Synthèse de fonctions booléennes générales

P. Deschizeaux

THESES

présentées à

LA FACULTE DES SCIENCES DE L'UNIVERSITE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR INGENIEUR

par

P. Deschizeaux

Ingénieur I.R.G.

Première thèse :

SYNTHESE DE FONCTIONS BOOLEENNES GENERALES

Deuxième thèse :

PROPOSITIONS DONNEES PAR LA FACULTE

Thèses soutenues le 27 Juin 1967 devant la commission d'examen

Monsieur J. KUNTZMANN	Président
Monsieur B. VAUQUOIS	Examinateurs
Monsieur R. PERRET	

THESES

présentées à

LA FACULTE DES SCIENCES DE L'UNIVERSITE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR INGENIEUR

par

P. Deschizeaux

Ingénieur I.R.G.

Première thèse :

SYNTHESE DE FONCTIONS BOOLEENNES GENERALES

Deuxième thèse :

PROPOSITIONS DONNEES PAR LA FACULTE

Thèses soutenues le 27 Juin 1967 devant la commission d'examen

Monsieur J. KUNTZMANN	Président
Monsieur B. VAUQUOIS	Examinateurs
Monsieur R. PERRET	

FACULTE DES SCIENCES

LISTE DES PROFESSEURS

DOYENS HONORAIRES :

M. MORET

M. WEIL

DOYEN :

M. BONNIER E.

PROFESSEURS TITULAIRES :

MM.	NEEL Louis	Chaire de Physique Expérimentale
	HEILMANN René	Chaire de Chimie
	KRAVTCHENKO Julien	Chaire de Mécanique Rationnelle
	CHABAUTY Claude	Chaire de calcul différentiel et intégral
	BENOIT Jean	Chaire de Radioélectricité
	CHENE Marcel	Chaire de Chimie Papetière
	WEIL Louis	Chaire de Thermodynamique
	FELICI Noël	Chaire d'Electrostatique
	KUNTZMANN Jean	Chaire de Mathématiques Appliquées
	BARBIER Reynold	Chaire de Géologie Appliquée
	SANTON Lucien	Chaire de Mécanique des Fluides
	OZENDA Paul	Chaire de Botanique
	FALLOT Maurice	Chaire de Physique Industrielle
	KOSZUL Jean-Louis	Chaire de Mathématiques M.P.C.
	GALVANI O.	Mathématiques
	MOUSSA André	Chaire de Chimie Nucléaire
	TRAVNARD Philippe	Chaire de Chimie Générale

	SOUTIF Michel	Chaire de Physique Générale
	CRAYA Antoine	Chaire d'Hydrodynamique
	REULOS R.	Théorie des Champs
	BESSON Jean	Chaire de Chimie
	AVANT Yves	Physique Approfondie
	GALLISSOT	Mathématiques
Melle	LUTZ Elisabeth	Mathématiques
MM.	BLAMBERT Maurice	Chaire de Mathématiques
	BOUCHEZ Robert	Physique Nucléaire
	LLIBOUTRY Louis	Géophysique
	MICHEL Robert	Chaire de Minéralogie et Pétrographie
	BONNIER Etienne	Chaire d'Electrochimie et d'Electrométallurgie
	DESSAUX Georges	Chaire de Physiologie Animale
	PILLET E.	Chaire de Physique Industrielle et Electrotechnique
	YOCCOZ Jean	Chaire de Physique Nucléaire Théorique
	DEBELMAS Jacques	Chaire de Géologie Générale
	GERBER R.	Mathématiques
	PAUTHENET R.	Electrotechnique
	VAUQUOIS B.	Chaire de Calcul Electronique
	BARJON R.	Physique Nucléaire
	BARBIER Jean-Claude	Chaire de Physique
	SILBER R.	Mécanique des Fluides
	BUYLE-BODIN Maurice	Chaire d'Electronique
	DREYFUS B.	Thermodynamique
	KLEIN J.	Mathématiques
	VAILLANT F.	Zoologie et Hydrobiologie
	ARNOUD Paul	Chaire de Chimie M.P.C.
	SENGEL P.	Chaire de Zoologie
	BARNOUD F.	Chaire de Biosynthèse de la Cellulose
	BRISSONNEAU P.	Physique
	GAGNAIRE Didier	Chaire de Chimie Physique

Mme	KOFLER L.	Botanique
MM.	DEGRANGE Charles	Zoologie
	PEBAY-PEROULA J.C.	Physique
	RASSAT A.	Chaire de Chimie Systématique

PROFESSEURS SANS CHAIRE :

MM.	GIDON P.	Géologie et Minéralogie
	GIRAUD P.	Géologie
	PERRET R.	Servomécanismes
Mme	BARBIER M.J.	Electrochimie
Mme	SOUTIF J.	Physique
MM.	COHEN J.	Electrotechnique
	DEPASSEL R.	Mécanique des Fluides
	GASTINEL N.	Mathématiques Appliquées
	ANGLES-d'AURIAC P.	Mécanique des Fluides
	DUCROS P.	Minéralogie et Cristallographie
	GLENAT R.	Chimie
	LACAZE A.	Thermodynamique
	BARRA J.	Mathématiques Appliquées
	COUMES A.	Electronique
	PERRIAUX J.	Géologie et Minéralogie
	ROBERT A.	Chimie Papetière
	BIAREZ J.P.	Mécanique Physique
	BONNET G.	Electronique
	CAUQUIS G.	Chimie Générale
	BONNETAIN L.	Chimie Minérale
	DEPOMMIER P.	Etude Nucléaire et Génie Atomique
	HACQUES Gérard	Calcul Numérique
	POLOUJADOFF M.	Electrotechnique

MAITRES DE CONFERENCES :

MM.	DODU J.	Mécanique des Fluides
	LANCIA Roland	Physique Automatique
Mme	KAHANE J.	Physique
MM.	DEPORTES C.	Chimie
Mme	BOUCHE L.	Mathématiques
MM.	SARROT-RAYNAUD J.	Géologie Propédeutique
Mme	BONNIER M.J.	Chimie
MM.	KAHANE A.	Physique Générale
	DOLIQUE J.M.	Electronique
	BRIERE G.	Physique M.P.C.
	DESPRE P.	Chimie S.P.C.N.
	LAJZEROWICZ J.	Physique M.P.C.
	VALENTIN P.	Physique M.P.C.
	BERTRANDIAS J.P.	Mathématiques Appliquées T.M.P.
	LAURENT P.	Mathématiques Appliquées T.M.P.
	CAUBET J.P.	Mathématiques Pures
	PAYAN J.J.	Mathématiques
Mme	BERTRANDIAS F.	Mathématiques Pures M.P.C.
MM.	LONGEQUEUE J.P.	Physique
	NIVAT M.	Mathématiques Appliquées
	SOHM J.C.	Electrochimie
	ZADWORNY F.	Electronique
	DURAND F.	Chimie Physique
	CARLIER G.	Biologie Végétale
	AUBERT G.	Physique M.P.C.
	DELPUECH J.J.	Chimie Organique
	PFISTER J.C.	Physique C.P.E.M.
	CHIBON P.	Biologie Animale
	IDELMAN S.	Physiologie Animale
	BLOCH D.	Electrotechnique
	BRUGEL L.	I.U.T.
	SIBILLE R.	I.U.T.

R E M E R C I E M E N T S

* * * * * *

Qu'il me soit permis d'exprimer ici toute ma gratitude envers Monsieur le Professeur Kuntzmann dont les critiques et les encouragements m'ont aidés à terminer cette étude.

J'adresse mes plus vifs remerciements à Monsieur le Professeur Vauquois et à Monsieur le Professeur Perret qui ont accepté de faire partie du jury.

Je remercie particulièrement Messieurs Lustmann et Chein qui ont collaboré à ce travail et dont les suggestions m'ont été précieuses et, tous mes amis de l'Equipe de Logique qui ont su par une ambiance sympathique, faciliter mon travail.

Ce travail a été effectué dans le cadre d'un contrat entre la Délégation Générale à la Recherche Scientifique et Technique et le Laboratoire de Calcul de l'Université de Grenoble.

P R E F A C E

* * *

Le travail que nous présentons ici a pour origine un fait précis : La demande de plus en plus pressante en ordinateurs de jour en jour plus puissants et plus diversifiés rend difficile la tache des constructeurs. Au fur et à mesure de leur conception et de leur fabrication, les nouvelles machines se démodent. Dans ces conditions, la création de matériel nouveau deviendrait une entreprise risquée sans l'aide qu'apporte justement la puissance nouvelle des ordinateurs. Cette puissance permet en effet une évolution des méthodes employées pour la conception de nouveaux systèmes. Actuellement, si la conception de la structure générale d'un ordinateur est du domaine strict de l'homme, par contre, celle des réseaux réalisant cette structure est du domaine de la machine. Cette utilisation des calculateurs nécessite la mise au point de méthodes mathématiques nouvelles.

Monsieur le Professeur Kuntzmann et sous sa direction une partie importante de l'Equipe de Logique de Grenoble s'est attaché à développer des théories et algorithmes relatifs aux calculs des circuits digitaux. En particulier, un problème précis, posé par des constructeurs de machines, a été abordé : le problème de la synthèse automatique de réseaux réalisant des fonctions données, ces réseaux étant construits avec des éléments technologiques précis. Ce travail a comporté plusieurs phases :

Le problème de l'existence des réseaux, en fonction des éléments de base et des fonctions à réaliser, a été traité par Monsieur Kuntzmann, en utilisant la théorie des familles de fonctions booléennes due à Post.

Le problème mathématique de la minimisation des formes polynomiales de fonctions booléennes a été développé par MM. Benzaken et Tison. Cette étude conduisit Monsieur Benzaken à écrire des programmes permettant une minimisation automatique des fonctions.

Enfin, avec MM. Lustman, Macheras et Chein, nous avons étudié à proprement parler le problème de la synthèse des réseaux combinatoires en ne perdant pas de vue l'aspect technologique du problème. Nous fûmes naturellement conduits à étudier spécialement la synthèse à l'aide d'éléments connus. Des méthodes théoriques diverses nous ont permis d'écrire des programmes de synthèse sur IBM 7044.

Le travail s'est effectué en deux étapes :

Une phase préliminaire du travail a consisté à chercher des méthodes efficaces de synthèses de réseaux réalisant une seule fonction à la fois. Nous avons envisagé le problème sous deux aspects : recherche d'un réseau de coût rigoureusement minimum, on recherche par des méthodes heuristiques de réseaux de coût faible.

Dans une deuxième phase nous avons recherché des méthodes permettant la synthèse de réseaux dits à plusieurs sorties, c'est-à-dire réalisant plusieurs fonctions à la fois.

L'objet du présent travail est de développer ces méthodes. A cette occasion nous présentons une rapide revue des méthodes envisagées par MM. Macheras et Lustmann pour passer ensuite a des méthodes originales, de synthèse de réseaux combinatoires à plusieurs sorties.

Les méthodes générales proposées sont effectivement programmées nous en donnons à la fin de cet ouvrage un aperçu.

Pour conclure, remarquons que ce travail bien qu'étant assez général, n'est qu'une approche du problème plus vaste de la synthèse des réseaux séquentiels. Ce problème est abordé théoriquement par de nombreux chercheurs mais n'a guère reçu de solution effectivement programmée. Il reste là, un domaine de recherche pratiquement vierge, et pourtant abordable grâce à la grande puissance des calculatrices actuelles.

1 - INTRODUCTION

Le problème posé est d'origine technique. Il s'agit de développer des méthodes permettant le calcul des réseaux électriques internes des calculatrices digitales. Notre souci a été essentiellement de développer des algorithmes efficaces et utilisables dans l'industrie des calculateurs. Ce souci nous amène à considérer la rentabilité de nos méthodes, qui dépend de deux facteurs : le temps de calcul et le coût des circuits réalisés.

Cette notion de coût nous conduit à définir de façon plus précise ce qu'on appelle un réseau.

2 - DEFINITION D'UN RESEAU DIGITAL

1 - Un réseau digital R est défini au point de vue fonctionnel par :

a) un ensemble X de n grandeurs booléennes $x_1 \ldots x_n$ dénomées variables, définissant l'entrée du réseau.

b) un ensemble Z de p grandeurs booléennes $g_1 \ldots g_p$ appelées variables internes du réseau, définissant l'état interne de R.

c) un ensemble Y de q grandeurs booléennes $y_1 \ldots y_q$ définissant la sortie du réseau, ces grandeurs étant fonction des variables et des variables internes ce que nous noterons :

$$Y = F(X,Z)$$

ou encore :

$$\begin{aligned} y_1 &= f_1(x_1 \ldots x_n,\ z_1 \ldots z_p) \\ &\vdots \\ y_q &= f_q(x_1 \ldots x_n,\ z_1 \ldots z_p) \end{aligned}$$

Par définition nous appellerons fonction générale l'ensemble F des q fonctions $f_1 \ldots f_q$.

3 - "FONCTIONS INCOMPLETES"

Il arrivera fréquemment que la valeur des $y_1 \ldots y_q$ soit indifférente pour certaines valeurs des variables $x_1 \ldots x_k$, on dira alors que la fonction générale est incomplètement spécifiée. Cela ne signifie nullement que la sortie du réseau sera indéterminée. Nous choisirons seulement au mieux une détermination particulière de la fonction.

4 - RESEAUX COMBINATOIRES

Notre étude sera en fait limitée au cas où le réseau n'a qu'un seul état interne, c'est-à-dire au cas ou la sortie Y est fonction des seules variables d'entrées X.

Un réseau digital sera défini au point de vue constituants par des "opérateurs" et des "connexions" entre ces opérateurs.

5 - OPERATEURS

Un opérateur sera toujours considéré comme une "boîte noire" c'est-à-dire un élément technologique ayant des entrées $x_1 \ldots x_k$ booléennes et une sortie booléenne fonction des seules entrées :

$$y = f(x_1 \ldots x_k)$$

L'opérateur est caractérisé par la fonction f.

6 - ORIENTATION

Un opérateur sera considéré comme orienté des entrées vers la sortie.

Remarque : Nous ne considèrerons jamais le détail de la structure de l'opérateur. Seules ses caractéristiques externes nous intéressent.

7 - CONNEXIONS ENTRE OPERATEURS

Nous supposerons les opérateurs connectables entre eux, c'est-à-dire que la sortie d'un opérateur peut être reliée à une ou plusieurs entrées d'opérateurs. On bornera éventuellement le nombre d'entrées connectables à une même sortie.

Nous ne considérerons que des réseaux sans boucles c'est-à-dire tels qu'il n'existe pas de chemin respectant l'orientation des opérateurs, qui soit fermé.

On pourra considérer plusieurs types de boîtes définies par leur fonction de sortie. On s'intéressera, entre autre, aux opérateurs suivants :

Opérateur "somme" à deux variables d'entrées x_1 et x_2 dont la sortie est

$$y = x_1 + x_2 \text{ (union logique)}$$

Opérateur "produit" à deux variables d'entrées x_1 et x_2 dont la sortie est

$$y = x_1 \cdot x_2 \text{ (intersection logique).}$$

Opérateur "complément" à une variable d'entrée x dont la sortie est $y = x'$ (complément de x).

Opérateurs "Ni" à k variables d'entrées $x_1 \ldots x_k$ dont la sortie y est

$$y = x_1' \cdot x_2' \cdot \ldots \cdot x_k'$$

On pourra considérer plusieurs valeurs de k.

Opérateur "Majorité" à 3 variables d'entrées x_1, x_2, x_3 dont la sortie est :

$$y = x_1 \circ x_2 + x_1 \circ x_3 + x_2 \circ x_3$$

Opérateur "U" à 3 variables d'entrées x_1, x_2 et x_3 dont la sortie est :

$$y = x_1 \circ x_2 + x_1' \circ x_3$$

8 - PROBLEMES DE LA SYNTHESE

Le problème que nous nous posons est le suivant :

Soient :

- Y = F(X) une fonction générale incomplète.
- A un catalogue d'opérateurs
- C un cahier des charges (comportant les conditions telles que : prix de revient minimum, sécurité de fonctionnement optimale, etc ... qui seront précisées plus loin).

Nous cherchons à réaliser un réseau R dont l'entrée est X, la sortie G(X) compatible avec F(X), forme d'opérateurs de A et vérifiant les conditions C.

Le problème de l'existence de R ne sera pas abordé ici. On pourra se reporter à l'ouvrage de Monsieur Kuntzmann : "Algèbre de Boole" (Ed. Dunod); nous supposerons toujours le problème possible.

9 - CONDITIONS SUR R

Pour être réalisable industriellement, le réseau R devra respecter certaines contraintes dont les plus fréquentes seront les suivantes :

a) - Contrainte de coût.

Cette contrainte, extrêmement délicate à définir, peut être grossièrement approchée par celle du nombre minimum d'opérateurs utilisés. En fait, le coût de R sera en général fonction également de sa structure.

b) - Contrainte de temps.

Le temps de calcul de F(X) par le réseau est fonction du nombre maximum d'opérateurs qu'il est possible de trouver en série dans R. Il sera donc fréquent de chercher à minimiser ce nombre.

c) - Contraintes de sécurité.

On imposera fréquemment que R ait une structure redondante. En particulier, dans les réseaux asynchrones, on imposera à R d'être sans aléa.

d) - Citons enfin, les contraintes de planarité de R, contraintes sur la longueur des connections, etc... qui ne seront pas abordées ici.

P R E M I E R E P A R T I E

SYNTHESE DES FONCTIONS SIMPLES

* * * *

* * *

INTRODUCTION

Nous nous posons en premier lieu le problème suivant : étant donnée une fonction booléenne F(en général incomplètement spécifiée), nous cherchons un réseau R formé d'opérateurs précisés dont la sortie est compatible avec F, le nombre d'opérateurs de R étant aussi faible que possible.

REMARQUES

(1) Nous pouvons prendre deux attitudes vis-à-vis de cette contrainte de coût :

- nous pouvons chercher un minimum rigoureux du nombre d'opérateurs.
- nous pouvons chercher R ayant un nombre d'opérateurs faible.

Ces deux problèmes seront en général traités de façon fort différentes.

(2) Nous limitons en général notre étude au cas des opérateurs monotones et plus précisément :

a) somme - produit - complément

b) majorité

c) ni à k entrées

Dans le cas (a) nous n'envisageons pas le cas de réseaux 2 couches (forme polynomiale) qui a été déjà abondamment traité.

PRINCIPES GENERAUX

Nous avons utilisé deux types de méthodes :

(1) méthode par composition :

Elle consiste à composer de toutes les façons possibles les variables entre elles au moyen des opérateurs donnés, jusqu'à trouver la fonction cherchée. Des critères d'élimination de fonctions intermédiaires à priori inintéressantes permettent de réduire considérablement le nombre de compositions à envisager.

(2) méthode par fractionnement :

Cette méthode consiste à décomposer la fonction donnée en sous fonctions plus simples jusqu'à obtenir les variables. Des critères permettent de choisir les sous-fonctions pour obtenir un réseau de coût raisonnable.

REMARQUE

Les applications de ces deux méthodes sont distinctes.

- les méthodes par composition sont en général lourdes, mais permettent dans certains cas d'obtenir des réseaux de coût minimal.
- les méthodes par fractionnement sont plus rapides mais ne permettent pas jusqu'à présent d'obtenir un minimum rigoureux.

CHAPITRE 1

*

METHODE PAR COMPOSITION

Cette méthode proposée par Ackers a été particulièrement développée par MM. Kuntzmann et Lustman (voir bibliographie en Annexe).
Comme nous le verrons elle s'est révélée efficace dans le domaine de la synthèse en opérateurs croissants.

PRINCIPE

Soit f(X) la fonction à réaliser O l'ensemble des opérateurs dont nous disposons. Dans une première phase nous composons les variables x_i à l'aide des opérateurs O_j pour obtenir un ensemble Y_1 de fonctions intermédiaires.

- Si f(X) ne se trouve pas dans Y, on itère le procédé en faisant agir O sur l'ensemble $X \cup Y_1$ pour obtenir Y_2, etc...
- Si nous supposons que f(X) est réalisable à l'aide des opérateurs de O , au bout d'un nombre fini d'itérations, on obtiendra une fonction égale à f(X).

REMARQUES

(a) Cette méthode qui consiste à construire exhaustivement tous les réseaux est très lourde et ne sera utilisable qu'à l'aide de certains critères de simplifications.

(b) Cette méthode est destinée à être programmée sur ordinateur, et de ce fait nécessite un codage particulier des fonctions à traiter.

1 - REPRESENTATION DES FONCTIONS COMPLETES

Toutes les fonctions (et en particulier les variables) sont représentées par des vecteurs b o o l é e n s "colonnes" définis de la manière suivante :

a On fait choix d'un ordre des variables qui seront appelées $x_1 \ldots x_n$.

b A chaque point X_i de l'espace H des variables, de coordonnées $x_{i_1}\ x_{i_2}\ \ldots\ x_{i_n}$ on associe l'entier $N_i = \sum_{j=1}^{n} x_{ij} 2^j$, on range les points X_i par N_i croissants (cet ordre sera appelé ordre naturel).

(c) A chaque fonction $f(x_1 \ldots x_n)$ on associe la suite des valeurs de f aux points rangés dans l'ordre naturel cette suite ordonnée sera disposée en colonnes.

EXEMPLE

Pour 3 variables, les points sont rangés dans l'ordre :

000, 001, 010, 011, 100, 101, 110, 111.

La fonction $x_1 + x_2 \cdot x_3$ sera donc représentée par la suite de ses valeurs :

0, 0, 0, 1, 1, 1, 1, 1

et sera donc représentée par le vecteur colonne :

$$V_f = \left|\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{array}\right|$$

REMARQUE

L'intérêt de cette représentation réside dans la simplicité des opérations de composition à l'aide d'opérateurs donnés et de comparaison entre colonnes : ces opérations se font composantes à composantes sur les vecteurs.

2 - REPRESENTATIONS INCOMPLETES

Alors que la représentation des fonctions complètes était indépendante de la fonction à réaliser, nous abordons ici une représentation spécifique du problème à traiter.

Soit $f = \underline{f} + \Phi\, \bar{f}$ la fonction incomplète à réaliser. Soit P l'ensemble des points de l'hypercube H où f n'est pas spécifiée. Nous nous désintéressons des valeurs sur P des fonctions intermédiaires (variables comprises) obtenues au cours de l'opération de composition. On pourra donc caractériser chacune de ces fonctions par la suite de ses valeurs aux points de H - P ordonnées comme précédemment.

REMARQUE

Pour pouvoir spécifier les points absents, on sera obligé de considérer simultanément la colonne d'une fonction et les colonnes des variables.

On est donc ammené à ranger toutes ces colonnes dans un même tableau T.

EXEMPLE

$$f = x_1\, x_2 + x_1\, x_3 + \emptyset\, (x_1 + x_2')$$

f est non spécifiée sur les points 000, 001, 100.

f est égale à zéro sur les points 010, 011.

f est égale à un sur les points 101, 110, 111.

Le tableau T dans lequel nous rangerons toutes les colonnes obtenues par compositions sera :

T =

x_1	x_2	x_3	.....	f
0	1	0		0
0	1	1		0
1	0	1		1
1	1	0		1
1	1	1		1

nous constatons effectivement que la colonne de f ne représente f qu'associée aux 3 colonnes des variables.

3 - OPERATEURS CROISSANTS

(a) Si nous ne disposons que d'opérateurs croissants, nous ne pouvons par composition des variables obtenir que des fonctions croissantes.

Soit alors $f = \underline{f} + \emptyset\ \bar{f}$ où $\underline{f}$ et $\bar{f}$ ne sont pas croissant, la condition de possibilité du problème est que la plus petite fonction croissante supérieure $(\hat{\underline{f}^*})$ à $\underline{f}$ soit inférieure à la plus grande fonction croissante $(\hat{\bar{f}_*})$ inférieure à $\bar{f}$:

$$(\hat{\underline{f}^*}) \qquad (\hat{\bar{f}_*})$$

Si cette condition est remplie, le problème se ramène à réaliser la fonction

$$f\ g = (\hat{\underline{f}^*}) + \emptyset\ (\hat{\bar{f}_*})$$

On est ramené au cas où les bornes sont croissantes.

(b) Si outre les variables X, nous disposons de leur compléments X', nous pouvons considérer f(X) comme une fonction g(X, X') croissante par rapport à X et à X',

$$g(XX') = \underline{g}(X,X') + \emptyset\ \bar{g}(X, X')$$

et en considérant $X \cup X' = Y$ comme un ensemble de 2n variables y_i indépendantes, on est ramené à la synthèse de la fonction g(Y) dont les bornes sont croissantes et qui n'est définie que sur les points ou $\forall\ i\ ,\ y_i = (y_{2-i})'$

(c) Représentation :

Dans tous les cas nous sommes ramenés au problème de la synthèse d'une fonction g dont les bornes $\underline{g}$ et $\bar{g}$ sont croissantes. Or ces fonctions croissantes peuvent être définies par leurs seuls points caractéristiques. On peut aussi définir $\bar{g}$ par la liste des points caractéristiques de $\bar{g}'$.

Soit Q l'ensemble des points non caractéristiques de $\underline{g}$ ou de $\bar{g}'$. Nous nous désintéresserons de la valeur des fonctions sur l'ensemble Q puisque ces valeurs sont déterminées par la condition de croissance. Nous sommes donc ramenés à la synthèse d'une fonction spécifiée uniquement sur H-P-Q, problème qui a déjà été traité.

Chaque fonction φ intermédiaire dans la synthèse de g sera donc représentée par une colonne d'un tableau T : chaque composante de cette colonne représentant la valeur de φ sur les points caractéristiques de $\underline{g}$ ou de $\bar{g}$.

4 - VARIANTE : METHODE DU PIVOT

Nous pouvons représenter toute colonne φ_i par la colonne $\Psi_i = \varphi_i \oplus f'$

Dans cette transformation (qui est biunivoque) f est codée par $g = f \oplus f'$ donc par une colonne uniquement constituée par des 1.

Le problème de l'obtention de colonne proches de f revient à celui de la fabrication de colonnes aussi grandes que possible. Dans cette représentation chaque opérateur $\alpha(x_1\ x_2\ \ldots\ x_k)$ sera remplacé par l'opérateur $\beta(x_1, x_2, \ldots, x_k) = \alpha(x_1\ x_2\ \ldots\ x_k) \oplus f'$.

En particulier si $\varphi_1\ \varphi_2\ \varphi_3$ sont trois colonnes codant f_1, f_2, f_3, on formera la colonne φ correspondant à $Maj(f_1, f_2, f_3)$ en formant :

$$\varphi = Maj(f_1, f_2, f_3) \oplus f'$$
$$= Maj\ (f_1 \oplus f', f_2 \oplus f', f_3 \oplus f')$$
$$= Maj\ (\varphi_1, \varphi_2, \varphi_3)$$

Nous remarquons que dans le cas d'opérateurs majorité, la représentation ne complique pas la formation de nouvelles colonnes.

REMARQUE

Nous pouvons changer d'ordre des points caractéristiques pour regrouper ceux où f=1 et ceux où f=0, on utilisera l'opérateur α', sur les autres l'opérateur α. Cette représentation intéressante a été proposée par Ackers et développée et utilisée avec succès par Monsieur Lustman.

5 - COMPOSITION ENTRE COLONNES

Composer des fonctions à l'aide d'un opérateur booléen revient donc à composer termes à termes les colonnes correspondantes. Chaque nouvelle colonne ainsi fabriquée est introduite dans le tableau T et considérée ou pas suivant comme une nouvelle variable.

6 - CRITERES D'ELIMINATION

Il va de soi que cette méthode ne sera applicable (même en machine) que si nous pouvons limiter le nombre de colonnes ainsi formées. Soit par exemple une fonction de 10 variables à réaliser à l'aide d'un opérateur à 2 entrées. Le premier pas de notre méthode nous conduit à construire C_{10}^{2} colonnes nouvelles soit 45. Le deuxième pas à C_{45+10}^{2} colonnes soit 1475. Si parmi ces colonnes on ne trouve pas notre fonction, nous en considérerons $C_{1475+45+10}^{2}$ soit un nombre de l'ordre de 2 millions, etc...

Nous allons donc chercher à éliminer certaines colonnes. Deux types de critères vont apparaître ici : des critères dits "rigoureux" qui consistent à éliminer les colonnes qui ne figurent certainement pas dans un réseau de coût minimum, et des critères dits "approchés" qui consistent à éliminer les colonnes qui ne figurent probablement pas dans un réseau minimum.

Naturellement les critères approchés permettent d'éliminer plus de colonnes que les rigoureux.

Notons que nous n'avons malheureusement pas de critères d'élimination rigoureux dans le cas général : nous n'en verrons apparaître que dans le cas de réseaux arborescents d'opérateurs monotones.

(1) Réseaux arborescents, critère 1 -

Si φ et Ψ sont deux colonnes égales, on peut éliminer la plus chère. En effet, un réseau contenant la plus chère n'est certainement pas minimal.

Si les coûts sont égaux on éliminera l'une des deux au choix.

Notons au passage que ce critère est rigoureux et qu'il ne s'applique pas si le réseau n'est pas arborescent.

(2) Réseaux arborescents, opérateurs croissants, critère 2 -

Citons sans démonstration, le critère démontré par Monsieur Kuntzmann :

- Si on recherche un réseau arborescent d'opérateurs croissants réalisant la colonne f et si φ et Ψ sont deux colonnes vérifiant :

a) $\varphi_i = f_i$ entraîne $\Psi_i = f_i$ (en désignant par φ_i et Ψ_i les diverses composantes de φ et Ψ.)

b) Le coût du réseau arborescent réalisant φ est supérieur ou égal à celui du réseau réalisant Ψ.

Alors on peut éliminer φ qui n'apparaît certainement pas dans un réseau arborescent de coût minimum.

Notons que la condition (a) peut s'écrire $Maj(\varphi, \Psi, f) = \Psi$ expression qui est aisément vérifiable en machine.

REMARQUE :

Dans la représentation du pivot, le test d'élimination se transforme: si $Maj\ (f_1, f_2, f) = f_2$

et si $\varphi_1 = f_1 \oplus f'$, $\varphi_2 = f_2 \oplus f'$, $\varphi = f \oplus f' = 1$

alors $Maj\ (\varphi_1 \oplus f', \varphi_2 \oplus f', 1 \oplus f') = f_2$

ou encore $Maj\ (\varphi_1, \varphi_2, 1) = f_2 \oplus f' = \varphi_2$

$$\varphi_1 + \varphi_2 = \varphi_2$$

ce qui se ramène à :

$$\boxed{\varphi_1 < \varphi_2}$$

condition extrèmement simple à vérifier, et qui fait l'intérêt de la méthode du pivot.

REMARQUES :

1) - Ce critère est rigoureux, mais ne conduit malheureusement pas à un grand nombre d'éliminations. Disons à titre indicatif qu'on peut arriver à éliminer 99 % des colonnes ce qui est insuffisant si on a des millions de colonnes à considérer.

2) - Ce critère est long à vérifier car il nécessite une comparaison de chaque nouvelle colonne avec toutes les anciennes, ce qui peut devenir prohibitif s'il y a trop de colonnes anciennes.

(3) Réseaux arborescents, opérateurs monotones, critère 3

Dans un réseau arborescent d'opérateurs monotones réalisant f, si φ est une fonction intermédiaire, f est soit croissante, soit décroissante en fonction de φ. Autrement dit, on a soit :

$$F = A\varphi + B \qquad (1)$$

soit

$$F = A\varphi' + B \qquad (2)$$

si le cas (1) se produit, on sait que φ peut être avantageusement remplacée par Ψ avec : Maj $(\varphi\Psi F) = \Psi$ et coût $(\Psi) \leq$ coût (φ).

si le cas (2) se produit, on peut exprimer F' en fonction croissante de φ

$$F' = (A' + \varphi) + B'$$
$$= C\varphi + D$$

autrement dit si

Maj$(\varphi, \Psi, F') = \Psi$ on peut éliminer φ devant Ψ.

Nous ne savons pas, en cours de synthèse, laquelle des 2 éventualités 1 ou 2 se produit.

On ne pourra éliminer φ que s'il existe deux colonnes Ψ et Θ vérifiant :

$$Maj(\varphi, \Psi, F) = \Psi, \qquad coût(\Psi) \leq coût(\varphi)$$

$$Maj(\varphi, \Theta, F') = \Theta, \qquad coût(\Theta) \leq coût(\varphi)$$

REMARQUE :

On constate immédiatement que le critère d'élimination est ici plus strict que dans le cas d'opérateurs croissants : il y aura moins de colonnes éliminées.

