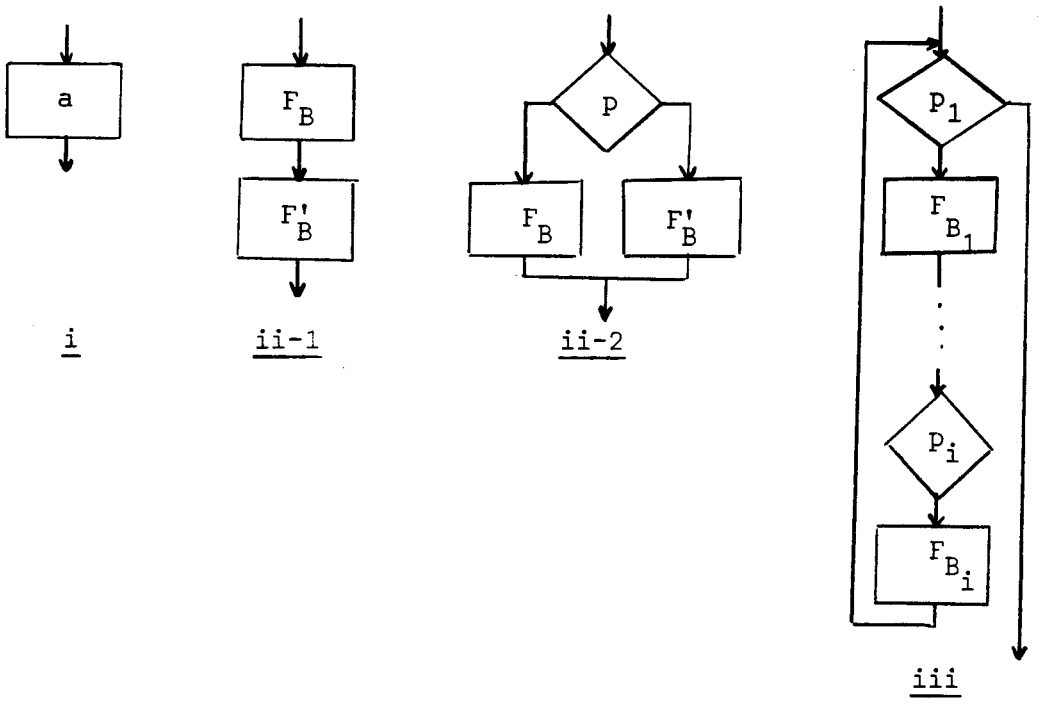
Autrement dit, les BJ_n-schémas sont construits sur :

$$F \cup P \cup \overline{P} \cup \{;, \text{si}, \text{sinon}, \text{alors}, \text{fsi}, \text{faire}, \text{fait}, \longrightarrow\} \quad .$$

Syntaxiquement, le langage BJ des BJ_n-schémas est solution de :

$$BJ = F \cup BJ; BJ \cup \text{si } (P \cup \overline{P}) \text{ alors } BJ \text{ sinon } BJ \text{ fsi}$$
$$\bigcup_{1 \leqslant i \leqslant n} \text{faire } [\overline{P} \longrightarrow BJ;]^{i-1}\ P \longrightarrow BJ \text{ fait}.$$

Les fragments d'organigrammes associés aux BJ_n-schémas :



i ii-1 ii-2 iii

IV-3 GP_n-SCHEMAS ("GP = GENERALYED PAGE").

Pour chaque $n \geq 1$, la classe de GP_n-schémas est la plus petite classe définie comme ci-dessous :

i chaque action est un GP_n-schéma

ii si $F_{G_1}, F_{G_2}, \ldots, F_{G_u}$, pour $u \geq 1$, sont GP_n-schémas et $P_1, P_2, \ldots, P_i$ $i \leq n$ sont les tests :
chaque schéma (avec une entrée et une sortie) composé de $P_1, P_2, \ldots, P_i$ (la multiple occurence des tests est autorisée) et $F_{G_1}, F_{G_2}, \ldots, F_{G_u}$ est un GP_n-schéma.

Alors chaque (1-1)-schéma composé de n'importe quel nombre d'action et de pas plus que de n tests (multiples occurences sont autorisées et en conséquence, on peut avoir plus que n unités de test) est un GP_n-schéma ; en substituant un GP_n-schéma par une action, on obtient toujours un GP_n-schéma.

IV-4 RE_n-SCHEMAS ("RE = REPEAT-EXIT")

Lorsque plusieurs itérations sont imbriquées, on voudrait pouvoir sortir directement des i-itérations englobantes : on introduit un nouveau type d'instruction exit (i) ($1 \leqslant i \leqslant n$).

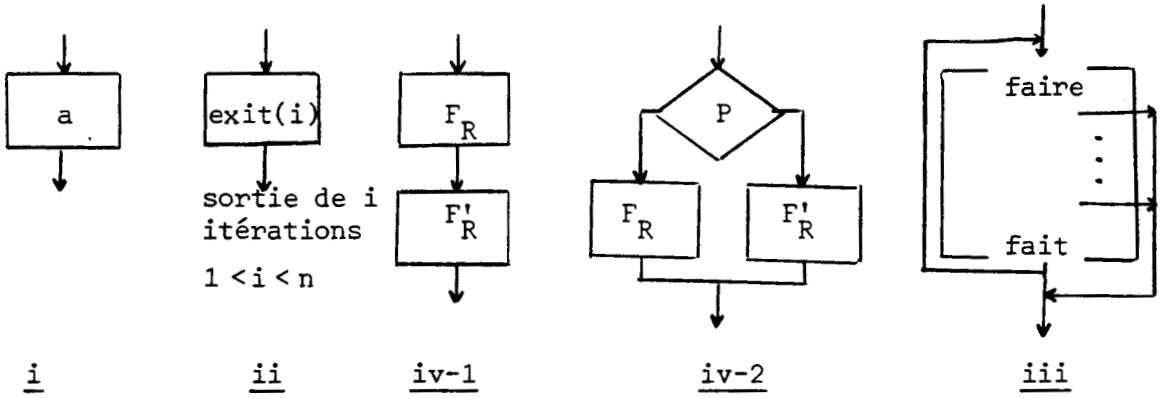
Supposons que exit(i) soit placé dans une itération qui admet k itérations englobantes ; pour $i < k$ exit(i) correspond à une rupture de séquence et passage à la fin de la i-ième itération englobante ; pour $i > k$ exit(i) correspond à une rupture de séquence à la fin de l'itération englobante la plus externe.

Pour chaque $n > 1$, la classe de RE_n-schéma est la plus petite classe définie, ci-dessous :

