Nouvelles méthodes de synthèse logique

Pascal Sicard

THESE

présentée par

Pascal SICARD

pour obtenir le titre de **DOCTEUR**

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
(arrêté ministériel du 5 Juillet 1984)

Spécialité: **Informatique**

NOUVELLES METHODES DE SYNTHESE LOGIQUE

Date de soutenance: 2 Septembre 1988

Composition du jury:

Madame	Gabrièle SAUCIER	
Messieurs	Alain COSTES	Président et rapporteur
	François ANCEAU	
	René DAVID	
	Jacques MOSSIERE	
	Nguyen Huy XUONG	Rapporteur

Thèse préparée au sein du laboratoire URCSI.

A toute ma famille.

Remerciements

Je tiens à remercier :

Madame G. SAUCIER, Professeur à l'ENSIMAG
qui m'a accueilli dans son équipe de recherche et qui m'a encadré tout au long de l'élaboration de cette thèse,

Monsieur A. COSTES, Professeur et Directeur du LAAS
d'avoir bien voulu accepter d'être rapporteur et de me faire l'honneur de présider la commission d'examen de cette thèse,

Monsieur N. H. XUONG, habilité à diriger des recherches au CNRS
pour l'examen et la critique qu'il a bien voulu apporter à ce travail en tant que rapporteur et membre du jury,

Monsieur F. ANCEAU, Professeur,
Monsieur R. DAVID, Directeur de recherche CNRS et Directeur du LAG,
Monsieur J. MOSSIERE, Professeur à l'ENSIMAG
qui ont accepté d'être membres du jury,

Je remercie aussi toute l'équipe de recherche dont j'ai apprécié la collaboration tout au long de ces 3 années, en particulier C. DUFF et M. CRASTES qui m'ont apporté un concours précieux.

Je remercie également toutes les personnes qui ont pu participer de près ou de loin à l'élaboration de ce travail.

INTRODUCTION

Les recherches en synthèse logique sont très actives depuis l'avènement des calculateurs digitaux et se développent en suivant les différentes évolutions technologiques.

Les premières études sur les fonctions logiques remontent au siècle dernier, c'est le mathématicien Boole qui s'y intéresse pour la première fois en 1850 et qui élabore une théorie sur les fonctions logiques : l'algèbre de Boole.

Il faudra attendre les années 1950, avec la naissance des premiers circuits logiques pour voir apparaître les premières techniques de minimisation logique. Les portes logiques sont coûteuses, il est alors primordial de minimiser le coût de l'implémentation d'une fonction booléenne (nombre de diodes et de résistances). La minimisation devient un domaine de recherche primordial.

Karnaugh et Veitch [KAR53] développent une technique manuelle basée sur une représentation des fonctions par tableaux. Il est possible sur de tels tableaux de déterminer visuellement les plus grands groupements de 2^k points (monômes premiers d'une fonction). Le nombre d'entrées était malheureusement limité à 5. Plus tard, des techniques automatiques plus sophistiquées étaient mises au point par Quine et M^{C}Cluskey [MCC65] pour obtenir une implémentation 'deux couches' avec le minimum de portes. Ces algorithmes s'appuyaient sur trois étapes principales:

-La fonction est décomposée en points élémentaires (monômes canoniques).

-A partir de l'ensemble des points, le calcul de tous les monômes premiers est effectué.

-A partir de l'ensemble de tous les monômes premiers, l'extraction de toutes les bases irrédondantes est réalisée pour trouver la plus petite.

Le problème de ces algorithmes est l'explosion rapide (en place mémoire et temps d'exécution), dûe à la décomposition de la fonction en points élementaires. Une nouvelle théorie est développée par Tison et Kuntzmann [KUNT68] : le consensus. Elle permet, étant donné un ensemble quelconque de monômes, de déterminer tous les monômes premiers sans pour cela avoir besoin de redescendre aux points

élémentaires de la fonction.

Jusqu'à la fin des années 70, le coût des portes ayant baissé, ces techniques de minimisation suffisent largement. La recherche dans le domaine de la minimisation semble être arrivée à son apogée; on n'en parle presque plus pendant 10 ans.

La fin des années 70 voit arriver l'introduction des circuits à structures régulières tel que le PLA, qui permettent des implémentations de fonctions à coût très réduit. La minimisation a alors un impact direct sur la surface du circuit, un monôme équivalant à une ligne de PLA . De plus, la construction de tels circuits permet d'implémenter des ensembles de fonctions logiques de plus en plus importants (nombre d'entrées et de fonctions supérieur à 30, nombre de monômes supérieur à 100). Les techniques de minimisation des années 60 explosent. En effet, on peut montrer que le nombre de monômes premiers pour une fonction simple à n variables peut atteindre $3^n/n$. De plus, le problème d'extraction d'une base irrédondante minimale est NP complet; de tels algorithmes sont alors impraticables, même pour des fonctions de tailles moyennes (10-15 variables) .

La recherche sur la minimisation reprend alors toute son ampleur. Puisque l'on ne peut plus espérer une minimisation exacte dans des temps et une place mémoire convenables, le problème est abordé à l'aide de différentes heuristiques.

Les premières tentatives maintiennent tout d'abord la génération de tous les monômes premiers par une technique plus efficace. L'extraction d'une base irrédondante minimale approchée est ensuite élucidée grâce à différentes heuristiques. Mais cela s'avère insuffisant, le nombre de monômes premiers peut être trés grand même pour des fonctions de tailles moyennes (nombre de variables inférieur à 15).

Une seconde approche est élaborée, on essaye simultanément d'identifier et de sélectionner des monômes (pas forcément premiers) pour s'approcher d'une couverture minimale. Beaucoup de tentatives sont lancées à partir de cette idée ([ROT80], [HON74], [RHY77], [ARE78], [BRO81]).

Par exemple [RHY77] et [ARE77] emploient à peu près la même méthode . Ils sélectionnent un point de la fonction. Ce point est étendu à un monôme premier. Tous les points couverts par ce monôme sont supprimés. Une itération sur ces trois étapes est effectuée jusqu'à ce que tous les points de la fonction soient couverts. Dans

[RHY77], à partir du point sélectionné, l'ensemble de tous les monômes premiers qui contiennent ce point est calculé. Dans cet ensemble un monôme est ensuite sélectionné, soit parce qu'il est essentiel, soit parce qu'il répond à des critères d'heuristiques. Cette méthode peut être très inéfficace au vu du grand choix possible parmi les monômes premiers. Dans [ARE78], seulement un sous-ensemble de tous les monômes premiers issus du point traité est généré. Cette méthode est plus rapide mais moins efficace que la précédente.

Ces deux méthodes ont un inconvénient majeur, elles redescendent aux points élémentaires de la fonction. Et elles ne peuvent traiter que des fonctions de petites tailles (nombre de variables inférieur à 10).

Dans MINI [HON74] puis PRESTO [BRO81] on essaye de pallier le problème de la décomposition et du stockage des points. La stratégie est la même, mais au lieu de redescendre aux points de la fonction, ils partent des monômes qui leur sont fournis. Un monôme est choisi puis étendu, les monômes couverts sont alors éliminés. Cette première étape ne donnant que des résultats médiocres, les monômes restants sont réduits, tout en gardant une couverture de la fonction pour pouvoir à nouveau les étendre, espérant ainsi un nouveau gain. Le problème de l'explosion en temps et en place mémoire est résolu; on ne redescend plus aux points de la fonction, on ne génere plus tous les monômes premiers, l'extraction d'une base irrédondante n'est plus effectuée réellement. La base obtenue n'est pas toujours minimale mais les résultats paraissent convaincants.

En France, on recommence alors à s'intéresser au problème, devant la demande de plus en plus pressante des industriels et l'avance des autres pays (principalement des Etats-Unis) dans ce domaine. En particulier au LCS, on développe un outil de minimisation deux couches performant dans un système de synthèse logique plus complet (ASYL) [HAN85]. En parallèle, aux Etats-Unis (Université de Berkeley), la recherche dans continue: grâce à une expérience dans le domaine acquise depuis de longues années et aprés une comparaison poussée des différentes stratégies des deux minimiseurs les plus connus de l'époque (MINI et PRESTO), Espresso-II [BRA84] est élaboré deux années plus tard. L'idée générale est la même, les techniques et les heuristiques sont choisis suivant les résultats des deux minimiseurs sur un grand nombre d'exemples. On ajoute des étapes permettant de résoudre correctement

certains types particuliers de cas, l'approche devient lourde et fastidieuse (voir [BRA84]). Un peu plus tard, d'autres mimiseurs tels que [DAG85] se veulent de nouveau exact, ils sont toujours basés sur une génération de tous les monômes premiers et une extraction des bases irrédondantes. Malgré l'amélioration des anciens algorithmes et les performances accrues des ordinateurs, ils ne peuvent trouver le minimum exact que pour des systèmes de fonctions de complexités moyennes (dépendant surtout du nombre de bases irrédondantes), et cela dans des temps nettement supérieurs à ceux des algorithmes à base d'heuristiques.

Les recherches de minimisation deux couches d'ASYL se sont faites dans une optique complètement différente et ont abouti à des résultats trés compétitifs. Nous voulions arriver à traiter des applications de trés grande taille (nombres de variables et de fonctions allant jusqu'à 50). Nous avons donc opté pour une recherche d'un minimum, limitant au maximum le nombre de monômes générés et s'approchant du minimum exact grâce à des heuristiques. Cette approche originale et plus élégante va consister à partir de monômes premiers des fonctions simples, de s'approcher du minimum exact par une itération sur deux étapes principales :

-adjonction de nouveaux monômes, créés par "intersection contrôlée" sur les monômes présents, à la base courante.

-extraction de base irrédondante minimale.

Cette méthode peut se comparer à un recuit simulé. On s'approche du minimum exact, en passant par des minimums locaux de plus en plus petits. Pour sortir d'un minimum local, on se permet d'augmenter la fonction coût en ajoutant des monômes, afin de retomber dans un minimum plus bas.

La maîtrise et l'amélioration des différents mécanismes de base du calcul booléen se sont avérées très fructueuses. Ceci a permis de réaliser un 'décollage' dans les recherches voisines. En particulier, les problèmes de synthèse sur d'autres cibles (PLD, PAL), qui seront évoqués dans la deuxième partie de la thèse, ont largement profité de ces bases solides. De même, les travaux sur l'optimisation du codage des états d'un contrôleur n'ont pu réussir que grâce à ces résultats de base.

La première partie de cette thèse sera consacré à la minimisation deux couches. On rappellera tout d'abord la problématique, ainsi que les définitions mathématiques

de base nécessaires à la compréhension de la méthode générale que nous proposons. Un chapitre sera ensuite consacré aux différentes techniques de base que nous avons employées dans l'algorithme général : test d'inclusion, complémentation, calcul de l'incouvert d'un monôme, détection des monômes essentiels. Nous aborderons ensuite la méthode générale qui est employée dans le minimiseur proposé. Les principales étapes, et les heuristiques qui sont employées dans chacune d'entre elles seront alors détaillées. Puis nous évoquerons l'aspect programmation. Nous donnerons un aperçu général du système complet (ASYL) dans lequel notre travail s'insère. Nous montrerons qu'un prototypage en PROLOG peut être intéressant et avantageux avant une réalisation finale dans un langage plus performant. Nous conclurons cette partie par des résultats montrant le comportement de notre algorithme et, une comparaison avec Espresso II et un minimum exact.

Une deuxième partie sera consacrée à la synthèse sur des réseaux programmables du commerce. L'utilisation de PLD (Programmable Logic Device) a pris ces dernières années un essor considérable. Ces circuits dont les dimensions sont figés permettent grâce à leurs structures programmables, une réalisation sur carte, aisée et performante d'une large gamme d'applications (contrôleur, logique variée) à moindre coût.