7 - MISE EN OEUVRE - METHODES RIGOUREUSES

Un programme rigoureux de synthèse en opérateurs croissants avec une structure arborescente a été écrit par MM. DESCHIZEAUX, LUSTMAN, MARTIN & VIVIER. Nous avons pour des raisons de commodité de programmation, limitée le nombre d'entrées des opérateurs à 5. Les critères d'élimination sont les critères rigoureux 1, 2 et 3.

Programme :

Le programme a été écrit en Algol sur IBM 7044. Une description plus complète est donnée au chapitre programmation.

Résultats :

Les limites d'exploitation de ce programme sont assez faibles. On ne peut guère dépasser 5 variables en moyenne avec des opérateurs croissants ou 4 variables en opérateurs monotones quelconques.

Cette limitation est imposée par le temps de calcul. En cours de synthèse, chaque nouvelle colonne est comparée à toutes les précédentes, ce qui nécessite à partir d'un certain nombre de colonnes, un temps important.

Exemple 1 -

Synthèse en opérateurs Majorité

Fonction incomplète de 8 variables dont la borne inférieure est :

F = CDEG + BDEG + ABCDG + BCEG + ACFG + ABCH + BCEH + ABDH + ACDFH + DEFH

Solution minimale

F = Maj(C,1,2)

1 = Maj(B,E,H)

2 = Maj(D,F,3)

3 = Maj(A,B,G)

Temps : (compilation non comprise : 52 secondes

Exemple 2 -

Opérateur somme et produit (croissants)

Fonction complète : F = ab + bc + ca

Solution : F = a(b+c) + b . c

Coût 4

Temps de calcul : 10 secondes

Exemple 3 -

Opérateur Ni à 2 entrées (monotones)

Fonction complète : F = ab + bc + ca

Solution : F = Ni(Ni(a,b), Ni(c, Ni(Ni(a,a), Ni(b,c))))

Coût 6

Temps d'exécution : 40 secondes

Exemple 4 -

Opérateur Ni a 2 entrées (monotones)

Fonction complète :

F = x ⊕ y ⊕ z

= xyz + xy'z' + x'yz' + x'y'z

Coût obtenu 24

Temps de calcul : 13 minutes

Etude Statistique

Nous pouvons dans des temps moyens d'une minute obtenir des réseaux de coût 8, ce qui correspond approximativement à des fonctions de 4 variables en opérateurs Ni et 8 variables en opérateurs Majorité. Les temps sont fonction exponentielle du nombre de variables et du nombre d'opérateurs utilisés.

Conclusion

Les résultats obtenus par cette méthode justifient l'emploi de programmes heuristiques de synthèse.

Il n'est cependant pas exclu qu'à l'aide de machines mieux adaptées au problème que la 7044, nous puissions traiter des problèmes plus importants.

8 - CRITÈRES D'ELIMINATION APPROCHEE - DISTANCE

Dans ce qui précède, les critères sont rigoureux. L'expérience montrant qu'ils ne permettent pas assez d'élimination, nous avons du rechercher des critères plus puissants.

DEFINITION

La distance entre deux colonnes φ et Ψ est le nombre de composantes par lesquelles φ et Ψ diffèrent.

PROPRIETE

Si Maj $(\varphi, \Psi, F) = \Psi$ alors pour toute composante de rang i, on a :

$$\varphi_i = F_i \Leftrightarrow \Psi_i = F_i$$

donc

$$d(\varphi,F) \geq d(\Psi,F)$$

l'égalité n'ayant lieu que si $\varphi = \Psi$).

CRITERES

Nous n'allons conserver que les colonnes dont la distance à F (ou a F') si on est en opérateurs monotones) est faible, de nombreuses variantes sont possibles :

a) - conserver à chaque itération, une seule colonne : celle dont la distance à la fonction est la plus faible.

b) - conserver les colonnes dont la distance à la fonction est inférieure à certaine valeur définie à l'avance.

c) - conserver les colonnes dont la distance à la fonction est inférieure à une valeur, fonction du coût de la colonne, etc ...

Notons que ces critères sont empiriques, et qu'ils ne se justifient que par la qualité des résultats obtenus.

9 - AMELIORATION PONCTUELLE D'UNE COLONNE PARTICULIERE φ

Les critères rigoureux d'élimination ont l'inconvénient de nécessiter une comparaison des anciennes et nouvelles colonnes. On peut alors chercher à construire des colonnes qui ne seront sûrement pas éliminées, ce qui nous évitera ces comparaisons.

Le principe est de choisir la colonne φ la plus proche de f parmi toutes les colonnes déjà obtenues et de chercher Ψ vérifiant :

$$Maj(\varphi, \psi, F) = \Psi$$

on choisit une composante i ou $\varphi_i \neq F_i$ et on cherche à former Ψ égale à F partout où $\varphi_j = F_j$ et de plus $\Psi_i = F_i$.

En général ce problème admet plusieurs solutions; on choisit alors celle qui se rapproche le plus de F.

10 - MISE EN OEUVRE DES METHODES HEURISTIQUES

De multiples programmes heuristiques ont été écrits; nous en distinguerons deux types dont les résultats sont très différents :

- programmes de synthèse en opérateurs croissants et en particulier Majorité.

- programmes de synthèse en opérateurs monotones et en particulier opérateurs Ni.

(a) Synthèse en opérateur majorité

Ces méthodes développées par Monsieur Lustman sont décrites dans sa thèse. Donnons en simplement les résultats généraux :

Trois programmes ont été écrits qui diffèrent par le test d'élimination, les résultats sont comparables au point de vue du temps de calcul qui est très inférieur à celui nécessaire aux programmes rigoureux.

Une étude statistique nous conduit à :

coût moyen 11	durée moyenne 10 secondes

ces valeurs étant grossièrement indépendantes du nombre de variables du problème. On peut chercher à comparer les coûts obtenus avec les coûts minimaux. Ces coûts minimaux sont en général inconnus. Cependant en étudiant certains cas (en particulier les exemples qu'il est possible de traiter par la méthode précédente) nous pouvons évaluer l'augmentation relative du coût à 15 % environ ce qui est peu en regard du gain de temps obtenu.

EXEMPLE

La fonction traitée précédemment :

F=CDEG + BDEG + ABCDG + BCEG + ACFG + ABCH + BCEH + CEFH + ABDH + ACDFH + DEFH

a) donne comme solution par la méthode "Maj Max"

F = Maj(1, g, 2)
1 = Maj(3, c, d)
2 = Maj(1, f, 4)
3 = Maj(e, h, 5)
4 = Maj(1, a, c)
5 = Maj (b, c, d)

Coût 6
Temps d'exécution : 12 secondes

b) Par la méthode Majorf :

F = Maj(1, 2, c)
1 = Maj(b, e, h)
2 = Maj(3, a, d)
3 = Maj(b, c, d)

Coût minimum
Temps d'exécution : 6 secondes

(exemples tirés de la thèse de Monsieur Lustman.)

(b) Synthèse en opérateur NI

Ces mêmes méthodes se sont avérées plus lentes en opérateurs Ni, en effet, cet opérateur non croissant nécessite d'une part une représentation, en colonne plus lourde, puisque nous pouvons ici réaliser toute fonction booléenne nous devons conserver tous les points de l'espace des variables. La dimension d'une colonne représentative d'une fonction complète de n variables est 2^n. D'autre part, le nombre total de colonnes qu'il est possible de former est beaucoup plus élevé (2^{2^n}); nous sommes alors placés devant le dilemme suivant :

- soit choisir un test d'élimination de colonnes comparables à ceux des méthodes précédentes, le nombre de colonnes intermédiaires formées dans le temps est alors grand.

- soit choisir un test qui permettrait d'éliminer beaucoup plus de colonnes, mais alors nous risquons de trouver un coût très loin du minimum voire de ne pas trouver de solution à l'aide des colonnes conservées.

En essayant de résoudre au mieux le problème, nous avons abouti aux résultats suivants.

EXEMPLE

$$f(xyz) = x \oplus y \oplus z = xyz + xy'z' + x'yz' + x'y'z$$

solution arborescente trouvée :

Coût : 21

durée de calcul : 43 secondes

Coût minimal : 14

d'une manière plus générale nous avons constaté qu'il n'est pas possible de traiter une fonction de 4 variables en moins d'une minute.

Ce résultat, nous a conduit à envisager d'autres méthodes que la composition pour la synthèse en opérateurs non croissants.

C H A P I T R E 2

* * * *

M E T H O D E S PAR F R A C T I O N N E M E N T

A - GENERALITES

1 - INTRODUCTION

Outre leur manque de rapidité, les méthodes par composition ont l'inconvénient de ne pas tenir compte des propriétés algébriques de la fonction à réaliser. Sauf dans le cas des opérateurs croissants, nous n'utilisons pas de renseignements tels que : nombre de monômes de la forme polynomiale, décompositions de la fonction, connexité des points représentatifs sur l'hypercube, etc... Les méthodes que nous allons exposer à présent sont au contraire prévues pour tenir compte des cas particuliers.

2 - OPERATEURS UTILISES

Les méthodes précédentes permettaient théoriquement de traiter les problèmes de synthèse en opérateurs monotones quelconques. Ici par contre nous devrons toujours spécifier les opérateurs utilisés. Nous nous sommes limités strictement aux 4 cas suivants :

- Opérateur U(xyz) = xy + x'z
- Opérateur Majorité (xyz) = xy + xz + yz
- Opérateur Ni à k entrées au plus
- Opérateurs somme et produits.

Le premier de ces opérateurs a été étudié par Monsieur MACHERAS dans sa thèse; le quatrième fait l'objet d'une étude menée actuellement par Monsieur CHEIN. De notre côté nous avons étudié l'opérateur majorité et surtout l'opérateur Ni.

3 - PRINCIPE DES ALGORITHMES

Etant donné un opérateur A à k entrées, et $f(x_1\ x_2\ \ldots\ x_n)$ la fonction à réaliser, on cherche k fonctions intermédiaires notées $f_1, f_2, \ldots, f_k$ vérifiant :

$$f(x_1\ x_2\ \ldots\ x_n) = A(f_1(x_1\ \ldots\ x_n) \ldots f_k(x_1\ \ldots\ x_n))$$

ceci fait, on itère le procédé en supposant que nous avons à faire la synthèse des fonctions intermédiaires $f_1, f_2, \ldots, f_k$. On arrête le procédé quand on arrive à n'avoir que des variables comme fonctions intermédiaires.

REMARQUE :

Ces méthodes conduisent par nature à des réseaux arborescents: une fois le 1er fractionnement effectué, on traite les fonctions $f_1 \ldots f_k$, indépendamment les unes des autres. Nous obtiendrons des réseaux quelconques, dans la 2ème partie de cette étude en considérant que nous avons à faire simultanément la synthèse de $f_1 \ldots f_k$, donc la synthèse d'une fonction générale.

4 - FRACTIONNEMENT

Nous déterminerons les fonctions intermédiaires $f_1 \ldots f_k$ à partir d'une forme minimale particulière de f; par exemple : forme lexicographique irredondante pour les opérateurs U ou base première irredondante pour les opérateurs NI. Nous n'envisagerons pas ici le problème de cette minimisation qui a été traitée théoriquement et qui a donné lieu à divers programmes écrits en particulier par Monsieur BENZAKEN.

REMARQUE :

Nous aurons des méthodes distinctes suivant que les fonctions sont complètes ou incomplètes : dans le cas de fonctions incomplètes nous considérerons en général une forme minimale de chaque borne de la fonction.

B - METHODE DE FRACTIONNEMENT SIMPLE

Cette méthode permet de faire la synthèse d'une fonction f(X) simple et complète.

1 - PRINCIPE

Une théorie antérieure nous permet de trouver les écritures minimales de f(X) en sommes de produits. Nous allons utiliser ces écritures pour trouver des réseaux de coût faible.

La méthode que nous proposons ici est valable pour les opérateurs cités plus haut : U, Majorité, NI, Sommes, Produits.

2 - OPERATION FONDAMENTALE : PARTAGE EN SOMMES

L'opération élémentaire de la méthode est le partage d'une fonction f en k fonctions $f_1 \ldots f_k$ avec :

$$f = f_1 + f_2 + \ldots f_k$$

Nous partirons d'une base de f :

$$f = \sum_{i \in I} m_i \qquad I = \{1, 2, \ldots, p\}$$

Nous déterminerons un k-recouvrement $\{I_1, I_2, \ldots, I_k\}$ de l'ensemble d'indice I :

$$I_1 \cup I_2 \ldots\ldots \cup I_k = I$$

nous poserons :

$$f_j = \sum_{i \in I_j} m_i$$

et nous aurons : $f = \sum_{j=1}^{k} f_j$

Il y a de multiples manières de déterminer ces f_j, nous en décrirons quelques unes par la suite.

3 - OPERATIONS DERIVEES

De cette opération de partage en somme, nous pouvons en déduire:

- le partage en produit qui sera le partage en somme de f*.
- le partage en Ni qui sera le partage en somme de f'.
- le partage en majorité qui reviendra à un partage d'une forme redondante de f.
- le partage en opérateurs U qui se ramènera à un partage en sommes d'une forme lexicographique de f.

4 - PARTAGES OPTIMAUX

Il est très délicat de définir quel sera la partage conduisant à un réseau de coût minimal. Nous pouvons cependant donner déjà une idée des méthodes utilisées : On obtiendra des réseaux de coût raisonnable en déterminant des partitions de bases premières irrédondantes.

5 - ITERATION DES PARTAGES

Ayant déterminé un partage de f suivant un opérateur, c'est-à-dire ayant trouvé $f_1 \ldots f_k$ vérifiant :

$$f = (f_1, \ldots f_k).$$

On peut itérer la méthode en fractionnant successivement $f_1 \ldots f_k$ suivant éventuellement d'autres opérateurs, et, poursuivant jusqu'à trouver comme fonctions intermédiaires, des variables.

Exemple :

Synthèse de la fonction f = ab + bc + ca en opérateurs Ni à 2 entrées. Le fractionnement de f en opérateurs Ni à 2 entrées revient à fractionner f' en opérateurs sommes à 2 entrées.

$$f' = a'b' + b'c' + c'a'$$

On posera par exemple :

$$f_1 = a'b' + b'c' \qquad f_2 = c'a' = Ni(a,b)$$

On calcul de même f_1' :

$$f_1' = b + ac$$

On posera encore :

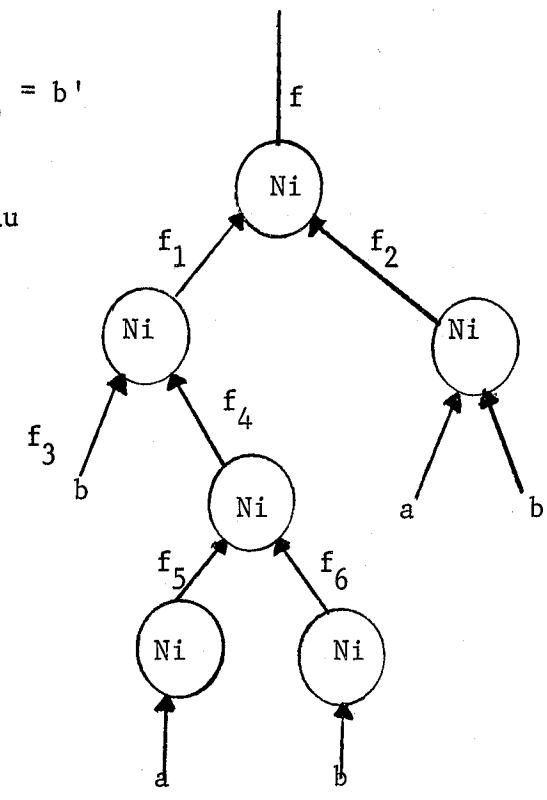
$$f_3 = b, \qquad f_4 = ac.$$

On calcule f_4' :

$$f_4' = a' + c'$$

$$f_5 = a' \qquad f_6 = b'$$

ce qui nous conduit en définitive au réseau ci-contre :



6 - PROBLEME DE LA CONVERGENCE

Il est certain qu'on ne peut pas toujours obtenir un réseau par ce procédé :

ⓐ Les opérateurs doivent permettre la synthèse de la fonction. Nous ne traiterons pas ici ce problème qui est résolu par la théorie des familles due à Post et développée par Monsieur Kuntzmann.

ⓑ Il est nécessaire que le fractionnement soit strict , c'est-à-dire qu'aucune des fonctions f_i ne soit égale à f.

Remarquons que cette condition n'est pas assurée par le critère de partage strict de l'ensemble des monômes, car la base peut être redondante.

Exemple :

$$f = ax + bx' + ab$$

fonction à réaliser en opérateurs Ni à 2 entrées. La méthode nous conduit par exemple à :

$f' = a'x + b'x' + a'b'$

$f_1 = a'x + b'x'$ $\qquad$ $f_2 = a'b' = Ni(a,b)$

$f_1' = ax + bx' + ab$

$f_3 = ax + bx'$ $\qquad$ $f_4 = ab$

$f_3' = a'x + b'x' + a'b'$

.....

etc ...

On constate qu'avec les partages choisis, notre méthode ne converge pas.

7 - CONDITION NECESSAIRE ET CONVERGENCE

Pour qu'il y ait convergence quelque soit le partage strict effectué il faut que la base choisie pour le partage en somme soit irredondante.

En effet, si elle ne l'était pas il existerait un partage qui regroupe les monômes redondant d'une part, et une base irredondante d'autre part, et on aurait éventuellement une suite stationnaire de fonctions.

8 - PREMIERE CONDITION SUFFISANTE DE CONVERGENCE

La condition précédente permet simplement d'affirmer que pour tout opérateur du réseau, la sortie n'est pas égale à une entrée. Elle ne permet pas de dire si, à partir d'une fonction f, et après plusieurs fractionnements on ne retrouve pas f.

Nous pouvons dans le cas d'opérateurs sommes et produits donner une condition suffisante :

Condition :

Si de f et de f*, on partage celui dont le coût en somme de produits est le plus faible, la méthode converge.

En effet, soit C le coût de la moins chère. Si nous fractionnons la base de coût C de cette fonction en f_1, f_2, ..., f_k, chacun des f_i aura un coût strictement inférieur à C. Autrement dit à chaque itération de la méthode, le coût diminue strictement. Or ce coût ne pouvant être négatif, les fractionnements successifs finissent par le coût 0, c'est-à-dire les variables.

Ceci est vrai quelque soit la façon dont on évalue le coût.

En particulier nous pourrons envisager le coût comme :

- nombre de monômes
- nombre de lettres au total
- nombre de lettres sous forme directe + 2 X nombre de lettres accentuées, etc ...

Nous utiliserons ces coûts pour obtenir des convergences plus ou moins rapides et tenir compte de données technologiques précises.

Remarque :

Le critère choisi permet par ailleurs d'affirmer que le coût total obtenu est inférieur au coût en somme de produits.

9 - DEUXIEME CONDITION SUFFISANTE DE CONVERGENCE

Si à chaque fractionnement les fonctions obtenues dépendant au minimum d'une variable de moins que la fonction de départ f, la méthode converge.

Cette condition évidente sera utilisée en particulier en opérateur U où on écrit à chaque pas :

$$f = x_i \, f_{(1)} + x_i' \, f_{(0)}$$

$$= x_i \, f_1 + x_i' \, f_2$$

f_1 et f_2 dépendent d'une variable de moins que f.

10 - CHOIX DES FRACTIONNEMENT

Les critères précédents nous laissent une grande liberté dans le choix du partage. Nous allons alors utiliser des méthodes heuristiques pour déterminer quel est le meilleur fractionnement.

(a) Fractionnement équilibré

Nous chercherons à déterminer le partage de manière à ce que $f_1 \ldots f_k$ aient des coûts équivalents. Pour ceci nous prendrons la règle suivante :

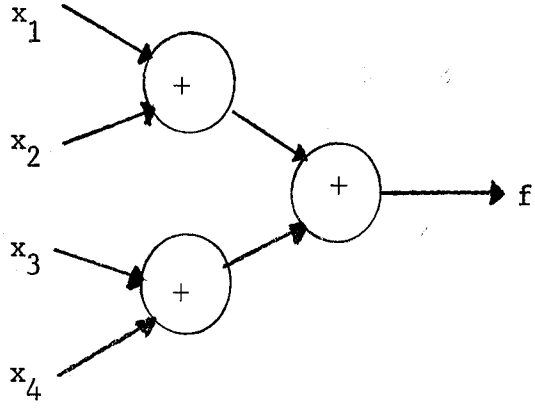
$\forall_i$, f_i comporte à peu près $\frac{P}{k}$ monômes premiers de f. (ou p est le nombre de monômes de la base considérée de f).

Ce fractionnement nous permet de minimiser grossièrement le nombre de couches du réseau.

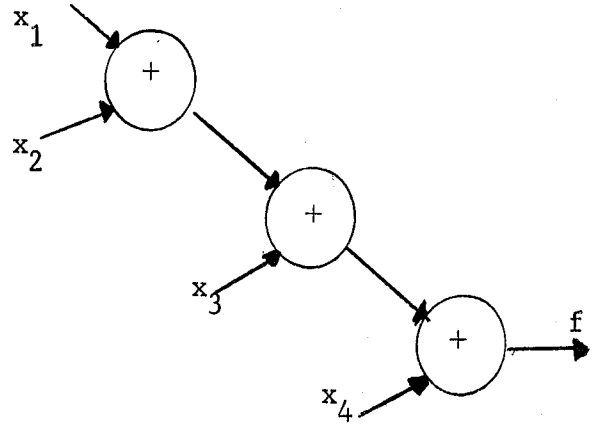
Exemple :

$$f = x_1 + x_2 + x_3 + x_4$$

Cette fonction peut être réalisée en sommes par les 2 réseaux suivants :



partage équilibré



partage déséquilibré

le deuxième réseau est de coût égal au premier mais plus "lent" puisque l'information traverse 3 couches au lieu de 2.

ⓑ Fractionnement lexicographique

Nous serons souvent amenés à regrouper dans un même f_i tous les monômes contenant une lettre x. En particulier, on cherchera à mettre en facteur les lettres les plus fréquentes.

Dans le cas de l'opérateur U on est amené à mettre dans une composante f_1 les monômes contenant x dans f_2 les monômes contenant x'. Tous les monômes devant apparaître, on est donc amené à envisager une forme lexicographique.

ⓒ Fractionnement en composantes connexes

Cette méthode repose sur le théorème suivant du à Seshu et Reed.

THEOREME : En opérateurs + et X à 2 entrées

> Si $f(X,Y) = h(X) + g(Y)$ avec $X \cap Y = 0$
> alors on obtient un réseau minimum de f à partir de réseaux minimaux de f et de g.

Ce théorème suppose que le coût est égal en nombre de lettres figurant dans l'expression en sommes et produits.

Dans ce cas, en effet, si α et β sont les coûts minimaux de h et g et si on suppose une forme moins chère $\varphi(X,Y)$ où les lettres de X et Y sont en nombres α' et β' avec $\alpha+\beta > \alpha'+\beta'$, alors un des deux, α' par exemple est inférieur à α.

Si on considère une valeur Y_o de Y qui annule g(Y) on a : $\varphi(X,Y_o) = h(X)$ mais $\varphi(X,Y_o)$ est une expression de coût au plus α' puisque toutes les lettres de Y sont remplacées par des constantes, donc il existe une expression de h(X) de coût plus petit que α ce qui est impossible.

Remarque :

Le théorème n'est pas exact en opérateurs autres que ceux précisés, par exemple :

- en opérateur + et X à 3 entrées - si on remplace y par une constante dans un tel opérateur, on ne peut pas toujours supprimer cet opérateur.

- en opérateur Ni à 2 entrées, si on remplace y par o, on ne peut pas supprimer l'opérateur qui doit complémenter x.

Conséquence :

(1) - En opérateur sommes et produits, on recherchera systématiquement les décompositions de f en sommes et pour ceci on exprimera toujours f sous forme de base première, puisque c'est sous cette forme que les décompositions sont évidentes.

(2) - S'il n'y a pas de telles décompositions, nous recherchons des décompositions non disjointes de la forme :

$$f(XYZ) = h(X,Z) + g(Y,Z)$$

où les variables Z apparaîssent effectivement dans deux fonctions.

En désignant par $N_{(X)}$, $N_{(Y)}$, $N_{(Z)}$ le nombre de variables de X, Y, Z le coût de l'expression est :

$$C \geq N_{(X)} + N_{(Y)} + 2\,N_{(Z)}$$

Nous rechercherons alors les décompositions ou le nombre de lettres de la partie commune est minimum.

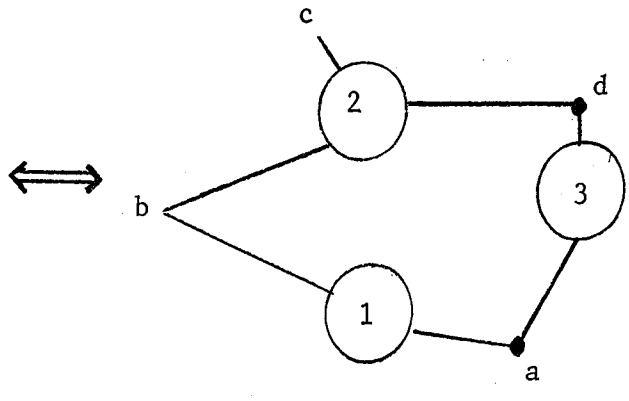
Recherche des décompositions - Réseau associé

Cette recherche se fait de façon simple en associant à chaque base de la fonction un réseau d'articulation la représentant, de la façon suivante :

Dans ce réseau les étoiles seront les monômes et les noeuds les variables. Une variable est associée à un noeud unique. Un monôme est représenté par étoile dont les branches sont reliées aux noeuds correspondants aux lettres du monôme.

Exemple :

$f = ab + bcd + ad$
$\quad\;\; 1 \quad 2 \quad 3$



Remarque :

Il y a correspondance biunivoque entre le réseau et la base de f.

PROPRIETE 1

Rechercher les décompositions disjointes en sommes :

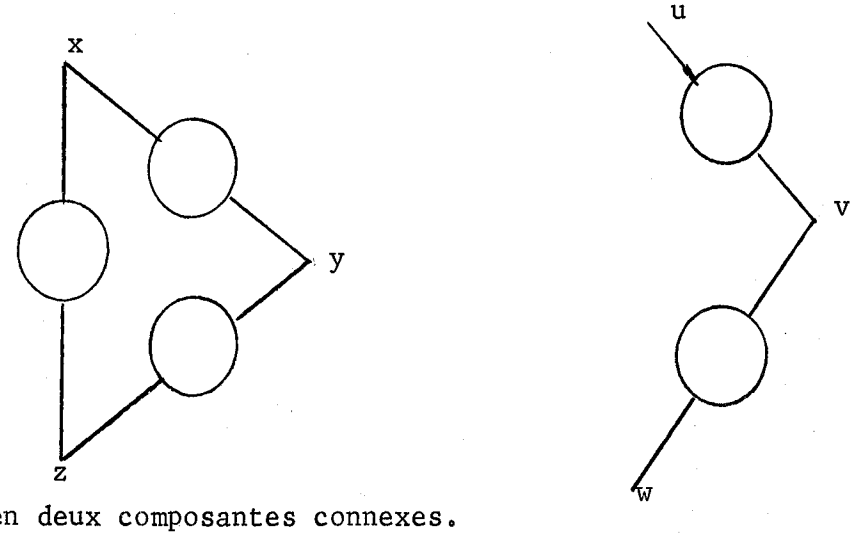
$$f = h(X) + g(Y)$$

revient à chercher si le réseau associé à une base première est connexe.

Exemple :

$$f = xy + yz + zx + uv + vw$$

dont le graphe associé est



qui a bien deux composantes connexes.

PROPRIETE 2

Rechercher les décompositions non disjointes de f de la forme :

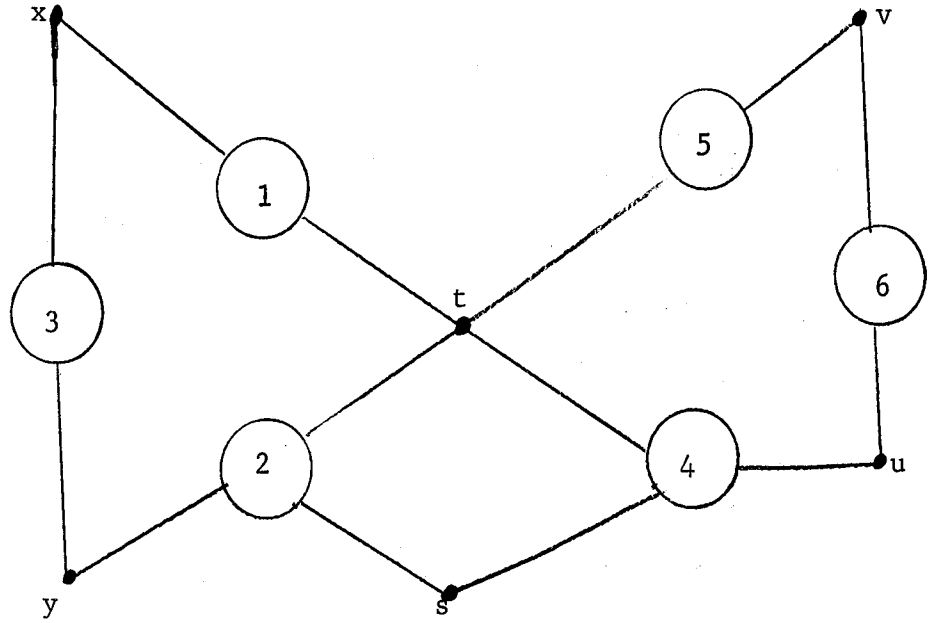
$$f(X,Y,Z) = h(X,Z) + g(Y,Z)$$

revient à chercher à partager le réseau associé à une base première de f en 2 parties, en dédoublant les noeuds associés aux variables $z \in Z$.

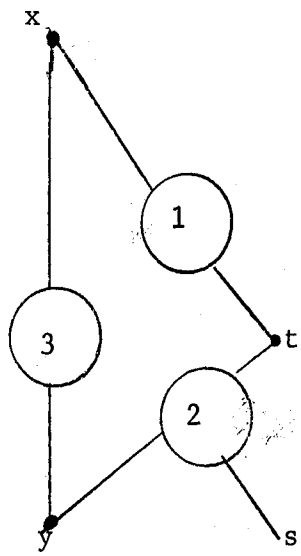
Exemple :

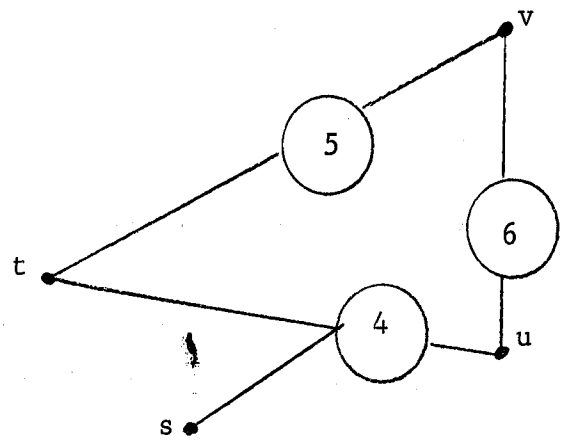
$$f = xt + yts + xy + stu + tv + uv$$

dont le graphe associé est :



en séparant en 2 les noeuds t et s on obtient :





$f_1 = xy + xt + yts$

$f_2 = stu + tv + uv$

Définition :

On appelle fracture du réseau l'ensemble des noeuds qui sont dédoublés pour disconnecter en deux parties appelées hypercoupures.

Le problème posé revient donc à rechercher les fractures minimales du réseau associé. Ce problème est traité en tant que problème de graphes, il est actuellement étudié par Monsieur Chein, qui a en particulier écrit des programmes permettant d'obtenir les coupures minimales.

11 - FRACTIONNEMENT EN OPERATEURS Ni

Les méthodes de fractionnement ont été étudiées principalement en vue de la synthèse en opérateurs Ni, pour lesquels la méthode par composition s'est avérée inefficace.

L'algorithme utilisé est le suivant :

(a) Calcul du complément f' de la fonction à étudier sous forme de base première irredondante.

b Recherche de la moins coûteuse des bases minimales de f et de f'. (le coût pouvant être interprété de différentes manières).