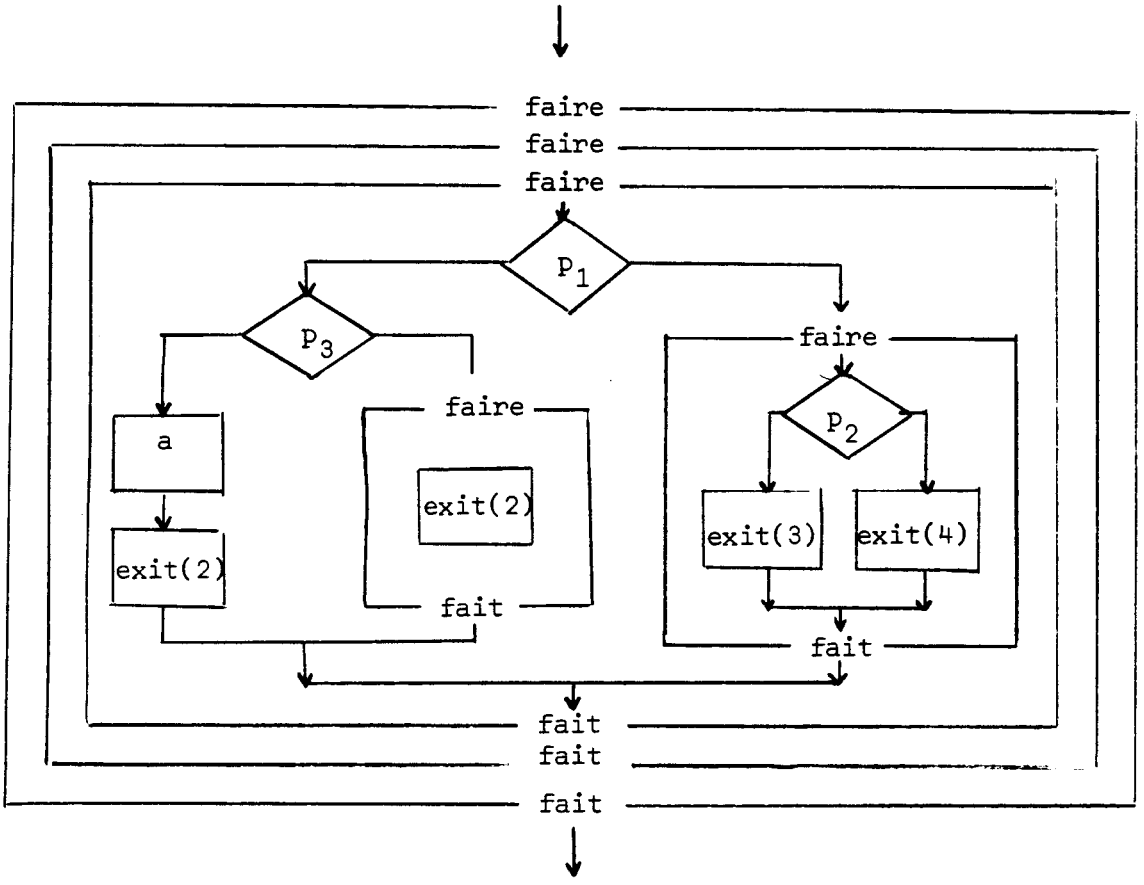
- i — chaque action est un RE_n-schéma.
- ii — l'entrée et la sortie avec instruction exit(i) $1 < i < n$ est un RE_n-schéma.
- iii — si F_R est un RE_n-schéma, séquence (faire ; F_R ; fait) est un RE_n-schéma.
- iv — si P est un test et F_R et F'_R sont deux RE_n-schémas :
  - iv-1 séquence (F_R ; F'_R) et
  - iv-2 si P alors F_R sinon F'_R fsi sont RE_n-schemas.

Syntaxiquement, le langage RE de RE_n-schémas est défini sur $\tilde{F} \cup P \cup \bar{P} \cup$ si, alors, sinon, fsi, faire, fait ; où $\tilde{F}$ contient des actions et les instructions exit(1),..., exit(n).

$RE = \tilde{F} \cup RE$; $RE \cup$ si ($P \cup \bar{P}$) alors RE sinon RE fsi $\cup$ faire RE fait.



i ii iv-1 iv-2 iii



Exemple d'un RE_4-schéma

IV-5 DRE_n-schémas (DRE = Repeat-Exit with Do While).

La classe de DRE_n-schémas est la classe de RE_n-schémas en ajoutant l'instruction "tant que P faire F'_R fait"où P est un test et F'_R est un DRE_n-schéma.

IV-6 GRE_n-schémas

Dans les RE_n-schémas, nous ne pouvons avoir l'instruction d'exit que pour sortir de au plus n-itérations englobantes.

Pour enlever cette restriction, nous généralisons la classe de RE_n-schémas de la façon suivante :

Au lieu d'avoir justement exit(i) pour $1 \leqslant i \leqslant n$, nous pouvons avoir exit (i pour $\forall i \geqslant 0$, mais nous remplaçons la restriction que aucune sous-itération ne peut pas avoir plus que n valeurs terminales différentes.

Par exemple le RE_4-schéma situé page 134 est un GRE_3 schéma.

Un exemple d'un GRE_{n+1}-schéma est la solution de système d'équations suivante :

$$\alpha_n^i = T_{n+1}^{\ i}(\alpha_n^{(i)}, OUT, \alpha_n^{(i+1)}, \ldots, \alpha_n^{(n)}, \alpha_n^{(0)}, \alpha_n^{(1)}, \ldots, \alpha_n^{(i-1)})$$

pour $i \in \{0,1, \ldots,n\}$

pour n = 2

$$\alpha_2^{(0)} = T_3^{(0)}(\alpha_2^{(0)}, OUT, \alpha_2^{(0)}, \alpha_2^{(2)})$$

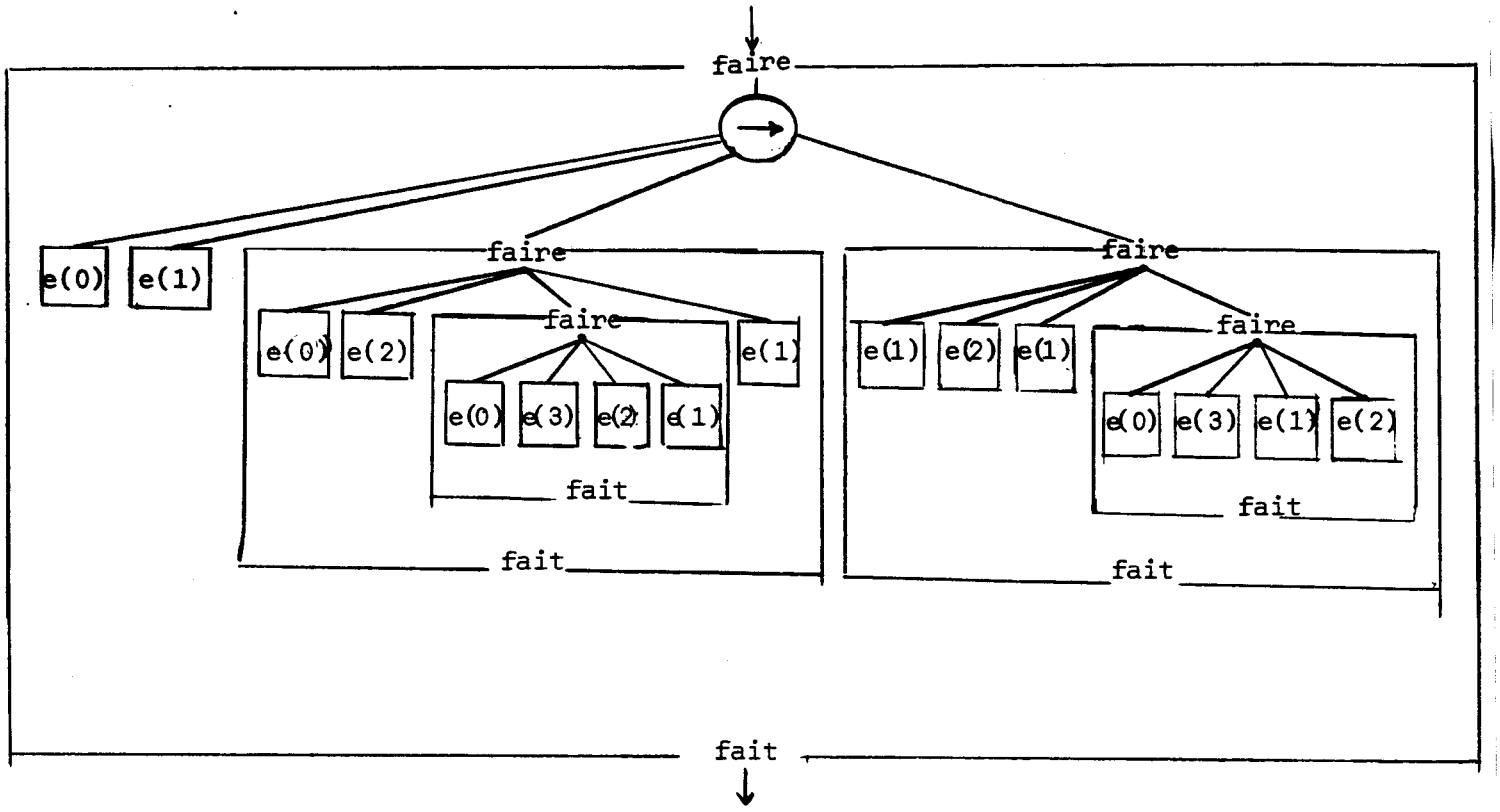
$$\alpha_2^{(1)} = T_3^{(1)}(\alpha_2^{(1)}, OUT, \alpha_2^{(2)}, \alpha_2^{(0)})$$

$$\alpha_2^{(2)} = \alpha_3^{(2)}(\alpha_2^{(2)}, OUT, \alpha_2^{(0)}, \alpha_2^{(1)})$$