Nous nous sommes intéressés à une des familles de réseaux programmables les plus utilisée actuellement appelés PAL. Nous détaillerons tout d'abord les caractéristiques variées de ce type de circuit (PLA de taille fixe à 'étage OU' figé).

Nous définirons ensuite la problématique liée à la synthèse de fonctions booléennes sur de tels circuits. Le but est de réaliser un ensemble de fonctions booléennes grâce à un ensemble minimal de boitier pris dans une bibliothèque donnée. Ce type de synthèse, tout en utilisant des techniques et des algorithmes liées à la minimisation deux couches de la première partie, introduit de nouveaux problèmes tels que la décomposition et le placement. La décomposition s'inspire de théories récentes utilisées pour les problèmes de minimisations multi-niveaux sur portes complexes et précaractérisées [BRA87], [KIN87].

Nous proposons une méthode originale décomposée en trois étapes distinctes.

-De part la structure particulière de ce type de circuits (étage OU figés, et différentes polarités des sorties), une complémentation et une minimisation indépendante de chaque fonction simple sont effectuées.

-Les dimensions des PALs étant figées, certaines fonctions peuvent ne pas être réalisables directement (trop de monômes ou trop de variables). Il faut alors les décomposer en sous-fonctions implémentables grâce aux circuits de la bibliothèque donnée. Notre méthode de décomposition est basée sur une factorisation de variables, par recherche des noyaux des fonctions. L'originalité de cette méthode consiste, d'une part à tenir compte de la structure des circuits en biliothèque au moment de la décomposition, et d'autre part, à choisir les sous-fonctions de telle manière que leur nombre ainsi que le temps de réponse de la fonction décomposée soient minimisées.

-Une fois toutes les fonctions rendues réalisables, la phase finale est le placement. Il faut choisir un ensemble minimal de boitiers dans la bibliothèque proposée, et placer effectivement les fonctions en affectant aux broches des boitiers les noms des entrées et des sorties de l'application.

Une programmation automatique des différents circuits choisis est alors ensuite possible.

PREMIERE PARTIE

LA MINIMISATION DEUX COUCHES

1. INTRODUCTION A LA MINIMISATION DEUX COUCHES

La minimisation deux couches a pour but de trouver les expressions d'un ensemble de fonctions booléennes simples permettant une implémentation optimale sur un type particulier de circuit, le PLA. Les critères d'optimalité d'implémentation sont premièrement la taille et deuxièmement les performances du circuit final.

1.1. UN CIRCUIT A STRUCTURE REGULIERE : LE PLA

Nous entendons ici par PLA, un circuit de structure régulière dite à deux couches (figure 2). Ce type de circuit peut être réalisé dans différentes technologies de façon trés simple et automatique de part sa structure même fort régulière. Il permet ainsi de réaliser de façon simple et peu coûteuse des systèmes de fonctions devenant de plus en plus complexes.

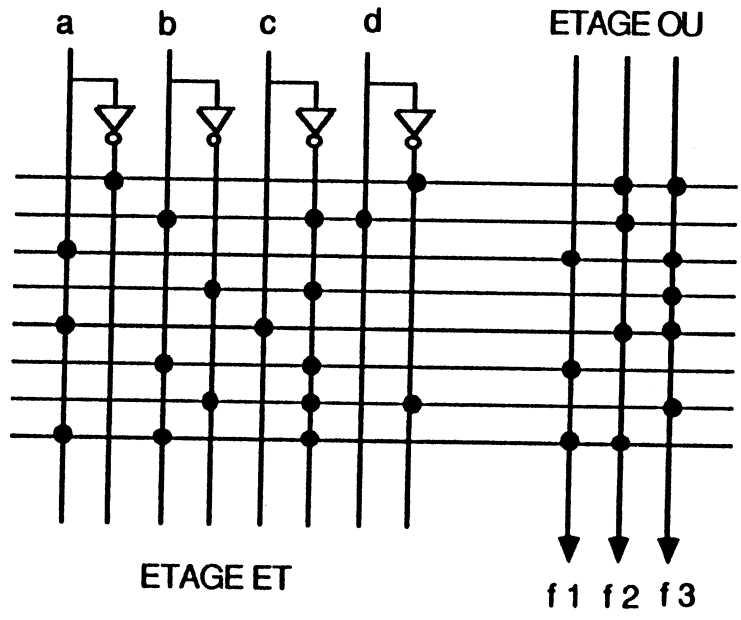


Figure 2 : un PLA

Une première couche appelée 'étage ET' est composée d'un certain nombre de lignes de transistors passant ou non suivant la configuration des entrées. Une ligne réalise ainsi un 'ET' logique entre les différentes variables d'entrée qui y figurent. La deuxième couche appelée 'étage OU', réalise les fonctions proprement dites. Chaque

ligne réalise un 'OU' logique entre certaines lignes de l'étage 'ET'.

L'optimalité d'un tel circuit est liée tout d'abord à la surface occupée sur le silicium mais aussi aux performances du circuit, c'est à dire à ses temps de réponse et à sa dissipation thermique (consommation). Ces performances pour une technologie donnée dépendent de trois paramètres :

- la charge des entrées
- la charge des sorties
- la charge des lignes

La charge d'une entrée est le nombre de transistors qui lui sont connectés. La charge d'une ligne est le nombre de transistors qui la connecte aux sorties. Celle d'une ligne est le nombre de transistors qui lui sont connectés.

D'une manière générale, on admet que les performances d'un tel circuit sont meilleures, d'autant que le nombre de transistors qu'il contient est petit.

1.2. DEFINITIONS MATHEMATIQUES DE BASE

La terminologie étant variée dans la littérature internationale dans le domaine des fonctions logiques et pour une compréhension claire et complète de tout ce qui va suivre, nous allons brièvement rappeler les définitions mathématiques de base concernant les fonctions booléennes et applicables à la minimisation deux couches. Dans un premier temps nous nous intéresserons aux fonctions booléennes simples pour ensuite généraliser à un ensemble de fonctions (ou fonction générale). Pour de plus amples détails se référer à [KUN68].

1.2.1. FONCTION BOOLEENNE SIMPLE

1.2.1.1. EXPRESSION BOOLEENNE ET MONOME

Rappel 1 :

Soit B= { 0, 1 } .

Tout n-uplet $X= (x_1,...,x_n) \in B^n$ est appelée variable booléenne générale.

Toute fonction totale $f : B^n \rightarrow B$ est dite booléenne simple à n variables.

Une valeur donnée de X est appelée point de la fonction. On dit que f couvre

Rappel 2 :

Toute fonction partielle f : B^n -> B est dite incomplète.

On a l'habitude de poser f(X)= ϕ (superposition de 0 et de 1). f n'est alors pas définie en X.

Pour cette raison une fonction incomplète est aussi appelée phi_booléenne.

Rappel 3 :

Une fonction booléenne incomplète f est définie par trois ensembles de points:

- L'ensemble C_1 des points X tel que f(X) = 1
- L'ensemble C_0 des points X tel que f(X) = 0
- L'ensemble C_ϕ des points X tel que f(X) = ϕ

C_1 , C_0 et C_ϕ sont disjoints , et $C_1 \cup C_0 \cup C_\phi = B^n$.

On appelle couverture à 1 (resp. 0 et ϕ) la fonction booléenne qui vaut 1 pour les points de C_1 (resp. C_0 et C_ϕ) et 0 ailleurs.

La couverture à 1 est encore appelée borne inférieure de f, l'union de la couverture à 1 et à ϕ borne supérieure de f, et la couverture à 0 complément de f.

Une fonction incomplète est entièrement spécifiée par la donnée de deux couvertures parmi ses trois .

Une fonction incomplète f est une tautologie si $C_0 = \emptyset$ (elle sera notée 1), ou autrement dit, si $C_1 \cup C_\phi = B^n$. Elle est nulle si $C_0 = B^n$ (notée 0).

Rappel 4 :

Toute fonction booléenne simple est exprimable par une expression booléenne. Une expression booléenne est construite à partir des variables booléennes simples de la fonction et de l'ensemble des opérateurs logiques : union (somme,+), intersection (produit, .) et négation (NON , $\neg$) .

Rappel 5 :

Un monôme est un produit de p variables simples distinctes ($n \geq p \geq 1$) sous forme normale (x) ou complémentée ($\neg x$).

p est le degré (ou longeur) du monôme.

La taille d'un monôme est le nombre de points qu'il couvre (2^{n-p}) .

Rappel 6 :

Une expression booléenne est dite polynômiale si elle est sous forme de somme de mônomes.

Rappel 7 :

Une relation d'ordre partiel sur les fonctions booléennes :

$f \leq g \Leftrightarrow f.g = f \Leftrightarrow f+g = g$

Tous les points couverts par f sont couverts par g.

Rappel 8 :

Une relation d'ordre partiel sur les fonctions booléennes incomplètes :

Soient f (de borne inférieure inf_f) et g (de borne supèrieure sup_g) deux fonctions incomplètes .

f est inférieure à g $\Leftrightarrow$ inf_f $\leq$ sup_g

On dit aussi souvent qu'un monôme m est "inclus" dans une fonction incomplète f s'il est inférieur à la borne supérieure de f. Tous les points qu'il couvre, sont couverts par la couverture à 1 ou à ϕ de la fonction.

Rappel 9 :

Un monôme canonique d'une fonction f est un monôme inférieur à f où toutes les variables de f sont présentes. Il couvre un seul point de f.

Rappel 10 :

Un monôme m est premier dans une fonction incomplète f si et seulement si m est inférieur à f, et s'il n'existe pas de monôme m' supérieur à m tel que m' soit inférieur à f.

1.2.1.2. BASES D'UNE FONCTION PHI-BOOLEENNE SIMPLE

Rappel 11 :

On appelle base d'une fonction incomplète f, une somme de monômes premiers de f telle que chaque point de la couverture à 1 de f soit couvert par au moins un de ces monômes. La base composée de tous les monômes premiers de f est appelée base complète de f.

Il est a remarquer que les monômes d'une base de f peuvent aussi couvrir des points de la couverture à ϕ de f.

Rappel 12 :

Un monôme est obligatoire dans une base d'une fonction incomplète s'il est le seul à couvrir au moins un point de la couverture à 1 de la fonction.

Rappel 13 :

On appelle base irrédondante d'une fonction f une base de f telle que tous ses monômes soient obligatoires. On ne peut enlever un monôme d'une base irrédondante sans qu'elle ne soit plus une base de f.

Rappel 14 :

Un monôme essentiel est un monôme obligatoire dans la base complète.

1.2.2. EXTENSION A UN ENSEMBLE DE FONCTIONS SIMPLES

Nous ne pouvons aborder l'étude d'un ensemble de fonctions sans définir quelques notions qui généralisent les définitions précédentes au sens global de cet ensemble.

Rappel 15 :

Soit B= { 0, 1 } . Toute fonction totale f : B^n -> B^m est dite booléenne générale.

Rappel 16 :

Toute fonction générale partielle f : B^n -> B^m est dite incomplète ou phi-booléenne.

Une fonction générale définit une séquence de fonctions simples et inversement. Nous pourrons parler indifférement d'une fonction générale ou d'une séquence de fonctions simples.

f_g : B^n -> B^m définit m fonctions simples f_1, ..., f_m avec f_i : B^n -> B. L'expression d'une fonction générale est donc un vecteur de m expressions booléennes représentant les fonctions simples f_1, ..., f_m.

Exemple :

F_g= [a.¬b + c , a + ¬c.d] nous avons f_1= a.¬b + c et f_2= a + ¬c.d

Ainsi, il existe un ordre partiel sur les fonctions générales définit à partir de l'ordre sur les fonctions simples :

$(f_1, ..., f_n) \leq (f'_1, ..., f'_n)$ <=> $f_1 \leq f'_1$, ... et $f_n \leq f'_n$

Pour des raisons de visions globales des monômes dans la séquence des fonctions (f_1,...,f_m), il est possible d'employer une autre forme d'expression.