(c) Si la base la moins coûteuse est f', on la fractionne suivant un des procédés décrits précédemment.

Si la base la moins coûteuse est f, on ne fractionne pas, c'est-à-dire que l'opérateur Ni utilisé est à 1 seule entrée.

(d) Pour chaque fonction intermédiaire f_i obtenue (il peut n'y en avoir qu'une) on étudie :

- s'il existe une variable x_j égale à f_i
- s'il n'en existe pas on itère le procédé en traitant la fonction f_i considérée comme f.

Remarques :

a) - Nous obtiendrons des méthodes distinctes en utilisant diverses méthodes de fractionnement. Toutes les méthodes décrites précédemment peuvent être utilisées.

b) - La partie la plus longue dans cet algorithme est la détermination des bases premières irredondantes des fonctions. Nous remarquons que si la fonction donnée est croissante, alors par notre procédé, nous n'obtenons que des fonctions croissantes. Il est inutile alors, ayant une base première, de rechercher une base irredondante puisque la base première est unique.

Par ailleurs, nous sommes amenés à calculer le dual f* de la fonction f traitée. Ce dual calculé par la formule de Morgan est obtenu directement sous forme de base première (théorème du à Monsieur Kuntzmann) nous n'avons donc pas besoin d'un calcul supplémentaire.

Un programme a été écrit en langage Algol sur IBM 7044. Les caractéristiques en sont les suivants :

A) - Longueur de programme : 4600 unités synthaxiques.

B) - Procédures utilisées (voir annexe)

a) procédure en M.A.P. pour le traitement logique de l'information infaisable directement en Algol.

b) procédures de base en Algol : Lecture et écriture de fonctions sous forme polynomiale, calcul des duales, calcul des bases premières, etc ...

C) - Temps de compilation : 1 minute 20 secondes.

D) - Nombre maximum de variables des fonctions traitables (en théorie) : 18

E) - Nombre d'entrées des opérateurs Ni : non limité pas le programme

F) - Complexité des problèmes traitables.

Indépendamment du temps de calcul, on est limité en machine par l'encombrement des formes polynomiales envisagées : à chaque instant la machine doit pouvoir contenir une base complète de la fonction et de son complément. On peut estimer à environ 5000 monômes premiers les dimensions maximum des fonctions traitables. En fait, cette valeur ne sera jamais envisageable pour des problèmes de durée.

G - Temps de calcul.

La durée du calcul croît à peu près linéairement avec le coût des solutions obtenues indépendamment du nombre de variables (contrairement aux méthodes par composition ou le temps croît exponentiellement avec le coût).

Exemples :

- fonctions de coût 6 - temps moyen 1 seconde
- fonctions de coût 15 - temps moyen 2 secondes
- fonctions de coût 50 - temps moyen 6 secondes

Remarque :

Ces temps bien que très inférieurs à ceux obtenus en méthode par composition sont encore relativement grands, si l'on pense qu'en moyenne le coût obtenu est lui-même fonction exponentielle du nombre de variables.

Exemple n° 1

Fonction à réaliser en opérateurs Ni à 3 entrées au plus :

F = BCDE'F' + B'CD E'F' + AB'C'D'EF + AB'CE'F' + ABC'E'F + A'BC'DE + ABC'DE'

Solution en notation préfixée :

3 3 2 2 B C 2 1 B A 3 2 C E 2 B C 1 D 2 1 E 1 A 3 3 B D 2 1 C 1 F 3 1 B 2 D F 2 1 C 1 F 3 1 2 1 C 1 A B 3 2 A C 1 E 3 B 1 F C .

Réseau arborescent de coût 34

Temps : 6 secondes

Réseau minimum non arborescent obtenu :

Coût : 25

Exemple n° 2

Fonctions de 8 variables ayant 55 monômes dans sa base première minimale en opérateur Ni :

à 4 entrées	coût	161	Temps	61 secondes
5 -		163		47 -
6 -		126		44 -
7 -		120		33 -
8 -		92		27 -

Résultats généraux :

D'une façon générale, on peut dire que les coûts sont raisonnables. Dans les rares cas où nous connaissons le minimum rigoureux (c'est-à-dire les fonctions dont le coût ne dépasse par 10), nous constatons qu'en général le minimum est trouvé, et qu'au maximum l'écart est de 10 %.

H) - Forme particulière des résultats.

(1) On peut en général passer à une forme non arborescente du réseau, en réunissant certains noeuds. En particulier, les compléments des variables (1 ni pour chaque occurence) sont très fréquents :

Exemple :

$F = W'XZ + V'WZ' + VWX'Z + V'W'X' + XY + WX'Y' + X'Y'Z'$

coût arborescent obtenu : 31

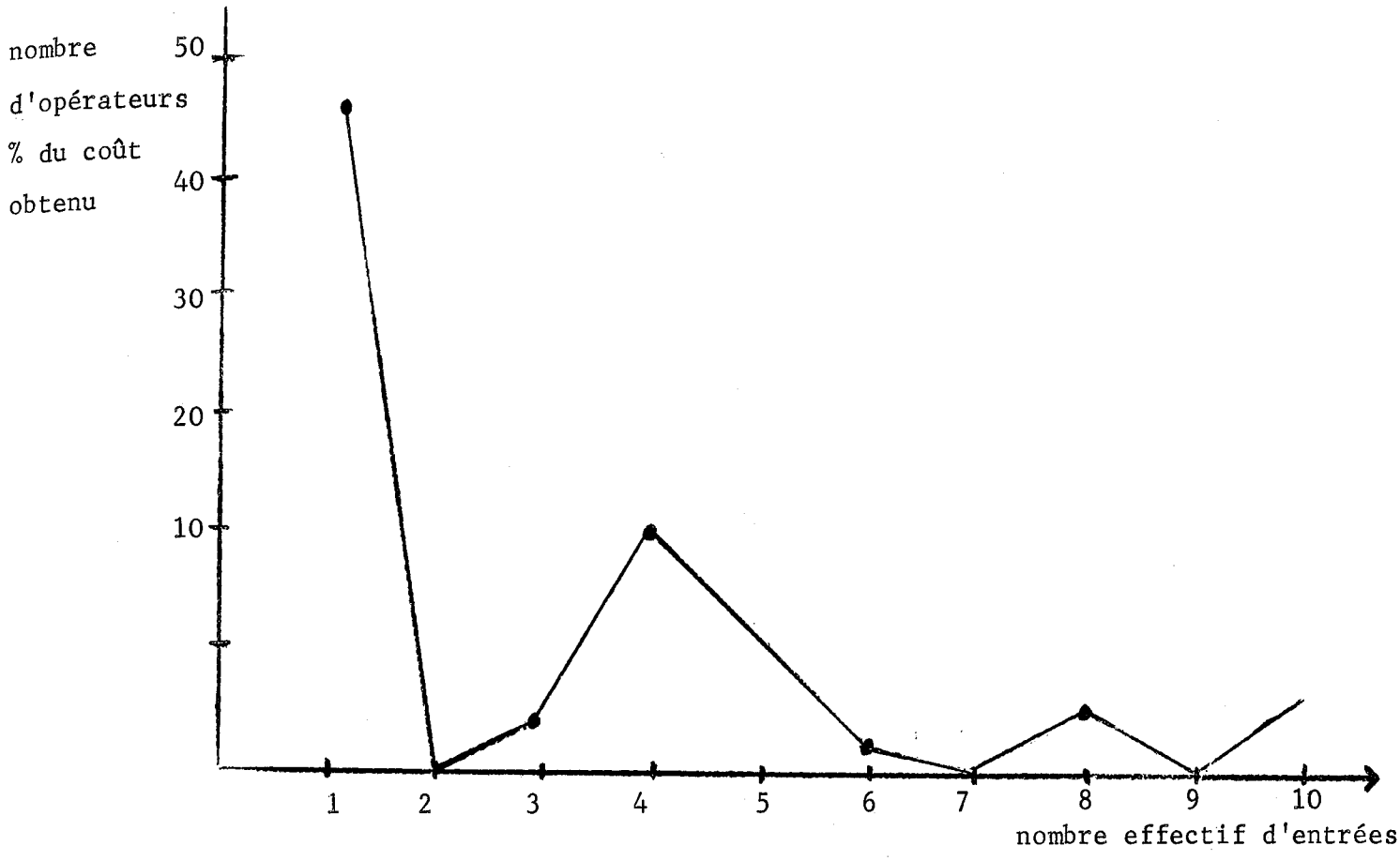
Dans cette solution :

Z' est fabriqué 4 fois
X' 4
W' 3
V' 2

on peut gagner 9 opérateurs en réunissant ces noeuds. Nous avons remarqué par ailleurs que des fonctions intermédiaires de coût supérieur peuvent rarement être réutilisées.

(2) Dans le cas d'opérateurs à nombre d'entrées k borné, nous constations les réseaux obtenus n'utilisent jamais toutes les possiblités d'entrées :

- sur 50 fonctions de 6 variables étudiées par les programmes, avec des opérateurs Ni à 10 entrées ou plus, nous n'avons jamais utilisé de Ni à 2 entrées ni à 7. Nous avons par ailleurs tracé la courbe statistique suivante :



Opérateur à au plus 10 entrées

Exemple :

$$F = XWV' + U + Z'W + Y'W + YX'W'V + Z'X'V + ZYW'V'$$

Nombre maximum d'entrées	Nombre effectif									
	1	2	3	4	5	6	7	8	9	10
4	13	10	1	5						
5	17	4	4	4	2					
6	19	5	4	4	0	2				
7	15	4	1	4	1	1	1			
8	14	0	1	4	3	0	0	1		
9	14	0	1	4	3	0	0	1	0	
10	14	0	1	4	3	0	0	1	0	

pour des opérateurs de "fan in" 8, 9 ou 10, on n'utilise pas les opérateurs de 2, 6 et 7 entrées effectives.

Ce résultat statistique (qui par ailleurs n'a pas été démontré) est intéressant pour le constructeur de circuits. Il signifie en effet, qu'on peut prévoir une certaine standardisation des opérateurs Ni. Par exemple, dans le cas de 6 variables, on n'utilisera que des opérateurs Ni à 1, 4, 5, 8, 10 entrées.

I) - Structure particulière : Réseau sans Aléa

La grande souplesse des méthodes de fractionnement, nous permet sans difficulté d'aborder les problèmes de sécurité et en particulier celui de la suppression des aléas statiques dans un réseau.

Définition :

Soit R un réseau réalisant une certaine fonction f(xX).
Soit X_0 une valeur de X telle que $f(x,X_0) = f(x,X_0) =$.

On dit qu'il y a aléa si au cours du changement de valeur de la variable x, la sortie du réseau prends transitoirement la valeur .

THEOREME :

Soit $f = Ni(f_1, f_2)$

f_1 et f_2 vérifiant les hypothèses :

a) f_1 et f_2 sont sans aléa,

b) toutes les fois que f_1 contient un monôme x m_1 et f_2, x'm_2 alors, soit f_1, soit f_2 contient leur consensus m_1 m_2 .

La fonction f est alors elle-même sans aléa.

Remarque :

On n'impose pas la présence du consensus de deux monômes d'une même fonction.

En effet :

(a) Montrons qu'il ne peut y avoir de fausse valeur 1 sur f(xX)

$$f'(xX) = f_1(xX) + f_2(xX)$$

Si f'(xX) prenait momentanément la valeur 1 cela signifirait que l'une au moins des 2 fonctions $f_1(xX)$ ou $f_2(xX)$ p r e n d momentanément la valeur 1 ce qui est exclu par l'hypothèse (a) .

(b) Montrons qu'il ne peut y avoir de fausse valeur 1.

Comme précédemment on ne peut avoir aléa sur f_1 ni sur f_2 mais il est possible de trouver X tel que :

$$f_1(xX) = 0 \qquad f_2(xX) = 1$$
$$f_1(x'X) = 1 \qquad f_2(x'X) = 0$$

Or si $f_1(x'X) = 1$ cela entraîne l'existence d'un monôme $m_{1(X)}x' = 1$ et si $f_2(xX) = 1$ cela entraîne l'existence de $m_{2(X)} \subset f_2(xX)$ et $m_2(X) \cdot x = 1$.

D'après l'hypothèse (b) leur consensus $m_{1(X)} \cdot m_{2(X)} = 1$ est présent par exemple dans $f_1(xX)$; donc on ne peut avoir $f_1(xX) = 0$; il y a contradiction, les deux fonctions f_1 et f_2 ne peuvent changer de valeur simultanément.

Corollaire :

Si on impose les hypothèses (a) et (b) à chaque pas de la méthode de fractionnement, on obtient un réseau sans aléa. En effet, l'hypothèse est toujours automatiquement vérifiée pour le 1^{er} étage du réseau qui est constitué par des variables.

Application :

Cet algorithme a été programmé et donne de bons résultats. L'obtention de réseaux sans aléa se fait avec une augmentation du coût d'environ 1,5 en moyenne.

Exemple :

$$F = A'BC'D + BCF' + A'D'E + B'D'E'$$

Une méthode normale a donné une solution de coût 16.
Une méthode de synthèse sans aléa a donné une solution de coût 23.

Or si on compare avec une méthode équivalente en double couche, on obtient en somme de produits :

- structure minimale : 13 lettres
- structure sans aléa :
  (base première complète) coût : 33 lettres

l'augmentation du coût est plus faible qu'en réseaux 2 couches.

REMARQUES :

Pour des raisons de simplification, nous pourrons remplacer l'hypothèse (b) par l'hypothèse (b') . Toutes les fois que deux monômes de f ont un consensus, ce consensus est présent dans une ou deux fonctions. Cette condition englobe la condition (b). Elle présente le gros avantage de supprimer complètement l'opération très lourde de base irredondante. La synthèse de réseaux sans aléa apparaît alors comme une simplification de la méthode normale !

11 - MISE EN OEUVRE DE CES METHODES EN OPERATEURS SOMMES ET PRODUITS

L'algorithme utilisé est le suivant :

(a) soit f la fonction à réaliser : on calcule f et f* sous forme de base première irredondante.

(b) On recherche quelle est de ces deux bases la moins coûteuse.

(c) On fractionne la base la moins coûteuse suivant un procédé quelconque; si on a fractionné f on a utilisé un opérateur +, sinon on a utilisé un opérateur X.

(d) On envisage successivement chacun des f_i :
- on regarde si f_i est une variable
- si f_i n'est pas une variable, on itère le procédé en (a) en remplaçant f par f_i.

Remarque :

La comparaison du coût de f et f* en vue de déterminer l'opérateur à utiliser peut être omise, un nombre borné de fois sans nuire à la convergence.

En particulier, on ne fera pas cette comparaison si une décomposition en sommes est trouvée. La convergence est assurée, puisque chaque

fonction d'un partage respectant cette décomposition, dépend de moins de variables que la fonction de départ.

Caractéristiques des programmes

Le programme écrit en Algol 60 par Monsieur Chein sur 7044 comporte 8000 unités syntaxiques environ.(La durée de compilation en est de 4 minutes).

Trois types de procédures sont utilisées :

- Procéduresde traitement logique de l'information en M.A.P. pour les opérations infaisables directement en Algol.
- Procédures de traitement de fonctions booléennes : dualisation, lecture, écriture, etc ...
- Procédures de traitement de réseaux spécifiques de la méthode, notamment :
  - recherche des composantes connexes
  - recherche des hypercoupures d'un réseau
  - décomposition d'un réseau suivant une fracture minimale, ou en deux sous-réseaux connexes ayant même nombre d'éléments.
  - écriture des fonctions en opérateurs + et X.

Possibilités

Le programme écrit actuellement traite uniquement les fonctions croissantes.

Le nombre de variables est limité à 36.

Le nombre de monômes est limité par la forme du réseau associé à la fonction. Dans certains cas (réseau dit complet) s'il y a un monôme, nous devons envisager 2^{m-1} fractures). Le nombre de fractures à envisager est d'ailleurs très inférieur en général.

Résultats :

Il a été envisagé trois variantes de la méthode.

Méthode 1 -

Partage suivant la fracture minimale

Méthode 2 -

Partage suivant une fracture telle que les deux parties obtenues aient même nombre d'éléments.

Méthode 3 -

Partage suivant la fracture minimale si le rapport des nombres de monômes des deux parties obtenues est compris entre $\frac{1}{3}$ et 3. Sinon : partage suivant la méthode 2.

Essais :

Les 3 méthodes ont été essayées sur des fonctions de 6, 7 et 8 variables. Dans le cas de 6, on a pu comparer avec le coût minimum :

(a) Ecart moyen relatif par rapport au coût en somme de produits (2 couches).

Méthode	6 variables	7 variables	8 variables
1	31 %	40 %	38 %
2	34 %	42 %	43 %
3	35 %	41 %	42 %

on constate que pour les 3 méthodes les résultats sont meilleurs pour un plus grand nombre de variables.

ⓑ Temps moyen de calcul

Méthode	6 variables	7 variables	8 variables
1	6,3 secondes	8,6 secondes	14,4 secondes
2	3,3 secondes	4,4 secondes	8,2 secondes
3	3,4 secondes	4,5 secondes	8,5 secondes

On constate que les méthodes 2 et 3, tout en donnant des coûts plus faibles sont plus rapides.

ⓒ Ecart par rapport au coût minimum : pour 6 variables

Méthode	Ecart
1	13 %
2	8 %
3	5, 7 %

Exemple :

Fonction de 8 variables :

F = adg + abcdf + bcefi + cegi + abefi + abgi + aegi + abcdfi + acgi + degi +
+ abdef + abdef + bdefi

de coût 53

La solution trouvée est :
(f+g) (a+e) (i+ad) (b+g) (c+(e+g)(d+a(b+e)))

de coût 16

Le temps de calcul est de 15 secondes.(Cette solution est arborescente).

Conclusion :

Nous avons ici une méthode des synthèses très efficace puisqu'en des temps courts on arrive à des écarts par rapport au minimum de 6 % en moyenne (Une méthode par composition donnant le minimum rigoureux est environ 20 fois plus lente).

Cette méthode n'a pas encore été envisagée pour des fonctions non croissantes outre les difficultés de programmation, nous rencontrons dans ce problème la difficulté du choix des bases premières à fractionner. Ce problème n'est pas encore résolu.

12 - FRACTIONNEMENT EN OPERATEUR MAJORITE

Principe :

Le principe est ici très simple. Soit $f = \sum_{i:I} m_i$ une fonction impaire.

Nous allons fractionner f en 3 sommes disjointes :

$$\varphi_1 = \sum_{i \in I_1} m_i, \quad \varphi_2 = \sum_{i \in I_2} m_i, \quad \varphi_3 = \sum_{i \in I_3} m_i$$

avec $I_1 \cup I_2 \cup I_3 = I$.

Ce fractionnement se fera suivant les méthodes habituelles. Nous posons ensuite :

$$f_1 = \varphi_1 + \varphi_2$$

$$f_2 = \varphi_2 + \varphi_3$$

$$f_3 = \varphi_1 + \varphi_3$$

Nous remarquons alors que tout monôme de f est dans deux fonctions f_i donc :

$$\boxed{f = \text{Majorité } (f_1, f_2, f_3)}$$

et que d'autre part, chaque f_i est sous-impaire.

Comme précédemment nous fractionnerons une base première minimale.

On peut itérer le procédé tant qu'il reste au moins trois monômes dans les fonctions intermédiaires.

A la fin du procédé, nous avons obtenu un réseau R d'opérateur majorité dont toutes les entrées f_i sont sous-impaires et ont 2 monômes au plus; nous utilisons alors un théorème démontré par Monsieur Lustman pour constater que 2 tels monômes dans une fonction sous-impaire ont une lettre en commun x_i. Nous pouvons alors écrire pour tout i

$$f_i = x_i \cdot g_i$$

(la propriété est évidente pour les f_i réduits à un monôme).

R est un réseau d'opérateurs croissants donc si on remplace à l'entrée les f_i par les fonctions supérieures x_i, on obtient une fonction F supérieure ou égale à f.

Mais F étant réalisée à l'aide d'opérateurs majorité à partir des variables est impaire. Deux fonctions f et F impaires ne pouvant être comparables que si elles sont égales, le réseau R réalise la fonction cherchée.

Remarque :

Nous pourrons arrêter une itération si on trouve un f_i (ayant éventuellement plus de 2 monômes), mais contenant une lettre x_i en facteur, ce qui donnera un coût moindre.

Exemple :

$$F = xyz + xy'z' + x'yz' + x'y'z$$

on pose :

$$f_1 = xyz + xy'z' + x'yz'$$
$$f_2 = xy'z' + x'yz' + x'y'z$$
$$f_3 = xyz + x'y'z = z(xy + x'y')$$

tout monôme de f et dans 2 fonctions f_i, donc :

$$F = \text{Majorité } (f_1, f_2, f_3)$$

en itérant, on obtient :

$$f_{11} = xyz + xy'z' = x(yz + y'z')$$
$$f_{12} = xyz + x'yz' = y(xz + x'z')$$
$$f_{13} = xy'z' + x'yz' = z'(xy' + yx')$$

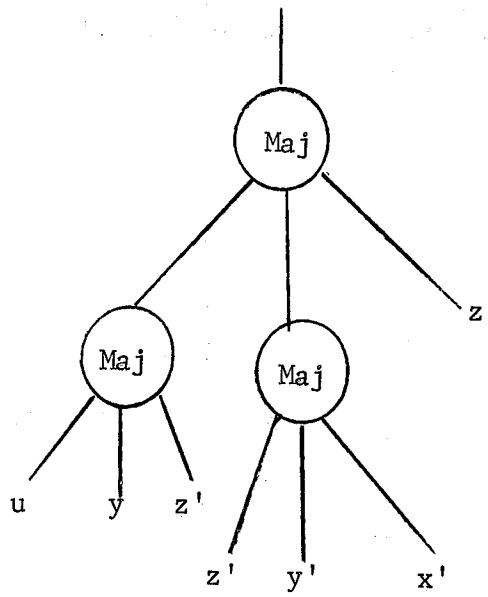
et

$$f_{21} = xy'z' + x'yz' = z'(xy'z' + x'y)$$
$$f_{22} = xy'z' + x'y'z = y'(xz' + x'z)$$
$$f_{23} = x'yz' + x'y'z = x'(yz' + y'z)$$

alors en remplaçant :

f_3 par z
f_{11} par x
f_{12} par y
f_{13} par z'
f_{21} par z'
f_{22} par y'
f_{23} par x'

on obtient le réseau suivant :



ou encore l'expression :

$$x \oplus y \oplus z = Maj(z, Maj(xyz'), Maj(x'y'z'))$$

expression qui est <u>minimale</u>, puisque toute expression à 2 opérateurs aurait seulement 5 entrées libres donc une des variables x, y ou z ne serait présente que sous une seule forme, ce qui est impossible puisque la fonction $x \oplus y \oplus z$ n'est monotone pour aucune variable.

Remarques :

Cet algorithme d'ailleurs simple à utiliser à la main n'a pas été programmé.

Dans la plupart des problèmes traités, nous avons obtenus des coûts comparables à ceux obtenus par des méthodes de composition.

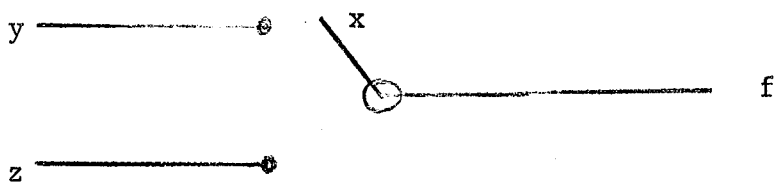
14 - METHODE DE FRATIONNEMENT - OPERATEUR U

Cet opérateur a été étudié par Monsieur Macheras.

On désigne par U un opérateur à trois entrées réalisant la fonction :

$$u(xgz) = xy + x'z$$

Une interprétation physique simple de cet opérateur est donné par le schéma suivant :



où X est la variable de commande d'un commutateur à deux positions mettant en contact une extrémité f avec l'une des deux extrémités y,z; on peut convenir par exemple :

si x = 1 f = x

si x = 0, f = z .

et on a bien

$$f = xy + x'z$$

Toute fonction booléenne peut être réalisée par un schéma n'utilisant que des opérateurs U, à condition de se donner les variables simples et les constantes 0 et 1. En effet, soit f(X) cette fonction; si x est une variable de X et Y l'ensemble des autres variables, on a les identités suivantes :

$$\begin{aligned} f(X) &= f(x,Y) \\ &= x\, f(1,Y) + x' f(0,Y) \\ &= U[x, f(1,Y), f(0,Y)] \quad ; \end{aligned}$$

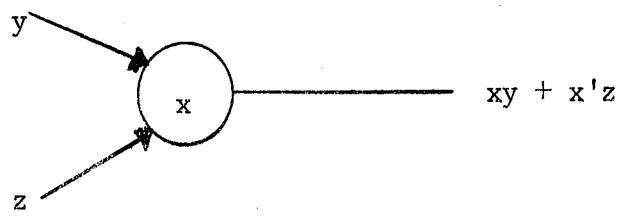
on peut poursuivre cette décomposition sur f(1,Y) et f(0,Y), et ainsi de suite jusqu'à n'avoir plus que des constantes : 0 ou 1 (le nombre maximum d'opérations U à effectuer est $1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1$).

On remarquera que la démonstration se borne à utiliser l'opérateur sous la forme restreinte :

$$U(x, Y, Z) = xY + x'Z,$$

où x est une variable simple et Y et Z des fonctions. Nous nous limiterons à cet usage de l'opérateur que nous noterons Ux.

On représentera un opérateur Ux par un cercle à l'intérieur duquel est inscrit le nom de la variable x, Y étant l'entrée supérieure et Z l'entrée inférieure :



DEFINITION :

On appellera "RESEAU U" un réseau sans boucles de noeuds fonctionnels constitués par des opérateurs Ux, les seules entrées indépendantes du réseau étant 0 et 1.

Ainsi la fonction :

$$f = a(b+b'c') + a'dc'$$

peut être réalisée par le réseau U ci-dessous (figure 1).

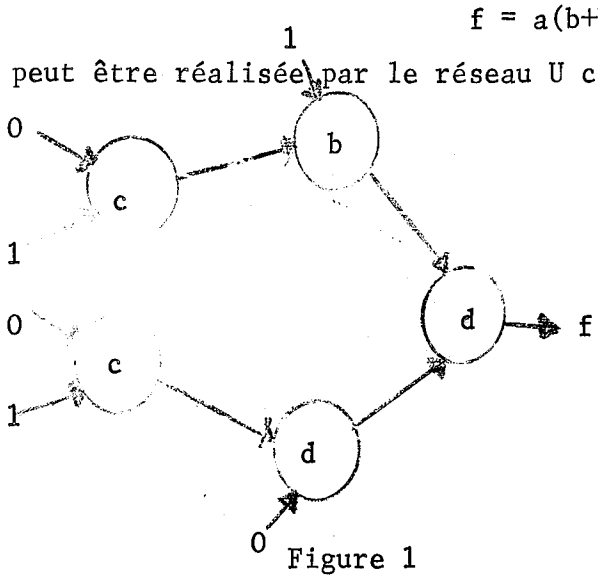


Figure 1

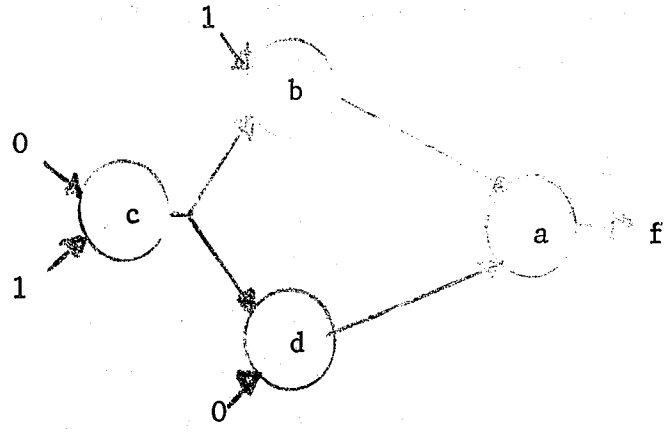


Figure 1 bis

qui est "arborescent", c'est-à-dire où la sortie d'un opérateur n'est utilisé comme entrée que dans un seul autre opérateur; on peut le simplifier en remarquant que deux sous-réseaux identiques peuvent être remplacés par un seul dont on utilisera deux fois la sortie : on obtient le réseau U de la figure 1 bis.

Nous savons déjà que toute fonction booléenne peut être réalisée par un réseau U, et la question qui se pose est de pouvoir la réaliser avec un réseau de coût le plus faible possible.

Remarquons que d'après notre définition les deux fonctions non dégénérées d'une variable ont un coût minimum de réalisation égal à 1, puisque :

$$x = Ux(1, 0) \quad \text{et} \quad x' = Ux(0, 1),$$

ce qui pour la première ne correspond pas à la réalité physique où x est données, donc de coût nul; mais d'une part, on pourra toujours faire la correction une fois le réseau obtenu, d'autre part, cette convention facilite l'étude théorique de la question. Elle permet en particulier d'énoncer les deux propriétés ci-après :

PROPRIETE 1

Deux fonctions "de même type" (c'est-à-dire telles qu'on puisse passer de l'une à l'autre par des permutations et complémentations sur les variables) ont des réalisations par U de même coût minimum.

PROPRIETE 2

Si dans un réseau U réalisant une fonction f on inverse les entrées 1 ou 0, on obtient un réseau U réalisant son complément f'.

Une conséquence des propriétés 1 et 2 est qu'une fonction, son complément et sa duale ont des réalisations par U de même coût minimum.

15 - BORNE SUPERIEURE DE COUT

THEOREME

Le coût minimum de réalisation par un réseau U d'une fonction est compris entre :

- le nombre de variables dont elle dépend effectivement (borne inférieure),
- le coût minimum en lettres d'une expression en somme-produit complément de cette fonction (borne supérieure).

Ainsi, f = b c(a+d) + c a d' a un coût minimum de réalisation compris entre 4 et 7.

La borne inférieure est évidente; l'autre se démontre à partir du théorème suivant :

THEOREME :

Si une fonction peut s'écrire sous la forme :

$$\varphi(Z) = \Psi\ g(X),\ Y$$

(X et Y étant deux parties Z, non forcément disjointes), si $R\Psi$ est un réseau U réalisant $\Psi(g, Y)$ (où g est une variable simple) Rg un réseau U réalisant $g(X)$, on obtient un réseau U réalisant $\varphi(Z)$ en remplaçant dans $R\Psi$ tout opérateur Ug par un réseau Rg réalisant $g(X)$ et dont les entrées 1 et 0 sont remplacées respectivement par les entrées supérieure et inférieure de Ug.

Exemple :

Soit :

$$\varphi = (a+b)\ c + a'\ b'\ d$$

La fonction $\Psi(g, c, d) = gc + g'd$ a la réalisation suivante (figure 2) :

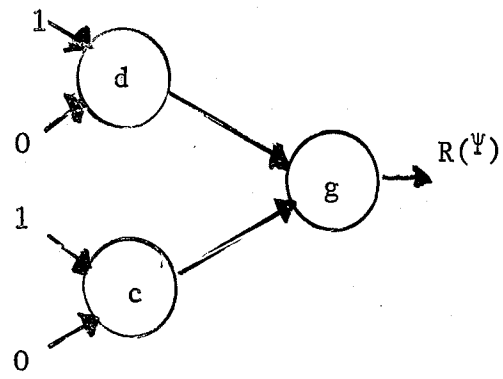


Figure 2

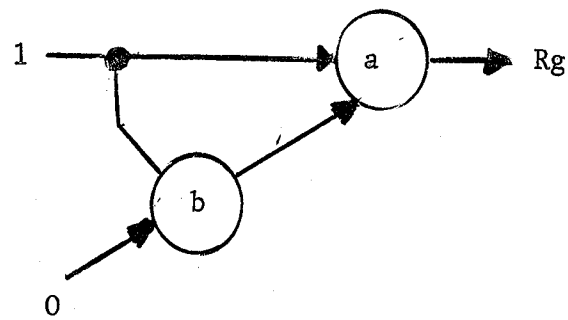


Figure 2 bis

Soit Rg un réseau réalisant $g(a, b) = a + b$ (figure 2 bis) on obtient un réseau réalisant φ en remplaçant dans $R\Psi$ 1'opérateur Ug par Rg (figure 3).