$S = \{T_3^{(0)}(exit(0),exit(1),\{T_3^{(1)}(exit(0),exit(2),\{T_3^{(2)}(exit(0),exit(3),exit(2),$
$exit(1))\},exit(1)\},\{T_3^{(2)}(exit(0),exit(2),exit(1),\{T_3^{(1)}(exit(0)exit(3),exit(1)$
$exit(2))\}\}$.

exemple d'un GRE_3 schéma

voir page suivante ...



IV-7 $DGRE_n$-schémas

La classe de $DGRE_n$-schémas est la classe de GRE_n-schémas en ajoutant l'instruction "tant que P faire F_G fait" ou P est un test et F_G est un $DGRE_n$-schéma.

CHAPITRE V

LES RELATIONS ENTRE LES STRUCTURES DE SCHEMAS DE PROGRAMMES

CHAPITRE V

LES RELATIONS ENTRE LES STRUCTURES DE SCHEMAS DE PROGRAMME

V-1 THEOREME DE BOHM & JACOPINI

V-1-1 $D \underset{P}{\equiv} S$

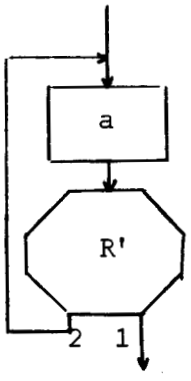
Preuve : Comme tout D-shéma est associé à un schéma de programme on a :

$$D \underset{P}{\leqslant} S.$$

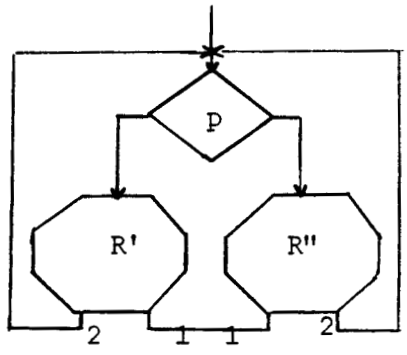
Il suffit donc de prouver $S \underset{P}{\leqslant} D$, c'est-à-dire que pour tout schéma F, on peut construire un D-schéma PATERSON équivalent.

Nous allons raisonner par induction sur la structure de schéma de programme.

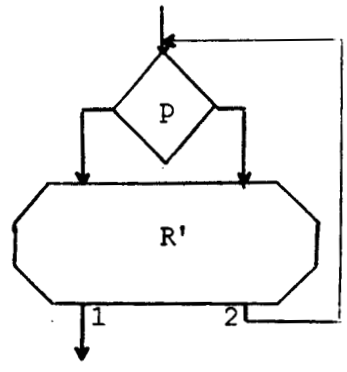
On peut décomposer un schéma de programme non vide en isolant sa première action ou son premier test : trois cas sont possibles qui sont représentés, ci-dessous :



TYPE 1



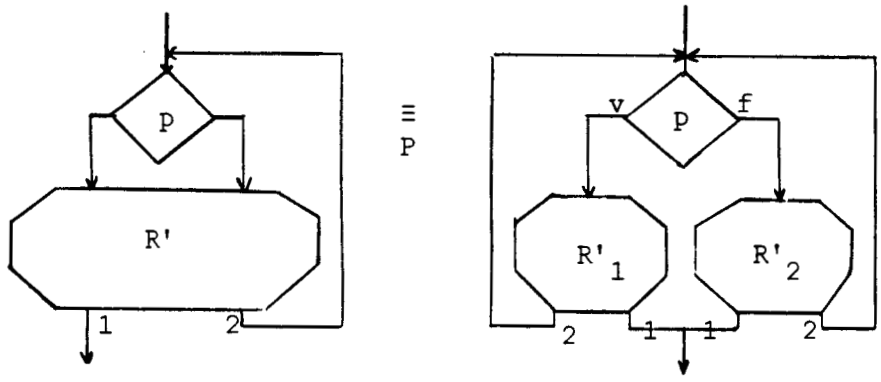
TYPE 2



TYPE 3

R' & R" désignent le reste de schéma de programme, les liaisons notées 1 ou 2 qui ne sont pas toujours représentées. Néanmoins de chaque R, au moins, un arc doit commencer.

Notons que les schémas du type 3 peuvent être ramenés à des schémas d type 2 de la façon suivante



où R'_1 & R'_2 sont les sous-parties accessibles respectivement par les vraies et fausses sorties de p.

(Notons que cette équivalence dans le cas infini n'est pas acceptable

Nous nous limiterons donc par la suite à des schémas du type 1 ou 2.

Intuitivement, pour construire un D-schéma PATERSON équivalent, il faut "retenir" les chemins utilisés.

Pour cela, nous allons introduire une nouvelle donnée BOOLENNE b et deux actions v et f qui seront interprétées respectivement comme les affectations b ⟵ vrai & b ⟵ faux.

Enfin un nouveau test est adjoint à P et interprété par :
s = vrai ⟺ b = vrai.

Pour chaque schéma de programme F du type 1 ou 2, supposons que

Δ_F = {x : x ε ($F \cup P \cup \{v,f,s\}$) & x est une unité initiale qui figure dans F}

Nous allons raisonner par récurrence sur $\# \Delta_F$.

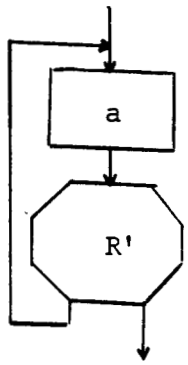
Pour $\# \Delta_F = 0 \Rightarrow F = \emptyset$ est évident.

Supposons que le théorème est vrai pour chaque schéma F avec $\# \Delta_F \leqslant$

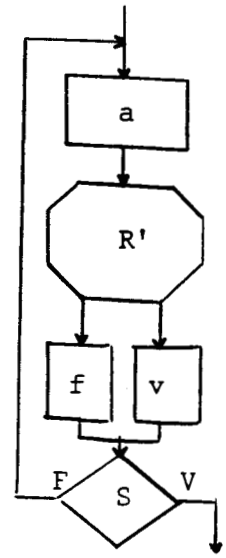
Pour # Δ_F = n+1

TYPE 1

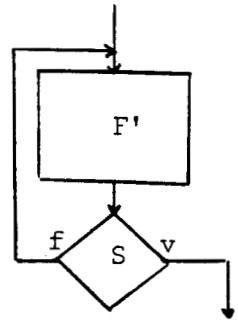
F =



transformons en F_1 =

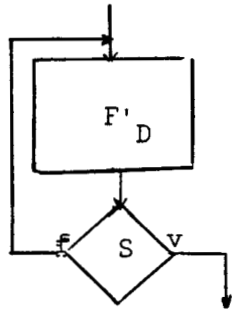


F_1 est de la forme

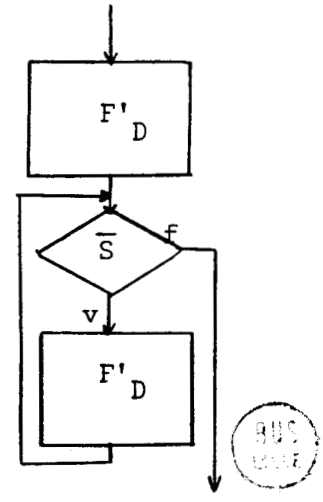


où F est un graphe orienté, alors on peut le considérer comme un schéma.
$\Delta_{F'}$ = n d'après l'hypothèse de récurrence F' est transformable en un D-schéma F'_D.