A) MONOME GLOBAL

Rappel 17 :

Un monôme global est une fonction générale définie par le vecteur (f_1,...,f_m) où pour tout i, f_i = m ou 0 (m étant un monôme de B^n -> B).

Nous représenterons ce monôme global par un couple formé du monôme m et d'un vecteur booléen $I = (e_1, e_2, ..., e_m)$ tel que :

si $e_i = 1$ alors $f_i = m$ sinon $f_i = 0$.

Ce vecteur de B^m est appelé vecteur d'inclusion du monôme global.
Dans [KUN68] un monôme global est appelé monôme général et son vecteur d'inclusion, coefficient du monôme.

Exemple :

Le monôme global (a. b.¬c, [1,0,1]) définit les fonctions simples:

f_1= a. b.¬c , f_2= 0 , f_3= a. b.¬c

Nous dirons que f_1 et f_3 apparaîssent dans le vecteur d'inclusion du monôme global.

Rappel 18 :

Un monôme canonique global est un monôme global (m,I) tel que :
m est un monôme canonique et le vecteur d'inclusion I ne possède qu'une composante égale à 1. Le point qu'il couvre est couvert par la fonction simple apparaissant dans son vecteur d'inclusion.

Pour une fonction simple, on dit qu'un monôme couvre un ensemble de monômes canoniques (ou points). De la même manière nous dirons qu'un monôme global couvre un ensemble de monômes canoniques globaux. Le monôme canonique global est l'élément de base pour une fonction générale. Par abus de langage nous l'appellerons aussi 'point'.

Rappel 19 :

Un monôme global (m, I) couvre tous les monômes canoniques globaux (mc_j, IC_j) tel que $mc_j \leq m$ et $IC_j \leq I$.

La taille d'un monôme global de p variables est le nombre de points qu'il couvre.

C'est à dire $2^{n-p} * k$ où k est le nombre de 1 dans le vecteur d'inclusion du monôme global.

Exemple :

Avec n = 3 et m = 3 :

$(x_1 . \neg x_2 , [1,0,1])$ couvre l'ensemble des monômes canoniques :

$\{(x_1 . \neg x_2 . x_3 , [1,0,0]), (x_1 . \neg x_2 . x_3 , [0,0,1]),$

$(x_1 . \neg x_2 . \neg x_3 , [1,0,0]), (x_1 . \neg x_2 . \neg x_3 , [0,0,1])\}$

Sa taille est égale à 4.

Rappel 20 :

Soit $f = (f_1,\ldots,f_m)$, si on met les f_i sous forme polynômiale alors f est exprimable par une somme de monômes globaux. On l'appelle forme polynômiale globale.

Exemple :

$f_1 = a.b + c.d$

$f_2 = a.b + \neg c.e$

$f_3 = a + \neg c.e + c.d$

On a l'expression globale :

$E = (a.b ,[1,1,0]) + (c.d ,[1,0,1]) + (\neg c.e , [0,1,1]) + (a , [0,0,1])$

Il est possible de généraliser la proprieté de monôme premier aux monômes globaux .

Exemple :

soit $f = (f_1, f_2)$ avec

$f_1 = a + \neg a.c$ $\quad$ $f_2 = a + \neg b.c$

(a, [1,0]) n'est pas premier puisque (a, [1,1]) est inclus dans f, de même ($\neg$a.c, [1,0])

n'est pas premier puisque (c, [1,0]) est inclus dans f. Par contre (¬b.c, [1,1]) est un monôme global premier pour f.

B) BASES D'UNE FONCTION GENERALE

Nous avons les définitions équivalentes à celles d'une fonction simple.

Rappel 21 :

On appelle base (globale) d'une fonction générale incomplète $f = (f_1, f_2, ..., f_m)$, une somme de monômes globaux premiers telle que pour chaque fonction f_j, la somme des monômes pour lesquels la $j^{ième}$ composante du vecteur d'inclusion est à 1 couvre tous les points de la couverture à 1 de f_j. La base constituée de tous les monômes globaux premiers est appelée base complète globale.

Rappel 22 :

Un monôme global est obligatoire dans une base globale si cette base cesse d'en être une quand on lui enlève ce monôme global. Il est le seul à couvrir au moins un monôme canonique global (mc, IC) tel que mc couvre un point de la couverture à 1 de la fonction simple apparaissant dans IC.

Rappel 23 :

On appelle base irrédondante d'une fonction générale, une base globale composée uniquement de monômes globaux obligatoires.

Rappel 24 :

Un monôme global est essentiel s'il est obligatoire dans la base complète globale.

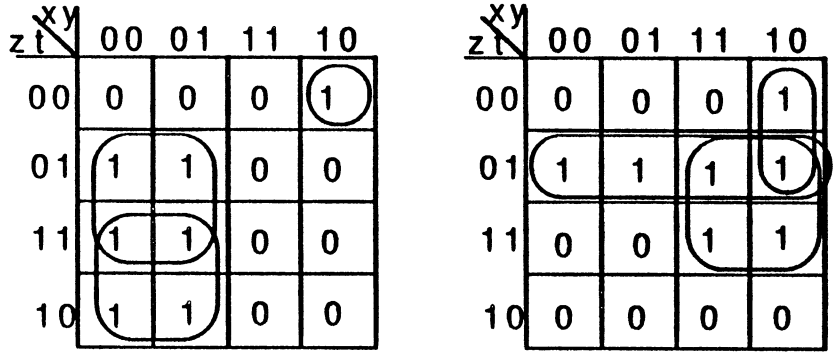
Remarque :

Il est à noter qu'il existe peu de lien entre des propriétés locales à une fonction simple apparaissant dans une fonction générale et les propriétés de celle-ci.

pour une fonction générale comprenant f (quel que soit son vecteur d'inclusion). Ou encore un monôme premier d'une fonction générale peut n'être premier dans aucune des fonctions simples de son vecteur d'inclusion.

C) APPLICATION A UN EXEMPLE

Soit deux fonctions f_1 et f_2 de quatre variables x, y, z, et t représentées sur les tableaux de Karnaugh suivants :



Les bases complètes de f1 et f2 sont représentées sur ces tableaux.

La base complète de f_1 est $\neg x.t + \neg x.z + x.\neg y.\neg z.\neg t$, tous les monômes sont essentiels.

La base complète de f_2 est $\neg z.t + x.\neg y.\neg z + x.t$, tous les monômes sont essentiels.

Toutes deux sont des bases irrédondantes.

La base complète globale de $f = (f_1, f_2)$ est :

$(\neg x.t, [1,0]) + (\neg x.z, [1,0]) + (x.\neg y.\neg z.\neg t, [1,1]) +$
$(\neg z.t, [0,1]) + (x.\neg y.\neg z, [0,1]) + (x.t, [0,1]) +$
$(\neg x.\neg z.t, [1,1])$

Les monômes globaux essentiels de f sont :

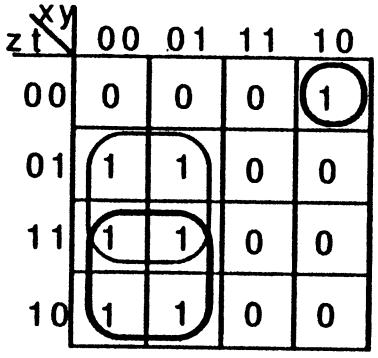
$(\neg x.z, [1,0])$, $(x.\neg y.\neg z.\neg t, [1,1])$ et $(x.t, [0,1])$

f possède deux bases globales irrédondantes représentées sur les tableaux de

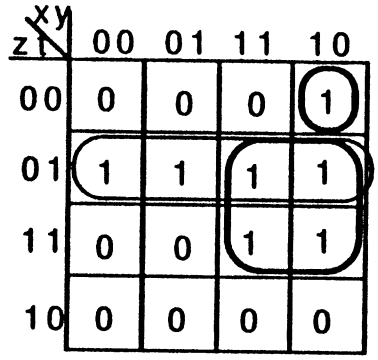
Karnaugh suivants : (les trois monômes globaux essentiels sont marqués en gras)

Première base irrédondante : 5 monômes globaux

{ (¬x.z, [1,0]), (x.¬y.¬z.¬t, [1,1]), (x.t , [0,1]), (¬z.t, [0,1]), (¬x.t, [1,0]) }



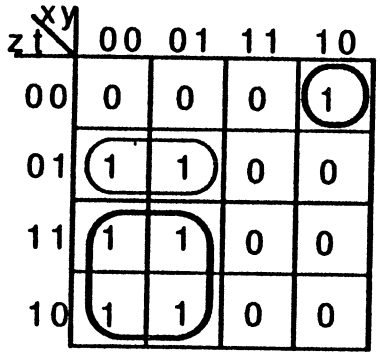
f1



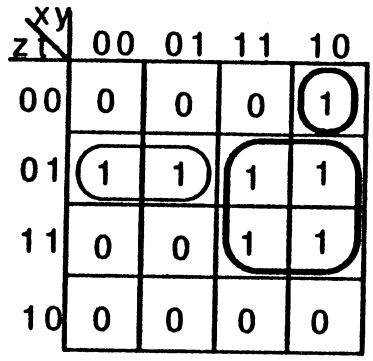
f2

Deuxième base irrédondante : 4 monômes globaux

{ (¬x.z, [1,0]), (x.¬y.¬z.¬t, [1,1]), (x.t , [0,1]), (¬x.¬z.t, [1,1]) }



f1



f2

1.3. PROBLEMATIQUE DE LA MINIMISATION DEUX COUCHES

D'après ces définitions, nous pouvons constater qu'un PLA permet d'implémenter une fonction générale. La donnée d'une forme polynômiale globale de la fonction définit exactement le PLA. Un monôme global correspond à une ligne de PLA, la partie monôme correspondant à l'étage 'ET' du PLA, et le vecteur d'inclusion à l'étage 'OU'.

Le premier critère d'optimalité est la surface du circuit. La surface d'un PLA est fonction du nombre d'entrées, du nombre de sorties, et de la hauteur du PLA (nombre de lignes). Si l'on admet qu'en général le nombre d'entrées et de sorties sont fixes, le problème de la minimisation deux couches consistera donc à trouver un ensemble minimal de monômes globaux nécessaire à définir l'ensemble de fonctions de l'application donnée.

Ce problème peut être résolu grâce à la notion mathématique de base irrédondante globale. En effet, si le problème de la minimisation est de trouver une représentation de coût minimal, la fonction coût étant ici fonction du nombre de monômes globaux, nous avons le théorème suivant :

On peut toujours obtenir une représentation de coût minimum au moyen d'une base irrédondante.

En effet, soit une représentation de coût minimum et M un monôme qui y figure. Si M n'est pas premier, on peut trouver un monôme premier M' > M, donc une base de même coût. Soit maintenant une base non irrédondante. Par définition un monôme au moins peut être supprimé, il existe donc une base irrédondante de coût inférieur.

Une fonction générale peut posséder plusieurs bases irrédondantes (ayant un nombre de monômes différent), il faudra donc trouver dans cet ensemble celle possédant le moins de monômes pour être sûr d'obtenir un PLA de taille minimale.

Pour répondre au deuxième critère lié aux performances du circuit (nombre de transistors), il nous faut définir une autre fonction coût. Elle est égale au nombre de variables des monômes v (nombre de transistors de l'étage 'ET') plus le nombre de 1 dans les vecteurs d'inclusions i (nombre de transistors de l'étage 'OU').
Il est encore possible à partir d'une base irrédondante globale de diminuer cette deuxième fonction coût sans augmenter le nombre de mônomes. Il est certain que pour chaque monôme présent dans une base irrédondante le nombre de variables ne peut être diminué puisque ces monômes sont premiers. Par contre le nombre de 1 dans leurs vecteurs d'inclusion peut être diminué.