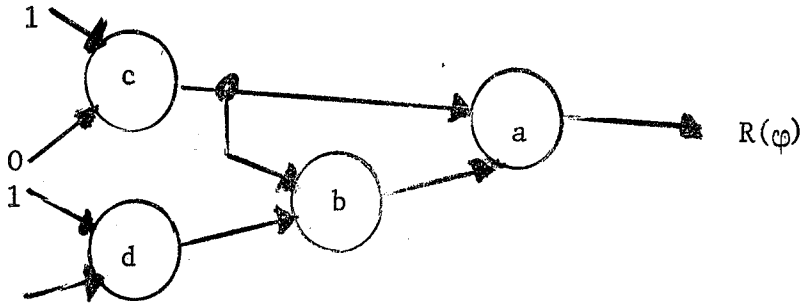
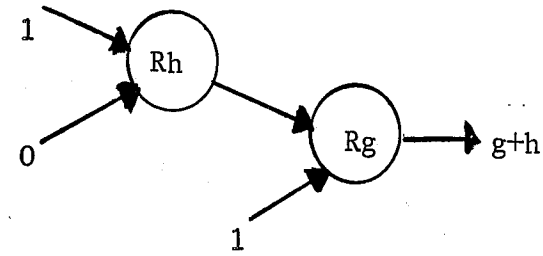
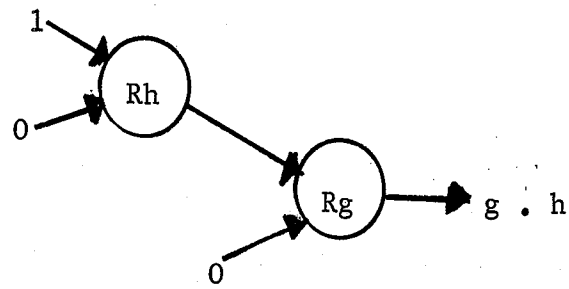


figure 3

Ce théorème s'applique en particulier à une somme : g(X) + h(Y) et à un produit : g(X) h(Y), pour lesquels on obtient les réalisations suivantes :



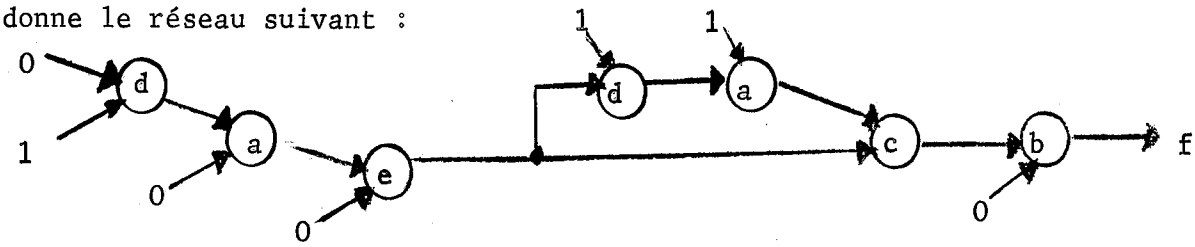


Ayant une expression en somme-produit-complément d'une fonction, on peut employer successivement ce procédé à tous les produits et sommes qui figurent dans cette expression : on obtiendra ainsi un réseau U qui réalise la fonction, et dont le coût est égal au nombre de lettres de l'expression.

Les réseaux qu'on obtient ainsi ne sont pas forcément de coût minimum : ainsi l'expression :

$$f = bc(a+d) + ead'$$

donne le réseau suivant :



de coût 7, or on peut réaliser cette fonction par un réseau de coût 6; remarquons cependant qu'ils ont une structure en chaîne qui peut présenter de l'intérêt.

16 - METHODE PRATIQUE DE SYNTHESE

1 - RESEAU ARBORESCENT :

La recherche d'un réseau arborescent de coût minimum réalisant une fonction donnée est simplifiée par les deux propriétés suivantes :

- a) Un tel réseau est nécessairement formé de sous-réseaux arborescents de coût minimum. En effet, un sous-réseau d'un réseau arborescent R n'est rattaché au reste du réseau que par sa sortie; s'il n'est pas de coût minimum R ne l'est pas non plus puisqu'on peut l'améliorer en amélirant le sous-réseau.
- b) Si x_i est sa variable "de tête" autrement dit si l'opérateur le plus en aval est Ux_i, elle n'apparaît plus dans le reste du réseau.

Il existe à partir de là plusieurs méthodes pour obtenir un réseau arborescent de coût minimum. Nous en exposerons une seule; elle utilise les trois remarques suivantes :

1- Si f est indépendante de x_i, cette variable n'intervient dans aucune représentation de coût minimum de;

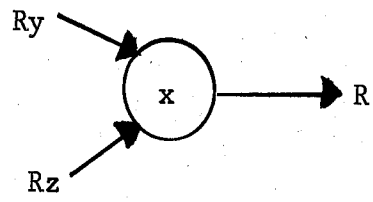
2- On dira que x_i est équivalente à x_j si on a l'identité suivante :

$f(..., x_i\ 1, ..., x_j = 0, ...) = f(..., x_i = 0, ..., x_j = 1, ...)$;

si deux variables x_i et x_j sont telles que x_i soit équivalente, soit à x_j, soit à x_j', à tout réseau réalisant f et ayant x_i en tête il correspond un réseau de même coût réalisant f et ayant x_j en tête. Il est donc inutile d'essayer ces deux variables à la fois comme variables de tête.

3- Si $f = U(x, Y, Z)$ et si Y et Z sont deux fonctions ne dépendant effectivement d'aucun argument commun, il suffira de chercher les représentations de coût minimum de f ayant x en tête.

En effet,[1] soit R le réseau ayant pour variable de tête x, pour entrée supérieure de Ux la sortie d'un réseau RY de coût minimum réalisant Y et pour entrée inférieure la sortie d'un réseau RZ de coût minimum réalisant Z.



Soient NY et NZ les coût respectifs de RY et RZ, celui de R est

$$C = 1 + NY + NZ$$

Soit R' un réseau quelconque réalisant la même fonction $f = xY + x'Z$, et soient N'x, N'Y et N'Z les nombres respectifs d'opérateurs Ux, Uy_i (y_i = argument de Y), z_i (z_i = argument de Z) de ce réseau. Le coût de R" est

$$C' = N'x + N'Y + N'Z$$

montrons que $c' \geq C$.

Y est différent de Z, donc f dépend de x et il y a au moins un opérateur Ux dans R'. Donc $N'x \geq 1$.

Si en fait x = 1, f devient identique à Y, quelles que soient les valeurs que prennent les arguments de Z. Or dans R' faire x = 1 revient à supprimer tout opérateur Ux et à joindre sa sortie à son entrée supérieure d'après l'identité :

$$U(1, \varphi, \Psi) = \varphi;$$

---

(1) La démonstration qui suit ne suppose pas que le réseau est arborescent : la remarque 3) est donc vraie pour un réseau quelconque.

On peut donner aux arguments de Z cette même valeur 1, ce qui revient de la même façon à supprimer les opérateurs Uz_i; on obtient un réseau U réalisant Y et où il n'y a plus que des opérateurs Uy_i, en nombre :

$$N'' Y \leq N'Y;$$

Comme NY est le coût minimum de réalisation de Y,

on a :

$$N''Y \qquad N''Y \geq NY,$$

donc :

$$N'Y \geq NY.$$

On démontrerait de même :

$$N'x \geq Nx.$$

On a donc bien :

$$C' \geq C.$$

Comme cas particuliers intéressants, citons les fonctions :

$$x + g(Y);\ xg(Y)\ ;\ xy + x'z$$

qui admettent les représentations de coût minimum avec x en tête.

METHODE DE SYNTHESE :

Soit $\{x_1, x_2, \ldots x_k\}$ l'ensemble des arguments de f réduit après application des remarques (1) et (2). A partir de la représentation canonique, on écrit la fonction sous les K formes successives :

$$\begin{aligned} f &= U(x_1, Y_1, Z_1) \\ &= U(x_2, Y_2, Z_2) \\ &\quad \ldots\ldots\ldots\ldots \\ &= U(x_K, Y_K, Z_K) \end{aligned}$$

- si $(Y_1 = 1;\ Z_1 = 0)$ ou $(Y_1 = 0;\ Z_1 = 1)$ la forme $U(x_1, Y_1, Z_1)$ est minimale et a pour coût 1.

- s'il existe x_i ($1 \leq i \leq K$) tel que les variables dont dépendent effectivement Y_i et Z_i forment deux ensembles d'arguments disjoints, d'après 3) le problème se ramène à la recherche des synthèses minimales de deux fonctions de n-1 variables (si n est le nombre de variables de f), Y_i et Z_i, et le coût pour f sera égal à la somme de leurs coûts + 1,

- sinon il faudra chercher : une synthèse minimale avec x_1, en tête, de même avec x_2, ..., de même avec x_K, et comparer les coûts des K synthèses obtenues : le problème se ramène donc à la recherche des synthèses minimales de 2K fonctions de n-1 variables.

2 - SYNTHESE A PARTIR D'UN RESEAU ARBORESCENT :

Le réseau arborescent une fois obtenu, on peut trouver un réseau moins coûteux réalisant la même fonction en remarquant que si deux sous-réseaux sont identiques on peut en supprimer un et rattacher l'extrémité du réseau laissée vacante à la sortie de l'autre. On montre facilement que le réseau obtenu en effectuant toutes les réductions possibles est unique, et qu'il suffit pour l'avoir d'appliquer jusqu'à épuisement de toutes les possibilités la règle suivante :

Si deux opérateurs sont identiques et ont les deux mêmes entrées dans le même ordre, on en supprime un et on rattache l'extrémité du réseau laissée vacante à la sortie de l'autre :

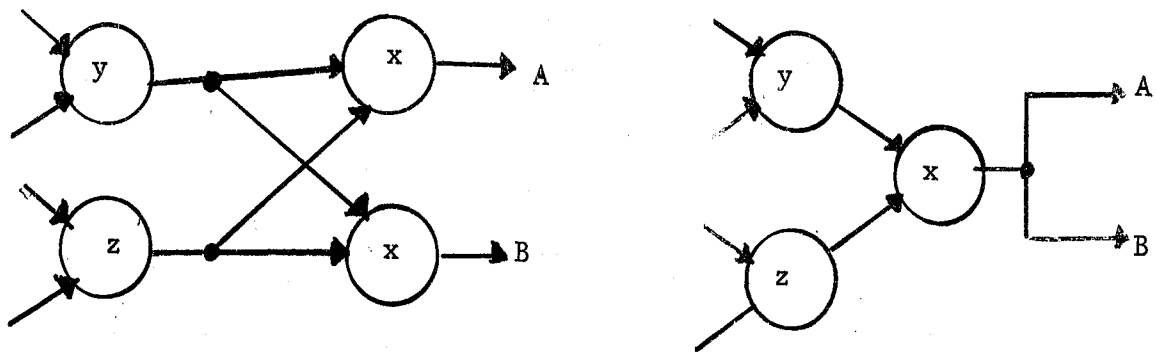


Figure 4

Remarquons qu'on a souvent intérêt pour obtenir le plus de réductions possibles à garder le même ordre des lettres dans les différents embranchements. C'est le cas par exemple si l'on veut réaliser le produit disjonctif :

$$x \otimes y \otimes z \otimes t;$$

en partant de l'ordre (x, y, z, t) on obtient le réseau de la figure 5, de coût 7, alors que le coût du réseau arborescent était 15.

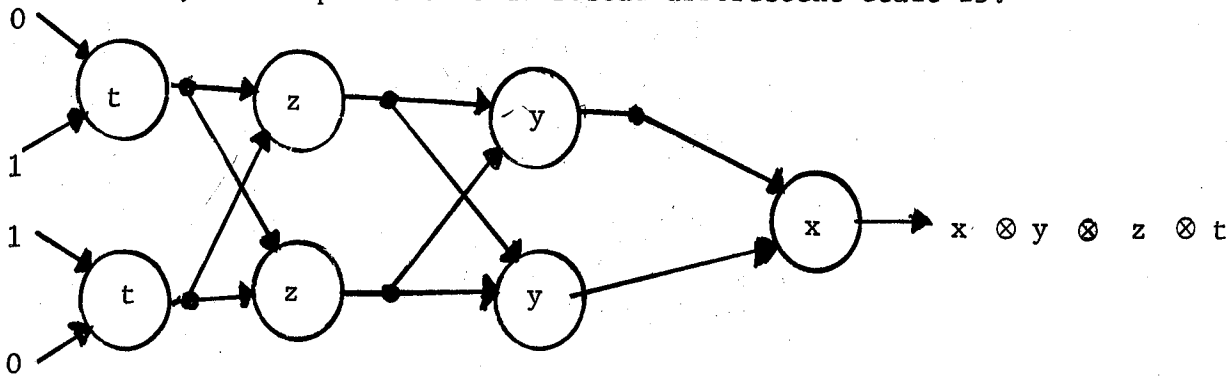


Figure 5

Bien que ne donnant pas forcément un réseau U de coût minimum, le procédé ci-dessus donne de bons résultats : sur 40 fonctions de 4 variables nous n'avons pu en trouver qu'une pour laquelle il ne donnait pas le coût minimum.

Dans le cas où l'on s'impose le même ordre des lettres dans les différents embranchements, le tableau suivant prévoit une borne supérieure du coût minimum de réalisation pour un nombre de variables n donné.

n	1	2	3	4	5	6	7	8	9	10	11	12
Borne supérieure du coût.	1	3	5	8	16	29	45	77	141	269	509	765

Pour $n \leq 4$ la borne supérieure est attente par au moins une fonction.

CONCLUSION

L'opérateur U semble avantageux à utiliser à cause de la simplicité de l'algorithme de synthèse. Il est certainement à regretter que cet opérateur ne soit pas fabriqué réellement par les constructeurs.

La méthode n'a pas été programmée; le travail consisterait en fait à concevoir un programme d'écriture d'une fonction sous une forme lexicographique locale ou non minimale.

Remarquons enfin que la méthode proposée par Monsieur Macheras est la seule connue pour résoudre ce problème : l'opérateur U n'étant pas monotone, les algorithmes par composition ne sont pas applicables.

CONCLUSION SUR CETTE METHODE DE FRACTIONNEMENT

La méthode de fractionnement exposée s'est avérée très efficace en ce qui concerne la synthèse de fonctions simples et complètes en réseaux arborescents. Elle a, d'autre part, l'avantage de la souplesse d'emploi : moyennant des modifications très simples nous avons pu obtenir des structures particulières (réseaux sans aléa, réseaux redondants, etc...).

Nous remarquons cependant les inconvénients suivants :

- A chaque fractionnement nous sommes obligés de choisir - parfois à priori - les fonctions intermédiaires, donc perdre de la généralité.

- Nous ne pouvons pas dans notre réseau arborescent, trouver les sous-fonctions calculées plusieurs fois (sauf dans les cas très simples). La conséquence de ceci est qu'on ne pourra pas utiliser cette méthode pour la synthèse des fonctions générales.

- La méthode ne permet pas de tenir compte du fait que si une fonction :

$$f = \sum_{i \in I} m_i \text{ a été fractionné en}$$

$$f_1 = \sum_{i \in I_1} m_i \quad \text{et} \quad f_2 = \sum_{i \in I_2}$$

avec $I_1 \cup I_2 = I$

On peut a volonté ajouter à f_1 des monômes quelconques de f_2 et à f_2 des monômes de f_1.

En fait, chaque partie f_i de f devrait être considérée comme une fonction incomplète :

$$\boxed{F_i = f_i + \emptyset\, f}$$

Enfin, nous remarquons que dans notre méthode s'introduit le calcul des duales et bases premières irredondantes de fonctions, calculs très lents en machine.

Ces différents inconvénients nous ont conduit à envisager la deuxième méthode dite des bornes.

C - FRACTIONNEMENT PAR LA METHODE DES BORNES

1 - INTRODUCTION

Cette méthode est essentiellement destinée à traiter le problème de la synthèse de fonctions incomplètes.

La méthode que nous proposons est dérivée d'un article de S.B. Ackers paru dans les I.E.E.E. Transaction on Electronic Computers en février 1965.

Pour développer cette méthode, nous avons étudié une représentation particulière des fonctions incomplètes.

2 - DEFINITIONS - NOTATIONS

Soit $F = \underline{f}$ et $\emptyset\ \bar{f}$ une fonction booléenne incomplètement spécifiée dont les bornes inférieures et supérieures sont respectivement $\underline{f}$ et $\bar{f}$.

Soit $\bar{f}^*$ le dual de $\bar{f}$.

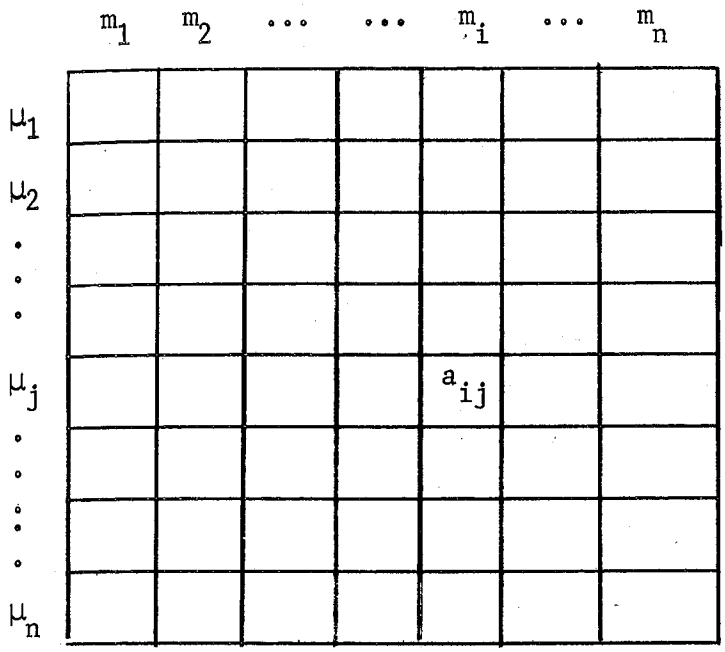
Nous considérons deux bases quelconques :

$$\sum_{i=1}^{n} m_i = \underline{f}$$

et

$$\sum_{j=1}^{p} \mu_j = \bar{f}^*$$

et le tableau ci-après :



dans la case de coordonnées i,j se trouvent les lettres communes aux monômes m_i et μ_j.

On désigne par A_{ij} l'ensemble de ces lettres et par a_{ij} le monôme produit de ces lettres.

Exemple :

$F = ab + bc + ca + \emptyset\ (a + bc)$

$\underline{f} = ab + bc + ca$

$\underline{f}^* = ab + ac$

	ab	bc	ca
ab	a,b	b	a
ac	a	c	a,c

Remarque :

Une fois le tableau dréssé, on fera abstraction des monômes origines m_i et μ_j.

DEFINITION

F sera dite la fonction génératrice de T et T le tableau généré par F.

NOTATION

T sera souvent noté $\{a_{ij}\}$.

LEMME 1 -

Aucun des A_{ij} n'est vide, en effet, si un monôme m_i de $\underline{f}$ et un monôme μ_j de $\underline{f}^*$ n'avaient aucune lettre en commun, cela entraînerait qu'entre m_i $\underline{f}$ et $\tilde{\mu}_j$ le monôme correspondant à μ_j dans $(\bar{f})'$ (obtenu à partir de μ en changeant toutes les accentuations), il n'y a pas une même lettre sous deux aspects donc μ_j et m_i peuvent valoir simultanément 1 ce qui est exclu puisque $\underline{f}$ et $(\bar{f})'$ sont disjoints.

LEMME 2 -

Le tableau généré par F* est le transposé du tableau généré par F, on le notera T*.

Remarque :

Le tableau ne représente pas biunivoquement la fonction comme le montre l'exemple suivant :

$$\begin{cases} \underline{f} = abc'd + a'bcd + ab'c'd \\ \underline{f}^* = abc'd' + ab'cd \end{cases}$$

$$\begin{cases} \underline{g} = abc'd + bcd + ab'c'd \\ \underline{g}^* = abc' + ab'cd \end{cases}$$

dont les tableaux sont égaux au tableau ci-contre :

	m_1	m_2	m_3
μ_1	a,b,c'	b	a,c'
μ_2	a,d	c,d	a,b',d

LEMME 3 -

Soit T un tableau de n lignes et P colonnes noté :

$$\{a_{ij}\} \qquad \begin{matrix} 1 \le i \le n \\ 1 \le j \le p \end{matrix}$$

dans lequel a_{ij} désigne le produit des lettres présentées dans la case i,j.

Considérons les deux fonctions :

$$g = \sum_{i=1}^{n} \prod_{j=1}^{p} a_{ij} \qquad \text{et} \qquad \sum_{j=1}^{p} \prod_{i=1}^{n} a_{ij}$$

Ces deux fonctions vérifient $g \le h^*$ ou encore :

$$\boxed{\sum_{i=1}^{n} \prod_{j=1}^{p} a_{ij} \le \prod_{j=1}^{p} \sum_{i=1}^{n} a^*_{ij}}$$

où a^*_{ij} est le dual du monôme a_{ij}.

a) Supposons d'abord les a_{ij} variables indépendantes donc $a^*_{ij} = a_{ij}$ dans ce cas l'inégalité :

$$\sum_{i=1}^{n} \prod_{j=1}^{p} \le \prod_{j=1}^{p} \sum_{i=1}^{n} a_{ij}$$

est évidente car dans le dévoloppement en somme de produit du 2ème membre on retrouve tous les termes du 1er.

b) Si les a_{ij} sont des monômes et non des variables, alors $a_{ij} \le a^*_{ij}$.

Or le 2ème membre de l'inégalité est fonction croissante des a_{ij} donc :

$$\prod_{j=1}^{p} \sum_{i=1}^{n} a_{ij} \leq \prod_{j=1}^{p} \sum_{i=1}^{n} a^*_{ij} ,$$

ce qui démontre le lemme.

Remarque :

Le lemme est démontré quelque soit T; on n'a pas eu besoin de supposer l'existence d'une fonction génératrice de T.

DEFINITION

On appelle fonction générée par T la fonction incomplète :

$$G = g + \emptyset\ h^*;$$

ou encore, en posant $\underline{g} = g$ et $\overline{g} = h^*$,

$$G = \underline{g} + \emptyset\ \overline{g}.$$

D'après la remarque précédente tout tableau génère une fonction G.

THEOREME

G est indépendant de la forme de départ de $\underline{f}$ et $\overline{f}^*$.

Montrons que toutes les fonctions G obtenues pour des bases différentes de $\underline{f}$ sont égales à G_o obtenu à partir d'une forme canonique de $\underline{f}$.

Pour ceci montrons qu'il est possible de remplacer dans toute forme $mx + mx^*$ par m.(on sait que cette opération, à partir de la forme canonique, peut conduire à toute base); désignons par : $A_i x$, $B_j x$ et C_k les monômes de $\overline{f}^*$ contenant respectivement x, x', et ni x, ni x'.

Désignons par a_i, b_j et c_k les monômes des lettres communes de m avec respectivement A_i, B_j, C_k (voir figure).

Dressons le tableau F dans lequel m x, m x* et m figureraient simultanément.

	mx	mx'	m
A_i . x	a_i . x	a_i	a_i
B_j x'	b_j	b_j . x'	b_j
C_k	c_k	c_k	c_k

soient n_1, n_2 et n_3 monômes de la fonction $\underline{g}$ générée par T correspondant à mx, mx', m; on a :

$$n_1 + n_2 = \prod a_i x \cdot \prod b_j \cdot \prod c_k + \prod a_i \cdot \prod b_j x' \cdot \prod c_k$$

$$n = \prod a_i \cdot \prod b_j \cdot \prod c_k$$

On constate qu'on a : $n_1 + n_2 = n$ autrement dit on peut remplacer mx + mx' par m dans $\underline{f}$ sans changer G, donc sans changer G, passer d'une base canonique à une base quelconque.

LEMME 4 -

Supposons T généré par F. La fonction G générée par T est compatible avec la fonction f génératrice de T; c'est-à-dire :

$$\underline{f} \leq \underline{g} \leq \overline{g} \leq \overline{f}$$

(a) Montrons $\underline{f} \leq \underline{g}$

Il y a le même nombre de monômes dans les deux fonctions.

Dans la colonne i du monôme m_i de $\underline{f}$ ne se trouvent que des monômes a_{ij} supérieures ou égaux à m_i (ils ont moins de lettres) donc leur produit $\prod_{j=1}^{p} a_{ij}$ est encore supérieure ou égal à m_i, ceci pour tout i donc $\underline{g} \geq \underline{f}$.

(b) On montre de la même manière que $\overline{g}^* \geq f^*$ donc $\underline{g} \leq \overline{f}$

COROLLAIRE

Si la fonction génératrice est complète elle est égale à la fonction générée.

LEMME 5 -

Condition nécessaire et suffisante pour avoir $\underline{f} \neq \underline{g}$

Soit

$$m_i \in \underline{f} \quad \text{et} \quad m_i^* \in \prod_{j=1}^{p} a_{ij}$$

pour avoir $m_i \neq m_i^*$ il faut et il suffit qu'au moins une lettre a de m_i ne figure pas dans m_i^*, c'est-à-dire ne figure dans aucun des a_{ij}, c'est-à-dire ne figure dans aucun des monômes μ_j. On peut donc énoncer la condition suivante :

Une condition nécessaire et suffisante pour avoir $\underline{f} \neq \underline{g}$ est qu'il existe une variable a telle que $\underline{f}$ et $\overline{f}$ vérifient une des conditions suivantes :

(1) $\underline{f}$ non monotone en a et $\overline{f}$ monotone en a.

(2) $\underline{f}$ fonction de a et $\overline{f}$ indépendante de a.

On aurait les mêmes conditions pour $\overline{g}$ et $\overline{f}$.

COROLLAIRE :

I -(1)- Si $\overline{f}$ est monotone en a alors g est l'enveloppe supérieure monotone en a de $\underline{f}$.

(2)- Si $\overline{f}$ est indépendante de a alors $\underline{g}$ est l'enveloppe supérieure indépendante en a de $\underline{f}$.

D'après la terminologie de Monsieur Lapscher, on dira que $\underline{g}$ est l'enveloppe incidente par rapport aux lettres figurant dans $\overline{f}$.

II De même pour la borne $\overline{g}^*$.

Sous-tableau de T.

Soit un tableau T à p lignes et n colonnes.

Soient I et J les ensembles d'indices :

$$I = \{ i,\ 1 \leq i \leq n \}$$

$$J = \{ j,\ 1 \leq j \leq p \}$$

et soient :

$$I_1 \subseteq I \quad \text{et} \quad J_1 \subseteq J.$$

on notera :

T_{I_1,J_1} le tableau de la fonction

$$F_1 = \sum_{i \in I_1} \left(\prod_{j \in J_1} a_{ij} \right) + \phi \left[\sum_{j \in j_1} \left(\prod_{i \in I_1} a_{ij} \right) \right]^*$$

ce tableau est un sous-tableau de $T_{I,J}$.

LEMME 6 -

Soient I_1 et I_2 deux ensembles d'indices vérifiant $I_1 \quad I_2 = I$ et considérons les sous-tableaux : $T_{I_1,J}$ et $T_{I_2,J}$ et les fonctions générés :

$$F_1 = \sum_{i \in I_1} \prod_{j \in J} a_{ij} + \phi \left[\sum_{j \in J} \prod_{i \in I_1} a_{ij} \right]^*$$

$$F_2 = \sum_{i \in I_2} \prod_{j \in J} a_{ij} + \phi \left[\sum_{j \in J} \prod_{i \in I_2} a_{ij} \right]^*$$

Soient F_1 et F_2 fonctions complètes compatibles avec F_1 et F_2.

Lemme : $F_1 + F_2$ est compatible avec F.

(a) $F_1 \geq \sum_{i \in I_1} \prod_{j \in J} a_{ij}$ et $F_2 \geq \sum_{i \in I_2} \prod_{j \in J} a_{ij}$

donc :

$$F_1 + F_2 \geq \sum_{i \in (I_1 \cup I_2)} \left(\prod_{j \in J} a_{ij} \right) = \underline{g} \geq \underline{f}$$

ⓑ $F_2 \leq (\sum_{j \in J} \prod_{i \in I_1} a_{ij})^*$ et $F_2 \leq (\sum_{j \in J} \prod_{i \in I_2} a_{ij})^*$

donc :

$$F_1 + F_2 \leq \left[\sum_{j \in J} \prod_{i \in I_1} a_{ij} \cdot \sum_{j \in J} \prod_{i \in I_2} a_{ij}\right]^* = A^*$$

le dual du deuxième membre A est un produit de sommes qu'il est possible de développer; on obtient parmi d'autres tous les termes de la forme :

$$\prod_{i \in I_1} a_{ij} \cdot \prod_{i \in I_2} a_{ij} = \prod_{i \subset (I_1 \cup I_2)} a_{ij}$$

donc $A \geq \sum_{j \in J} \prod_{i \in I} a_{ij}$

donc en passant au dual :

$$A^* \leq (\sum_{j \in J} \prod_{i \in I} a_{ij})^* = \bar{g} \leq \bar{f}$$

ce qui termine la démonstration du lemme.

COROLLAIRE

Par dualité on obtient la même propriété pour F* :

Si $J_1 \cup J_2 = J$ et si F_1 et F_2 sont compatibles avec T_{I,J_1} et T_{I,J_2}, alors $F_1 \cdot F_2$ est compatible avec F.

Définition :

On dira que F est compatible avec T si F est compatible avec la fonction G générée par T.

THEOREME

Le lemme 6 et sa corollaire permettent d'énoncer le théorème suivant :

Si $I_1 \subset I$, si $J_1 \subset J$ et si φ est compatible avec le tableau T_{I_1,J_1}, alors il existe deux fonctions A et B telles que

$F = A\varphi + B$ soit compatible avec T.

compatible (chute de lemme)

En effet, considérons I_2 complémentaire de I_1 dans I et J_2 complémentaire de J_1 dans J.

Soient A et B compatibles respectivement avec les tableaux :

$$T_{I_1,J_2} \quad \text{et} \quad T_{I_2,J}$$

D'après le lemme précédent on a :

A . F compatible avec la fonction G_1 générée par le tableau $T[I_1,J]$.

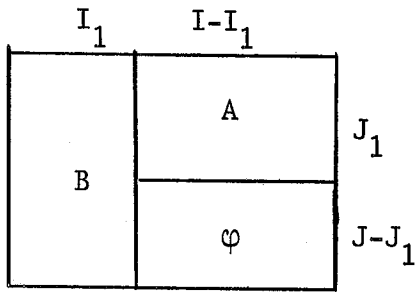
D'après le même lemme, A . F + B est compatible avec G généré par $T_{I,J}$ donc avec F.

Décomposition

THEOREME 2 - Décomposition croissante;

soient T(I,J) et deux ensembles d'indices I_1 et J_1 tels que les tableaux $T(I_1,J)$, $T(I-I_1, J_1)$ ne contiennent pas les lettres x_i de X et $T(I-I_1,J-J_1)$ ne contiennent pas les lettres y_j de Y alors il existe une décomposition :

$$f = A(Y,Z)\ \varphi(X,Z) + B(Y,Z).$$



Ce théorème est une corollaire immédiate du théorème (1).

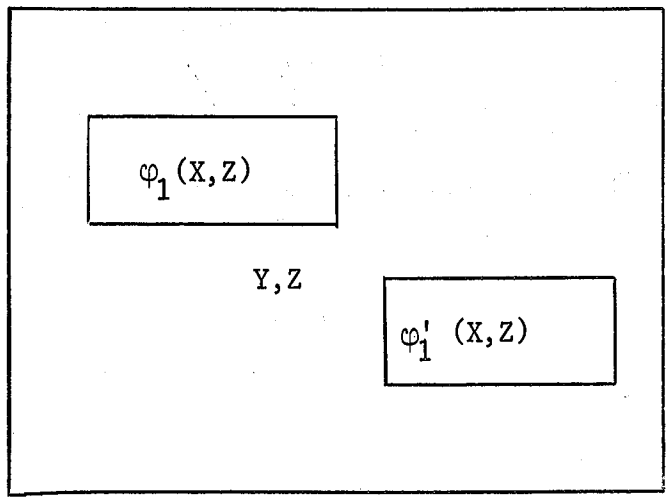
Remarque :

Nous trouvons ainsi des décompositions particulières : f est croissant en φ.