Alors F devient



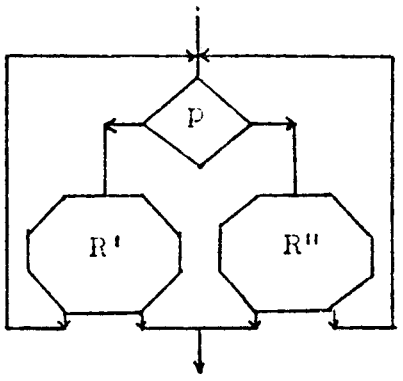
est équivalent à



Ce qu'est un D-schéma PATERSON équivalent à F.

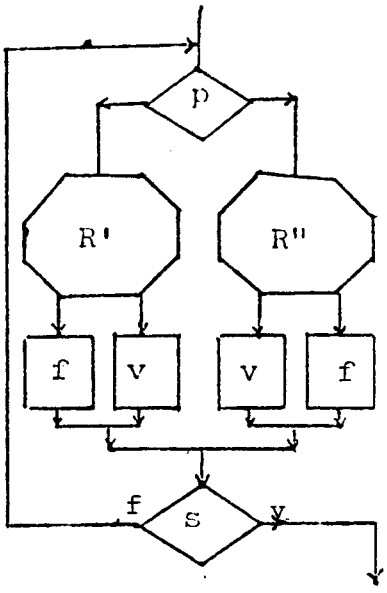
...

TYPE 2

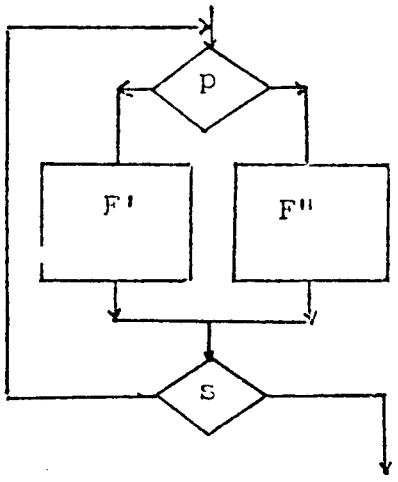


F du type 2.

transformé en F_1 =

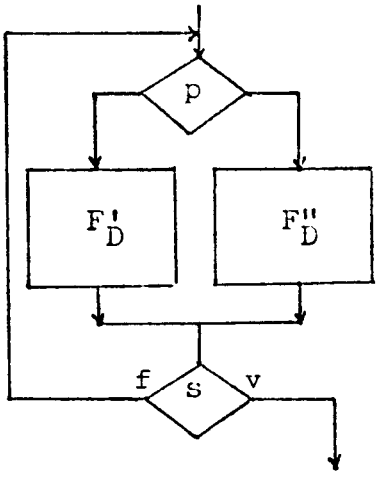


D'après l'hypothèse de récurrence F' et F" sont transformables en en deux D-schémas F'_D et F''_D.