Exemple :

Soit M1 = (a.d, [1,1]) et M2 = (a, [1,0]) deux monômes qui forment une base

irrédondante de (f1, f2). v + i = 6

M1'= (a.d, [0,1]) et M2 forment encore une expression de (f1, f2) qui a une fonction coût v + i = 5.

Nous verrons dans la dernière étape de l'algorithme de minimisation, comment nous pouvons minimiser cette deuxième fonction coût à partir d'une base irrédondante.

2. LES TECHNIQUES DE BASE DE LA MINIMISATION

Avant d'aborder la méthode générale du minimiseur deux couches, nous allons nous intéresser aux principaux problèmes que nous avons pu rencontrer tout au long de cette étude.

Beaucoup de procédures ont pu devenir performantes grâce à une stratégie simple mais fondamentale : la décomposition de Shannon [SHA48]. Des exemples d'utilisations de cette technique en vue de manipulations de fonctions logiques peuvent être trouvés dans [MOR70] et [HON72]. Nous commencerons par en rappeler le principe. Nous montrerons ensuite comment cette décomposition a permis de réaliser efficacement les procédures de bases de notre minimiseur : test d'inclusion, calcul d'incouvert, calcul de complément.

Notations :

Soit f une fonction booléenne simple d'une variable générale X .

$X = (x_1, ..., x_n)$ où x_i est une variable booléenne simple.

Une expression booléenne de f sera notée f(X) ou $f(x_1,... , x_n)$ (et souvent f pour ne pas surcharger les notations).

$\sim x_i$ pourra représenté indifférement la forme normale (x_i) ou complémentée ($\neg x_i$) d'une variable simple.

On notera $f(x_1,... ,v_i ,..., x_n)$ une expression booléenne de f où toute occurence de x_i aura été remplacée par une valeur booléenne v_i (1 ou 0).

$f(x_1,... ,1 ,..., x_n)$ (resp. $f(x_1,... ,0 ,..., x_n)$) pourra aussi être noté $fx_i(X)$ (resp. $f\neg x_i(X)$).

2.1. COFACTEUR ET DECOMPOSITION DE SHANNON

Théorème de Shannon :

$$f(X) = x_i \cdot fx_i(X) + \neg x_i \cdot f\neg x_i(X)$$

$fx_i(X)$ (resp. $f\neg x_i(X)$) est appelé cofacteur de f(X) par rapport à x_i (resp.$\neg x_i$)

On peut généraliser la notion de cofacteur par rapport à un monôme.

Définition du cofacteur :

Soit m un monôme, on appelle cofacteur de f par rapport à m l'expression $f_m(X)$ qui est obtenue en remplaçant dans f(X) chaque occurence des variables x_i apparaîssant dans m par :

1 si elle est sous forme normale dans m .

0 si elle est sous forme complémentée dans m .

Autrement dit, les variables apparaîssant dans m sont remplacées dans f, par les valeurs qui rendent m égal à 1. On trouve aussi souvent la notation f(X/m=1).

Exemple :

$f(X) = x_1.\neg x_2.x_3 + x_2.\neg x_3 + \neg x_1$ et $m = x_1.\neg x_2$

alors $f_m(X) = x_3$

Théorème du cofacteur :

$m.\ f(X) = m\ .\ f_m(X)$

Démonstration :

Lemme 1 :

$x_i\ .\ f(X) = x_i\ .\ fx_i(X)$ et $\neg x_i\ .\ f(X) = \neg x_i\ .\ f\neg x_i(X)$

La démonstration grâce au théorème de Shannon est immédiate.

A partir de ce lemme, la généralisation à plusieurs variables est immédiate.

2.2. APPLICATIONS AU PROCESSUS DE MINIMISATION

2.2.1. LE TEST D'INCLUSION

Ce test permet de savoir si un monôme est inférieur ou égal à une fonction (en

d'autres termes, inclusion du monôme dans une fonction). Il est utilisé dans la plupart des étapes de l'algorithme général. C'est lui qui va occuper la plus grande partie du temps de calcul dans l'algorithme général. Il était donc primordial de le rendre extrémement efficace, ceci a pu être fait grâce à la décomposition de Shannon. Il est à remarquer que tester l'inclusion d'un monôme global (m, I) dans un ensemble de fonctions revient à tester l'inclusion de m dans chaque fonction correpondante à une composante à 1 du vecteur d'inclusion I.

Théorème du test d'inclusion :

$m \leq f \Leftrightarrow f_m = 1$

Démonstration :

On rappelle que $m \leq f \Leftrightarrow m.\ f = m$.

(=>) $m \leq f \Rightarrow m.\ f = m$

$\Rightarrow (m.\ f)_m = m_m$

$\Rightarrow m_m.\ f_m = m_m \Rightarrow 1.\ f_m = 1 \Rightarrow f_m = 1$

(<=) $m.\ f_m = m \Rightarrow m.\ f = m$

$\Rightarrow m \leq f$

Donc prouver que m est inférieure ou égal à f revient à prouver que le cofacteur de f par rapport à m est une tautologie.

2.2.2. LA PREUVE DE TAUTOLOGIE

Théorème de la tautologie :

$f(x_1, \ldots, x_n)$ est une tautologie si et seulement si fx_i et $f\neg x_i$ sont des tautologies.

Démonstration :

(=>) Pour tout point de B^n, f(X) = 1.

Soit un point X tel que $x_i = 1$ alors $f(X) = 1.\ fx_i(X) + 0.\ f\neg x_i(X) = fx_i(X) = 1$ (de même avec xi = 0 ,nous obtenons $f\neg x_i(X) = 1$)

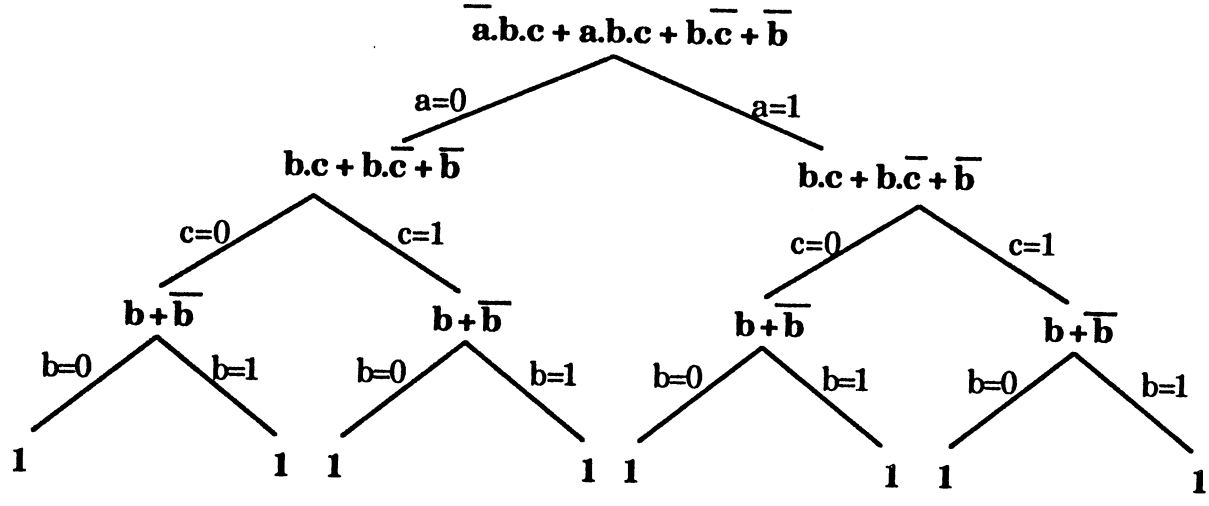
(<=) $fx_i = 1$ et $f\neg x_i = 1$ => $f = x_i.fx_i + \neg x_i.f\neg x_i = x_i + \neg x_i = 1$

On peut donc à partir de ce théorème élaborer une procédure récursive qui reconnaît si une espression booléenne est une tautologie. On remarque que dès qu'une branche de l'arbre de récursion aboutit à un échec, on peut immédiatement réfuter le test de tautologie pour la fonction de départ.

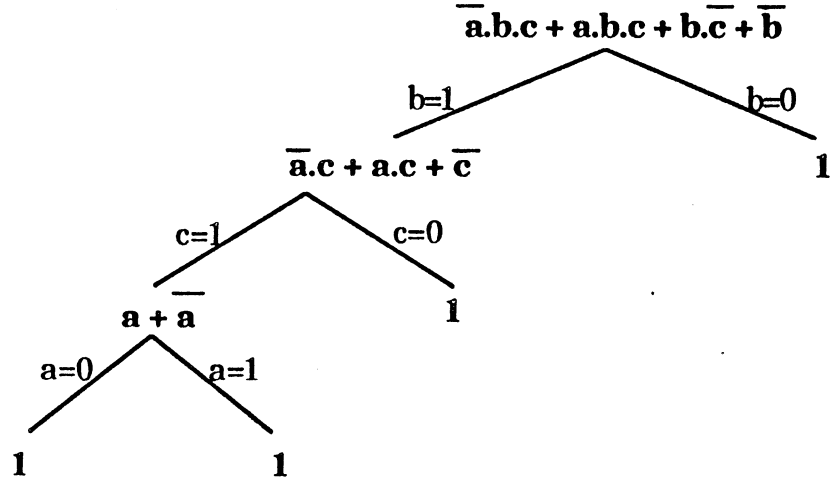
A chaque pas de la récursivité, il faut choisir une variable parmi celles présentes afin d'appliquer la décomposition de Shannon. La variable choisie est appelée variable de coupure. Ces choix successifs ont une importance capitale dans l'aboutissement plus ou moins rapide de l'algorithme.

Exemple :

Les deux arbres de récursion suivant montre la différence significative que l'on peut obtenir suivant l'ordre des variables de coupure.



Si nous choisisons comme ordre de variables de coupure (b, c, a), l'arbre de récursion devient nettement plus simple :



2.2.2.1. DEUX FORMES PARTICULIERES INTERESSANTES

La détection de deux formes particulières d'expression booléenne à chaque niveau de décomposition peut accélérer considérablement le processus :

1) Un monôme composé d'une seule variable apparaît dans l'expression booléenne :

$f(X) = \sim x_i + g(X)$

2) Une variable x_i apparaît dans l'expression sous une seule forme (normale ou complémentée) :

$f(X) = \sim x_i . g(X/x_i) + h(X/x_i)$

(notation : X/x_i signifie que la variable x_i n'apparaît plus dans l'expression).

Ces deux formes particulières sont intéressantes car elles simplifient la récursion. Comme nous allons le démontrer dans les théoremes suivants, dans ces deux cas nous n'avons besoin de descendre que dans une des deux branches de l'arbre de récursion.

Théorème_cas1 : (même théorème avec $\neg x_i$)

Si $f(X) = x_i + g(X)$ alors :

$$f(X) = 1 \Leftrightarrow f_{\neg x_i}(X) = 1$$

Démonstration :

$fx_i(X) = 1$ d'où le résultat.

Théorème_cas2 :

Si $f(X) = x_i.g(X/x_i) + h(X/x_i)$ alors :

$$f(X) = 1 \Leftrightarrow f_{\neg x_i}(X) = 1$$

(même théorème avec $\neg x_i$)

Démonstration :

$fx_i = g(X/x_i) + h(X/x_i)$

$f\neg x_i = h(X/x_i)$

donc $f\neg x_i = 1 \Rightarrow h(X/x_i) = 1 \Rightarrow g(X/x_i) + h(X/x_i) = 1$

$\Rightarrow fx_i = 1 \Rightarrow f = 1$

La détection et l'application de ces deux cas particuliers accélèrent considérablement la preuve de tautologie puisque le travail est divisé par deux et ceci le plus tôt possible. Dans l'exemple précédent, b puis c (au deuxième niveau) vérifient le théorème_cas1.