3 - GENERALISATION - DECOMPOSITION QUELCONQUE

Une décomposition quelconque sera simplement caractérisée par deux sous-tableaux $T(I,J_1)$ et $T(I_2,J_2)$ (avec $I_1 \cap I_2 = 0$, $J_1 \cap J_2 = 0$) dont les fonctions générées φ_1 et φ_2 ne dépendent que de certaines variables et vérifient :

$$\varphi_1 = \varphi_2'$$



en effet, on pourra écrire successivement

$$f = A(Y,Z)\ \varphi_1(X,\ Y) + B(X,Y,Z)$$

et $$B = C(Y,Z)\ \varphi_1'(X,Y) + D(Y,Z)$$

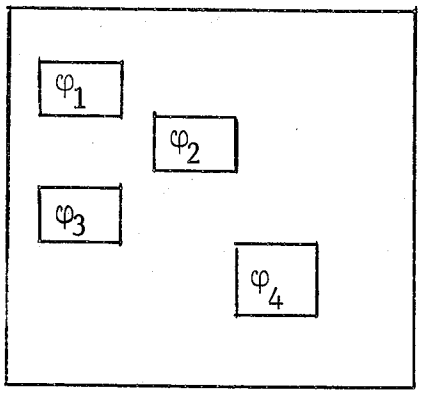
donc : $$f = A(Y,Z)\ \varphi_1(X,Y) + C(Y,Z)\ \varphi_1'(X,Y) + D(Y,Z)$$

$$f = h(\varphi(X,Y),\ Y,\ Z)$$

h étant une fonction quelconque.

Décompositions multiples :

On trouverait sans difficulté des décompositions multiples en isolant non pas 2 mais k sous-tableaux ne contenant pas Y, leur complémentaire ne contenant pas X., et on aura des expression de la forme :



$$f = h(\varphi_1(X,Z)\ \varphi_2(X,Z)\ \ldots\ \varphi\ (X,Z),Y,Z)$$

Remarque :

Si sur des exemples simples ne contenant pas Y, il est facile de former des rectangles dans le tableau T quite à permuter certaines lignes ou colonnes, il est plus difficile de réaliser ceci en machine.

Il est cependant possible d'utiliser l'algorithme de Malgrange qui est bien adapté à ce genre de problème.

4 - APPLICATION A LA SYNTHESE PAR FRACTIONNEMENT

1 - Rappel sur les méthodes de fractionnement

Etant donné une fonction F incomplète, nous cherchons un réseau d'opérateurs (par exemple sommes et produits) dont la sortie est compatible avec F.

- Une méthode de "fractionnement" en sommes et produits consistera à déterminer k fonctions incomplètes $F_1, F_2, \ldots, F_k$ telles que :

$\sum_{i=1}^{k} F_1$ ou $\bigcap_{i=1}^{k} F_i$ soit compatible avec F.

- Pour chacun des F_i on peut itérer cette méthode de fractionnement, le test d'arrêt de l'itération consistant à chercher s'il existe une variable compatible avec la fonction incomplète considérée.

Remarque :

Dans chaque méthode de fractionnement la convergence devra être assurée par une condition sur les F_i.

Cas particulier du tableau d'Ackers :

(1) Définition du partage

(a) Il est possible de déterminer k fonctions F_1, F_2, ..., F_k telles que $\sum_{i=1}^{k} F_i$ soit compatible avec $G = \sum_{i=1}^{n} m_i + \emptyset \left(\sum_{j=1}^{p} M_j\right)^*$ donc avec F.

Il suffit de déterminer un k-recouvrement I_1, I_2, ..., I_k de l'ensemble d'indices $I = \{i,\ 1 \leq i \leq n\}$ et de considérer les k fonctions générées par les tableaux $T_{I_i,J}$. (lemme 7)

(b) De la même manière il est possible de déterminer la fonction F_i telle que $\prod_{i=1}^{k} F_i$ soit compatible avec G, donc en F en déterminant un k-recouvrement de l'ensemble d'indices J et en considérant les fonctions générées par les tableaux T_{I,J_i}.

Exemple :

$$f\ F = ab + bc + ac + \emptyset\ (a+bc)$$

$T =$

	ab	bc	ac	
ab	a,b	b	a	1
ac	a	c	a,c	2
	1	2	3	

$I = \{1, 2, 3\}$; $J = \{1, 2\}$

Les fonctions F_1 et F_2 générées par les tableaux suivants :

$T_{\{1,2\},\ J}$ =

	ab	bc	
ab	a,b	b	1
ac	a	c	2
	1	2	

$T_{\{2,3\},\ J}$ =

	bc	ac	
ab	b	a	1
ac	c	a,c	2

c'est-à-dire :

$$F_1 = ab + bc + \emptyset\ (ab+ac)^*$$

$$F_2 = bc + ac + \emptyset\ (ab+ac)^*$$

ont leur somme compatible avec F.

(2) Itération de la méthode - Convergence

On peut continuer à partager le tableau d'Ackers suivant la méthode précédente.

Si les sous-tableaux T_i formés sont stricts, c'est-à-dire : $I_i \neq I$ ou $J_j \neq J$ alors les dimensions des tableaux successifs diminuent strictement à chaque opération, on arrivera donc après un nombre fini d'itération à des tableaux 1×1 c'est-à-dire dont la fonction générée s'écrira :

$$F_{PQ} = m_P + \emptyset\ (\mu_q)^*\ P = \{p\}\ ;\ Q = \{q\}$$

or, on a vu que deux monômes m_p et μ ont toujours une variable x en commun. Il existe donc une variable x compatible avec F_{pq}.

On est donc assuré ici de la convergence de la méthode de fractionnement.

Remarque :

On testera en fait, à chaque itération, s'il existe une variable compatible avec la fonction $F_{P.Q}$ considérée, même si P ou Q ont plus d'un d'indice.

Exemple :

Soit F la fonction complète :

$$F = ab + bc + ca + \emptyset\ (ab + bc + ca).$$

T =

	ab	bc	ca
ab	a,b	b	a
bc	b	b,c	c
ca	a	c	c,a

on détermine F_1 et F_2 en partageant le tableau en 2 suivant le pointillé pour obtenir :

T_1 =

a,b	b
b	b,c
a	c

T_2 =

a
c
c,a

$F_1 = ab + bc + \emptyset\ (ab + bc + ac)^*$

$F_2 = ac + \emptyset\ (a + c)^*$

$F = F_1 + F_2$

on coupe encore T_1 pour obtenir :

T_3 =

a,b	b
b	b,c

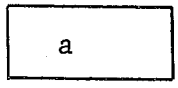
T_4 =

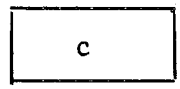
a	c

$F_3 = ab + bc + \emptyset(ab + bc)^*$ avec $F_1 = F_3 \cdot F_4$

$= b$

T_4 sera à son tour coupé en :

T_5 = 

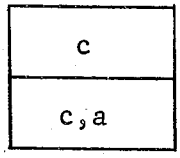
T_6 = 

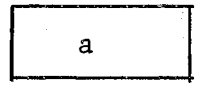
$F_5 = a + \emptyset\ a$
$= a$

$F_6 = c + \emptyset\ c$
$= c$

avec $F_4 = F_5 + F_6$
$= a + c$

Enfin, T_2 sera partagé en :

T_7 = 

T_8 = 

$F_7 = ac + \emptyset\ (c + ac)^*$
$= c$

$F_8 = a + \emptyset\ a$
$= a$

avec $F_2 = F_7 \cdot F_8$

on a donc en définitive :

$F = F_1 + F_2$
$= F_3 \cdot F_4 + F_2$
$= F_3\ (F_5 + F_6) + F_7 \cdot F_8$
$= b\ (a+c) + ac$

Forme de la solution :

(a) Par construction la fonction obtenue est compatible non seulement avec F mais aussi avec G. Or $\underline{g}$ et $\overline{g}^*$ sont plus simples que $\underline{f}$ et $\overline{f}^*$ (ils s'obtiennent par suppression de certaines lettres) ils contiennent éventuellement moins de monômes. On aura donc intérêt à faire la synthèse de G et non de F, c'est-à-dire, à dresser le tableau de G.

(b) Par construction également, la fonction trouvée ne s'écrit qu'avec des lettres du tableau T. On peut donc dire que si une des bornes est écrite sous forme monotone ou indépendante en a, la solution sera respectivement monotone ou indépendante en a.

(on ne peut rien dire si une borne monotone par exemple est écrite sous forme non monotone).

Variables nécessaires à l'expression de f.

Nous avons vu qu'au cours de la synthèse, nous sommes amenés à choisir dans chaque monôme a_{ij} de T, une lettre x_k particulière. Ceci pouvant se faire de diverses manières, nous avons un problème de couverture à résoudre.

A chaque variable x_k du problème, nous associons une variable y_k auxiliaire qui sera égale à 1 si x_k est utilisée effectivement dans la fonction complète. On appelle b_{ij} le monôme obtenu en remplaçant les x par les y. Dans a_{ij} nous devons choisir au moins une lettre, donc un y_i est égal à 1 dans b_{ij} donc :

$$(b_{ij})^* = 1 \text{ pour tout couple } i,j$$

donc :

$$\Psi = \prod_{i \in I,\ j \in J} (b_{ij})^* = 1$$

En développant cette expression (sans considérer que les termes en $y_k\ y'_k$ sont nuls) et en éliminant les multiples, nous obtenons une base don les monômes représentent les lettres nécessaires.

Remarques :

(1) Ce problème déjà traité par Monsieur Kuntzman dans son livre d'algèbre de Boole (page 30) est résolu ici dans un cas particulier : nous recherchons non pas toutes les bases de variables possibles, mais celles qui peuvent être obtenues par la méthode.

(2) Quand nous aborderons ce problème pour des fonctions générales, nous pourrons considérer le produit des diverses fonctions caractéristiques Ψ_k des fonctions élémentaires f_k :

$$\Psi = \prod_k \Psi_k = \prod_k \left(\prod_{i,j} (b_{k_{ij}})^* \right)$$

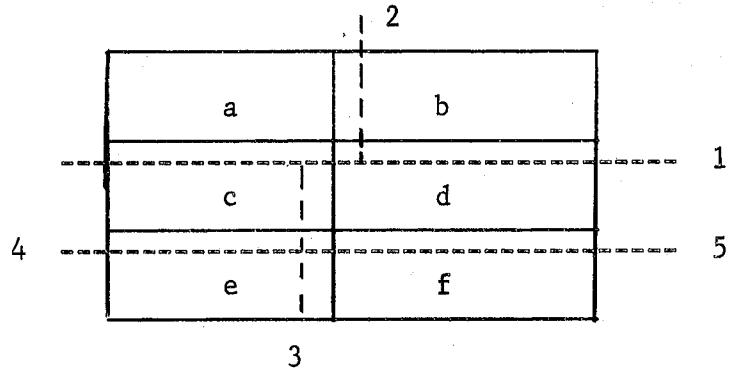
donc comme précédemment les monômes nous donnent les variables utiles au problème.

Coût maximum de la solution :

Nous avons vu que les fractionnements s'arrêtent quand on arrive à un tableau 1 X 1. Il y a au plus n X p tels tableaux si n = Card(I) et p = Card(J). Ce nombre de sous-tableaux 1 X 1 est le nombre de variables de la solution ou opérateurs + et X trouvée.

Ce nombre sera effectivement atteint pour un tableau dans lequel toutes les cases sont occupées par des variables distinctes.

Exemple :



$n \times p = 6$

notons en pointillé les coupures du tableau avec leur numéro d'ordre.

$f = (a+b)(ce + df)$ dont le coût est bien 6.

Remarque :

Notons qu'en général cette borne est peu réaliste.

Choix des bases de départ :

D'après la borne du coût nous voyons qu'il sera intéressant de limiter le plus possible le nombre de monômes de chacune des deux bornes $\underline{f}$ et $\overline{f}^*$ (en particulier, on choisira des bases irredondantes).

Par ailleurs, on trouvera d'autant plus facilement une variable compatible avec un sous-tableau donné, que les bornes sont plus éloignées l'une de l'autre, on aura donc intérêt à choisir des monômes les plus petits possible.

On prendra donc une base 1ère minimale de chaque borne et on cherchera à remplacer chaque monôme m_i par un monôme n_i le plus petit possible sans changer les autres monômes de la base.

Synthèse en opérateur Ni

Il est possible de développer une théorie identique en opérateur Ni, en considérant le tableau construit sur les fonctions $\underline{f}$ et $\overline{f}'$; dans la case a_{ij}, on place les lettres qui différencient $\underline{f}$ et $\overline{f}'$, sous la forme ou elles apparaissent dans $\underline{f}$.

La seule différence avec précédemment est la suivante. Si T est généré par F. $\underline{f} + \emptyset\ \overline{f}$, le tableau généré par F' est obtenu en transposant T et de plus en complémentant toutes les lettres.

On pourra s'affranchir de cette complémentation en comptant le nombre de couches, c'est-à-dire le nombre de coupures effectuées pour obtenir $T\left[I,J\right]$. Si ce nombre est impair, on complémente toutes les lettres de $T\left[I,J\right]$.

Exemple :

$$F = xy + yz + zx$$

$$\underline{f} = xy + yz + zx$$

$$\overline{f}' = x'y' + y'z' + z'x'$$

T =

	xy	yz	zx
x'y'	x,y	y	x
y'z'	y	y,z	z
z'x	x	z	z,x

1

chercher $F = Ni(F_1, F_2)$ revient à chercher $F' = F_1 + F_2$. Nous allons donc couper le complément de la borne supérieure suivant 1 .

T_1 =

x,y	y	x
y	y,z	z

T_2 =

x	z	z,x

T_1 et T_2 sont respectivement les tableaux de F_1' et F_2' , nous éviterons l'opération de complémentation en coupant cette fois verticalement :

T_4 =

x,y	y
y	y,z

$F_4 = y$

T_5 =

x
z

coupé en lui-même en

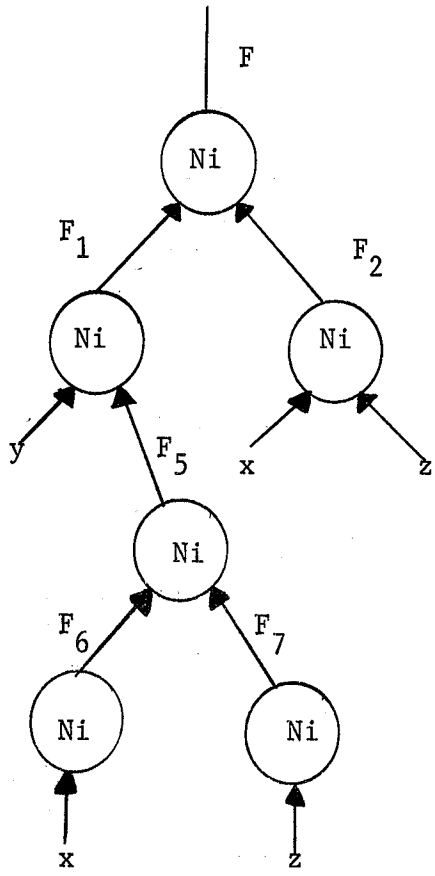
T_6 = | x | et T_7 = | z |

Ces tableaux obtenus après un nombre de coupures impaires représentant respectivement les fonctions x' et z'. Enfin T_2 sera coupé en :

T_8 = | x | T_9 = | z | z,x |

obtenus après 2 coupures, donc représentant les fonctions x et z.

On a obtenu le réseau :



5 - PROGRAMMATION DE LA METHODE

(1) Représentation des fonctions

Nous remarquons en premier lieu que le tableau T généré par une fonction f de n variables peut comporter jusqu'à :

$$M = 2^{n-1} \times 2^{n-1} \text{ monômes } a_{ij}.$$

Or chacun des a_{ij} ne sert qu'une fois, puisque les partages se font sans recouvrement. Nous n'avons pas intérêt à construire effectivement le tableau T, mais à partager directement les bornes des fonctions.

(2) Algorithme utilisé :

Une fonction $f = \underline{f} + \emptyset\, \overline{f}$ tant donnée nous opérerons de la façon suivante :

(a) On met $\underline{f}$ et $\overline{f}'$ sous forme de base première irredondante.

(b) On remplace éventuellement $\underline{f}$ et $\overline{f}'$ par leurs enveloppes $\underline{g}$ et $\overline{g}'$ définies précédemment, ce qui revient à supprimer dans l'une de ces fonctions toute lettre qui ne figure pas dans l'autre.

(c) Eventuellement on recalcule $\underline{g}$ et $\overline{g}'$ sous forme de base première irredondante.

(d) On détermine une partition des monômes de $\overline{g}'$ pour déterminer k fonctions intermédiaires :

$$f_i = (\overline{g}')_i + \emptyset\, \underline{g}'$$

dont la borne inférieure est $(\overline{g}')_i$ et la borne supérieure $(\underline{g}')$.

Cette partition pourra se faire par exemple suivant une méthode lexicographique.

(e) Pour chaque f_i, on regarde s'il existe une variable x_j qui est comprise entre les bornes. Sinon pour f_i considéré on itère la méthode.

(f) Quand un f_i a été déterminé, c'est-à-dire quand on a trouvé φ fonction complète compatible avec f_i, on pourra remplacer les bornes inférieures $\underline{f_{i+j}}$ par $\underline{f_{i+j}} \cdot \varphi'$ puisque les points de φ n'ont pas à être couverts une deuxième fois.

Mise en oeuvre

Comme les précédents cet algorithme a été programmé en Algol sur IBM 7044. Les temps de calcul sont très courts à cause de l'absence des calculs des compléments et des duales.

Sur 10 exemples traités par les deux méthodes : fractionnement simple et fractionnement des bornes, la deuxième méthode va en moyenne 10 fois plus vite.

Les coûts obtenus sont légèrement supérieurs (5 à 10 % d'augmentation) à ceux obtenus par la première méthode. Ceci est probablement du à l'absence de test de comparaison entre f et f* qui améliore considérablement le coût trouvé par la 1ère méthode).

Conclusion :

Cette méthode est intéressante par sa simplicité et la rapidité des programmes écrits. Si elle semble donner des résultats comparables aux précédents, elle offre l'énorme avantage d'être la seule méthode généralisable aux problèmes de synthèse de plusieurs fonctions.

Enfin, elle nous a permis d'envisager d'une manière simple le problème de la synthèse de réseaux redondants.

6 - SYNTHESES REDONDANTES EN RESEAUX D'OPERATEURS Ni.

INTRODUCTION

Le problème de la synthèse d'un réseau redondant est résolu en théorie depuis longtemps. Diverses méthodes sont employées effectivement, mais elles présentent toutes certains défauts.

La redondance pure (qui consiste à répéter un même réseau $2k + 1$ fois si on désire corriger k erreurs) est intéressante car la correction se fait de manière simple par un opérateur à seuil. Cependant, l'augmentation de matériel est importante.

Le codage permet une augmentation de matériel minime, mais les circuits de correction sont en général très complexes.

Remarque :

Dans les méthodes usuelles, on n'utilise en général pas le fait que certains réseaux sont naturellement redondants, (tout au moins de façon partielle). Cette propriété peut alors être utilisée pour améliorer les méthodes classiques. Nous nous intéresserons au cas de la redondance pure.

METHODE DE SYNTHESE

Soit F la fonction à réaliser et $F' = m_1 + m_2 + \ldots + m_p$ une base 1ère irredondante.

La méthode de fractionnement classique consiste à déterminer une partition des monômes m_i en k classes disjointes.

La méthode redondante consistera à déterminer un recouvrement des monômes de F' tel que chaque monôme soit dans 2 parties distinctes au moins.

Exemple :

$$F' = m_1 + m_2 + m_3 + m_4$$

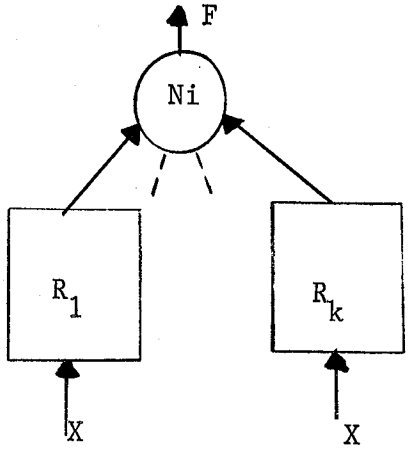
on écrira par exemple :

$$F' = (m_1 + m_2) + (m_2 + m_3) + (m_3 + m_4) + (m_4 + m_1)$$

Nous appellerons F_1, F_2, ..., F_k les sommes de monômes ainsi définies.

LEMME

Soit R_1, R_2, ..., R_k les réseaux réalisant F_1, ... , F_k indépendemment les uns des autres.



Considérons une valeur X_0 quelconque des entrées X telle que $F(X_0) = 0$ ou encore $F'(X_0) = 1$

Un des monômes au moins de F' est alors égal à 1, et par construction des f_i, 2 fonctions F_{i_1} et F_{i_2} sont égales à 1.

Supposons alors une défaillance de R_{i_1} seul, telle que $F_{i_1}(X_0)$ prenne la valeur fausse 0; nous avons alors :

$$F(X_0) = F_1(X_0) + \ldots + F_{i_1}(X_0) + F_{i_2}(X_0) + \ldots + F_k(X_0)$$

$$= F_{i_2}(X_0) = 1$$

$F'(X_0)$ est donc correct malgré l'erreur donc également F.

Nous dirons que le circuit total corrige les erreurs $0 \rightarrow 1$ de F.

Corollaire :

(1) Si nous répétons l'opération sur les F_i, nous corrigeons les erreurs $0 \rightarrow 1$ sur ces fonctions donc les erreurs $1 \rightarrow 0$ sur F.

(2) Si les deux dernières couches d'opérateurs sont supposées fiables, on corrige toute erreur sur F.

Remarque :

Rien a priori ne permet d'affirmer que les réseaux obtenus seront moins chers que ceux obtenus par redondance pure.

Nous remarquerons simplement que les formes minimales en somme de produit constituent déjà un recouvrement redondant des points de la fonction, c'est-à-dire qu'on passe d'une partition des monômes à un double recouvrement des monômes, le nombre de points dans chaque classe n'augmente pas beaucoup (en général beaucoup moins de 2 fois).

Exemple :

$$F = B'C + CD + A'BC' + AD'$$

le réseau de coût arborescent optimum non redondant est de 12.

Le réseau redondant est lui de coût 26. Si on triplait le minimum et qu'on prenne la majorité on trouverait :

$$3 \times 12 + 6 = 42$$

Conséquence :

Dans la synthèse de la $i^{ème}$ fraction de f, on n'a pas besoin de couvrir les points couverts déjà deux fois dans deux f_{j_1} et f_{j_2} distincts, avec $j_1, j_2 < i$, en fait on remplacera alors la borne inférieure $\underline{f_i}$ de f_i par

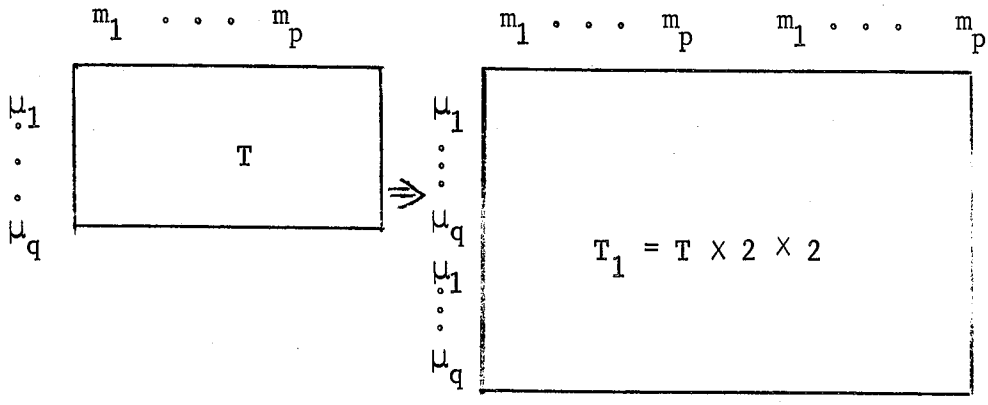
$$\underline{f_i} \times \Big(\sum_{\substack{j_1 \neq j_2 \\ j_1 < i \\ j_2 < i}} f_{j_1} \cdot f_{j_2} \Big)'$$

la nouvelle fonction f_i étant rendu plus incomplète sera donc réalisé par un réseau de coût au plus égal au précédent.

On itère une fois ce procédé, et quand on traite les fonctions de niveau supérieur à 3 on reprend la méthode développée précédemment en remplaçant $\underline{f_i}$ par $\underline{f_i} \cdot \sum_{j<i} f_j)'$.

Programmation :

La méthode a été programmée d'une façon très simple en remarquant qu'on peut rendre les écriture des fonctions des couches 1 et 2 redondantes simplement en doublant le tableau T dans ses deux dimensions :



si on partage T en k parties sans changer l'ordre des monômes; on obtient les conditions précisées précédemment.

Les résultats obtenus sont très variables suivant les fonctions étudiées. Dans les cas les plus mauvais, on obtient une augmentation, du coût de 4 comme le laissent prévoir les dimensions du tableau.

Dans certains cas par contre, l'amélioration est plus faible. Il existe des fonctions dont le minimum est sous forme partiellement redondante.

Exemple 1

$$f = ac + bc + ad + bd + ec + ef + cf$$

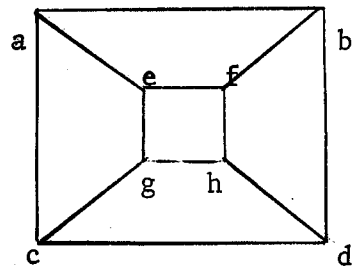
s'écrit sous la forme minimale suivante :

$$f = (a+b)(c+d) + (e+b)(c+f)$$

Exemple 2

$$f = ab + ae + ac + ef + eg + fh + gh + bf + hd + bd + cd + cg$$

dont le réseau associé est le suivant :



cette fonction possède une forme minimale :

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

ou chaque terme couvre une maille. En ajoutant les formes (a+g)(e+c) et (f+d)(b+h) on obtient une couverture des 6 mailles donc totalement redondante.

L'augmentation du coût est :

$$\frac{\Delta c}{C_{min}} = \frac{8}{16} = \frac{1}{2}$$ au lieu de 1 dans les méthodes classiques.

7 - CONCLUSION

Nous avons développé deux méthodes dont les résultats sont très différents.

La première a l'avantage d'être très bien adaptée au traitement machine. Par ailleurs, la simplicité relative de la théorie a permis de développer des algorithmes conduisant au coût minimum. En regard de ces avantages, nous trouvons des temps de calcul longs, dus au fait que le calcul est dans une large mesure indépendant de la fonction traitée.

La deuxième méthode est tout au contraire plus lourde à programmer, et d'une théorie plus difficile. Par contre, elle a permis d'utiliser les propriétés algébriques des fonctions à traiter, et par cela, simplifier considérablement les calculs. Elle ne permet malheureusement pas en général de trouver des réseaux de coût minimum.

Malgré leurs difficultés théoriques les méthodes de fractionnement seront en fait préférées aux premières à cause essentiellement de leur rapidité.

DEUXIEME PARTIE

CHAPITRE 3

SYNTHESE DES FONCTIONS GENERALES

A - INTRODUCTION

Le problème posé est le suivant : Etant donnée une fonction booléenne générale (incomplète ou non), c'est-à-dire p fonctions booléennes simples, on cherche un réseau comportant le moins d'opérateurs possible, dont les sorties sont compatibles avec les p fonctions.

Remarquons qu'en général le problème n'est pas résolu par la synthèse séparée des diverses fonctions, puisque certains résultats intermédiaires de l'une pourront servir pour les autres.

Nous utiliserons cependant les résultats obtenus dans la synthèse des fonctions simples pour étudier le problème général. Nous retrouverons en particulier les deux types de méthodes : par composition et par fractionnement.

PRINCIPES GENERAUX

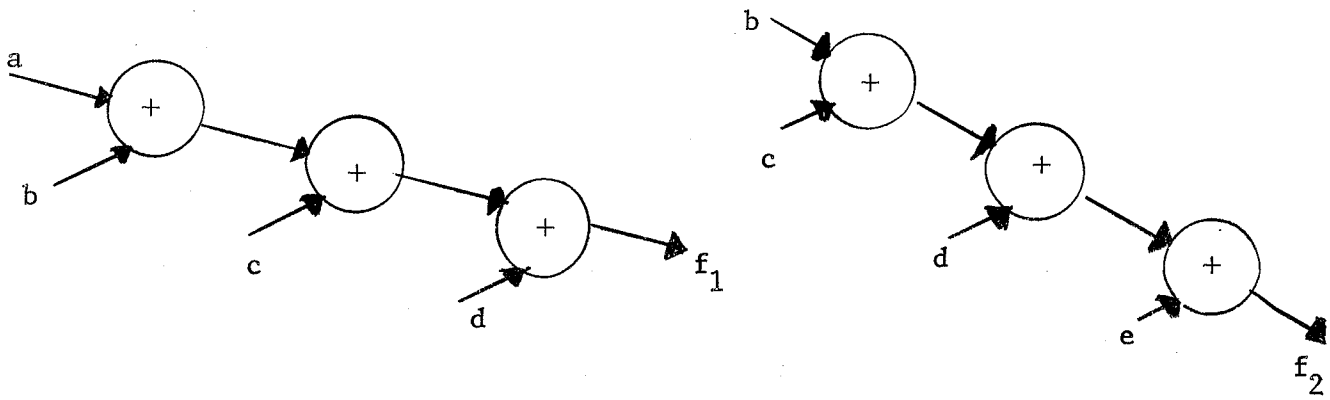
Nous développerons un certain nombre d'algorithmes pouvant se ranger dans les classes suivantes :

ⓐ Algorithmes de synthèse_séparée_:

Nous réalisons séparément les diverses fonctions booléennes; nous cherchons ensuite les fonctions intermédiaires communes aux divers réseaux. Cette méthode ne conduit jamais à de bons résultats : il peut arriver que des fonctions même très voisines n'aient aucune partie commune dans leurs réseaux :

Exemple :

$f_1 = a + b + c + d$, $f_2 = b + c + d + e$



on ne peut trouver aucune partie commune aux deux réseaux dessinés.

Cette méthode servira essentiellement de référence pour juger de la qualité des autres méthodes.

b) Algorithme de synthèse successive :

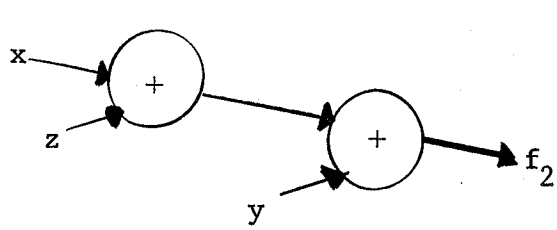
Nous pouvons choisir un ordre pour les fonctions à réaliser; ayant réalisé la première, nous cherchons un réseau réalisant la deuxième en utilisant le plus possible les résultats intermédiaires de la première, puis la troisième ayant la première et la deuxième et ainsi de suite jusqu'à épuisement des fonctions à traiter.

L'ordre dans lequel on traite les fonctions est évidemment important. Si par exemple on réaliser les deux fonctions :

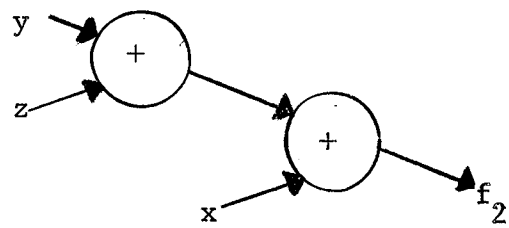
$$f_1 = x + y + zt \qquad \text{et} \qquad f_2 = x + y + z$$

si on réalise f_1 en premier, on pourra toujours réaliser f_2 en ajoutant simplement z à f_1.

Si par contre on réalise d'abord f_2 par les réseaux suivants :



ou



on ne peut pas utiliser de résultats intermédiaires pour réaliser f_1.

c) Algorithme de synthèse alternée

Cet algorithme est une variante du précédent : après application de la méthode (b) on reprend la synthèse de f_1 en utilisant les résultats intermédiaires de f_2, f_3, ..., f_p (y compris ceux communs à un f_i et à f_1 au premier tour). On itère le procédé tant que le coût ainsi défini décroît.

(d) Algorithme de synthèse composée

Cet algorithme consite à introduire une fonction dite composée par introduction de variables auxiliaires. Dans le cas de 2 fonctions f_1 et f_2, on considèrera :

$$F = \alpha f_1 + \alpha' f_2$$

α étant une variable auxiliaire.