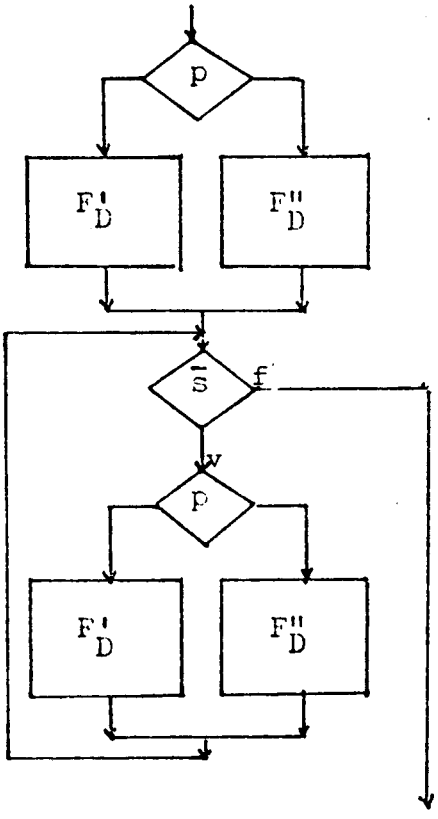


Forme de F_1

Alors F devient

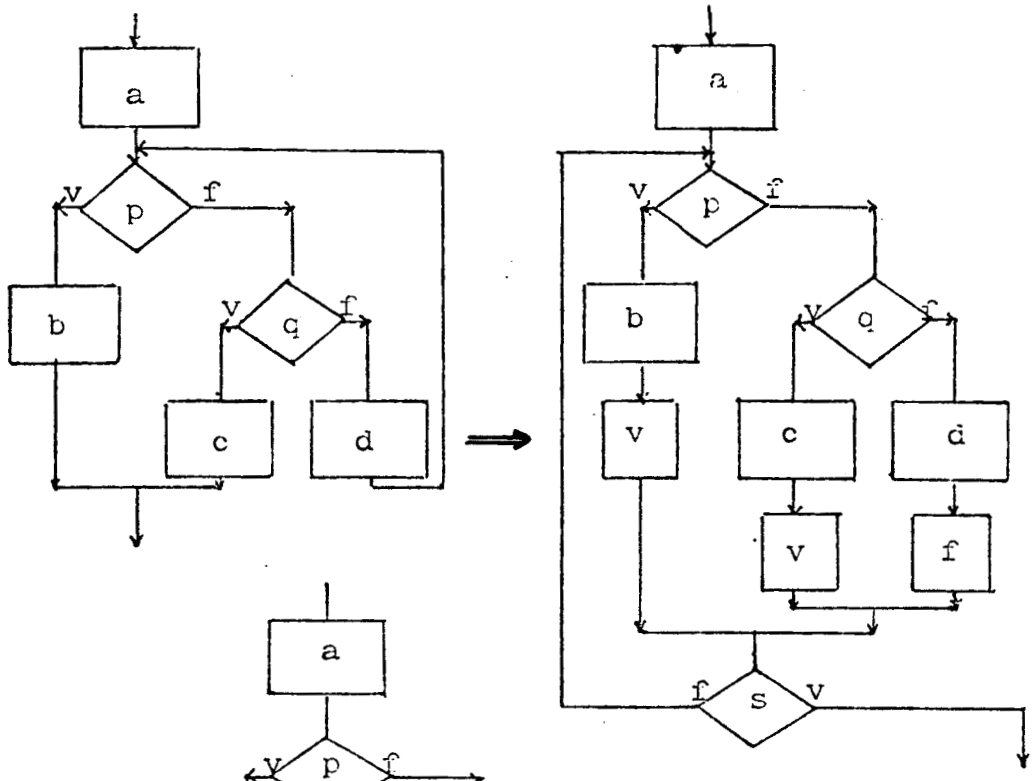


≡ P



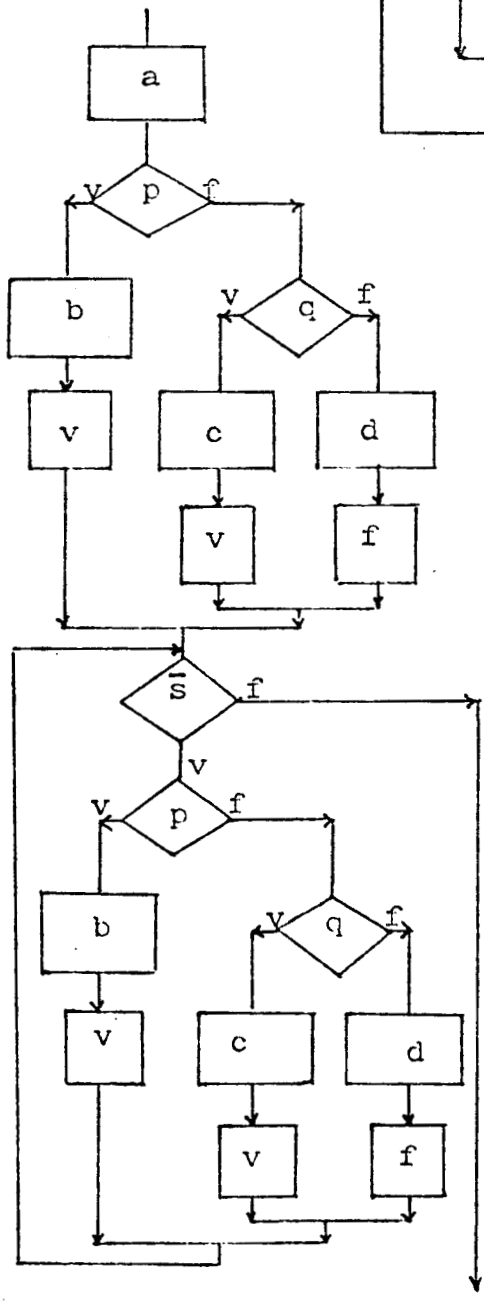
Ce qui est un D-schéma PATERSON équivalent à F.

F =



une étape de la transformation.

F_D =



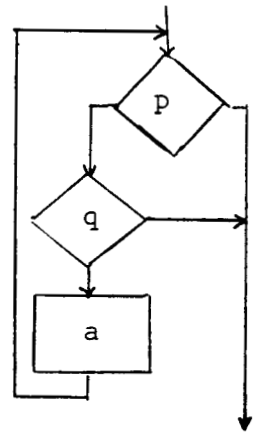
D-schéma PATERSON équivalent à F

V-1-2 $D <_I S$

Preuve : A tout D-schéma, on peut associer un schéma de programme, d'où $D \leqslant_I S$.

Pour montrer que $S <_I D$, il suffit de trouver un schéma F et une interprétation de ce schéma telle qu'aucun D-schéma (comportant les mêmes actions et tests que F) ne calcule la même fonction pour cette interprétation.

Choisissons F, représenté par le schéma ci-dessous :



Choisissons l'interprétation I = <N,I> suivante :

$I(p)_v(x) = \{(x,x) : \exists\ n\ \varepsilon\ N,\ x = 2^n\}$

$I(q)_v(x) = \{(x,x) : \exists\ n\ \varepsilon\ N,\ x = 2^n + 2^{n-1}\}$

$I(a)(x) = \{(x,y) : \quad y = x + 1\}$

Avec cette interprétation, on trouve immédiatement :

pour $2^n < d \leqslant 2^n + 2^{n-1}$ $\qquad I(F)(d) = 2^n + 2^{n-1}$

" $2^n + 2^{n-1} < d \leqslant 2^{n+1}$ $\qquad I(F)(d) = 2^{n+1}$

Supposons qu'il existe un D-schéma F_D IANOV équivalent à F et F_D contient k occurence de a.

Choisissons un entier m tel que $k+1<2^{n-2}$ et une donnée $d = 2^m-(k+1)$, il est clair que $2^{m-1} + 2^{m-2}<d\leqslant 2^m$ car la plus grande valeur de d est 2^m (k+1=0) et la plus petite valeur de $d = 2^m -(2^{m-2}-1) = 2^{m-1} + 2^{m-2} +1 \Longrightarrow d>2^{m-1} + 2^{m-2}$.

$$I(F)(d) = 2^m \;\&\; F \underset{I}{\equiv} F_D \text{ alors } I(F_D)(d) = 2^m$$

Nécessairement, le calcul d'initialisation d pour F_D contient K + 1 occurences de l'action a ; comme a admet k occurences dans F_D, alors on trouve dans F_D au moins une instruction du type :
tant que $\overline{p}$ faire...a... fait.

Considérons la première occurence d'une instruction de ce type utilisée dans le calcul de d ; avant cette instruction, dans le calcul de d, il ne peut y avoir d'instruction itérative effectuée au moins une fois car une telle instruction serait de la forme :
tant que $\overline{q}$ faire...a...fait.

et son résultat intermédiaire serait $I(F_D)(d) = 2^m + 2^{m-1}$ qui est supérieur à 2^m, ce qui est impossible.

Pour la valeur initiale $d' = 2^m + 2^{m-1} - (k+1)$ tel que $k + 1 < 2^{m-2}$, on a :

$$2^m<d'\leqslant 2^m + 2^{m-1}$$

car la plus grande valeur de d' (k+1=0) est $2^m + 2^{m-1}$, et la plus petite valeur de d' est $2^m + 2^{m-1} - 2^{m-2} + 1 = 2^m + 2^{m-2} + 1 \quad d > 2^m$.

$$I(F)(d') = 2^m + 2^{m-1} = I(F_D)(d')$$

D'après le même raisonnement, on trouve :

$I(F_D)(d') = 2^{m+1}$ qui est supérieur à $2^m + 2^{m-1}$.

Exemple :

On considère les entiers
5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
Pour avoir le résultat $2^3 = 8$, d peut être au moins 7 car si $d = 6 = 2^2 + 2$ il sortira par la vraie sortie de q.
Ainsi pour avoir le résultat $2^3 + 2^2 = 12$, d' peut être au moins 9 car si $d' = 8 = 2^3$, il sortira par la vraie sortie de p.

V-1-3 $\quad D \underset{C}{\equiv} BJ_1$

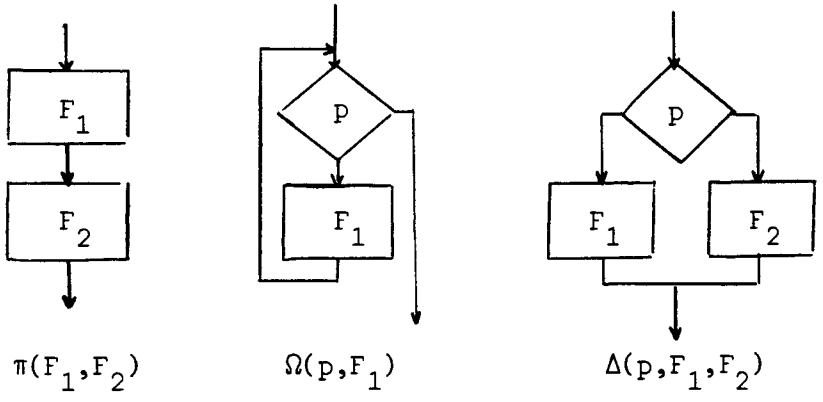
Les cycles de BJ_1-schémas sont exactement de la même forme que ceux des D-schémas.

V-2 THEOREME DE KOSARAJU

V-2-1 $\quad BJ_1 \underset{I}{<} BJ_2$

Le schéma F dans le théorème V-1-2 est un BJ_2-schéma, il a déjà été dit q ce n'est pas un D-schéma. De plus, il est impossible de le réduire à un D-schéma.