Nous pouvons généraliser ces deux théorèmes à plusieurs variables.

Théorème_1g :

si $f(X) = \sim x_l + ... + \sim x_k + g(X)$ alors

$$f(X) = 1 \Leftrightarrow f_m(X) = 1 \quad \text{avec } m = \neg\sim x_l . ... .\neg\sim x_k$$

Démonstration :

Elle est immédiate par application successive du théorème_cas1 sur les variables $x_l ... x_k$.

Nous pouvons faire de même pour le théorème_cas2:

Théorème_2g :

Si x_l ... x_k sont monoformes dans une fonction f (de forme $\sim x_l$ ... $\sim x_k$) alors

$$f = 1 \Leftrightarrow f_m(X) = 1 \quad \text{avec } m = \neg\sim x_l . \ldots . \neg\sim x_k$$

Démonstration :

De même que pour le théorème précedent la démonstration se fait aisément par application successive du théorème_cas2 sur les variables x_l... x_k .

Il est ainsi possible d'éviter k étapes de récursions grâce à cette généralisation. Il est bien sûr possible d'appliquer ces deux proprietés en même temps en prenant comme ensemble de variables de coupure, les variables vérifiant l'une ou l'autre de ces deux propriétés. Nous détecterons donc à chaque pas ces deux formes particulières pour pouvoir obtenir dés que possible ce gain de complexité.

2.2.2.2.CHOIX DE LA VARIABLE DE COUPURE DANS LE CAS GENERAL

Quand aucune des deux formes particulières précédentes n'est détectés, le choix de la variable de coupure peut encore jouer un rôle important dans la complexité de l'algorithme. Il n'est pas possible de choisir cette variable, en vue de la simplification de l'algorithme plusieurs niveaux plus loin, sans ajouter des calculs complexes qui ne feraient qu'augmenter la complexité de l'algorithme. Nous essayons donc de réduire la complexité des deux cofacteurs (fx_i et $f\neg x_i$) de l'étape de récursion suivante.

Les expressions que nous aurons à traiter seront des formes polynômiales. A chaque étape de la récursion le travail effectué est donc l'instanciation de la variable de coupure dans chacun des monômes de l'expression. Ce travail est fonction avant tout du nombre de monômes de celle-ci. Il faut donc choisir la variable de coupure x_i telle que fx_i et $f\neg x_i$ comportent le moins possible de monômes, puisque la récursion va

se faire sur ces deux nouvelles expressions.

Soit D_i (resp. C_i) le nombre de monômes de f où la variable x_i apparait sous forme normale (resp. complémentée). Soit L le nombre de monômes de f. Si l'on considére que le temps T du travail éffectué à la prochaine étape ne dépend que du nombre de monômes des deux expressions fx_i et $f\neg x_i$, nous avons $T= t(L-D_i) + t(L-C_i)$. Nous pouvons estimer que la fonction t est proportionnelle au nombre de monômes, il faudra minimiser $L-D_i + L-C_i$, c'est à dire maximiser $D_i + C_i$.

2.2.2.3. AUTRES AMELIORATIONS

Il y a dans la plupart des cas réfutation du test. La preuve de tautologie nous servant principalement pour le test d'inclusion, il y a fort peu de chance qu'un monôme quelconque soit inclus dans une fonction. Dans la pratique, nous avons pu constater que 90 pour 100 des tests aboutissent pour notre application à une réfutation. Nous avons donc essayer de détecter une potentielle réfutation le plus tôt possible dans l'arbre de récursion.

Théorème 1 :

Une fonction à n variables ne peut être une tautologie si la somme des tailles des monômes qui la représente est infèrieur à 2^n .

Démonstration:

Une fonction à n entrées est une tautologie si elle couvre les 2^n points possibles. Chaque monôme couvre un ensemble de points de cardinalité égale à sa taille. Donc si l'ensemble des points couverts par la totalité des monômes a une cardinalité infèrieure à 2^n, on est sûr que cet ensemble de monômes ne pourra couvrir B^n .

Remarque : ce test trés rapide ne sera appliqué que si n n'est pas trop grand ($n \leq 10$), n diminuant dans la récursion.

Théorème 2 :

Soit $f(x_1 \ldots x_n)$ une fonction à n variables .

Si toutes les variables $x_1 \ldots x_n$ sont présentes sous une seule forme dans f $(x_1 \ldots x_n)$ alors forcément f n'est pas une tautologie.

Démonstration :

Soit $\sim x_1, \ldots, \sim x_n$ les formes des respectives des variables dans f $(x_1 \ldots x_n)$.

D'après le théorème_2g :

$$f = 1 \quad \Leftrightarrow \quad f_m = 1 \text{ avec } m = \neg(\sim xl). \ldots .\neg(\sim xn)$$

Or $f_m = 0$ puisque les variables sont sous la forme (~xl), ..., (~xn) dans f.

Théorème 3 :

Si une variable x_i est présente dans tous les monômes d'une fonction sous une seule forme $\sim x_i$, alors f ne peut être une tautologie.

Démonstration:

D'après le théorème_cas2 : $f = 1 \quad \Leftrightarrow \quad f\neg(\sim x_i) = 1$

Or x_i appartient à tous les monômes de f donc $f\neg(\sim x_i) = 0$ donc f ne peut pas être une tautologie.

2.2.3. LA COMPLEMENTATION D'UNE FONCTION BOOLEENNE

La complémentation d'une fonction booléenne est un problème qui peut être d'une trés grande complexité si l'on utilise par exemple les opérateurs classiques de l'algèbre de Boole (Loi de Morgan). Nous allons montrer comment la décomposition de Shannon peut s'adapter au problème de la complémentation et permet d'obtenir un algorithme trés efficace.

Théorème 1 :

$$\neg f = x_i . \neg (f x_i) + \neg x_i . \neg (f \neg x_i)$$

Démonstration:

D'après le théorème de Shannon nous avons :

$$f = x_i.\ fx_i + \neg x_i.\ f\neg x_i$$

En appliquant le théorème de De Morgan :

$$\neg f = (\neg x_i + \neg (fx_i)) . (x_i + \neg (f\neg x_i))$$

$$= x_i.\neg (fx_i) + \neg x_i.\ \neg (f\neg x_i) + \neg (fx_i).\neg (f\neg x_i) + x_i.\neg x_i$$

$$= x_i.\neg (fx_i) + \neg x_i.\ \neg (f\neg x_i) + \neg (fx_i).\neg (f\neg x_i) \quad \text{puisque } x_i.\neg x_i=0$$

or $\neg (fx_i).\neg (f\neg x_i) = \neg (fx_i).\neg (f\neg x_i).\ (x_i + \neg x_i)$ puisque $x_i + \neg x_i = 1$

donc en développant :

$$\neg f = x_i.\neg (fx_i) + \neg x_i.\ \neg (f\neg x_i) + \neg (fx_i).\neg (f\neg x_i).\ x_i + \neg (fx_i).\neg (f\neg x_i).\ \neg x_i$$

et nous avons

$$\neg (f_{xi}).\neg (f_{\neg xi}).\ x_i \leq x_i.\neg (f_{xi}) \text{ et } \neg (f_{xi}).\neg (f_{\neg xi}).\ \neg x_i \leq \neg x_i.\ \neg (f_{\neg xi})$$

donc

$$\neg f = x_i.\ \neg (f_{xi}) + \neg x_i.\ \neg (f_{\neg xi})$$

A partir de cette méthode récursive, nous pouvons améliorer le processus en regardant un cas particulier.

Théorème 2 :

si une fonction f comprend un ou plusieurs monômes de longueur 1 , $\sim x_l ,\ldots,\sim x_k$ alors nous avons :

$$\neg f = m\ .\ \neg(f_m) \quad \text{avec} \quad m = \neg(\sim x_l)\ .\ \ldots\ .\neg(\sim x_k)$$

Démonstration:

Lemme : si $f(X) = \sim x_i + g(X)$ alors $\neg f = \neg(\sim x_i).\neg(f\neg(\sim x_i))$

Démonstration :

nous avons :

$f{\sim}x_i = 1 + g{\sim}x_i = 1$

donc $\neg f = {\sim}x_i . \neg (1) + \neg{\sim}x_i . \neg (f\neg{\sim}x_i)$

$= \neg{\sim}x_i . \neg (f\neg{\sim}x_i)$

En appliquant par récurrence ce lemme sur le nombre k de monômes, nous démontrons le théorème.

Pour limiter au maximum, à chaque étape de récursion le nombre de monômes des sous_fonctions à complémenter, nous choisirons (de la même manière que dans la preuve de tautologie) la variable la plus appropriée.

Pour éviter l'explosion dûe au nombre de mônomes qui remontent à chaque niveau de l'arbre de récursion, nous appliquons une sorte de minimisation à chaque niveau. Ceci peut être obtenu simplement grâce aux deux théorèmes suivants.

Théorème 3 :

S'il existe un monôme m élément de $\neg fx_i$ et de $\neg (f\neg x_i)$ alors

$\neg f = x_i . (\neg (fx_i) - \{m\}) + \neg x_i . (\neg (f\neg x_i) - \{m\}) + m$

où la notation f - {m} représente l'expression booléenne de f où le monôme m a été supprimé.

Démonstration :

$\neg f = x_i . (\neg (fx_i) - \{m\}) + \neg x_i . (\neg (f\neg x_i) - \{m\}) + x_i . m + \neg x_i . m$

et $x_i . m + \neg x_i . m = m$

La détection d'un monôme identique dans les deux expressions $\neg (fx_i)$ et $\neg (f\neg x_i)$ permet ainsi une simplification rapide mais significative de la fonction résultante à chaque niveau de la récursion (gain d'un monôme et d'une variable).

Théorème 4 :

S'il existe un monôme m élément de $\neg fx_i$ et un monôme m' élément de $\neg (f_{\neg xi})$ tel que $m' \leq m$ alors :

$$\neg f = x_i . \neg (fx_i) + \neg x_i . (\neg (f\neg x_i) - \{m'\}) + m'$$

Démonstration :

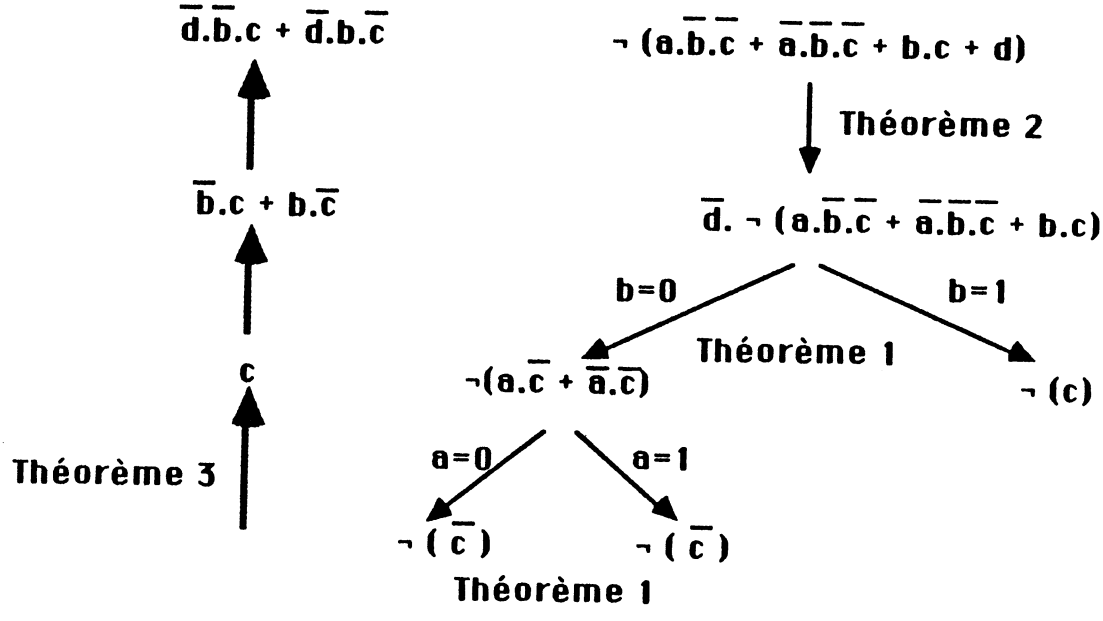
De la même façon que pour le théorème précédent nous obtenons :

$$x_i.m + \neg x_i.m' = x_i.m + m'$$

Quelques améliorations ont été apportées à cet algorithme de base telle que : Si la fonction ne comprend plus qu'un monôme, alors la loi de De Morgan est appliquée directement.