Une méthode quelconque nous permet de déterminer un réseau R réalisant F. Si dans R' nous donnons à α les valeurs 0 et 1 nous obtenons respectivement f_2 et f_1; mais si α est une constante en général le réseau R se simplifie, on appelle R_1 et R_2 les réseaux obtenus après simplification. Ces réseaux ont comme partie commune toute partie invariante dans cette simplification, c'est-à-dire toute partie indépendante de α. Cet algorithme nous permet de déterminer a priori les parties communes aux deux réseaux.

(e) Algorithme de synthèse simultanée

Cet algorithme qui utilise une méthode par composition consiste à former des fonctions proches à la fois de f_1, f_2, ..., f_k en utilisant des critères adéquats de suppressions de fonctions intermédiaires.

Remarque :

Ces algorithmes donnent des résultats très variables suivant la méthode (fractionnement ou composition) utilisée. Nous rencontrerons en particulier des difficultés pratiques dues au temps de calcul qui sera considérablement augmenté par rapport aux méthodes précédentes.

B - SYNTHESE DES FONCTIONS GENERALES PAR COMPOSITION

1 - INTRODUCTION

Nous pouvons reprendre le principe des méthodes par composition, nous composerons de toutes les façons possibles les variables entre elles pour obtenir des fonctions intermédiaires φ_i. Nous envisagerons divers critères d'élimination des φ_i permettant de simplifier le problème. Nous continuerons la composition jusqu'à obtenir une fonction φ_i compatible avec chaque fonction donnée f_j. Comme précédemment nous utiliserons une représentation en colonnes.

2 - OPERATEURS UTILISES

Les résultats obtenus pour les fonctions simples, nous montrent qu'il sera impossible pour des fonctions générales de traiter des problèmes de synthèse par des opérateurs non croissants. Nous utiliserons donc uniquement des opérateurs croissants dans cette étude, et principalement l'opérateur majorité.

3 - REPRESENTATION DES FONCTIONS

Nous avons vu dans la première partie qu'il est possible de représenter les fonctions intermédiaires (variables comprises) par leurs valeurs sur les points caractéristiques de première et deuxième espèce de la fonction à réaliser. Ici nous devrons spécifier les valeurs sur les points caractéristiques de chacune des fonctions à traiter. Nous voyons ici apparaître une difficulté "d'encombrement" : le problème posé nécessite un codage plus lourd que précédemment. Lors de l'exploitation en machine des résultats le nombres de mémoires utilisées sera plus important que dans le cas particulier d'une fonction.

Remarque :

Nous ne pouvons pas utiliser ici la méthode du pivot décrite précédemment puisqu'il y a diverses fonctions à réaliser.

4 - SYNTHESE SIMULTANEE

Cette méthode est la généralisation directe des méthodes pour fonctions simples. Comme précédemment nous composerons de toutes les façons possibles les variables entre elles, en éliminant celles qui ne sont pas intéressantes. Nous distinguerons encore deux types de critères d'élimination :

ⓐ Critère d'élimination rigoureux :

On élimine une colonne φ si elle ne peut apparaître dans aucun minimum. Ceci se produira s'il existe une colonne Ψ de coût moindre qui peut remplacer partout φ, sans changer les résultats finaux, c'est-à-dire si pour tout i, on a :

$$\text{Maj}(\varphi, \Psi, f_i) = \Psi$$

Remarque :

On peut étendre ce critère en remarquant que si pour tout i on peut trouver Ψ_i qui élimine φ dans la synthèse de f_i et si de plus le coût de φ est supérieure au coût total des Ψ_i :

$$\begin{cases} \forall_i, \exists \Psi_i, \text{Maj}(\varphi, \Psi_i, f_i) = \Psi_i \\ \text{et } \Sigma \text{ coût } \Psi_i < \text{coût de } \varphi. \end{cases}$$

Ces critères d'élimination rigoureux sont très stricts. En fait, la proportion de colonnes éliminées sera beaucoup plus faible que dans la synthèse d'une fonction unique. Nous ne pourrons pas espérer compte tenu des résultats antérieurs, trouver le réseau de coût minimum réalisant une fonction générale.

ⓑ Critères d'élimination approchée :

Ces critères utilisent la distance entre deux colonnes définie précédemment. On définira la distance composée D d'une colonne φ à la fonction générale F donnée par

$$D - (\varphi, F) = f(d(\varphi, f_1), \ldots, d(\varphi, f_k))$$

g étant une fonction croissante. En particulier on considérera :

$$D1 = \sum_{j=1}^{k} d(\varphi, f_j)$$

et

$$D2 = \prod_{j=1}^{k} d(\varphi, f_j)$$

cette dernière étant d'ailleurs préférable puisqu'elle est nulle si φ est une des fonctions f_j et d'autre part, elle permettra d'éliminer les nombreuses colonnes à égale distance de deux f_j.

Remarque :

Nous serons encore plus que précédemment, placé devant le choix entre un critère d'élimination puissant qui diminue le nombre de colonnes à calculer et un critère plus strict qui assure la convergence de la méthode.

ⓒ Mise en oeuvre :

La méthode programmée comme les précédentes en Algol sur IBM 7044 a donné des résultats décevants.

Comme dans toutes les méthodes de ce type, le nombre de colonnes nouvelles à envisager à chaque pas varie sensiblement comme la puissance $k^{ième}$ du nombre de P de colonne en place, si k est le nombre d'entrées des opérateurs du problème. Avec, l'opérateur majorité qui comporte 3 entrées nous avons à envisager C_p^3 nouvelles colonnes.

Nous n'avons pas pu en fait trouver un critère qui permette d'éliminer beaucoup de colonnes, les temps de calcul seront donc grands.

Exemple :

Deux fonctions quelconques de 4 variables réalisées en opérateur majorité à l'aide des variables, variables complémentées et constantes 0 et 1.

Temps moyen : 3 minutes 15 secondes.

On constate en modifiant les critères d'élimination, que :

- soit les 2 fonctions sont réalisées par des réseaux pratiquement disjoints (méthode séparée)
- soit les deux réseaux utilisant un même sous réseau commun mais alors le nombre de colonnes conservées (donc le temps de calcul) est grand.

Conclusion :

Cette méthode qui n'est pas a priori mauvaise en elle-même est inférieure à celles exposées par la suite.

5 - SYNTHESE SUCCESSIVE

La méthode de synthèse simultanée nous oblige à conserver un grand nombre de colonnes intermédiaires. En particulier dans le cas de la synthèse de 2 fonctions, on doit choisir entre :

- conserver les colonnes intermédiaires entre les deux fonctions à réaliser, et ces colonnes sont très nombreuses (si f_1 et f_2 diffèrent par p points, il y a $C_p^{\text{entier } (p/2)} \# 2^p$ colonnes à égale distance entre f_1 et f_2),
- Eliminer ces colonnes et risquer de réaliser de manière complètement séparée les deux fonctions.

Pour éviter ces ennuis nous pouvons utiliser la méthode suivante :

On réalise <u>successivement</u> les diverses fonctions : la fonction f_1 est traitée comme si elle était seule. On obtient un réseau R_1 qui la réalise, et φ_1, φ_2, ..., φ_p des fonctions intermédiaires de R_1. Dans la synthèse de f_2 on considère ces fonctions comme des variables : on supposera qu'il y a n+p variables et on réalise f_2 par la même méthode de composition que f_1. Plus généralement pour la synthèse de f_i, on considérera comme des variables les colonnes intermédiaires nécessaires à la synthèse de f_1 ... f_{i-1}.

<u>Remarques</u> :

(a) Considérer une colonne intermédiaire φ_i comme une variable revient à dire que son coût est ramené à zéro.

(b) Dans la synthèse de f_i, on peut ne considérer que les colonnes non améliorables (comme elles ont toutes le coût zéro, cela revient à dire qu'il n'existe pas Ψ_j tel que :

$$\text{Maj } (\Psi_j, \varphi_j, f_i) = \Psi_j$$

(c) <u>Complexité du problème</u> :

Dans la synthèse de f_i sont considérées comme variables les anciennes variables et les fonctions intermédiaires du réseau réalisant f_1, f_2, ..., f_{i-1}. Ces nouvelles variables sont à composer entre elles de toutes les façons possibles. Le nombre de compositions varie donc à peu près <u>exponentiellement</u> avec le nombre de variables. En fait, si les premières fonctions f_1 ... f_{i-1} sont réalisées par des réseaux de coût élevé, on ne pourra pas envisager la synthèse de f_i par le procédé décrit.

(d) Dans une méthode approchée on peut s'imposer de ne conserver dans la synthèse de f_i, que les φ_j dont la distance à f_i est faible. On ne prendra par exemple que les 10 plus proches, ou ceux dont la distance à f_i n'excède pas une certaine valeur donnée. On évitera ainsi l'écueil signalé en (c) .

MISE EN OEUVRE

Nous avons programmé cette méthode en utilisant comme opérateur majorité. L'algorithme de synthèse de fonction simple est du à Monsieur Lustmann (méthode "Majmax"). Cet algorithme a été choisi car c'est lui qui nécessite le moins de colonnes intermédiaires.

Contrairement à la méthode précédente, nous avons obtenus ici de bons résultats.

Exemple :

Synthèse en majorité de 3 fonctions de 4 variables :

- durée du calcul : 57 secondes
- coût total : 10
- coût en synthèse séparée : 12
- nombre maximum de colonnes présentées simultanément en machine : 20.

RESULTATS

Nous avons essayé cette méthode sur quelques exemples.

Pour des triplets de fonctions complètes de 4 variables, nous avons obtenus pour un coût moyen de 10, une amélioration de 10 % en moyenne par rapport à la synthèse séparée. En fait les résultats sont trop peu nombreux et les coûts trop faibles pour qu'on puisse réellement faire une étude statistique.

Pour des fonctions de 5 variables, les coûts croissants fortement, il a été impossible de traiter le problème.

6 - SYNTHESE ALTERNEE

ⓐ Cette méthode est une variante de la méthode de synthèse successive. Elle consiste, pour s'affranchir du choix d'un ordre sur les f_i, à reprendre plusieurs fois le problème. Par exemple, si F_1 et F_2 sont deux fonctions à réaliser, on réalise d'abord F_1 puis F_2 en utilisant au mieux les résultats de F_1. On reprend ensuite la synthèse de F_1 en utilisant les résultats de F_2. Il faut alors supprimer de la liste des colonnes tous les φ_i qui ont servi à la synthèse de F_1 seul au premier pas (on gardera ceux qui servent à F_2). On itère le procédé en considérant alternativement F_1 et F_2.

La méthode se généralise facilement à plus de 2 fonctions. On n'a pas ici à choisir un ordre de traitement : on considèrera par exemple pour 3 fonctions, la succession suivante :

$$f_1,\ f_3,\ f_2,\ f_3,\ f_2,\ f_1,\ \text{etc ...}$$

On pourra par exemple à chaque itération choisir au hasard la fonction à traiter.

ⓑ Arrêt des itérations

Rien ne permet de dire que la méthode converge vers une solution minimum. Cette méthode est simplement un moyen d'obtenir un grand nombre de solutions à peu de frais. On arrêtera les itérations quand par exemple, on aura dépassé un temps fixé ou quand on n'obtient plus d'amélioration du coût total depuis plusieurs itérations.

ⓒ Résultats

La méthode programmée comme la précédente donne des résultats intéressants.

En reprenant l'exemple précisé dans la première méthode, nous obtenons par exemple, après 3 itérations une amélioration du coût de 10 %.

Nous remarquons cependant que le temps augmente beaucoup (il est à peu près proportionnel au nombre d'itérations). Or nous avons vu que le temps était déjà relativement important dans la méthode initiale. Nous ne pourrons guère faire plus de 3 ou 4 itérations dans des temps machine raisonnables.

Remarquons enfin, que cette méthode nous a obligé à alourdir considérablement les programmes. Nous sommes en particulier obligés de noter les antécédents et les descendants de toute colonne nouvelle φ afin de faire les suppressions indispensables entre chaque itération.

CONCLUSION

Les méthodes de composition se révèlent peu adaptées à la synthèse des fonctions générales. Nous pouvons dégager deux raisons à ceci :

(a) En premier lieu, la représentation en colonne est considérablement alourdie :

- nous ne pouvons plus représenter une fonction sur ses seuls points caractéristiques,
- nous ne pouvons plus utiliser la représentation du pivot qui donnait lieu à des tests d'élimination très simples.

(b) Outre cet inconvénient formel, la méthode par composition ne marche en pratique que si beaucoup de colonnes intermédiaires sont éliminées. Pour une fonction simple nous avons trouvé des tests simples et efficaces qui permettent d'éliminer de nombreuses colonnes sans pourtant nous éloigner trop du coût minimum. Dans le cas général par contre, des tests d'élimination trop puissants conduisent pratiquement à une synthèse séparée.

C - SYNTHESE DES FONCTIONS GENERALES PAR FRACTIONNEMENT

1 - INTRODUCTION

Comme dans les méthodes par composition, on peut envisager une méthode de synthèse séparée. Dans le cas de deux fonctions f_1, f_2 par exemple, on réalise f_1 et f_2 séparément et on cherche parmi les fonctions intermédiaires celles qui sont présentes deux fois. Ici, nous voyons une difficulté apparaître, il est difficile de tester l'égalité de deux fonctions écrites sous forme polynomiales. En fait, on ne trouve facilement à la main que des fonctions ne dépassant pas 3 monômes.

D'autre part, les fonctions intermédiaires déterminées dans ces méthodes sont toujours données sous forme complète. Alors deux fonctions intermédiaires φ et Ψ égales dans le domaine de définition de f_1 et f_2 et différentes ailleurs ne seront jamais réunies.

Pour ces différentes raisons, nous serons amenés à "doubler" notre représentation polynomiale par un codage en colonnes.

Choix d'une méthode.

La méthode de fractionnement simple ne se prête pas à la synthèse de fonctions générales. Nous allons illustrer ceci par un exemple.

Soit à réaliser la fonction $f = xy + x'y'$ en opérateurs Ni à 1 ou 2 entrées (dont les coûts sont égaux). Un premier fractionnement nous conduit aux deux intermédiaires suivants :

$$f_1 = xy' \qquad f_2 = x'y$$

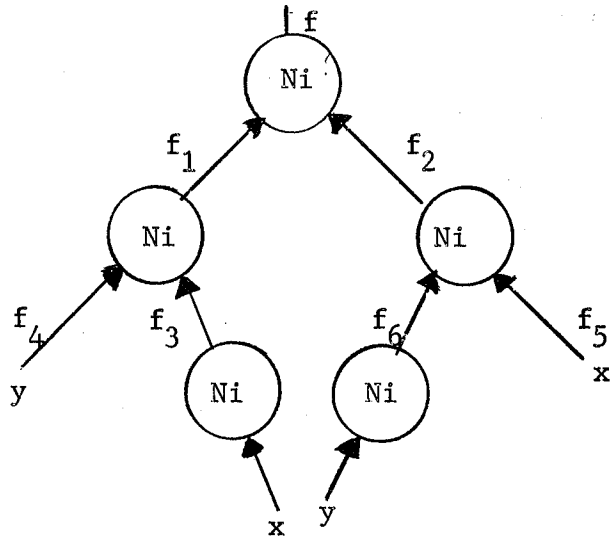
(nous sommes ici ramenés à un problème de synthèse de la fonction générale $F = \left| \begin{matrix} f_1 \\ f_2 \end{matrix} \right|$ par la suite on écrit :

$$f_1' = x' + y \qquad f_2' = x + y'$$

et on pose

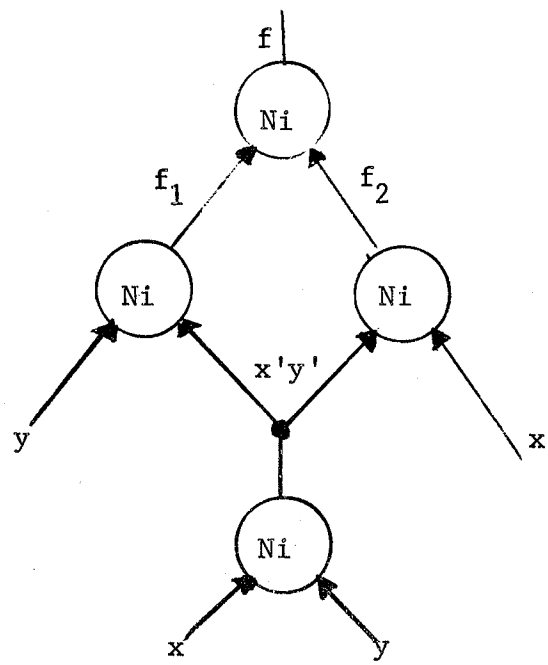
$$f_3 = x' \qquad f_4 + y \qquad f_5 = x \qquad f_6 = y'$$

le réseau obtenu est alors le suivant :



on ne peut ici réunir aucun noeud.

Pourtant, on peut sans changer f_1 remplacer f_3 par x' y' (de même coût et de même f_6 par x'y').



Il est cependant impossible, sans connaître le réseau de deviner que f_3 et f_6 peuvent être réunis.

Par contre, dans la méthode des bornes on aura :

$$f_1 = xy' + \emptyset\ (xy' + x'y)$$
$$f_2 = x'y + \emptyset\ (xy' + x'y)$$

en complémentant f_1 et f_2 on obtient :

$$f_1' = xy + x'y' + \emptyset\ (x' + y)$$
$$f_2' = xy + x'y' + \emptyset\ (x + y')$$

ce qui donne par partage de la borne inférieure :

$$f_3 = xy + \emptyset\ (x'+y) \qquad f_4 = x'y' + \emptyset(x'+y)$$
$$f_5 = xy + \emptyset\ (x+y') \qquad f_6 = x'y' + \emptyset(x+y')$$

On obtient toujours comme détermination

$$f_3 = y,\ f_5 = y$$

mais on obtient, outre :

$$f_4 = x',\ f_6 = y'$$

la détermination suivante :

$$f_4 = x'y' = f_6$$

Dans cette dernière méthode, on peut théoriquement trouver toutes les confusions de noeuds possibles. Dans tout ce qui suit, seule sera utilisée cette méthode.

OPERATEURS

La méthode utilisée ici n'a été envisagée actuellement qu'en opérateurs Ni.

2 - SYNTHESE SUCCESSIVE

(a) Principe

Dans la méthode décrite dans la première partie, nous avons toujours cherché pour chaque fonction intermédiaire f_i, s'il existe une variable comprise entre les bornes. Il vient alors naturellement l'idée de chercher si une variable ou une fonction déjà réalisée est comprise entre les bornes de f_i. On se heurte immédiatement à la difficulté de reconnaissance d'une inégalité de la forme :

$$\underline{f_i} \leq \varphi \leq \overline{f_i}$$

sur des polynômes si φ n'est pas une variable.

(b) Représentation des fonctions

Une fonction aura une double représentation : polynomiale et en colonne, nous ferons le fractionnement sur la forme polynomiale et les comparaisons sur les colonnes. Les colonnes seront définies partout où une des fonctions du problème est définie. (Nous écrirons évidemment un programme de conversion d'une forme à l'autre).

ALGORITHME

Toute fonction intermédiaire f_i sera traitée de la manière suivante :

(a) A partir de $f_i = \underline{f_i} + \emptyset\ \overline{f_i}$ on détermine par fractionnement k fonctions plus incomplètes :

$$g_j = \underline{g_j} + \emptyset\ \overline{g_j} \qquad 1 \leq j \leq k$$

en opérateurs Ni en particulier on fractionnera la borne inférieure de f'_j et on gardera la borne supérieure.

Les fonctions g_j seront dites de même niveau, et f_i de niveau supérieur à g_j.

(b) Pour tous les j on regarde successivement s'il existe une variable ou une fonction intermédiaire déjà réalisée comprise entre les bornes de g_j. Le test se fait sur les colonnes : on calcule à chaque fois les colonnes de $\underline{g_j}$ et de $\overline{g_j}$.

Si on a pu trouver φ_j compatible avec g_j, on passe à la fonction suivante de même niveau.

Si on n'a pas trouvé de tel φ_j on itère le procédé en considérant g_j comme f_i.

Si pour tout j, on a trouvé un φ_j compatible avec g_j, on a du même coup trouvé un φ compatible avec f_i qui est :

$\varphi = \mathcal{O}(\varphi_1\ \varphi_2 \ldots \varphi_k)$ où $\mathcal{O}$ est l'opérateur utilisé pour le fractionnement.

La colonne φ est calculée à partir des φ_i et considérée comme une fonction déjà réalisée.

(c) Changement de niveau : la fonction φ étant trouvée, on itère le procédé en passant aux fonctions f_{ip} de même niveau que f_i. Si f_i était la dernière de son niveau on passe après calcul du φ correspondant, au niveau supérieur et ainsi de suite.

CAS DES OPERATEURS Ni

Si on utilise des opérateurs Ni, on utilise certaines propriétés qui améliorent les résultats.

(a) Quand on a trouvé φ_i compatible avec g_i, les points de cette fonction n'ont plus à être couverts par $g_{i+1} \ldots g_k$, autrement dit on peut remplacer pour tout $j (i \leq j \leq k)$. $\underline{g_j}$ par $g_j \cdot \varphi_i'$, ce qui diminuant la borne inférieure, augmente les chances de trouver φ_j compatible avec g_j.

(b) Si on trouve plusieurs φ_i compatibles avec g_i, on choisira le plus grand puisque d'après (a), on aura ainsi le g_j le plus incomplet possible.

(c) Quand on n'a pas trouvé φ_i compatible avec g_i, on peut chercher s'il existe φ_i compatible avec g_i, ce qui signifierait qu'il est inutile de fractionner g_i mais qu'il suffit de le complémenter.

MIS EN OEUVRE

Nous avons programmé cette méthode toujours en Algol sur 7044. Ici le choix du langage Algol s'est révélé très intéressant : ce langage en effet, accepte la récursivité ce qui nous a grandement simplifié la programmation de ce problème déjà très complexe.

Nous avons écrit un programme comportant 6000 unités syntaxiques. Il comporte comme tous les précédents des procédures de manoeuvre élémentaires en M.A.P. et des procédures de traitement algébrique des fonctions en Algol.

La procédure principale est récursive, elle traite le problème du fractionnement d'une fonction incomplète et celui de la recherche des fonctions compatibles avec les intermédiaires ainsi formés.

RESULTATS

Les résultats obtenus par ce programme sont nettement supérieurs à tous ceux obtenus précédemment pour illustrer ceci citons des exemples :

Exemple 1 -

Nous traitons 3 fonctions générales de 5 variables, à 2 composantes complètes :

	Synthèse séparée	Synthèse successive
F_1	88	75
F_2	51	41
F_3	70	62
Temps Total	2 minutes	2 minutes 30 secondes

amélioration du coût due à la méthode en moyenne : 0,85.

Exemple 2 -

Une fonction générale à 5 composantes, fonctions complètes de 6 variables d'un coût total en Ni à 2 entrées en 2 couches de 270 secondes environ. Le réseau obtenu comporte 141 opérateurs Ni et a été calculé en 2 minutes 30 secondes.

Un tel coût en méthode par composition ne pourrait être obtenu qu'en un temps de plusieurs heures, voire plusieurs jours (dans ces programmes le temps est fonction exponentielle du coût).

Exemple 3 -

Une fonction générale à 2 composantes incomplètes de 6 variables. Chaque fonction étant spécifiée environ sur 70 % des points. On a obtenu un coût de 42 en 30 secondes. (le coût obtenu en somme de produit pour les moins chères des bornes est approximativement de 65).

RESULTATS STATISTIQUES

Sur des fonctions complètes générales à 2 ou 3 composantes la méthode permet d'économiser 15 à 20 % du coût en synthèse séparée.

Si le nombre de composante augmente, le gain augmente aussi, puisque le nombre de fonctions intermédiaires disponibles est plus grand. Pour 5 composantes on arrive facilement à une économie de 30 %.

APPLICATION - PROBLEMES DE CODAGE

Dans les problèmes de synthèse des systèmes séquentiels on se heurte au problème du codage des états. Si par exemple dans les systèmes asynchrones, tous les codages possibles sont valables théoriquement, il n'en reste pas moins que le choix du codage influe sur le coût de la réalisation matérielle.

En particulier, on constate fréquemment que ce n'est pas le codage utilisant le moins de variables secondaires qui conduit au réseau le moins coûteux. Ceci nous amène à envisager l'algorithme suivant :

PRINCIPE

On fera choix d'un codage initial éventuellement très redondant. Ce codage donne lieu à une fonction générale incomplète définissant l'état suivant A_{t+1} et les sorties S de la machine en fonction de l'état actuel A_t et les entrées actuelles X_t :

$$(S, A_{t+1}) = F(A_t, X_t)$$

Parmi toutes les déterminations de F, nous pouvons, par la méthode de synthèse exposée précédemment, choisir celle conduisant au réseau de coût optimum. Nous aurons d'autant plus de choix dans les réseaux que la fonction F est plus incomplète, c'est-à-dire qu'il y a plus de variables d'état.

Si par exemple, on prend pour le codage initial, une variable par état, s'il y a n états, la fonction F est définie pour tout X sur n points et non spécifiée sur 2^n-n.

DETERMINATION de F

Nous choisirons par exemple une variable a_i par état A_i, a_i valant 1 quand la machine est dans l'état A_i. La fonction F est donc indéterminée sur tous les points ou deux a_i au moins valent 1. En prenant comme valeur 0 et 1 sur ces points on obtiendra respectivement $\underline{F}$ et $\overline{F}$.

Exemple :

	X	X'	X	X'
A	A	C	1	0
B	B	A	0	1
C	A	B	1	1
	état suivant.		sortie	

$$\underline{a_{t+1}} = ab'c'x + a'b'cx + a'bc'x'$$

$$\overline{(a_{t+1})'} = a'bc'x + ab'c'x' + a'b'cx'$$

$$\begin{cases} \underline{b_{t+1}} = a'bc'x + a'b'cx' \\ \overline{(b_{t+1})'} = ab'c' + a'bc'x + a'b'cx \end{cases}$$

$$\begin{cases} \underline{c_{t+1}} = a\,b'\,c'\,x' \\ \overline{(c_{t+1})'} = x + a'bc'x' + a'b'c\,x' \end{cases}$$

$$\begin{cases} \underline{S_t} = ab'c'x + a'bc'x' + a'b'c \\ \overline{(S_t)'} = a'bc'x + ab'c'x' \end{cases}$$

CHOIX D'UNE SOLUTION PARTICULIERE

Nous pouvons chercher éventuellement les réseaux tels que la sortie F_0 vérifie :

(1) $F_0(A_t\ X_t)$ indépendant de $A_{1t} \subset A_t$

En effet, si (1) est vérifiée, on montre facilement par récurrence que la sortie S_t ne dépend d'aucune des variables d'état de A_1 donc ces variables sont sans intérêt et peuvent être supprimées.

Or nous avons décrit un procédé pour déterminer les ensembles irredondants de variables nécessaires à l'écriture d'une fonction quelconque. Nous pouvons donc déterminer des codages ayant moins de variables internes que celui prévu initialement.

CAS DES SYSTEMES ASYNCHRONES

Dans un codage de système asynchrone l'ensemble des vecteurs booléens définissant le codage est tel que la distance de deux vecteurs est au minimum 1. Il est alors impossible de supprimer des composantes à ces vecteurs sans en confondre au moins deux. Ce que nous avons dit précédemment ne s'applique pas ici, directement. Il serait cependant possible de supprimer des variables en confondant des états transitoires qui vont au même état stables équivalents. Nous n'avons pas du fait envisagé ici ce problème, mais il semble que l'étude de cette question soit intéressante et puisse conduire rapidement à des résultats.

CONCLUSION

Notre méthode de synthèse semble apte à traiter des problèmes beaucoup plus généraux que ceux envisagés jusqu'ici. En particulier, le problème du choix optimum du codage en fonction du coût est intéressant et n'a semble-t-il été traité jusqu'à présent.

3 - SYNTHESE COMPOSEE

La méthode que nous allons exposer dans le cas des Ni a été conçue pourpermettre de fixer d'avance quelles seront les parties communes des réseaux des diverses fonctions.

(a) Définition

Si $f_1(X)$, $f_2(X)$, ..., $f_p(X)$ sont les fonctions à réaliser, on définira une composée $F(X, \lambda_1, \ldots, \lambda_k)$ où les λ_j sont des variables auxiliaires, par les conditions :

1 - $F(X, \lambda_1, \ldots, \lambda_k)$ est réalisable par les opérateurs du problèmes.

2 - Pour tout i, on peut trouver une valeur λ_{i_j} à donner à chaque λ_j pour que :

$$F(X, \lambda_{i_1}, \lambda_{i_2}, \ldots, \lambda_{i_k}) = f_j(X)$$

Exemple :

Si on a trois fonctions à réaliser f, g et h on écrira :

$$F(X,\alpha,) = \alpha\ \beta\ f(X) + \alpha\ \beta' g(X) + \alpha'\ \beta\ h(X)$$

et on aura :

$$f(X) = F(X, 1, 1)$$
$$g(X) = F(X, 1, 0)$$
$$h(X) = F(X, 0, 1)$$

Remarque :

La définition est valable que les fonctions soient complètes ou non.

On définira par exemple pour deux fonctions :

$$f = \underline{f} + \emptyset\ \overline{f} \qquad \text{et} \qquad g = \underline{g} + \emptyset\ \overline{g}$$

$$F = \alpha\ f + \alpha'\ g$$
$$= \alpha\ \underline{f} + \alpha'\ \underline{g} + \emptyset\ (\alpha\ \overline{f} + \alpha'\ \overline{g})$$

dont la borne inférieure est $\alpha\ \underline{f} + \alpha'\ \underline{g}$ et la borne supérieure $\alpha\ \overline{f} + \alpha'\ \overline{g}$. Si alors H est une fonction complète compatible avec F, on peut l'écrire :

$$H = \alpha\ h_1 + \alpha' h_2 \text{ et on a :}$$

$$\alpha\ \underline{f} + \alpha'\ \underline{g} \qquad \alpha\ h_1 + \alpha'\ h_2 \qquad \alpha\ \overline{f} + \alpha'\ \overline{g}$$

ou encore pour $\alpha = 1$:

$$\underline{f} \qquad h_1 \qquad \overline{f}'$$

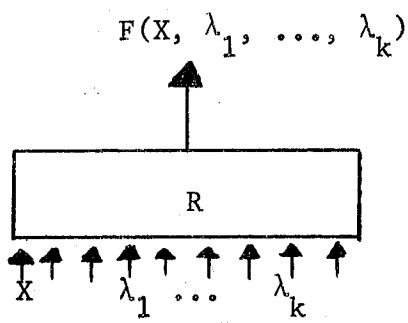
et de même pour $\alpha = 0$:

$$\underline{g} \qquad h_2 \qquad \overline{f}$$

autrement dit si $H(X, \alpha)$ est compatible avec $F(X, \alpha)$ alors $H(X, 1)$ est compatible avec $f(X)$ et $H(X, 0)$ avec $g(X)$.

(b) Synthèse de F

Par une méthode quelconque on réalise la synthèse de F(si F est complète on prendra une méthode de fractionnement simple, si elle est incomplète, on utilisera un fractionnement des bornes). Soit R le réseau obtenu.



ⓒ Simplification de R

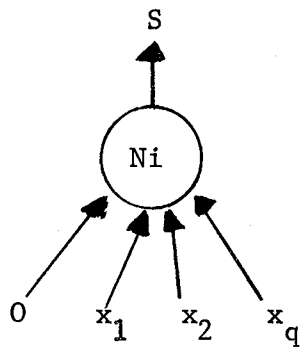
En donnant aux $\lambda_1 \ldots \lambda_k$ les valeurs $\lambda_{i_1} \ldots \lambda_{i_k}$ la sortie du réseau est $f_i(X)$. Mais en opérateurs Ni si on donne des valeurs particulières aux entrées d'un réseau, celui-ci peut toujours se simplifier.

LEMME 1

Si un opérateur Ni a une entrée identiquement nulle on peut supprimer cette entrée.

En effet, la sortie est :

$$S = 0' \cdot x_1' \quad x_2' \ldots x_q'$$
$$= 1 \cdot x_1' \quad x_2' \ldots x_q'$$

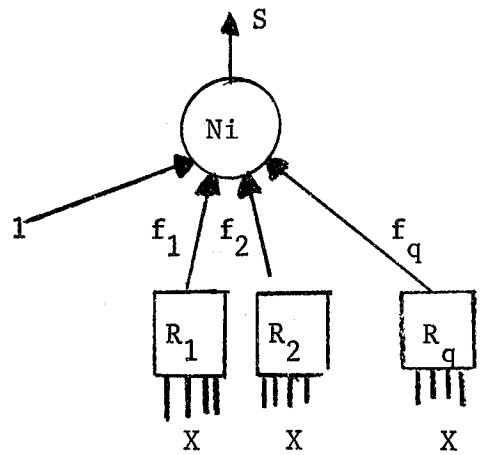


LEMME 2

Si une entrée d'un opérateur identique à 1 on supprime cet opérateur en remplaçant sa sortie par 1. Si le réseau en amont de l'opérateur ne sert qu'à former les entrées de cet opérateur, on le supprime également.