NOTE : BOHM & JACOPINI [6], ont présenté ces schémas comme schémas de base :



$\pi(F_1,F_2)$ $\quad \Omega(p,F_1)$ $\quad \Delta(p,F_1,F_2)$

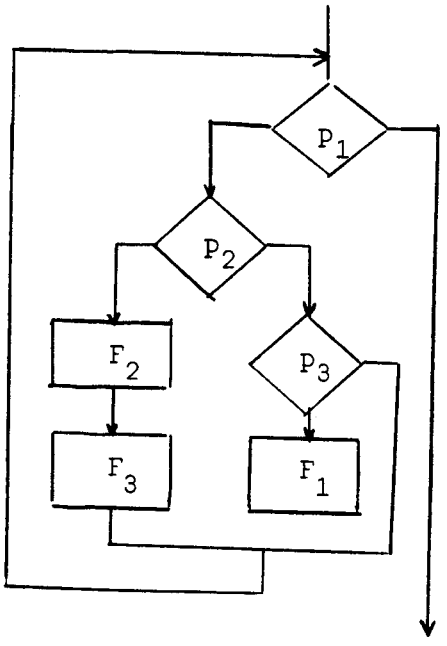
(ou p est un test et F_1 et F_2 sont deux schémas).

Ils ont partagé les schémas de programmes en deux parties :

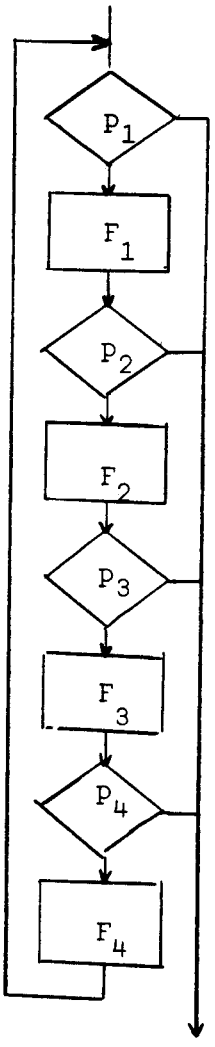
I - les schémas que l'on peut décomposer en $\pi.\Omega$ et Δ•

II - les schémas que l'on ne peut pas décomposer en π, Ω et Δ.

EXEMPLE :



$\Omega(p_1, \Delta(p_2, \Omega(p_3,F_1), \pi(F_2,F_3)))$



$\Omega_4(p_1,p_2,p_3,F_1,F_2,F_3)$

BOHM & JACOPINI [6] ont montré que :

n, il existe un Ω_{n+1}-schéma qui n'est pas réductible en Ω_n-schéma.

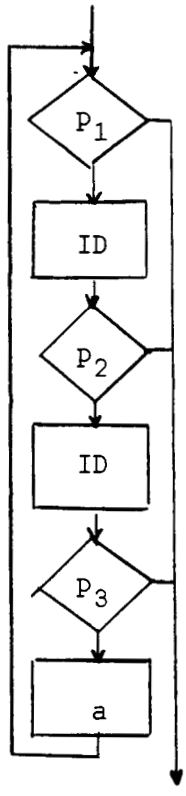
Dans le théorème V-2-2, nous allons voir que $\forall$ n, il existe un Ω_{n+1}-schéma (BJ_{n+1}) qui n'est réductible en aucun GP_n-schéma, ce qui est un résultat plus fort que le précédent.

V-2-2 $\quad BJ_i \underset{I}{<} BJ_{i+1} \qquad \forall\, i > 1$

Pour chaque n > 1, il existe un BJ_{n+1}-schéma qui n'est pas réductible en en aucun GP_n-schéma vis à vis d'équivalence de IANOV.

Preuve : On va démontrer le résultat pour n = 2 et le cas général sera évident, par la même.

Soit G =



C'est un BJ_3-schéma et nous voulons montrer qu'il ne peut pas être réductible en aucun GP_2-schéma (pour l'équivalence d IANOV).

Considérons l'interprétation I = N,I suivante :

$I(P_1)_v(x) = \{(x,x) : \exists\, m \in N,\ x = 2^m\}$
$I(p_2)_v(x) = \{(x,x) : \exists\, m \in N,\ x = 2^m + 2^{m-1}\}$
$I(p_2)_v(x) = \{(x,x) : \exists\, m \in N,\ x = 2^m + 2^{m-1} + 2^{m-2}\}$
$I(a)(x) \quad = \{(x,y) : \quad y = x + 1\}$

Comme dans le théorème V-1-2 les nombres de la forme 2^m, $2^m + 2^{m-1}$ et $2^m + 2^{m-1} + 2^{m-2}$ peuvent être considérés comme des "STOP", et pour chaque valeur initiale a de x, I(G)(a) est le plus proche et le plus grand (p.p & p.g) "STOP" de a.

Nous allons établir quelques lemmes pour compléter la démonstration du théorème.

LEMME 1 : Pour chaque contrôle de chemin composé de nombre fini de p_1, p_2, p_3 et a, la valeur finale sera supérieure à la valeur initiale.

Preuve : L'action a augmente toujours la valeur de x et les tests p_1, p_2, p_3 ne modifient pas x, alors la valeur finale sera supérieure à la valeur initiale.

LEMME 2 : Soit J un schéma composé de p_1, p_2, p_3 et a qui contient k occurrences de l'action a :

i - pour chaque valeur initiale a de x
si I(J)(a) est défini et plus grand que a + k, alors I(J)(a) est supérieur ou égal au p.p. et p.g. "STOP" de a.

ii - soient a et a' deux nombres, tel que :
a + k < au p.p. et p.g. "STOP" de a et
a' + k < au p.p. et p.g. "STOP" de a'.

Dans chaque t étapes, si x avec la valeur initiale a devient a + δ, $\delta \leq k$ avec un contrôle final en α, alors dans la même t-étapes, x avec la valeur initiale a' deviendra a' + δ avec un contrôle en α.

Preuve : i - D'une part, si a + k + 1 $\geq$ au p.p. et p.g. "STOP" de a, le lemme est évident. D'autre part, puisque I(J)(a) $\geq$ a + k + 1 et qu'il existe k occurrence de l'action a dans J, le contrôle du chemin doit passer par le circuit p_1, p_2, p_3.
Après le premier tour de circuit, la valeur de x peut augmenter au plus k, car dans ce tour chaque occurrence de a opère au plus une fois sur x.
Ainsi, les sorties fausses de p_1, p_2, p_3 peuvent être dans le circuit et le contrôle du chemin ne peut quitter le circuit qu'avec les vraies sorties de p_1, p_2, p_3 et d'après le lemme 1 I(J)(a) $\geq$ au p.p. et p.g. "STOP" de a.

...

Preuve : ii - Dans les premières t-étapes, le contrôle du chemin pour x avec la valeur initiale a ne contient pas les vraies sorties de p_1, p_2,p_3, puisque aucun a, a + 1,..., a + δ ne peut être un "STOP". Alors pour les premières t-étapes, le contrôle du chemin pour x avec la valeur initiale a' est aussi identique (ni a', a' + 1,.. a' + δ ne peut être un "STOP").
Alors x avec la valeur initiale a' deviendra a' + δ avec un contrôle en α.

LEMME 3 : Supposons que J est un GP_2-schéma composé de p_1,p_2,p_3 qui admet k occurences de l'action a. Pour chaque entier m qui vérifie $2^{m-3} > k + 1$, si

$$I(J)(2^m - k - 1) = 2^m$$
$$I(J)(2^m + 2^{m-1} - k - 1) = 2^m + 2^{m-1}$$
$$I(J)(2^m + 2^{m-1} = 2^{m-2} - k - 1) \text{ est défini}$$

Alors $I(J)(2^m + 2^{m-1} + 2^{m-2} - k - 1) > 2^m + 2^{m-1} + 2^{m-2}$

Preuve : On démontre par récurrence sur la classe de GP_2-schémas.