Exemple :

La figure suivante montre la descente et la remontée dans l'arbre de récursion obtenu par l'algorithme proposé pour un exemple trés simple.



2.2.4. CALCUL DE L'INCOUVERT D'UN MONOME

L'incouvert d'un monôme est une notion importante qui servira à plusieurs reprises dans l'algorithme de minimisation. Nous en donnerons tout d'abord une définition précise puis nous élaborerons une méthode de calcul efficace.

Définition 1 :

La partie incouverte d'un monôme global M = (m, I) par rapport à une somme de monôme globaux E (notée PI(M / E)) est définie par :

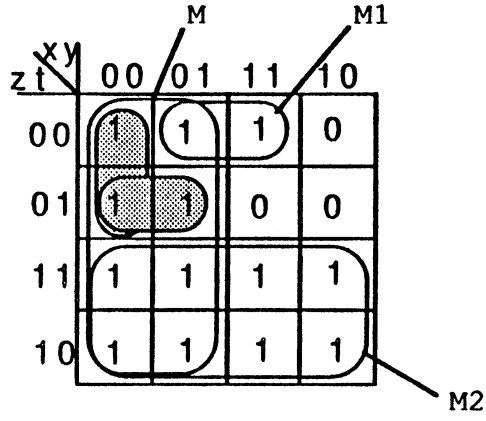
$$PI(M / E) = M - E = M.\neg E$$

PI(M / E) couvre tous les monômes canoniques globaux couverts par M et non couverts par E.

Remarque :

PI(M / E) n'est pas forcément un monôme.

Par exemple pour une fonction simple :



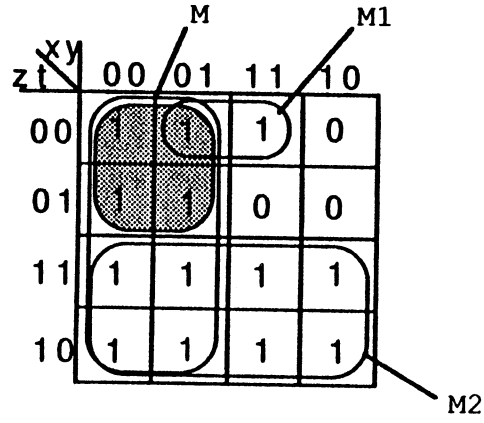
M =¬x et E= M1 + M2, nous avons PI(M / E)= ¬x.¬y.¬z + ¬x.¬.z.t

Le nombre de monômes de PI(M / E) peut devenir trés important, nous allons simplifier le problème en approximant la partie incouverte au plus petit monôme la contenant.

Définition 2 :

L'incouvert d'un monôme global M par rapport à une somme de monômes globaux E (noté INC(M/ E)) est le plus petit monôme global contenant PI(M / E).

Dans l'exemple précédent INC(M/ E) = $\neg x.\neg z$



Pour calculer l'incouvert d'un monôme efficacement, nous allons employer une fois de plus la décomposition de Shannon. Pour cela, il nous faut tout d'abord généraliser le théorème de Shannon et la notion de cofacteur pour une expression globale.

Théorème de Shannon généralisé :

Soit E une expression globale, nous avons :

$$E = X_i . Ex_i + \neg X_i . E\neg x_i$$

Où Ex_i (resp. $E\neg x_i$) est l'expression globale E où chaque occurrence de x_i (resp. $\neg x_i$) a été remplacé par 1 (resp. 0).

X_i (resp. $\neg X_i$) représente le monôme global (x_i ,[1,...,1]) (resp. ($\neg x_i$,[1,...,1])).

Définition 3 :

Soient un monôme global M= (m, I) et une expression globale E définie par le vecteur de fonctions simples [f1,..., fk] , le cofacteur de E par rapport à M (noté E_M) est

défini par le vecteur des cofacteurs des fonctions simples par rapport à m $[f1_m,\ldots,fk_m]$.

Théorème 1 :

Comme pour les expressions booléennes simples, nous avons l'égalité :

$$M.\,E = M.\,E_M$$

Démonstration :

En repassant aux expressions booléennes simples définies par E, la généralisation est immédiate.

Lemme 1 :

$$\neg(E_M) = (\neg E)_M$$

Démonstration :

Si E définit $(f_1,\ldots,f_k)$ et M= (m,I) alors $\neg(E_M)$ est définie par $\{\neg(f1_m),\ldots,\neg(fk_m)\}$ et $(\neg E)_M$ est définie par $\{(\neg f1)_m,\ldots,(\neg fk)_m\}$.

Il nous faut donc montrer que pour expression booléenne simple f, nous avons :

$$\neg(f_m) = (\neg f)_m$$

D'aprés le théorème de shannon nous avons : $\neg f = x.\,(\neg f)_x + \neg x.\,(\neg f)_{\neg x}$

mais nous avons aussi : $\neg f = x.\,\neg(f_x) + \neg x.\,\neg(f_{\neg x})$

En faisant x= 0 puis x= 1 nous obtenons $(\neg f)_x = \neg(f_x)$ et $(\neg f)_{\neg x} = \neg(f_{\neg x})$

La généralisation à un produit de variables est immédiate.

Définition 4 :

On définit PPMC (E) comme le plus petit monôme global tel que l'expression globale E est inférieure ou égale à M.

Nous sommes ici obligé d'étendre la définition de monôme simple : un monôme pourra aussi être la fonction tautologie 1. Un monôme global pourra aussi être (1, I) avec I = (1,...,1). Ainsi le PPMC (E) existe pour toute expression E puisque nous avons

quel que soit E : $E \leq (1, [1,\ldots,1])$.

Lemme 2 :

$PPMC(M.\ E_M) = M.\ PPMC\ (E_M)$

Démonstration :

Soit une variable x.

Nous avons $PPMC(x.\ E_x) = x.\ PPMC\ (E_x)$. Il suffit d'affecter x par 0 puis 1. (de même avec $\neg x$). La généralisation à plusieurs variables est immédiate.

Théorème 2 :

$INC\ (M/E) = M\ .\ PPMC\ (\neg(\ E_M)\ )$

Démonstration :

$INC\ (M/E) = PPMC\ (\ M.\ \neg E)$ par définition

$= PPMC\ (M.\ (\neg E)_M)$ (théorème 1)

$= M.\ PPMC(\ (\neg E)_M\ )$ (lemme 2)

$INC\ (M/E) = M\ .\ PPMC\ (\neg(\ E_M)\ )$ (lemme 1)

Pour calculer le INC(M/E), il suffit donc de savoir calculer le PPMC d'une expression. Le théorème suivant donne une méthode récursive analogue à la décomposition de Shannon.

Théorème 3 :

$$PPMC(E) = PPMC\ (\ X.PPMC(E_x) + \neg X.PPMC(E_{\neg x})\ )$$

Démonstration :

(1) $PPMC(E_x) \geq E_x$ et $PPMC(E_{\neg x}) \geq E_{\neg x}$

Donc $X.PPMC(E_x) + \neg X.PPMC(E_{\neg x}) \geq X.E_x + \neg X.E_{\neg x} = E$ d'après le théorème de Shannon généralisé (X représentant ici le monôme global (x, [1,...,1])).

D'où PPMC ($X.PPMC(E_X) + \neg X.PPMC(E_{\neg X})$) ≥ PPMC(E)

(2) PPMC(E) ≥ E = $X.E_X + \neg X.E_{\neg X}$

Donc PPMC(E) ≥ $X.E_X$ et PPMC(E) ≥ $\neg X.E_{\neg X}$

D'où PPMC(E) ≥ PPMC($X.E_X$) et PPMC(E) ≥ PPMC($\neg X.E_{\neg X}$)

Donc PPMC(E) ≥ PPMC($X.E_X$) + PPMC($\neg X.E_{\neg X}$) et

Ainsi PPMC(E) ≥ PPMC(PPMC($X.E_X$) + PPMC($\neg X.E_{\neg X}$))

Et on a PPMC($X.E_X$) = $X.PPMC(E_X)$ d'après le lemme 2 (idem pour $\neg x$)

Donc PPMC(E) ≥ PPMC($x.PPMC(E_X) + \neg x.PPMC(E_{\neg X})$)

Définition 5 :

On appelle PPMCC(E) le plus petit monôme global contenant le complément de l'expression globale E.

On a donc par définition PPMCC(E) = PPMC($\neg$E).

Théorème 4 :

PPMCC(E) = PPMC ($x.PPMCC(E_X) + \neg x.PPMCC(E_{\neg X})$)

Démonstration :

PPMCC (E) = PPMC ($\neg$E) = PPMC ($x.PPMC((\neg E)_X) + \neg x.PPMC((\neg E)_{\neg X})$)

d'après le théorème 3

PPMC($\neg$E) = PPMC ($x.PPMC(\neg(E_X)) + \neg x.PPMC(\neg(E_{\neg X}))$) d'après le lemme 1

= PPMC ($x.PPMCC(E_X) + \neg x.PPMCC(E_{\neg X})$) par définition du PPMCC.

Nous évitons ainsi le calcul de $\neg(E_M)$ dans le calcul du PPMC($\neg(E_M)$). Nous avons donc un algorithme récursif pour calculer le plus petit monôme global contenant le complément d'une expression globale. Nous savons donc calculer efficacement l'incouvert d'un monôme global puisque INC(M/E) = M. $PPMCC(E_M)$.

Il ne nous reste plus qu'à calculer le PPMC de la somme de deux monômes globaux.

Définition 6 :

Soit l'opération $\pm$ définie sur deux monômes globaux M=(m,l) et M'=(m',l') par :

$M \pm M' = (m'' , l'')$ où $l''= l + l'$ et

$$m''= \prod \sim x_i \quad \text{tel que } \sim x_i . m= m \text{ et } \sim x_i . m'= m'$$

S'il n'existe aucun variable x_i vérifiant cette propriété alors m''=1.

C'est à dire m'' est le produit des variables qui sont présentent dans m et m' sous la même forme.

Posons $(0, i) \pm (m', i')= (m', i')$ et $(1, i) \pm (m',i')= (1, i+i')$ où 0 représente le monôme vide et 1 la tautologie.

Exemple :

$(a.b.\neg c , [1,0,1]) \pm (\neg a.b , [0,1,1]) = (b , [1,1,1])$

Théorème 5 :

PPMC (M+ M') = $M \pm M'$

Démonstration :

Soit M''= (m'', l'')= $M \pm M'$, nous avons $M \leq M''$ et $M' \leq M''$

Montrons qu'il n'existe pas de monôme global $M_1=(m_1, l_1)$ strictement plus petit que M'' et tel que $M \leq M_1$ et $M' \leq M_1$.