Soit en effet, un opérateur dont les entrées sont f_1, f_2 ...f_q et 1; la sortie est :

$$S \equiv (1)' \cdot f_1' \quad f_2' \ldots f_q'$$
$$\equiv 0 \cdot f_1' \quad f_2' \ldots f_q'$$
$$\equiv 0$$



Si les fonctions intermédiaires des réseaux R_1, R_2, ..., R_q ne servent pas ailleurs, on pourra les supprimer, puisque S est indépendant des f_1, ..., f_q.

Conséquence :

R étant connu on détermine facilement les réseaux R_1 ... R_p, en donnant des valeurs particulières aux variables auxiliaires et en simplifiant ensuite.

ⓓ Parties communes aux Ri

Les réseaux R_i dérivant tous d'un même réseau R en donnant aux variables λ_i des valeurs particulières, tout sous réseau de R indépendant des λ_i peut être partie commune.

Exemple

Soit à réaliser deux fonctions f et g complètes en opérateurs Ni à 3 entrées. On considère :

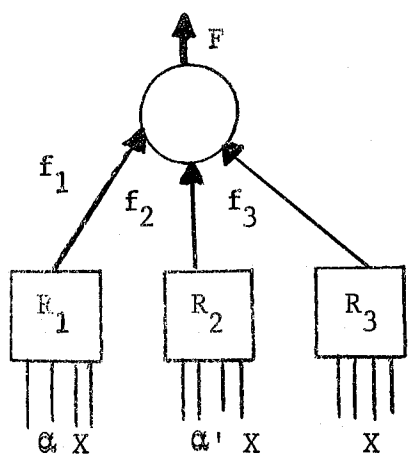
$$F = \alpha\, f + \alpha'\, g \text{ et son complément}$$

$$F' = \alpha\, \varphi_1 + \alpha'\, \varphi_2 + \varphi_3$$

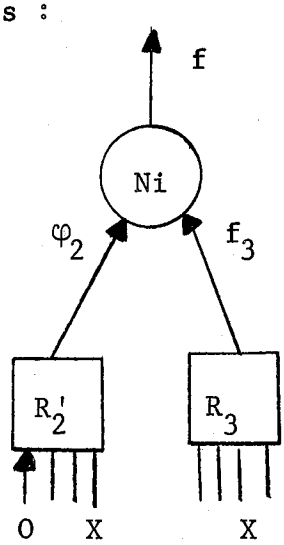
si $\varphi_3 \neq 0$ on fractionne en 3 parties :

$$f_1 = \varphi_1 \qquad f_2 = \alpha'\, \varphi_2 \qquad f_3 = \varphi_3$$

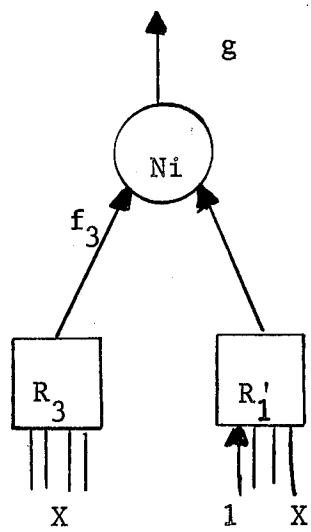
On itère le procédé en fractionnement f_1 f_2 et f_3. On obtient un réseau R dont la structure est la suivante :



En donnant à α les valeurs 0 et 1 on obtient respectivement les deux réseaux suivants :



et



dont la partie commune est R_3.

Remarque :

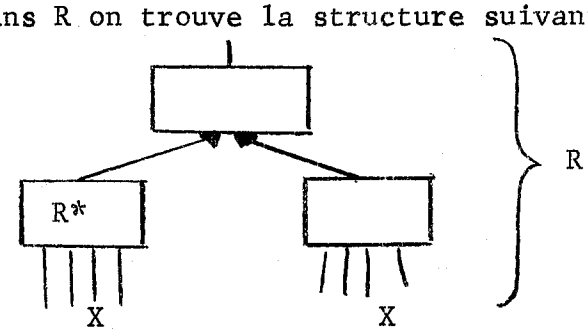
Dans le cas de 2 fonctions on peut énoncer le théorème suivant:

THEOREME

Tout sous réseau R* de R indépendant des variables auxiliaires et qui n'est pas sous réseau d'un sous réseau R** de R dépendant de ces variables, est partie commune.

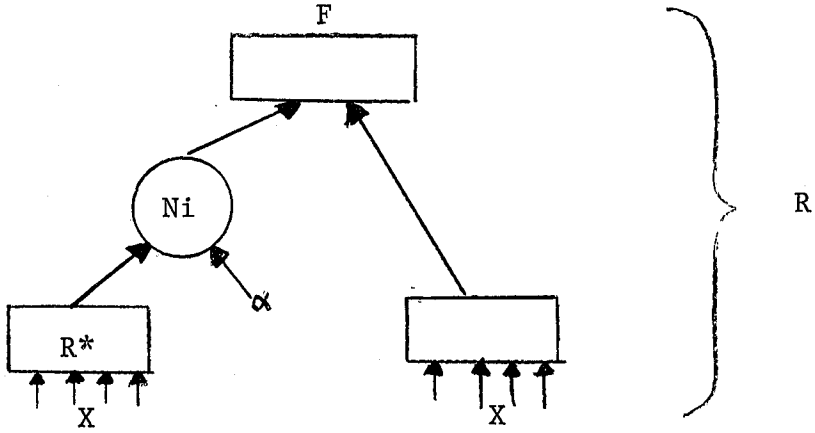
Exemple :

Si dans R on trouve la structure suivante :



le réseau R* se retrouvera intégralement dans les deux réseaux R_1 et R_2.

Si par contre on trouve la structure suivante :



où R* est indépendant de X, mais en amont d'un opérateur dont une entrée est . R* se retrouve dans R(α = 0) mais il est absent dans R(α = 1).

ⓔ Conséquence

On peut envisager une méthode de fractionnement qui respecte les parties communes possible :

- quand on a à traiter F = α f + α' g, on fractionnera une base première de F' :

$$F = \alpha\,\varphi_1 + \alpha'\,\varphi_2 + \varphi_3$$

en respectant autant que possible la partie φ_3.

Mise en oeuvre :

Cette méthode est facilement programmable : il suffit d'ajouter aux programmes précédents les procédures permettant de composer des fonctions et de simplifier le réseau obtenu par synthèse. Nous n'avons en fait écrit de programme que dans le cas de réseaux arborescents, réseaux pour lesquels les simplifications se font aisément en machine.

Remarque :

La simplification du réseau R obtenu est très facile sans l'aide d'ordinateur, il suffit en fait de dessiner le réseau pour que les simplifications soient évidentes. Pour les problèmes de synthèse traitables en machine, les réseaux restent assez simples pour être modifiés à la main.

Exemple :

$f = x + y + z$ $\qquad$ $g = y + z + t$

$F = x + t + (y+z)$

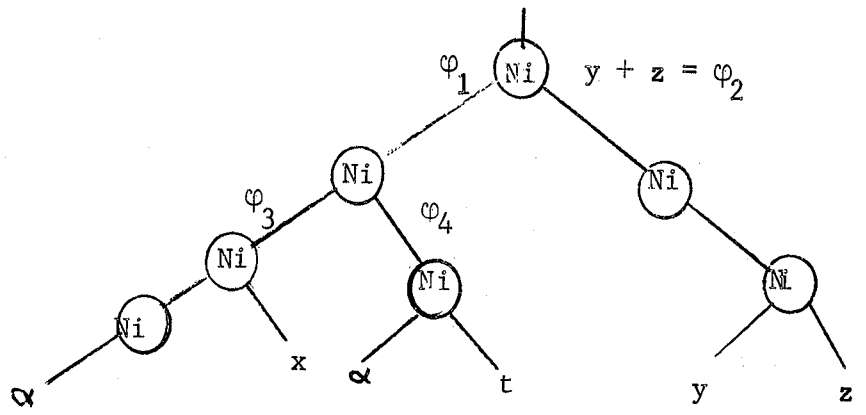
on peut utiliser une méthode en fractionnement simple :

$\varphi_1 = x + 't$ $\qquad$ $\varphi_2 = y + z$

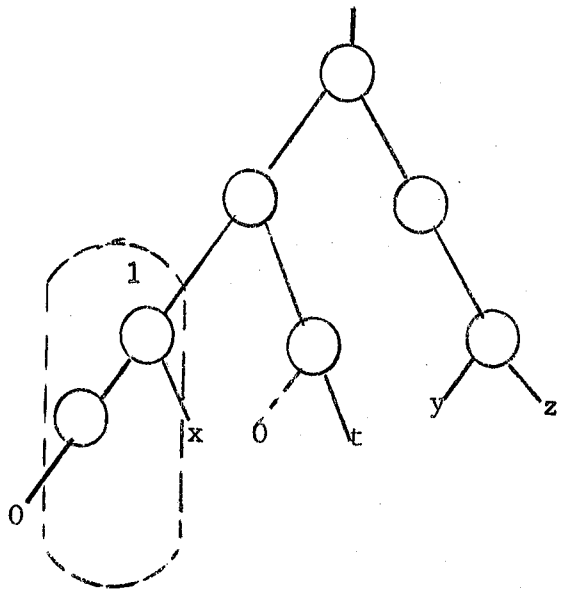
$\varphi_1' = x' + t'$

$\varphi_3 = x'$ $\qquad$ $\varphi_4 = t'$

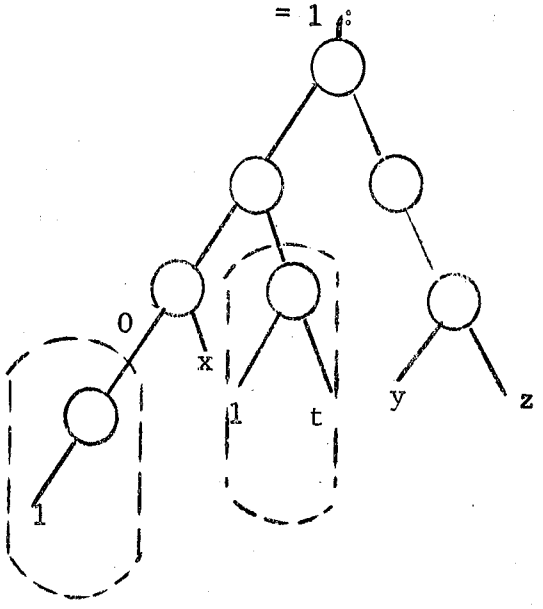
ce qui donne le réseau R suivant :



$= 0$:

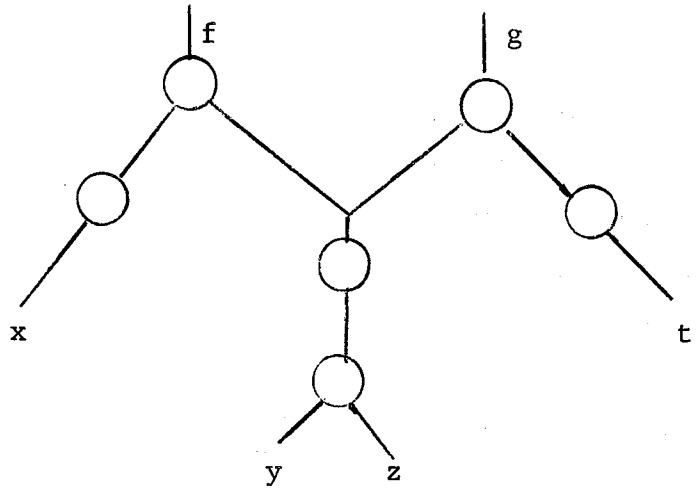


$= 1$:



Les parties entourées d'un trait pointillé étant à supprimer.

c'est-à-dire en définitive le réseau :



Conclusion :

Nous constatons que dans un cas semblable il est pratiquement impensable de rater le réseau minimum alors qu'en synthèse successive on a seulement une chance sur 3 de former y + z dans la première fonction à traiter. La méthode est donc manifestement intéressante.

RESULTATS :

Nous avons programmé cette méthode en fractionnement simple. Le programme obtenu est relativement complexe du fait des procédures de traitement des réseaux arborescents (7000 unités syntaxiques). Nous avons obtenus en moyenne une amélioration de 10 % du coût par rapport à la synthèse séparée pour deux fonctions de 6 variables.

La méthode est peu efficace par rapport à la précédente. Elle a cependant l'avantage d'être dérivée d'une méthode de fonctions simples, ce qui simplifie les problèmes de programmation.

4 - CONCLUSION

Comme dans le cas des fonctions simples, nous avons constaté ici que les méthodes de synthèse par fractionnement sont supérieures à celles par composition, tout au moins quand à la rapidité et à la souplesse d'emploi. Nous remarquons cependant qu'elles sont beaucoup plus délicates à programmer car les ordinateurs actuels sont peu adaptés au calcul symbolique. Enfin, rappelons que les méthodes par fractionnement ne s'appliquent que pour des opérateurs précisés, et non pour tous les opérateurs d'un type donné comme les méthodes par composition.

Il est certain qu'il y aurait encore des points à étudier dans ces méthodes; nous ne pouvons en effet, pas obtenir à coup sûr des réseaux de coût minimum, sauf dans des cas très restreints.

Par ailleurs, il y a encore des propriétés à découvrir dans le domaine des applications aux systèmes séquentiels, ainsi que dans le domaine de la synthèse de réseaux faibles.

Enfin, rappelons que nos hypothèses de départ étaient très théoriques, nous n'avons jamais étudié des problèmes tels que la standardisation des réseaux et des structures, ni des problèmes mettant en jeu un coût autre que le nombre d'opérateurs. Nos résultats sont plutôt une indication de choix de méthodes à envisager pour lesproblèmes précis à résoudre.

A N N E X E n° 1

PROGRAMMATION DE PROBLEMES BOOLEENS

ETUDE DU SYSTEME ECRIT PAR L'EQUIPE DE LOGIQUE DE GRENOBLE

E T U D E T H E O R I Q U E

1ère Partie

INTRODUCTION

Cette étude concerne un ensemble cohérent de programmes et sous-programmes permettant de traiter un grand nombre de problèmes logiques.

LANGAGE DE PROGRAMMATION

En abordant le problème notre objectif était triple :

a) - Efficacité

Désirant avoir des programmes efficaces, nous avions intérêt à utiliser un langage "proche de la machine", c'est-à-dire rapides et occupant une place aussi faible que possible en machine.

b) - Objectif pédagogique

Travaillant dans l'université, notre rôle était d'écrire des programmes compréhensibles par tous, spécialistes ou non. Pour satisfaire cet objectif un langage évolué genre Algol semblait adéquat.

c) - Maniabilité

Nos programmes concernant des recherches en cours, c'est-à-dire des résultats non encore définitifs. Pour cette raison nos programmes devaient être facilement modifiables. Là encore, un langage évolué semblait le meilleur.

Pour les deux dernières raisons à nos yeux essentielles, nous avons choisi le langage Algol 60.

Remarque :

Ce langage offre l'avantage de permettre une utilisation locale du Langage Machine, au sein de procédures. Cette solution de concilier les trois objectifs.

CHOIX D'UN CODAGE

Les quantités à coder ont été essentiellement des fonctions booléennes parfois, considérées sous forme polynomiales.

Pour l'efficacité des programmes, nous avons été amenés à considérer deux types de représentation machine, une représentation "polynomiale" proche de l'écriture littérale, et une représentation vectorielle.

1 - REPRESENTATION POLYNOMIALE

(a) Un polynôme booléen est représenté par un tableau entier T à une dimension et un nombre de monômes M. Chaque mémoire de T entre 1 et M (bornes comprises) représente un monôme.

(b) Monôme

Un monôme est représenté par une mémoire de la machine. A chaque variable x_i $1 \leq i \leq 18$ correspond à deux positions binaires dans la mémoire positions i et i+ 18.

La partie droite (position 1 à 18) contient les digits marquant la présence d'une lettre non complémentée dans le monôme.

La partie gauche (position 19 à 36) contient les digits marquant la présence des variables complémentées.

Exemple :

monômes x_1 x_2 x_3'

36	21 20 19 18	3 2 1
00	0 1 0 0	0 0 ..... 0 0 1 1

Remarque :

Le codage d'un monôme est indépendant du nombre total de variables du problème.

ⓒ Variable

Chaque variable est donc codée par un couple de deux digits :

x_i $\{0\ 1\}$

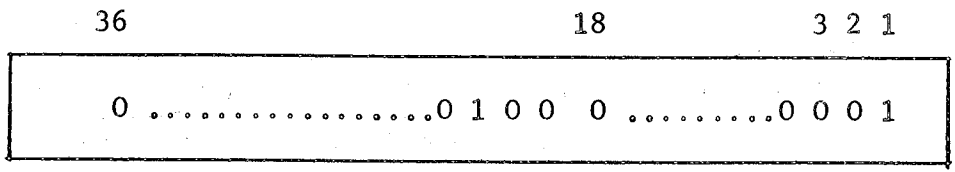
x_i' $\{1\ 0\}$

ⓓ Constantes

Si une variable x_i n'apparaît pas dans le monôme, les positions i et i+18 sont toutes deux nulles. On peut alors considérer que le couple $\{0\ \ 0\}$ représente la constante 1.

Exemple :

$x_1\ x_2'$ peut s'écrire $x_1 \cdot 1 \cdot x_3'$, le 1 marquant l'absence de x_2' et ce monôme s'écrit :



Si x_i et x_i' apparaissent simultanément, on a un monôme nul autrement dit la constante 0 est codée par un couple $\{1\ \ 1\}$ pour au moins un i.

ⓔ Codage machine

Ce vecteur de 2^n composantes sera rangé dans un tableau entier à 1 dimension déclaré de 1 à k = Entier $(2^n/36) + 1$. Les k-1 premières mémoires étant pleines, la kième remplie en commençant par la droite.

Les digits inutilisés dans la kième sont mis à 1.

(f) Opérations sur les fonctions

Cette représentation des fonctions présente le gros avantage sur la représentation polynomiale, de permettre facilement les opérations sur les fonctions :

α) Comparaison. Si deux fonctions sont égales, leurs vecteurs de vérité sont égaux mémoire à mémoire.

β) Composition. Si on peut calculer $g = h(f_1, f_2, \ldots, f_p)$ en calculant mémoire à mémoire le vecteur de vérité de g.

(g) Produit de monômes

Le produit de deux monômes s'obtient par union terme à terme des mémoires. On fera éventuellement un test de nullité du produit en recherchant les couples {1 1} .

Un monôme A non nul est tel que l'intersection terme à terme des parties droite et gauche du code de A est vide.

Exemple :

x y X y z

xy = | 0 ..... 0 0 0 0 ..... 0 1 1 |

yz' = | 0 .....0 1 0 0 0 ..... 0 1 0 |

.....

xyz = | 0 .....0 1 0 0 0 ..... 0 1 1 |

test de nullité

partie droite | 0 ... 0 0 1 1 |

partie gauche | 0 ... 0 1 0 0 |

intersection | 0 ... 0 0 0 0 |

Le test vérifie que le monôme est non nul.

2 - REPRESENTATION VECTORIELLE

Une fonction peut être représentée par la suite ordonnée de ses valeurs aux différents points de l'espace des variables.

a) Choix de l'ordre :

On choisit à priori un ordre pour les variables. A chaque point A de coordonnées $a_1, a_2, a_3, \ldots, a_{n-1}, a_{n_{n-i-1}}$, on associe l'entier $N_{(A)} = \Sigma\ 2 \times a_i$ on ordonne l'ensemble des points A par N croissants. Cet ordre est appelé ordre naturel.

b) Vecteur de vérité :

A la suite ordonnée des points, on associe terme à terme la suite ordonnée des valeurs de la fonction envisagée. On adopte une représentation vectorielle. Ce vecteur est appelé vecteur de vérité.

Exemple :

$$f(x_1, x_2, x_3) = x + yz$$

Suite des $N_{(A-i)}$	Suite des A_i			Suite des $f(a_{i_1}, a_{i_2}, a_{i_3})$
	a_1	a_2	a_3	
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	0	1	1
7	1	1	0	1
8	1	1	1	1

Exemple :

$$g = x + y \oplus x\ y\ z$$

x + y	x y z	g
0	0	0
0	0	0
1	0	1
1	0	1
1	0	1
1	0	1
1	0	1
1	1	0

Remarque :

Contrairement à ce qu'on pourrait croire, cette représentation est en moyenne moins encombrante que la représentation polynomiale.

Considérons par exemple des fonctions de 5 variables. Elles comportent en moyenne 8 monômes premiers environ, c'est-à-dire 8 mémoires en forme polynomiale, alors que la forme vectorielle comportant $2^5 = 32$ digits n'occupe qu'une mémoire.

OPERATIONS DE BASE

Le langage Algol ne permet pas les opérations d'unions, intersection et complément sur les mémoires entières. Nous avons alors tourné la difficulté en écrivant des procédures élémentaires en MAP.

Remarque :

Nous avons écarté la déclaration booléenne qui certes, permet sur notre compilateur Algol les opérations d'union et d'intersection, mais qui ne permet le complément et le test d'égalité que sur la 1ère position. Or l'expérience a montré que les tests d'égalité sont beaucoup plus fréquents que les opérations logiques, ce qui justifie notre choix.

La liste des procédures MAP est donnée dans l'annexe.

P R E S E N T A T I O N des P R O G R A M M E S

2ème partie

1 - SOUS-PROGRAMMES DE ROUTINE

Pour parer aux insuffisances du langage Algol, nous utilisons des procédures écrites en langage machine.

Ces procédures sont valables sur 7044 ayant le compilateur de Grenoble.

A - PROCEDURE D'ECRITURE

Ces procédures sont dues à Monsieur Colmerauer :

```
procédure RCH ;
commentaire  Cette procédure provoque un retour à la ligne;

procédure LIGNE(A,B); valeur A, B; Entier A, B;
commentaire  Cette procédure délimite la zone d'impression sur la feuille de
sortie : tous les textes suivants seront écrits entre les colonnes A et B.
(Si la procédure n'est pas appelée, on écrit sur toute la largeur de la feuille;

procédure Sorchaîne (A); chaîne A ;
commentaire  Ecrit la chaîne A entre les limites définies par Ligne, à la
suite des précédents textes, s'il y a la place, ou à la ligne si la chaîne
est plus longue que la place restante ;

procédure Sortexte (A,B,C); valeur A,B,C ;
entier A, B, C ;
commentaire  Ecrit A caractère dont les codes "DCB" sont dans la mémoire C à
partir du Bième caractère. La chaîne constitue est cadrée comme pour la
procédure sorchaîne ;

procédure Sorentier (A) ; valeur A; entier A ;
commentaire  écrit l'entier A à la suite des textes précédents, ou à la ligne
si la place est insuffisante ;
```

B - SOUS-PROGRAMMES DE TRAITEMENT LOGIQUE

```
entier procédure OU (A,B) ; valeur A, B; entier A, B;
commentaire union logique des mémoires A et B ;

entier procédure ET(A,B); valeur A, B; entier A, B;
commentaire  Intersection logique des mémoires A et B ;
```

```
Entier procédure NG(A) ; valeur A ; entier A ;
commentaire complément logique de la mémoire A ;

Entier procédure DJ(A,B) ; valeur A, B; entier A, B
commentaire  disjonction des mémoires A et B

Entier procédure TD(A, B) ; valeur A, B ;
entier A, B ;
commentaire  translation à droite de la mémoire A de B positions (les positions
libérées sont mises à 0) ;

entier procédure TG(A,B) ; valeur A, B;
entier A, B ;
commentaire rotation vers la gauche de la mémoire A de B position ;

Entier procédure Bit(A,B) ; valeur A, B ;
entier A, B ;
commentaire Bit=1 si le Bème bit de A (à partir des digits faible poids)
est 1 sinon Bit=0 ;

Entier procédure Poids (A); valeur A; entier A;
commentaire poids = nombre de digits non nuls de la mémoire A;
```