Supposons :
$a_1 = 2^m - k - 1$ $\quad s_1 = 2^m$
$a_2 = 2^m + 2^{m-1} - k - 1$ $\quad s_2 = 2^m + 2^{m-1}$
$a_3 = 2^m + 2^{m-1} + 2^{m-2} - k-1$ $\quad s_3 = 2^m + 2^{m-1} + 2^{m-2}$

- Pour ID et a le lemme est vrai.

- Supposons que le lemme soit vrai pour GP_2-schémas J_1,J_2,..., J_u avec k_1,k_2,...,k_u occurences de l'action a.

- On prouve ce théorème pour chaque GP_2-schéma J composé de J_1,J_2,..., J_u et chaque test q_1,q_2 ε {p_1p_2,p_3}.
Supposons que J a k occurences de a, ainsi $k = k_1 + k_2 + ... + k_u$.
Soit $I(J)(a_1) = s_1$, $I(J)(a_2) = s_2$ et $I(J)(a_3)$ est défini.
Considérons x avec la valeur initiale a_1, le contrôle du chemin passe par une séquence (probablement nulle) des q_1 et q_2, puis il entre dan un J_i et après certaines transformations sort de J_i et continue cette procédure.

Nous disons que le contrôle du chemin a un circuit global s'il existe un J_i qui est entré au moins deux fois.
On suppose que la séquence des J_i (entré) soit $J_{i_1}, J_{i_2}, J_{i_3}, \ldots J_{i_j}$ on trouve le minimum v tel que J_{i_v} augmente la valeur de x de plus que k_{i_v}, deux cas sont possibles :

α - $i_v = i_j$
β - $i_v \in \{i_1, i_2, \ldots, i_{j-1}\}$.

(un tel v doit exister car $s_1 - k_1 > k = k_1 + k_2 = \ldots + k_u$)

Soit $a + \delta$ la valeur de x avec un contrôle a l'entrée de j_{i_v}, $\delta \leqslant k_{i_1} + k_{i_2} + \ldots + k_{i_{v-1}}$, ainsi d'après le lemme 2-ii - pour x avec la valeur initiale a_2 ou a_3 le contrôle du chemin arrivera à l'entrée J_{i_v} avec les valeurs $a_2 + \delta$ ou $a_3 + \delta$ respectivement.
Si v vérifie α, d'après le lemme 2-ii, J_{i_v} augmente la valeur de x (avec la valeur initiale $a_2 + \delta$ (début de J_{i_v})), de plus que k_{i_v}. D'après le lemme 2-i, $I(J_{i_v})(a_2 + \delta) \geq s_2$.

J_{i_v} est le dernier schéma qu'augmente la valeur de a_2,
alors $I(J_{i_v})(a_2 + \delta) = s_2$.
D'après l'hypothèse de récurrence $I(J_{i_v})(a_3 + \delta) > s_3$
alors $I(J)(a_3) > s_3$.

Si v vérifie β, nous choisissons le premier circuit global dans le contrôle du chemin pour x avec la valeur initiale a_1.

La seule façon que x (avec la valeur initiale a_1 ou a_2) ait de quitter le circuit global est la vraie sortie de q_1 ou q_2. D'après lemme 1 et 3, $\{q_1, q_2\} = \{p_1, p_2\}$.

Puisque x, avec la valeur initiale a_3, rentre ainsi dans ce circuit il ne peut sortir que par la vraie sortie de q_1 ou q_2 i.e. vraie sortie de p_1 ou p_2, donc le résultat sera supérieur à s_3.

Alors d'après le lemme 3 aucun GP_2-schéma ne peut pas avoir les valeurs finales correctes pour trois valeurs a_1, a_2 & a_3, mais G peut les avoir.

...

V-2-3 $BJ_n \underset{C}{\leqslant} RE_1$

Démonstration : le schéma de programme : faire p $\longrightarrow \alpha$; q $\longrightarrow \beta$ fait, a le même calcul que le schéma : faire si p alors ; si q alors sinon exit(1) fsi sinon exit(1) fsi fait.

De même plus généralement :

faire $p_1 \longrightarrow \alpha_1 ; \ldots ; p_n \longrightarrow \alpha_n$ fait.

a le même calcul que :

faire si p_1 alors α_1; si p_2 alors $\alpha_2 \ldots$ sinon exit(1) fsi sinon exit(1) fsi fait.

V-3 THEOREME DE JACOB

V-3-1 $RE_1 \underset{C}{\equiv} GRE_1$

Preuve : Comme tout RE_1-schema(formule) est une GRE_1-formule on a : $RE_1 \underset{C}{\leqslant} GRE_1$.

Il suffit de démontrer que $GRE_1 \underset{C}{\leqslant} RE_1$ i.e. pour toute fourmule $\{h\} \in GRE_1$, on peut construire une RE_1-formule COUSINEAU-équivalente

Soit $\{g\} \in GRE_1$, telque $\{g\}$ est une sous-formule de $\{h\}$ et supposons que

- $S(\{g\}) = \{g\}$ si $\{g\}$ est une formule propre
- $S(\{g\}) = [\{g\} - i].\omega_i$ si l'unique valeure terminale de $\{g\}$ est $i \geqslant 1$
- $\Theta(\{h\}) = \{h\}[\{g\}:\{g\} \longleftarrow \{h\} \ \& \ \delta(\{g\},\{h\}) = 1/S(\Theta(\{g\}))]$

($\Theta(\{h\})$ est la formule obtenue en subsistuant $S(\Theta(\{g\}))$ à la sous-formule $\{g\}$(de profondeur 1) de $\{h\}$). Alors $\Theta(\{h\}) \in RE_1$

Il nous reste à démontrer que $\{h\} \underset{C}{\equiv} \Theta(\{h\})$

LEMME

$$1 \leqslant i \leqslant n \qquad \{f_i\} = \{f_i - i\}.\omega_i$$

Preuve : On fait une recurrence sur la valeur terminale i

- i=1 $\{f_1\} = \{f_1 - 1\}.\omega_1$ (un simple résultat de A4).

Supposons que le résultat soit vrai pour tout $i < n$.

Il faut démontrer que $\{f_n\} = \{f_n - n\}\omega_n$.

$\{f_{n-1}\} = \{f_{n-1} - n-1\}.\omega_{n-1}$ par hypothése de recurrence

$\Rightarrow \{\{f_n\}\} = \{\{f_{n-1} - n-1\}.\omega_n\}$ par A2

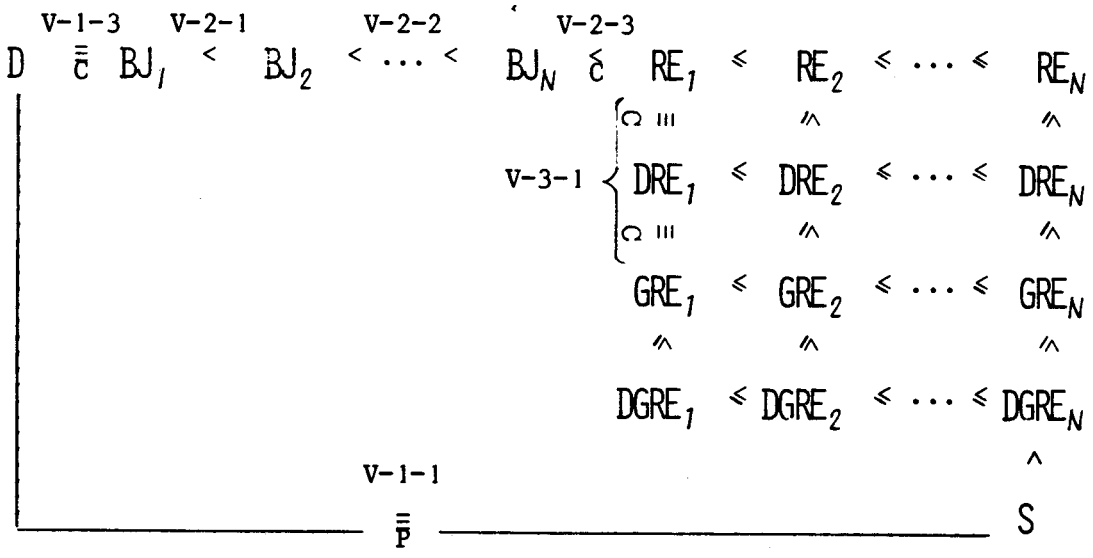
$\Rightarrow \{\{f_n\}\} = \{\{f_n - n\}.\omega\}_n$ par définition

$\Rightarrow \{f\}_n = \{f_n - n\}.\omega_n$ par A3

On a donc pour toute formule $\{g\} \in GRE_1$, $\{g\} \underset{c}{\equiv} S(\{g\})$ et d'après A1 & A2 : $\{h\} \underset{c}{\equiv} \Theta(\{h\})$.