(1) Si $M \leq M_1$ et $M' \leq M_1$ alors $l \leq l_1$ et $l' \leq l_1$

donc $l_1 \geq l + l' = l''$

donc si M1 existe forcément, $l_1= l''$

(2) Montrons que si $m \leq m_1$ et $m' \leq m_1$ alors $m'' \leq m_1$

si $m \leq m_1$ et $m' \leq m_1$ alors toute variable de m_1 est présente sous la même forme dans m et m' . Or par définition m" est composé de toutes les variables présentes dans m et m' sous la même forme, donc forcément m" comporte au moins toutes les variables (sous la même forme) de m_1. Donc $m''.m_1 = m''$, c'est à dire $m_1 \geq m''$.

2.3. DETECTION DES MONOMES ESSENTIELS

Les monômes essentiels sont les monômes qui sont les seuls à couvrir au moins un point parmi tous les monômes premiers. Ils sont donc dans toutes les bases. Dans la base complète, les monômes essentiels sont ceux qui sont obligatoires, il serait alors évident de les détecter. En effet, grâce au test d'inclusion, il est facile de détecter si chacun des monômes M est inférieur à la base complète moins M.

Dans l'algorithme de notre minimiseur, nous n'obtiendrons jamais la base complète. Il nous faut donc trouver une technique qui permet de détecter les monômes essentiels dans une base quelconque.

Différentes méthodes pour trouver les monômes essentiels dans une base quelconque peuvent être trouvées dans [BAH81], et [BRA84]. Nous avons repris l'algorithme employé dans [BRA84], grâce auquel nous pourrons aussi détecter une nouvelle catégorie de monômes qui possède des propriétés intéressantes pour notre algorithme : les monômes stables (voir chap. 3.5).

Il nous faut définir avant tout, un ensemble d'opérations sur les monômes globaux.

Rappel 1 :

Soient m_1 et m_2 deux monômes. Le consensus [TIS65] entre m_1 et m_2 existe si m_1 et m_2 possèdent une seule même variable x sous deux formes différentes. Il est alors défini par $r_1.r_2$ avec $m_1 = x.\, r_1$ et $m_2 = \neg x.\, r_2$

Soient M= (m, I), M'= (m', I') deux monômes globaux.

Rappel 2 :

Si il existe un consensus entre m et m'

et si $l \cdot l' \neq 0$ alors

le consensus global de M et M' est défini par : (consensus(m, m') , $l \cdot l'$)

Définition 1 :

Si $m \cdot m' \neq 0$ et $(l + l' = l$ ou $l + l' = l')$

On appellera produit-simple de M et M' , le monôme global :

($m \cdot m'$, $l \cdot l'$)

Le produit-simple couvre tous les points qui appartiennent à la fois à M et M'.

Définition 2 :

Si $m \cdot m' \neq 0$ et $(l + l' > l$ et $l + l' > l')$

On appellera produit-complexe de M et M', le monôme global :

($m \cdot m'$, $l + l'$)

Définition 3 :

Soit © l'opération entre deux monômes globaux M et M' définie par :

M © M' = consensus global(M, M') (s'il existe)

sinon

= produit-simple (M, M') (s'il existe)

sinon

= produit-complexe (M, M') (s'il existe)

sinon

= 0

2.3.1. L'ALGORITHME

Début

Pour tout monôme M d'une base B *faire*

COUV-ESS(M) <- { M © M' tel que M' ∈ (B - {M}) + PHI } ;

si M n'est pas inférieur ou égal à COUV-ESS(M) *alors*

M est essentiel

finpour

fin

B et PHI dénote respectivement une base quelconque et la couverture à ϕ de la fonction générale traitée.

2.3.2. JUSTIFICATION DE L'ALGORITHME

Rappel de la définition du monôme essentiel :

Un monôme est essentiel pour une fonction s'il est le seul parmi tous les monômes premiers à couvrir au moins un point de la borne inférieure de cette fonction.

Lemme :

Si M n'est pas essentiel, pour tout point P= (p, IP) de M qui n'est pas élément de PHI, il existe un point P'= (p', IP') de B + PHI qui n'est pas élément de M et tel que (p= p' et IP≠ IP') ou (p et p' adjacent et IP= IP').

Démonstration :

M n'est pas essentiel donc il existe M' premier tel que M' couvre P.

* si M > M' (avec M= (m, I) et M'= (m', I')) alors il existe un point P' élément de M' qui n'appartient pas à M et tel que P= P' et IP≠ IP' puisque M et M' sont premiers.

* sinon

Soit le monôme M. M' . Il existe puisque P en fait partie.

Soit une variable x_i de M. M' qui n'apparaît pas dans M'. Elle existe puisque M et M' sont premiers et m' n'est pas inclus dans m.

Soit le point P'=(p',IP') tel que

IP'= IP et (p_j'= p_j pour tout j différent de i et p_i'= $\neg p_i$).

P' est élément de B + PHI puisqu'il appartient à M', P' n'appartient pas à M et nous avons p adjacent à p' et IP= IP'.

Théorème :

M ≤ COUV-ESS(M) ⇔ M n'est pas essentiel.

Démonstration :

(=>)

M ≤ COUV-ESS(M)

=> Pour tout point P de M, il existe un monôme M' de COUV-ESS(M) tel que P ≤ M'

* Soit M' est le résultat d'un produit-simple de M et d'un monôme de PHI :

nous avons : P ≤ PHI ce n'est donc pas un point de la borne infèrieure.

* Soit M' est le résultat d'un produit-simple de M et d'un monôme M" de B-{M} :

nous avons P ≤ M" donc P est couvert par un monôme premier autre que M.

-Soit M' est le résultat d'un consensus global de M et d'un monôme M" de B-{MG} :

Nous avons : M' couvre des points qui ne sont pas couverts par M donc si nous agrandissons au maximum M', nous obtenons un monôme premier différent de M qui contient P.

-Soit M' est le résultat d'un produit-complexe de M et d'un monôme M" de B- {MG} :

Nous avons : si M'= (m', l') et M= (m, l) ; l' > l par définition du produit complexe donc M' contient des points qui ne sont pas couverts par M. Donc si nous agrandissons au maximum M', nous obtenons un monôme premier différent de M qui contient P.

Ceci étant valable pour tout point P de M, si P est couvert par COUV-ESS(M), il existe un monôme premier qui couvre P donc M n'est pas essentiel.

(<=) Il faut montrer :

Pour tout point P de M tel que P est un point de la borne inférieure, s'il existe un monôme premier M' (≠ M) qui couvre P alors P est couvert par COUV-ESS(M).

a) si M' est élement de B - {M} :

Donc P est couvert par le produit-simple de M et M' ; donc il est couvert par COUV-ESS(M).

b) si M' n'est pas élément de B - {M} :

Si P est couvert par M' alors il existe P' tel que : (Lemme)

*soit (p adjacent à p' et IP = IP')

Il existe un monôme alors M"=(m",I") de B + PHI qui couvre P' :

D'après la théorie sur les consensus de [KUN68], il existe un consensus entre m" et m (puisque p et p' sont adjacents) qui contient p et p'.

Donc il existe M" tel que le consensus global de M" et M contient P. CQFD

*soit (p= p' et IP≠ IP')

Il existe un monôme M" de B + PHI tel que P' est élément de M" :

puisque p= p' et I" + I n'est inférieur ni à I ni à I' (car IP ≤ I et IP' ≤ I' et IP≠ IP'). Il existe un produit-complexe entre M" et M qui contient donc P. CQFD

3. LA MINIMISATION DEUX COUCHES DANS ASYL

3.1. PRINCIPE ET ALGORITHME GENERAL

La minimisation en vue d'une implantation d'une fonction générale sur PLA doit répondre à deux critères :

-En premier lieu, la minimisation du nombre de monômes globaux dont dépend la taille du circuit.

- deuxièmement le nombre de variables et de 1 dans les vecteurs d'inclusion des monômes globaux, dont dépend le nombre de transistors dans le circuit pour répondre à des critères de performances.

L'extraction d'une base globale irrédondante minimale (paragraphe 1.3) répond au premier critère.

Une minimisation exacte consisterait à :

-Premièrement, générer tous les monômes globaux premiers

-Puis à extraire à partir de la base complète, toutes les bases irrédondantes et à choisir une de celles qui contient le moins de monômes.

Une telle méthode n'est pas concevable vu la taille des applications que nous voulons minimiser (nombre de variables $\approx$ 50 , nombre de fonctions simples $\approx$ 50).

Une fois convaincu de cet état des choses, il nous a fallu élaborer une méthode qui, d'une part limite le nombre de monômes premiers générés, d'autre part grâce à des heuristiques appropriées extrait une base irrédondante tendant vers la minimalité. Pour arriver le plus près possible de la minimalité absolue, et ceci en limitant le nombre de monômes générés, nous nous sommes inspiré de la méthode du recuit simulé. Notre approche va s'approcher du minimum exact en passant par des minimum locaux. On sortira d'un minimum local en augmentant la fonction coût (nombre de monômes) de manière à retomber sur un minimum local plus bas.

Nous trouvons une base globale grâce à l'obtention d'une base de chaque fonction simple (bases locales). A partir de cette base globale, nous extrayons une base irrédondante tendant vers une minimalité relative (au vu de la base de départ).

Nous sortons ensuite de ce minimum local en ajoutant de nouveaux monômes, afin de retomber si possible sur un minima plus bas par une nouvelle extraction de base irrédondante. Nous itérons sur ces deux étapes jusqu'à ce que l'on arrive à une solution stable. Comme nous le verrons au paragraphe 3.6, le calcul et le choix des nouveaux monômes sont primordiaux, mais facilement réalisables du fait que l'on soit parti de monômes premiers de chaque fonction simple.

Nous allons tout d'abord donner un résumé de la philosophie générale du minimiseur proposé, puis nous reverrons en détail chacune de ses étapes.

Etape 1 : Minimisation indépendante de chaque fonction .

Nous obtenons une base de chaque fonction simple.

Etape 2 : Obtention d'une base globale.

A partir de l'ensemble des bases de chaque fonction, nous formons une expression polynômiale globale de la fonction générale. Les vecteurs d'inclusion de chacun des monômes de cette expression sont agrandis au maximum; nous obtenons ainsi une base globale.

Etape 3 : Obtention d'une base irrédondante globale.

Nous extrayons de la base globale obtenue, une base irrédondante tendant vers une minimalité relative à la base initiale, c'est à dire sans adjonction de nouveaux monômes ou transformation des monômes présents.

Etape 4 : Détection des monômes globaux essentiels.

Nous effectuons une itération sur les étapes 3 et 5, il est intéressant de déterminer les monômes essentiels au moment de la première passe. Cette étape permet de gagner du temps, les monômes essentiels étant mis de coté une fois pour toute. Mais elle est aussi très importante pour l'étape 5, afin de limiter le nombre de nouveaux monômes introduits.

Etape 5 : Génération de nouveaux monômes globaux premiers

Nous rajoutons de nouveaux monômes choisis avec soins afin de sortir du minimum local atteint à l'étape 3, et de retomber dans un minimum plus bas.

ITERATION : On boucle ainsi sur l'étape 3 puis 5, jusqu'à ce qu'il n'y ait plus de gain en nombre de monômes, ou en nombre de variables et de 1 dans les vecteurs d'inclusions dans la totalité des monômes globaux.

Etape 6 : Retour à des ensembles irrédondants localement.

Nous avons obtenu une base irrédondante pseudo-minimale. Mais il est encore possible de gagner des transistors dans le PLA d'implémentation. Ceci est possible en réduisant tout d'abord les vecteurs d'inclusion des monômes globaux (ensembles irrédondants localement) et par la suite en réduisant le nombre de variables de ceux-ci.

3.2. PREMIERE ETAPE : OBTENTION DE BASES LOCALES

Nous commençons donc à obtenir une base pour chaque fonction simple définissant la fonction générale de départ. Comme nous le verrons par la suite, il est important d'obtenir des monômes premiers pour chaque fonction.
Deux optiques s'offrent à nous :

- nous ne voulons pas générer trop de monômes pour les grosses applications. Nous nous limiterons à générer dans cette étape des bases locales quelconques. Ces bases ne seront pas forcément irrédondantes. Cette propriété n'étant de toute façon pas conservée dans le passage à une base globale (étape 2), il est inutile de rendre les bases locales obtenues irrédondantes.