C - PROCEDURES DE CALCUL BOOLEEN

Ces procédures sont toutes écrites en Algol. Pour la plupart, elles sont dues à Monsieur Benzaken. Elles utilisent toutes le codage sous forme polynomiale des fonctions. Leur liste est donnée extensivement ci-dessous.

```
          PROCEDURE LECT1(F,M,N) ;
VALEUR N;
ENTIER M,N;
ENTIER TABLEAU F;
COMMENTAIRE CETTE PROCEDURE LIT LES M MONOMES D'UNE FONCTION
DE N VARIABLES ET LES RANGE DANS LE TABLEAU F[1:M] F SE
TERMINE PAR . NE PAS COMMENCER F PAR +,NE PAS SEPARER PAR UN
BLANC UNE LETTRE DE L'ASTERISQUE ;
DEBUT
   ENTIER T,I,J,G;
   ENTIER TABLEAU C[1:73];
   J:=1;
   F[J]:=0;
E: ENTREE(5,MOD1,72,C[1]);
MOD1:MODELE( " (72A1) " );
   POUR T:=1 PAS 1 JUSQUA 72 FAIRE
   C[T]:=TD(C[T],30);
   T:=0;
   POUR T:=T+1 TANTQUE  T ≤ 72 FAIRE
   DEBUT
      SI C[T]=27 ALORS ALLERA  E5
      SINON
      SI C[T]=48 ALORS ALLERA  E1
      SINON
      SI C[T]=16 ALORS ALLERA  E4
      SINON
      DEBUT
         POUR I:=1 PAS 1 JUSQUA N FAIRE
         DEBUT
            SI C[T]=NOM[I] ALORS ALLERA  E2;
         FIN ;
      FIN ;
   E2:G:=RANG[I];
      SI C[T+1]=12 ALORS ALLERA  E3
      SINON
      DEBUT
         F[J]:=OU(F[J],G);
         ALLERA  E1;
      FIN ;
   E3:G:=TG(G,18);
      T:=T+1;
      F[J]:=OU(F[J],G);
      ALLERA  E1;
   E4:J:=J+1;
      F[J]:=0;
   E1:
   FIN ;
   ALLERA  E;
E5:M:=J;
FIN LECT1 ;
```

```
PROCEDURE EXTRF1(F,M,N,LIGNE) ;
VALEUR M,N,LIGNE;
ENTIER M,N,LIGNE;
ENTIER TABLEAU F;
COMMENTAIRE CETTE PROCEDURE IMPRIME LES M MONOMES D'UNE
FONCTION DE N VARIABLES . LIGNE EST LE NOMBRE DE CARACTERES
PARLIGNE CE NOMBRE DOIT ETRE INFERIEUR A 120 ;
DEBUT
   ENTIER CAR,LIBRE,TAU,I,J,A,B,LA;
   CAR:=0;
   LIBRE:=LIGNE-2*N;
   RCH;
   POUR I:=1 PAS 1 JUSQUA M FAIRE
   DEBUT
      SI CAR > LIBRE ALORS
      DEBUT
         SAUTLIGNE;
         RCH;
         CAR:=0
      FIN ;
      CAR:=CAR+3;
      A:=F[I];
      B:=TD(A,18);
      A:=ET(A,262143);
      TAU:=OU(A,B);
      SI TAU=0 ALORS
      DEBUT
         ECRIRDCB(1);
         ALLERA  STOP
      FIN ;
      POUR J:=1 PAS 1 JUSQUA N FAIRE
      DEBUT
         SI ET(TAU,RANG[J]) ≠ 0 ALORS
         DEBUT
            LA:=NOM[J];
            ECRIRDCB(LA);
            CAR:=CAR+1;
            SI LA < 0 ALORS
            DEBUT
               LA:=TD(LA,6);
               ECRIRDCB(LA);
               CAR:=CAR+1
            FIN ;
            SI ET(B,RANG[J]) ≠ 0 ALORS
            DEBUT
               CAR:=CAR+1;
               ECRIRDCB(12)
            FIN
         FIN
      FIN ;
```

```
                SI I ≠ M ALORS
        DEBUT
            ECRIRDCB(48);
            ECRIRDCB(16);
            ECRIRDCB(48)
        FIN ;
    FIN ;
STOP:RCH;
FIN EXTRF1 ;
```

```
PROCEDURE INCLURE1(F,J,A) ;
VALEUR A;
ENTIER A,J;
ENTIER TABLEAU F;
COMMENTAIRE CETTE PROCEDURE INTRODUIT DANS LA FONCTIONF DE J
MONOMES , LE MONOME A . J EST RECALCULE . TOUTES LES
SIMPLIFICATIONS SONT FAITES NOTA: IL FAUT QUE DANS F TOUTE LES
SIMPLIFICATIONS AIENT ETEES FAITES ;
DEBUT
    ENTIER B,C,I,K;
    BOOLEEN RECUP;
    AIGUILLAGE AIG:=MOINS,PLUS;
    SI J=0 ALORS ALLERA  ALPHA;
    I:=0;
    K:=1;
    RECUP:= FAUX ;
DEPART:I:=I+1;
    B:=F[I];
    ALLERA  AIG[K];
MOINS:
    SI A=OU(A,B) ALORS ALLERA  STOP;
PLUS:C:=OU(A,B);
    SI C=B ALORS
    DEBUT
        SI I=J ALORS ALLERA  BETA;
        B:=F[J];
        J:=J-1;
        RECUP:= VRAI ;
        K:=2;
        ALLERA  PLUS
    FIN ;
    SI RECUP ALORS
    DEBUT
        F[I]:=B;
        RECUP:= FAUX
    FIN ;
    SI I=J ALORS ALLERA  ALPHA;
    ALLERA  DEPART;
ALPHA:J:=J+1;
BETA:F[J]:=A;
STOP:
FIN INCLURE1 ;
```

```
PROCEDURE DUALE1( F,Q1,M,N,G,P,DC,MAX) ;
VALEUR Q1,M,N,MAX;
ENTIER Q1,M,N, F,MAX;
ETIQUETTE DC;
ENTIER TABLEAU F,G;
COMMENTAIRE CETTE PROCEDUE DUALISE LA FONCTION F SITUEE ENTRE
LES MEMOIRES Q ET M DU TABLEAUF ET PLACE LEDUALE DANS G ENTRE 1
ET P ;
DEBUT
   ENTIER S,I,Q,P1,P2,J,K,R,L;
   ENTIER TABLEAU AUX[1:MAX];
   S:=F[Q1];
   P:=1;
   POUR I:=1 PAS 1 JUSQUA N FAIRE
   DEBUT
      Q:=ET(RANG[I],S);
      SI Q ≠ 0 ALORS
      DEBUT
         G[P]:=Q;
         P:=P+1
      FIN
      SINON
      DEBUT
         Q:=ET(RANG[I+18],S);
         SI Q ≠ 0 ALORS
         DEBUT
            G[P]:=Q;
            P:=P+1
         FIN
      FIN
   FIN ;
   P:=P-1;
   POUR I:=Q1+1 PAS 1 JUSQUA M FAIRE
   DEBUT
      S:=F[I];
      P1:=P2:=0;
      POUR J:=1 PAS 1 JUSQUA P FAIRE
      DEBUT
         SI ET(G[J],S) ≠ 0 ALORS
         DEBUT
            P1:=P1+1;
            G[P1]:=G[J]
         FIN
         SINON
         DEBUT
            P2:=P2+1;
            AUX[P2]:=G[J]
         FIN
      FIN ;
      P:=P1;
      SI P2=0 ALORS ALLERA  FIN1;
      POUR J:=1 PAS 1 JUSQUA N FAIRE
```

```
        DEBUT
            Q:=RANG[J];
            SI ET(Q,S)=0 ALORS
            DEBUT
                Q:=RANG[J+18];
                SI ET(Q,S)=0 ALORS ALLERA  FINJ
            FIN ;
            POUR K:=1 PAS 1 JUSQUA P2 FAIRE
            DEBUT
                R:=OU(Q,AUX[K]);
                SI ET(R,TO(R,18)) ≠ 0 ALORS ALLERA  FINK;
                POUR L:=1 PAS 1 JUSQUA P1 FAIRE
                SI OU(R,G[L])=R ALORS ALLERA  FINK;
                P:=P+1;
                SI P > MAX ALORS ALLERA  DC;
                G[P]:=R;
            FINK:
            FIN ;
        FINJ:
        FIN ;
    FINI:
    FIN
FIN DUALE1 ;
```

```
PROCEDURE BASEPREM1(F,M,N) ;
VALEUR N;
ENTIER M,N;
ENTIER TABLEAU F;
COMMENTAIRE CETTE PROCEDURE CALCULE UNE BASE IRREDONDANTE DE
LA FONCTION F , LE LE NOMBRE M DE MONOMES EST RECALCULE ;
DEBUT
   SI M=0 ALORS
   DEBUT
      ECRIRE( " FONCTION NULLE " );
      ALLERA  STOP;
   FIN ;
   DEBUT
      ENTIER I,A,K,J,X,I0,SAM,SAB,S,K0,K1,K2,ALPHA,BETA,I1,I2;
      ENTIER TABLEAU AUX,RD,RC[1:M];
      I:=M;
DEPART:A:=F[I];
      K:=0;
      POUR J:=1 PAS 1 JUSQUA I-1,I+1 PAS 1 JUSQUA M FAIRE
      DEBUT
         X:=OU(A,F[J]);
         SI ET(X,TD(X,18))=0 ALORS
         DEBUT
            K:=K+1;
            AUX[K]:=ET(X,NG(A))
         FIN
      FIN ;
      SI K < 1 ALORS ECHEC:
      DEBUT
         SI I=1 ALORS ALLERA  STOP;
         I:=I-1;
         ALLERA  DEPART
      FIN ;
      I0:=1;
      REDUC1(AUX,K);
TEST:
      SI K < 1 ALORS ALLERA  ECHEC;
      SAM:=REUNION(AUX,K);
      SAB:=ET(SAM,TD(SAM,18));
      SI SAB=0 ALORS ALLERA  ECHEC;
      SAM:=DJ(SAM,OU(SAB,TG(SAB,18)));
CHOIX:S:=RANG[I0];
      BETA:=TG(S,18);
      SI ET(S,SAB)=0 ALORS
      DEBUT
         SI I0=N ALORS ALLERA  ECHEC
         SINON
         DEBUT
            I0:=I0+1;
            ALLERA  CHOIX
         FIN
      FIN ;
```

```
        K0:=K1:=K2:=0;
        POUR J:=1 PAS 1 JUSQUA K FAIRE
        DEBUT
           ALPHA:=AUX[J];
           SI ET(SAM,ALPHA)=0 ALORS
           DEBUT
              SI ET(S,ALPHA) ≠ 0 ALORS
              DEBUT
                 K1:=K1+1;
                 RD[K1]:=DJ(S,ALPHA)
              FIN
              SINON
              SI ET(BETA,ALPHA) ≠ 0 ALORS
              DEBUT
                 K2:=K2+1;
                 RC[K2]:=DJ(BETA,ALPHA)
              FIN
              SINON
              DEBUT
                 K0:=K0+1;
                 AUX[K0]:=ALPHA
              FIN
           FIN
        FIN ;
        K:=K0;
        SI K1*K2=0 ALORS Z:
        DEBUT
           SI I0=N ALORS ALLERA  ECHEC;
           I0:=I0+1;
           ALLERA  TEST
        FIN ;
        POUR I1:=1 PAS 1 JUSQUA K1 FAIRE
        DEBUT
           ALPHA:=RD[I1];
           POUR I2:=1 PAS 1 JUSQUA K2 FAIRE
           DEBUT
              BETA:=OU(ALPHA,RC[I2]);
              SI BETA=0 ALORS ALLERA  SUCCES;
              SI ET(BETA,TD(BETA,18))=0 ALORS INCLURE1(AUX,K,
              BETA)
           FIN
        FIN ;
        ALLERA  Z;
     SUCCES:
        POUR J:=I+1 PAS 1 JUSQUA M FAIRE
        F[J-1]:=F[J];
        M:=M-1;
        ALLERA  ECHEC;
     FIN ;
:TOP:
IN BASEPREM1 ;
```

```
PROCEDURE REDUC1(F,M) ;
ENTIER M;
ENTIER TABLEAU F;
DEBUT
   ENTIER I,J;
   J:=1;
   POUR I:=2 PAS 1 JUSQUA M FAIRE
   INCLURE1(F,J,F[I]);
   M:=J
FIN REDUC1 ;


PROCEDURE COMPLEMENT(F,Q,M,N,G,P) ;
VALEUR Q,M,N;
ENTIER Q,M,N,P;
ENTIER TABLEAU F,G;
COMMENTAIRE CETTE PROCEDURE COMPLEMENTE LA PARTIE DU TABLEAUF
SITUEE ENTRE LES RANGS Q ET M ( BORNES COMPRISES ) . CE
COMPLEMENT EST PLACE DANS LE TABLEAUG ENTRE LES RANGS 1 ET P ,
SOUS FORME DE BASE COMPLETE ;
DEBUT
   ENTIER I,MAX;
   MAX:=2000;
   DUALE1(F,Q,M,N,G,P,DC,MAX);
   POUR I:=1 PAS 1 JUSQUA P FAIRE
   G[I]:=ROT(G[I],18);
   ALLERA  Z;
DC:ECRIRE( " DEBORDEMENT " );
FIN COMPLEMENT ;
```

```
          PROCEDURE COMPREM1(F,M,N,MAX,DC) ;
VALEUR N,MAX;
ENTIER M,N,MAX;
ENTIER TABLEAU F;
ETIQUETTE DC;
COMMENTAIRE CETTE PROCEDURE CALCULE LES COMPOSANTS PREMIERS DE
LA FONCTION. M EST RECALCULE . EN CAS DE DEPASSEMENT DE
CAPACITE ON OBTIENT DANS UNE BASE QUELCONQUE , ET ON EST
RENVOYE A L' ETIQUETTE DC ;
DEBUT
   ENTIER I,J,S,I1,I2,P,Q,R,N1,N2;
   ENTIER TABLEAU AUX[1:MAX];
   I:=0;
CAL:S:=REUNION(F,M);
   S:=ET(S,TD(S,18));
VAR:I:=I+1;
   SI I > N ALORS ALLERA  STOP;
   Q:=RANG[I];
   R:=RANG[I+18];
   SI ET(Q,S)=0 ALORS ALLERA  VAR;
   N1:=0;
   N2:=MAX+1;
   POUR J:=1 PAS 1 JUSQUA M FAIRE
   DEBUT
      P:=F[J];
      SI ET(Q,P) ≠ 0 ALORS
      DEBUT
         N2:=N2-1;
         AUX[N2]:=DJ(P,Q)
      FIN
      SINON
      SI ET(R,P) ≠ 0 ALORS
      DEBUT
         N1:=N1+1;
         AUX[N1]:=DJ(P,R)
      FIN
   FIN ;
   POUR I1:=1 PAS 1 JUSQUA N1 FAIRE
   DEBUT
      P:=AUX[I1];
      POUR I2:=N2 PAS 1 JUSQUA MAX FAIRE
      DEBUT
         R:=OU(P,AUX[I2]);
         SI ET(R,TD(R,18)) ≠ 0 ALORS ALLERA  FINI;
         INCLURE1(F,M,R);
         SI M=MAX ALORS ALLERA  DC;
      FINI:
      FIN
   FIN ;
   ALLERA  CAL;
STOP:REDUC1(F,M);
FIN COMPREM1 ;
```

```
PROCEDURE VARIABLE(X,N,DIMENSION) ;
VALEUR N,DIMENSION;
ENTIER N,DIMENSION;
ENTIER TABLEAU X;
COMMENTAIRE INTRODUIT DANS LE TABLEAU X LES VECTEURS DE VERITE
DES N VARIABLES ;
DEBUT
   ENTIER I,J,K,A,B;
   POUR I:=1 PAS 1 JUSQUA DIMENSION FAIRE
   POUR J:=1 PAS 1 JUSQUA N FAIRE
   X[J,I]:=0;
   POUR I:=0 PAS 1 JUSQUA MASQUE[N+1]-1 FAIRE
   DEBUT
      A:=I ./. 36+1;
      B:=I-36*(A-1)+1;
      POUR J:=1 PAS 1 JUSQUA N FAIRE
      X[J,A]:=OU(X[J,A],TG(BIT(I,J),B-1));
   FIN ;
   POUR I:=MASQUE[N+1]-36*(DIMENSION-1)+1 PAS 1 JUSQUA 36 FAIRE
   H1:=OU(H1,MASQUE[I]);
   POUR A:=1 PAS 1 JUSQUA N FAIRE
   X[A,DIMENSION]:=OU(X[A,DIMENSION],H1);
FIN VARIABLE ;


PROCEDURE VERITE(F,M,N,G,DIMENSION) ;
VALEUR M,N,DIMENSION;
ENTIER M,N,DIMENSION;
ENTIER TABLEAU F,G;
COMMENTAIRE PLACE DANS G[1:DIMENSION] LE VECTEUR DE VERITE
DU POLYNOME F[1:M] ;
DEBUT
   ENTIER A,B,C,D,I,J;
   D:=0;
   POUR I:=19 PAS 1 JUSQUA 18+N FAIRE
   D:=OU(D,MASQUE[I]);
   POUR I:=1 PAS 1 JUSQUA DIMENSION FAIRE
   G[I]:=0;
   POUR I:=0 PAS 1 JUSQUA MASQUE[N+1]-1 FAIRE
   DEBUT
      A:=OU(I,DJ(TG(I,18),D));
      POUR J:=1 PAS 1 JUSQUA M FAIRE
      SI OU(A,F[J])=A ALORS
      DEBUT
         B:=I ./. 36+1;
         C:=I-36*(B-1)+1;
         G[B]:=OU(G[B],MASQUE[C]);
         J:=M;
      FIN ;
   FIN ;
   G[DIMENSION]:=OU(G[DIMENSION],H1);
FIN VERITE ;
```

```
PROGRAMME:
   DEBUT
      ENTIER Q,S,I,J;
      ENTIER JMAX,NOMBRE;
      ENTIER NOMBRE1;
      ENTIER TABLEAU COUPURE[0:1];
      ENTIER QMAX,IMAX,QUALITE;
      N:=EDONNEE;
      PREPARE1(N);
      DIMENSION:=MASQUE[N+1] ./. 36+1;
      H1:=0;
      DEBUT
         ENTIER TABLEAU T[1:100,1:DIMENSION];
         ENTIER TABLEAU FF[1:DIMENSION];
         PROCEDURE SYNTHESE
         RESEAU(A,B,P,Q,T,S,K,PARITE,COUPURE,G) ;
         VALEUR A,B, F,Q,K,PARITE,COUPURE,G;
         ENTIER P,Q,S,K,PARITE;
         ENTIER TABLEAU A,B,T,COUPURE,G;
         COMMENTAIRE DECOMPOSE LA FONCTION DONT LA BORNE INF EST
         A ET LA BORNE SUP EST B EN F1=B+ØA1 ET F2=B+ØA2 AVEC
         A1+A2=A ;
         DEBUT
            ENTIER I,J,H,Z,P1,I1,I2,PAIR;
            BOOLEEN NON;
            P1:=
            SI P > 2 ALORS 2
            SINON P;
            PAIR:=ABS(PARITE-1);
            I1:=REUNION(A,P);
            I2:=REUNION(B,Q);
            SI ROT(I1,18) ≠ I2 ALORS
            DEBUT
               I1:=ROT(I1,18);
               I2:=ROT(I2,18);
               POUR I:=1 PAS 1 JUSQUA P FAIRE
               A[I]:=ET(I2,A[I]);
               POUR I:=1 PAS 1 JUSQUA Q FAIRE
               B[I]:=ET(I1,B[I]);
               REDUC1(A,P);
               REDUC1(B,Q);
            FIN ;
            COMMENTAIRE A ET B SONT REMPLACES PAR LEURS
            ENVELOPPES ;
            DEBUT
               COMMENTAIRE PARTAGE SUIVANT LA VARIABLE LA PLUS
               FREQUENTE ;
            PARTAGE:H:=0;
               POUR J:=1 PAS 1 JUSQUA N,19 PAS 1 JUSQUA 18+N FAIRE
               DEBUT
                  NOMBRE1:=NOMBRE:=0;
                  POUR I:=1 PAS 1 JUSQUA P FAIRE
```

```
            NOMBRE:=NOMBRE+BIT(A[I],J);
            POUR I:=1 PAS 1 JUSQUA Q FAIRE
            NOMBRE1:=NOMBRE1+BIT(B[I],
            SI J > 18 ALORS J-18
            SINON J+18);
            SI NOMBRE*NOMBRE1 > H ET
            BIT(COUPURE[PAIR],J)=0 ALORS
            DEBUT
               H:=NOMBRE;
               JMAX:=J;
            FIN ;
         FIN ;
         COUPURE[PAIR]:=OU(COUPURE[PAIR],
         MASQUE[JMAX]);
         COMMENTAIRE JMAX EST LE RANG DE LA VARIABLE LA
         PLUS FREQUENTE ;
         SI H=P ET P ≠ 1 ALORS ALLERA  PARTAGE;
         SI H < 1 ET P > 3 ALORS
         DEBUT
            NON:= VRAI ;
            P1:=2;
            H:=P ./. P1;
         FIN ;
         COMMENTAIRE SI UNE VARIABLE APPARAIT P FOIS , ON
         PASSE A LA SUIVANTE . SI TOUTES LES VARIABLES
         APPARAISSENT 1 FOIS , ON COUPE AU MILIEU . ;
COUPMIN:
         DEBUT
            ENTIER TABLEAU A2[1:P-H+1];
            ENTIER TABLEAU F[1:DIMENSION];
            I1:=I2:=0;
            SI NON ALORS
            DEBUT
               POUR I:=1 PAS 1 JUSQUA P-H FAIRE
               A2[I]:=A[I+H]
            FIN
            SINON
            DEBUT
               POUR I:=1 PAS 1 JUSQUA P FAIRE
               SI BIT(A[I],JMAX)=1 ALORS
               DEBUT
                  I1:=I1+1;
                  A[I1]:=A[I];
               FIN
               SINON
               DEBUT
                  I2:=I2+1;
                  A2[I2]:=A[I];
               FIN ;
            FIN ;
            COMMENTAIRE LES MONOMES CONTENANT LA VARIABLE
            DE RANG JMAX SONT RANGES DANS LE TABLEAU A ENTRE
```

```
    1 ET H , LES AUTRE DANS A2 ENTRE 1 ET P-H ;
    SI H=P ALORS P1:=1;
    SORTEXTE( " NI " );
    ECRIRDCB(P1);
    ECRIRDCB(48);
    COUT:=COUT+1;
VARIABLE1:VERITE(A,H,N,F,DIMENSION);
    POUR I:=1 PAS 1 JUSQUA DIMENSION FAIRE
    G[I]:=NG(G[I]);
    G[DIMENSION]:=OU(G[DIMENSION],H1);
    QMAX:=0;
    POUR I1:=S PAS -1 JUSQUA 1 FAIRE
    DEBUT
        QUALITE:=0;
        Z:=0;
        POUR J:=1 PAS 1 JUSQUA DIMENSION FAIRE
        SI ET(G[J],T[I1,J])=OU(F[J],T[I1,J])
        ALORS Z:=Z+1;
        SI Z=DIMENSION ALORS
        DEBUT
            COMMENTAIRE T[I1] EST UNE SOLUTION
            POSSIBLE . ;
            POUR J:=1 PAS 1 JUSQUA DIMENSION FAIRE
            QUALITE:=QUALITE+POIDS(T[I1,J]);
            SI QUALITE > QMAX ALORS
            DEBUT
                QMAX:=QUALITE;
                IMAX:=I1;
            FIN ;
        FIN ;
    FIN ;
    SI QMAX ≠ 0 ALORS
    DEBUT
        COMMENTAIRE T[I1] EST LA SOLUTION LA PLUS
        GRANDE POSSIBLE ;
        I1:=IMAX;
        SORENTIER(I1);
        ECRIRDCB(48);
        ALLERA
        SI P1=2 ALORS VARIABLE2
        SINON SOUSARBRE;
    FIN ;
    SYNTHESERESEAU(B,A,Q,H,T,S,K,PAIR,COUPURE,F);
    I1:=S;
    ALLERA
    SI P1=2 ALORS VARIABLE2
    SINON SOUSARBRE;
VARIABLE2:VERITE(A2,P-H,N,F,DIMENSION);
    POUR I2:=S PAS -1 JUSQUA 1 FAIRE
    DEBUT
        Z:=0;
        POUR J:=1 PAS 1 JUSQUA DIMENSION FAIRE
```

```
                SI ET(G[J],T[I2,J])=OU(F[J],T[I2,J])
                ALORS Z:=Z+1;
                SI Z=DIMENSION ALORS
                DEBUT
                   SORENTIER(I2);
                   ECRIRDCB(48);
                   ALLERA  SOUSARBRE;
                FIN ;
             FIN ;
             SYNTHESERESEAU(B,A2,Q,P-H,T,S,K,PAIR,COUPURE,F);
             I2:=S;
          SOUSARBRE:S:=S+1;
             POUR J:=1 PAS 1 JUSQUA DIMENSION FAIRE
             T[S,J]:=
             SI P1=2 ALORS NG(OU(T[I1,J],T[I2,J]))
             SINON NG(T[I1,J]);
             T[S,DIMENSION]:=OU(T[S,DIMENSION],H1);
          FIN ;
       FIN ;
FIN SYNTHESE RESEAU ;
```

C - PROCEDURES DIVERSES

Ces procédures écrites par Monsieur Chein concernent également la synthèse mais sont plus particulièrement destinées au traitement des réseaux. Nous n'en donnons qu'une liste sommaire.

1 -

boolean procedure CØNEX(F,M); valeur M; entier M; entier tableau F ;
commentaire CØNEX est vrai si le réseau représenté par le tableau (F,M) est connexe, faux autrement ;

2 -

procedure CØMC(F,M,C,NC,SØRT1); valeur M; entier M, NC;
entier tableau F, C; étiquette SØRT1 ;
commentaire Cette procédure cherche les NC composantes connexes de la fonction F de M monômes qu'elle range dans le tableau C à deux indices, le 1er indiquant la composante, on va en SØRT1 s'il y a plus de 10 composantes ou 20 éléments dans une composante ;

3 -

procedure CØMC1(F,M,C,NC,SØRT1); valeur M; entier N,NC;
entier tableau F,C; étiquette SØRT1 ;
commentaire Cette procédure range dans le tableau C les NC composantes connexes du réseau représenté par le tableau F de M mémoires, on va en SØRT2 dans les mêmes conditions que CØMX ;

4 -

procédure ARTI1(F,M,AD,VA,N,H,TRØP); valeur M, N; entier M, N, H ;
entier tableau F, AD, VA; étiquette TRØP;
commentaire Cette procédure range dans le tableau AD de longueur H les hypercoupures simples du tableau F de M mémoires à N bits utilisés et dans VA les fractures associées, on ira à TRØP s'il y a plus de 1000 hypercoupures;

5 -

procédure ARTI1(F,M,N,NO,HS,AHS,TEST1,TRØP); valeur M,N,NO;
entier M,N,NO,HS,AHS ; entier tableau F; booleen TEST1; étiquette TRØP;
commentaire Cette procédure range dans HS : l'hypercoupure simple ayant une fracture associée minimale AHS si TEST1 est vrai, l'hypercoupure simple maximale si TEST1 est faux. NO=N si la fonction n'est pas croissante sinon NO=-1 on va en TRØP s'il y a plus de 1000 HS ayant un nombre d'élément donné;

6 -

procédure DECP3(F,M,N,NO,HS,F1,F2,M1,M2, N1,N2); valeur M,N,NO;
entier M,N,NO,HS,M1,M2,N1,N2; entier tableau F,F1,F2;
commentaire Cette procédure découpe (F,M,N) en (F1,M1,N1) et (F2,M2,N2) suivant l'hypercoupure HS;

7 -

procédure SØP(F,M,N,R,EXIT); entier M,N; entier tableau F,R;
étiquette EXIT ;
commentaire Cette procédure récursive range dans le tableau R l'écriture en S et P de la fonction (F,M,N) complète quelconque obtenue par la méthode des fractures améliorée ;

A N N E X E - 2 -

EXEMPLES DE RESEAUX OBTENUS

1 - METHODE DE FRACTIONNEMENT SIMPLE - RESEAUX ARBORESCENTS

Exemple 1 -

Opérateur NI à 2 entrées.

Fonction complète :

Φ = BDE + ABCD + ABDF' + B'CD'E + B'CE'F' + A'BC'D + A'CDEF' + A'B'C'F + B'C'DF.

coût de cette expression en somme de produits (base première irredondante) :

36 lettres

Solution sans aléa (notation préfixée) :

Φ = 1 2 1 2 1 2 2 1 2 B 1 F 1 2 2 1 A D C 2 1 2 1 2 1 E 1 C 1 D 1 F A
1 2 2 1 2 1 B 1 D 1 A 2 A C 2 1 2 1 C B 1 2 F E 1 2 1 2 2 1 2 1 C B
1 2 1 E 2 1 2 1 B 1 A 1 2 1 D F 2 2 2 1 C 1 A F 1 2 1 B 1 D.

Coût : opérateur à 2 entrées : 30
opérateur à 1 entrée : 29

Total : 59

Temps de Calcul : 6, 3 secondes

En réunissant certaines parties du réseau on obtient :

2 entrées : 28
1 entrée : 17

Total : 45

Solution minimale (en notations préfixée) :

Φ = 2 1 2 1 2 2 1 B D 2 1 B F C 2 2 2 1 A D 2 1 2 1 B 1 A 1 F 1 2 C 2 1
E 1 D 1 2 1 2 2 1 2 1 C 1 B 1 2 A E 2 1 2 1 C B 1 2 1 D 1 F 2 1 2 1 E
C 2 1 D 2 1 A B

Coût : opérateur à 2 entrées : 25
opérateur à 1 entrée : 23
Total 48

Temps de calcul : 6, 4 secondes.
(en réunissant certains noeuds on peut arriver à économiser 10 opérateurs à 1 entrée : donc on obtient un réseau non arborescent de coût 38).

Augmentation relative du coût :

$$\eta = \frac{\text{coût sans aléa}}{\text{coût minimal}} = \frac{59}{48} = 1,23$$

En opérateur + et à 2 entrées

même fonction que précédemment, traitée par la méthode des composantes connexes.
Solution :

Φ = B' + D E + (A + C') (C + A' + F') . (B + C' + F' + D' E') .
. C + (D + A')(B + F) . F + E' + D(B + A')

coût : 23

Temps de calcul : 51 secondes.

2 - FRACTIONNEMENT PAR LA METHODE DES BORNES

a) Fonction de l'exemple 1 - Opérateurs à 2 entrées.

Borne inférieure :

BDE + ABCD + ABDF' + B'CD'E' + B'CE'F' + A BC'D + A'CDEF' + A'B'C'F + B'C'DF

Complément de la borne supérieure

BD' + A'BCE' + AC'D' + CD'E + B'CDF + B'C'F' + ABC'E'F + AB'CE

Solution

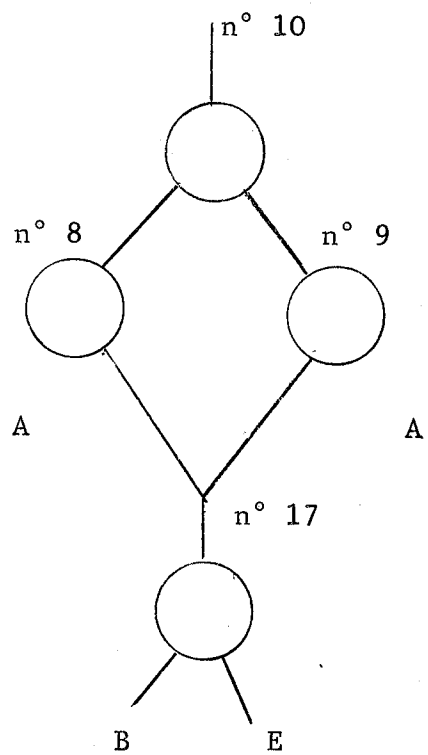
NI2 NI2 NI2 2 2 NI2 NI2 1 NI1 NI2 5 NI1 3 NI2 NI1 NI2 5 3 NI1 NI2 NI1 6
NI1 1 NI2 NI2 NI1 NI2 2 8 NI2 NI2 5 4 NI2 5 6 NI2 NI2 4 NI2 1 NI1 NI2 2 3
NI2 NI2 3 NI2 NI2 2 15 NI2 2 NI2 6 8

Coût : opérateur à 1 entrée : 8
2 entrées :25
Total : 33

Temps de calcul 21, 7 secondes

Remarque : Les 6 lettres sont codées par un chiffre de 1 à 6; les chiffres supérieurs à 6 correspondent à des noeuds déjà fabriqués.

Exemple :



sera écrit :

NI2 NI2 A NI2 BE NI2 7 A

Synthèse redondante

Fonction de 4 variables en opérateur NI a 3 entrées (on suppose donnés les compléments des variables)

F = B'C + CD + A'BC' + AD' + A'B'C' + A'BCD' + AC'D

Coût en + et : 19

Solution minimale :

F = 3 2 1 2 D' A' C 3 1 2 A C' D B' 2 1 2 A B C

Coût total : 10

Temps de calcul : 1 seconde

Solution redondante :

F = 3 3 2 3 D' A' C 3 A C' D 2 3 C D' A' 3 B' A C' 2 C' 2 D B' 3 2 3 A B C
3 D' A' C 2 3 C A B 3 C D' A' C 3 2 A C' D 3 A B C 2 3 B' A C' 3 C A B 2
2 D B' C'

Coût : 26

Temps de calcul : 2 secondes

Une solution majoritaire aurait coûté :

3 coût minimum + coût de la majorité

c'est-à-dire 3 10 + 6 = 36

Fonction générale incomplète :

5 fonctions incomplètes de 5 variables en opérateur NI à 2 entrées.
Coût total en sommes de produit : 69 (pour les bornes inférieures)

a) En synthèse séparée :

Complément de la borne supérieure
DE' + A'B'C' + A'BE + AC'E + BCD'E

Borne inférieure
B'CD' + AD'E' + A'BC'E + CDE

Solution
NI2 NI2 NI2 4 NI2 3 1 NI2 NI2 5 5 NI2 2 3 NI2 NI2 NI2 NI1 2 1 NI2 NI2 NI1
3 2 8 NI2 6 NI2 14 7

Complément de la borne supérieure
C'D'E + A'B'E + AD' + AB'E' + ABC'

Borne inférieure
A'C + A'E' + A'BD + AB'DE'+ BCD

Solution
NI2 NI2 NI2 NI2 3 NI1 5 1 NI2 NI2 4 3 NI2 2 1 NI2 8 NI2 9 NI2 NI2 1 1 NI2
NI2 5 2 NI2 10 3

Complément de la borne supérieure
AB'E + ABD + BC + CE + ACD' + A'B'C'D'

Borne inférieure
BC'D' + B'DE' + AC' D'E' + A'B'DE' + A'C'D

Solution
NI2 NI2 NI2 NI2 2 NI1 5 NI2 NI2 4 3 NI2 2 2 NI2 NI2 6 NI1 1 NI2 8 NI2 1
NI2 9 NI1 3 NI2 NI2 NI2 8 6 NI2 NI2 3 9 4 NI2 NI2 6 14 NI2 NI2 3 12 NI2 1 8

Complément de la borne supérieure

AB' + AD' + A'C + DE

Borne inférieure

A'C'D' + ABDE' + A'C'E'

Solution

NI2 NI2 NI2 1 1 NI2 NI2 2 6 NI2 4 6 NI2 NI2 3 4 NI2 NI2 1 NI1 3 5

Complément de la borne supérieure

C'E + B'D + CD + AE' BC'D'

Borne inférieure

A'CD' + CD'E + A'B'D'E' + A'BC'DE'

Solution

NI2 NI2 NI2 NI2 3 3 4 NI2 5 NI2 NI1 4 2 NI2 NI2 4 NI2 NI2 1 6 NI2
NI1 5 6 NI2 8 6 NI2 NI2 1 2 NI2 1 8

Coût : 86

Temps : 57 secondes

b) <u>En synthèse successive</u>

Complément de la borne supérieure

DE' + A'B'C' + A'BE' + AC'E + BCD'E

Borne inférieure

B'CD' + AD'E' + A'BC'E + CDE

Solution

NI2 NI2 NI2 4 NI2 3 1 NI2 NI2 5 5 NI2 2 3 NI2 NI2 NI2 NI1 2 1 NI2 NI2 NI1
3 2 8 NI2 8 NI2 6 NI2 14 7

Complément de la borne supérieure

C'D'E + A'B'C'E + AD' + AB'E' + ABC'

Borne inférieure

A'C + A'E' + A'BD + AB'DE + BCD

Solution

NI2 NI2 NI2 NI2 3 8 1 NI2 NI2 4 3 NI2 2 1 NI2 24 NI2 6 NI2 13 NI2 NI2 5 2 NI2 12 3

Complément de la borne supérieure

AB'E + ABD + BC + CE + ACD' + A'B'C'D'

Borne inféieure

AB'D' + B'DE' + AC'D'E' + A'B'CE' + A'C'D

Solution

NI2 NI2 NI2 31 27 6 NI2 NI2 31 NI2 3I 4 NI2 NI2 8 6 NI2 NI2 3 6 NI2 1 NI2 3 4

Complément de la borne supérieure

AB' + AD' + A'C + DE

Borne inférieure

A'C'D' + ABDE' + A'C'E'

Solution

NI2 NI2 6 27 NI2 42 NI2 NI2 1 6 5

Complément de la borne supérieure

C'E + B'D + CD + AE' + BC'D'

Borne inférieure

A'CD' + CD'E + A'B'D'E' + A'BC'DE'

Solution

NI2 NI2 NI2 14 4 NI2 5 NI2 25 2 NI2 NI2 4 NI2 43 NI2 8 14 NI2 18 NI2 NI2 1 2 43

Coût : 59

Coût total en somme de produit : 69 pour les bornes inférieures

Coû approximatif en réaliser + et à partir des NI : 150

A N N E X E - 3 -

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

ACKERS S.B. & ROBBINS T.C.

"Synthesis of combinational logic using 3 imput majority gates"
Computing review, vol 5 n° 5, sept-Octobre 1964

LUSTMAN F.

"Synthèse de fonctions à l'aide de l'opérateur majorité"
Algèbre de Boole et Machines Logiques - Dunod 1967
(Colloque d'Algèbre de Boole, janvier 1965).

KUNTZMANN J.

"Algèbre de Boole"
Dunod Avril 1965

KUNTZMANN J.

"Méthodes générales de réalisation de fonctions booléennes par des organes technologiques donnés".
Automatisme Tome X, pp. 58-60, février 1965

LUSTMAN F.

"Réalisation de fonctions booléennes avec l'opérateur majorité"
Thèse de Docteur-Ingénieur, Grenoble, avril 1966

MACHERAS J.

"Synthèse de fonctions booléennes par l'opérateur U"
Thèse de troisième cycle, Grenoble, juin 1966

CHARLET, THIRIEZ

"Génération de fonctions booléennes quelconques à l'aide d'un opérateur réalisant simultanément la somme et son complément"
Projet de fin d'Etude, Section Mathématiques Appliquées, Grenoble, 1966

BENZAKEN C.

"Algorithme de dualisation d'une fonction booléenne"

Chiffre 2ème trimestre 1966, pp. 119-128, 1966

MEO A.R.

"Sulla sintesi di reti Nand o Nor a molti livelli"

Calcolo tome 2, pp. 1-82, janvier 1965

GRASSELI A.

"An approximate method for the synthesis of multiple output combinational net works"

Calcolo Tome 2, pp. 347-368, 1965

EQUIPE DE LOGIQUE DE GRENOBLE

"Bibliographie portant sur l'Algèbre de Boole, l'algèbre séquentielle, le codage et certaines questions rattachées"

Université de Grenoble, Mathématiques Appliquées
Avril 1965

PICHAT E.

"Décomposition des fonctions booléennes"

Thèse de Docteur de Spécialité - Grenoble
janvier 1966

* * * * *

TABLE des MATIERES

Suite .../... 2

Deuxième partie

Chapitre 2

VU

Grenoble, le

Le Président de la Thèse

VU

Grenoble, le

Le Doyen de la Faculté des Sciences

VU, et permis d'imprimer,

Le Recteur de l'Académie de GRENOBLE