Les principaux résultats obtenus sont resumés dans le tableau ci-dessous, où

$<$, $<$, $\equiv$, sont mis pour $\underset{I}{\leqslant}$, $\underset{I}{\leqslant}$ et $\underset{I}{\equiv}$.



THEOREME

Toutes les structures présentées, ci-dessus, sont PATERSON-équivalentes

1. ABRAMSON G. (1978).

Interpolation theorem for program schemata
Information and control 36, p. 217-233.

2. ARSAC J., NOLIN L., RUGGIU G., VASSEUR J.P. (sep. 1974).

Le système de programmation structurée EXEL
Revue technique thomson-csf vol 6 n° 3, p. 715-736.

3. ASHCROFT E., MANNA Z., PAUELI A. (July, 1973).

Decidable properties of monadic functional schemas
J. A.C.M. vol. 20, n° 3, p. 489-499.

4. BLOOM S.L., TINDELL R. (1979).

Algebraïc and graph theoretic characterizations of structured flowchart schemes
Theoretical computer science 9, North Holland pub. com. p. 265-286.

5. BIRD M.R. (January, 1972).

The equivalence problem for two-tape, one way automata, and a related problem for flow diagrams.
Memorandum n° 19, computer science department uni. of swansea.

6. BOHM C., JACOPINI G. (May 1966).

Flow diagrams, turing machines and languages with only two formation rules.
C. A.C.M. vol. 9, n° 5, p. 366-371.

7. CHANDRA A.K., MANNA Z. (1972).

Program schemas with equality
Proc. 4th. Ann. A.C.M. sym. theory Comput. Danver, colo. p. 52-64.

8. COOPER D.C., (august 1967).

BOHM and JACOPINI's reduction of flow charts
C. A.C.M. vol. 10, n° 8, p. 463 & 473.

9. COURCELLE B. (1979-1980).

Cours du D.E.A. de math. appliquées de l'université de BORDEAUX I.

10. COURCELLE B. (1976).

Les ensembles algébriques d'arbres et languages déterministes ; quelques applications à la théorie des schémas de programmes.
Thèse de doctorat d'état à l'université de PARIS VII.

11. COUSINEAU G. (1977).

Les arbres à feuilles indicées : un cadre algébrique pour l'étude des structures de contrôle.
Thèse de doctorat d'état à l'université de PARIS VII.

12. COUSINEAU F.G. & RIFFLET I.M. (1974).

Schéma de programme : problème d'équivalence et complexité.
Thèse de doctorat de 3e cycle, Université de PARIS VII.

13. DAIGNEAULT A. (1974).

Théorie des modèles en logique mathématiques
North Holland Pub.

14. DEMILLO R.A., EISEINTAT S.C., LIPTON R.J. (January, 1980).

Space-time trade-offs in structured programming : an improved combinatorial embedding theorem.
J. A.C.M. vol. 27, n° 1, p. 123-127.

15. DIJKSTRA (March, 1968).

GO TO statement considered harmful
C. A.C.M. vol. 11, n° 3, p. 147-148.

16. ERSHOV A.P. (1972).

Theory of program schemata.
Information processing 71 Nor. HOLLAND Publishing company.

17. GARLAND S.J., LUCKHAM D.C. (1973).

Program schemas, recursion and formal languages
J.of Comp. and sys. sciences vol.7, p. 119-160.

18. HOARE C.A.R. (1969).

An axiomatic basis for computer programming.
C. A.C.M. vol. 12, n° 10, p. 576-580.

19. JACOB G. (1978).

Structural invariants for some classes fo structural programs.
MFCS, Zakopane ; lect. notes in Comput. Sci. n° 64.

20. JACOB G. (1978)

Programmes réductibles à une formule EXEL de rang donné
Colloque AFCET informatique CIF-SUR-YVETTE.

21. JACOB G. (1980).

Calcul du rang des w-arbres infinis réguliers.
LITP et Université de LILLE I (à paraître).

22. JACOB G. (1981).

Une technique de derecursivations des schémas de IANOV.
Laboratoire associé au CNRS n° 248.

23. JAFFE J.M. (1980).

The equivalence of R.E. program schemes and data flow schemes
J. of Com. and sys. sci. Vol. 21, p. 92-109.

24. GREIBACH S.A. (1975)

Theory of program structures : schemes, semantics, verification.
Lec. notes in Comp. Science n° 36.

25. KAPLAN D.M. (1969).

Regular expressions and the equivalence of programs.
J. of Comp. and sys. science Vol. 3, p. 361-386.

26. KASAI T. (1974).

Translatability of flowcharts in to while programs.
J. of Comp. and sys. science. Vol. 9, p. 177-195.

27. KFOURY A.J. (1974).

Translatability of schemes over restricted interpretation
J. of Comp. and sys. science. Vol. 8, p. 387-408.

28. KLENNE (1967).
Mathematical logic.

29. KOUSARAJU S.R. (1974).

Analyses of structured programs
J. of Comp. and sys. science. Vol. 9, p. 232-255.

30. LIVERCY C. (1978).

Theorie des programmes
DUNOD informatique. PARIS.

31. LUCKHAM D.C.,PARK D.M.R., PATERSON M.S. (1970).

On formalised Computer programs.
J. of Comp. and sys. science. Vol.4, p. 220-249.

32. MANNA Z (1974).

Mathematical theory of Computation.
Mc Grow Hill, New York.

33. MANNA Z., ASHCROFT E (1975).

Translating program schemes to WHILE programs.
SIAM J. COMPUT. Vol. 4, N° 2.

34. MENDENSON (1966).

Introduction to mathematical logic.

35. MILNER R. (1970).

Equivalence on program schemes.
J. of Comp. and sys. science. Vol. 4, p. 205-219.

36. NIVAT M. (1978).

Chartres, arbres, programmes itératifs.
Synthèse, manipulation et transformation de programmes IRIA.

37. PETERSON W W., KASAMI T., TOKUTA N (1973).

On the capabilities of WHILE, REPEAT, and exit statements.
C. A.C.M., Vol. 16, n° 8.

38. RUTLEDGE J.D. (1964).

On IANOV's program schemata
J. A.C.M. Vol. 11, n° 1, p. 1-9.

39. SALOMAA A. (1966).

Complete axiom systems for the algebra of regular events.
J. A.C.M. Vol. 13, n° 1, p. 158-169.

40. SCHAENFIELD J (1976).
Mathematical logic.

41. SCOTT D (1967).

Some definitional suggestions for automata theory
J. of Comp. and sys. sciences. Vol. 1, p. 187-212.

42. WEGBREIT B. (1974).

The synthesis of loop predicates.
C. A.C.M., Vol. 17, N° 2.

43. ZABETI F. (1980).

Etude de quelques structures de schémas de programmes
Mémoire du D.E.A. d'informatique, Université de BORDEAUX I.