- nous pouvons nous permettre une minimisation plus exacte et nous pourrons obtenir dans cette étape des bases locales complètes, nous rapprochant ainsi dans l'étape suivante de la base globale complète.

3.2.1. BASES LOCALES QUELCONQUES

Nous voulons obtenir pour chaque fonction simple composant la fonction

générale, une base, cela d'une manière efficace et de façon à ne pas augmenter le nombre de monômes de départ de chaque fonction simple. Pour ce faire, nous avons opté pour une méthode basée sur le test d'inclusion (parag. 2.2.1.). Elle s'est avérée trés rapide vu l'efficacité du test d'inclusion proposé. La méthode est simple, nous partons de la liste de monômes de la borne supérieure de la fonction, que nous rendons premiers par agrandissement maximal.

Elle peut se résumer en deux étapes principales :

- Prendre un monôme, le rendre premier en l'agrandissant au maximum.
- Supprimer tous les monômes inférieurs à ce monôme agrandi.

Agrandir un monôme consiste à essayer de lui "enlever" une ou plusieurs variables. Le monôme ainsi agrandi doit être encore inférieur à la borne supérieure de la fonction. Une fois le monôme ainsi agrandi dans toutes les directions possibles, nous sommes sûr d'obtenir un monôme premier de la fonction.

Résumé de l'algorihme :

1. Marquer les monômes de départ avec l'indicateur "non_premier."
2. Prendre un monôme m marqué "non_premier" s'il en existe;
   sinon terminer, la liste des monômes restant forme une base de la fonction.
3. S'il existe une des variables de m qui n'a jamais été supprimée alors générer le diviseur m' en supprimant cette variable de m. Sinon aller à l'étape 7.
4. Tester l'inclusion de m' dans l'ensemble des monômes. (voir paragraphe 2.2.1)
5. Si le test est positif alors remplacer m par m'.
6. Aller à l'étape 3.
7. Marquer m avec l'indicateur "premier".
8. Supprimer les monômes marqués "non_premier" inférieurs à m.
9. Aller à l'étape 2.

Exemple :

$f(x,y,z,t) = \neg x.\neg y.\neg z.\neg t + \neg x.y.\neg z.\neg t + \neg x.y.\neg z.t + \neg x.y.z.t + \neg x.y.z.\neg t$
$+ x.y.z.t + x.y.z.\neg t + x.\neg y.\neg z.\neg t + x.\neg y.z.t + x.\neg y.z.\neg t$

1. $\neg x.\neg y.\neg z.\neg t \rightarrow \neg y.\neg z.\neg t \leq f$ ($\rightarrow$ signifie "est agrandi en")
   On élimine $x.\neg y.\neg z.\neg t \leq \neg y.\neg z.\neg t$
2. $\neg x.y.\neg z.\neg t \rightarrow \neg x.\neg z.\neg t$
   Pas d'élimination puisqu'il n'existe aucun monôme de f inférieur à $\neg x.\neg z.\neg t$
3. $\neg x.y.\neg z.t \rightarrow \neg x.y.t \rightarrow \neg x.y$
   On élimine $\neg x.y.z.t$ et $\neg x.y.z.\neg t$
4. $x.y.z.t \rightarrow y.z.t \rightarrow y.z$
   On élimine $x.y.z.\neg t$
5. $x.\neg y.z.t \rightarrow x.z.t \rightarrow x.z$
   On élimine $x.\neg y.z.\neg t$.

Dans cet algorithme le nombre de monômes n'est jamais augmenté mais nous ne sommes pas sûr d'obtenir une base irrédondante. Dans l'exemple précédent y.z et $\neg x.\neg z.\neg t$ sont redondants.

Il n'existe pas de lien entre les propriétés d'irrédondance locale et globale, c'est à dire un ensemble de bases irrédondantes ne donneront pas forcément une base irrédondante globale. De plus nous avons par la suite une étape d'obtention de base irrédondante globale. Il était donc inutile, vu l'optique de cette étape, d'obtenir des bases locales irrédondantes.

3.2.2. BASES LOCALES COMPLETES

Si nous voulons faire une minimisation plus 'poussée' pour une fonction, nous pouvons générer tous les monômes premiers à partir de la base obtenue en a). En effet un algorithme exact de minimisation d'une fonction consisterait à générer tous les monômes premiers puis à extraire une base irrédondante minimale. Il est à rappeler que le nombre de monôme premiers pour une fonction à n variables peut aller jusqu'à

$3^n/n$. Cette option ne sera concevable que pour des fonctions de taille moyenne ($n \leq 10$). Si l'on considère cette étape au vu d'une minimisation globale, il peut être intéressant de générer tous les monômes premiers locaux, nous en parlerons plus en détail ultérieurement (chapitres 3.4 et 5).

Notre méthode s'appuie sur la méthode de Tison [TIS67] à base de consensus. Elle diffère de celle de Tison par le fait que nous partons d'une base (obtenue en 3.2.1), et que nous nous servons du test d'inclusion.

Résumé de l'algorithme :

On forme le consensus du monôme courant avec les monômes qui le suivent dans la liste courante. Si le consensus est inclus dans un monôme présent dans la liste on le supprime, sinon on l'agrandit au maximum et on le met en bout de liste (ce consensus agrandi est forcément un nouveau monôme premier). Ce processus est répété jusqu'à qu'il n'y ait plus de nouveaux monômes créés par consensus.

Exemple :

si l'on reprend l'exemple précédent après la première étape :

$\neg y.\neg z.\neg t$ (1) + $\neg x.\neg z.\neg t$ (2) + $\neg x.y$ (3) + $y.z$ (4) + $x.z$ (5)

Génération de tous les monômes premiers :

cons(1,3)= $\neg x.\neg z.\neg t$ existe en 2

cons(1,5) = $x.\neg y.\neg t$ pas d'agrandissement : monôme premier (6)

cons(2,4)=$\neg x.y.\neg t$ inclus dans 3

cons(3,5)= $y.z$ existe en 4

cons(2,6)=$\neg y.\neg z.\neg t$ existe en 1

cons(4,6)=$x.z.\neg t$

Dans cette méthode, contrairement à celle de Tison, le nombre de monômes présents au cours de l'algorithme ne dépasse jamais le nombre de monômes premiers de la base complète. De plus après la génération d'un consensus nous n'avons pas besoin de tester l'inclusion d'un monôme présent dans ce nouveau monôme puisque tous les monômes présents sont déjà premiers.

3.3. ETAPE 2 : OBTENTION D'UNE BASE GLOBALE

Nous avons obtenu pour chaque fonction une base (complète ou non). A partir de cet ensemble de bases locales nous voulons obtenir une base globale. Cela peut se faire très simplement en deux étapes.

3.3.1.OBTENTION DE MONOMES GLOBAUX

Nous transformons tout d'abord les monômes des bases locales en monômes globaux en leur ajoutant un vecteur d'inclusion. A chaque monôme m d'une des bases locales est associé un vecteur d'inclusion I pour former le monôme global (m, I) où la $i^{ème}$ composante du vecteur d'inclusion I sera égale à 1 si m apparaît dans la base de la fonction f_i trouvée à l'étape 1.

Exemple :

$f_1 = a.b + b.c$

$f_2 = a + b.c$

E= { (a.b, [1,0]), (b.c, [1,1]), (a, [0,1]) }

b.c apparait dans la base de f_1 mais aussi dans celle de f_2.

Les monômes globaux ainsi obtenus ne sont pas forcément premiers, mais ils sont inférieurs à la fonction générale définie par la séquence des fonctions simples $(f_1,...,f_k)$ par construction.

Dans l'exemple (a.b , [1,0]) n'est pas premier.

3.3.2. MONOMES GLOBAUX PREMIERS

Nous allons agrandir, les monômes globaux obtenus dans la phase précédente, de façon à les rendre premiers grâce à la proposition suivante :

Proposition :

Soit $b_1,\ldots, b_k$ des bases des fonctions $f_1,\ldots, f_k$; soit m un monôme appartenant à au moins une base b_i, soit le monôme global (m, I) où le vecteur d'inclusion I est défini par :

si $m \leq b_j$ alors la composante I_j de I est à 1, sinon elle est à 0.

Le monôme global (m, I) est alors premier dans la fonction générale définie par la séquence de fonctions $(f_1,\ldots, f_k)$.

Démonstration :

(m, I) est inclus dans $(f_1,\ldots, f_k)$ par construction.

Montrons qu'il ne peut exister de monôme (m', I') tel que

$(m', I') > (m, I)$ et $(m', I') \leq (f_1,\ldots, f_k)$.

m est premier pour au moins une fonction f_j.

Supposons qu'il existe $(m', I') > (m, I)$ et $(m', I') \leq (f_1,\ldots, f_k)$

$(m', I') > (m, I) \Leftrightarrow (m \leq m' \text{ et } I < I') \text{ ou } (m < m' \text{ et } I \leq I')$

Si $m \leq m'$ puisque m est premier dans f_j, la composante I'_j de I' est forcément à 0; donc $I < I'$ est impossible.

Si $I \leq I'$, la composante I_j de I est égale à 1 alors I'_j est forcément égale à 1, donc forcément $m' \leq m$ puisque m est premier dans f_j.

Il suffit donc pour tout monôme global obtenu dans la phase précédente d'agrandir au maximum son vecteur d'inclusion par tests d'inclusion.

Dans l'exemple précédent :

a.b est inclus dans f_2 donc (a.b , [1,0]) devient (a.b , [1,1]) qui est premier.

b.c est déjà inclus dans f_1 et f_2 , il est donc déjà premier.

(a, [0,1]) est déjà premier puisque a n'est pas inclus dans f_1.

Nous avons optimisé cette étape grâce à la proposition suivante .

Proposition :

Soit (m, I) un monôme global obtenu dans la première phase (3.2.1), soient $b_{i1},\ldots, b_{ir}$ les bases des fonctions où m n'apparait pas (les fonctions $f_{i1},\ldots, f_{ir}$ n'apparaîssent pas dans I).

Si m n'est pas inclus dans $b_{i1}+\ldots+ b_{ir}$ alors le vecteur d'inclusion I est déjà complet et donc (m, I) est déjà premier.

Démonstration :

Si M n'est pas inférieur à $b_{i1}+\ldots+ b_{ir}$ alors il ne peut être inférieur à chacune de ces bases.

Si un monôme global vérifie cette proposition, nous avons remplacé r tests d'inclusion par un seul. Cette optimisation est donc efficace seulement si les monômes globaux obtenus dans la première phase sont en large proportion déjà premiers par rapport au nombre de fonctions du système. Nous avons pu constater que dans la plupart des exemples, c'est bien le cas surtout pour les exemples possédant un nombre important de variables.

3.4. ETAPE 3 : EXTRACTION D'UNE BASE IRREDONDANTE

Nous voulons à partir d'une base globale, obtenir une base irrédondante tendant vers la minimalité (en nombre de monômes globaux), celle-ci étant un critère relatif à la base de départ. C'est à dire, cette base irrédondante sera extraite sans adjonction de nouveaux monômes, ou transformation des monômes présents. Pour des raisons de complexité, nous voulons éviter de trouver toutes les bases irrédondantes

existantes pour ensuite prendre une des plus petites. La base irrédondante sera donc trouvée grâce à une heuristique. La base de départ peut être issue soit de l'étape 1 suivie de l'étape 2, soit de l'étape 5.

Dans le premier cas, si l'on a choisi de générer tous les monômes premiers locaux dans l'étape 1, nous obtenons à l'entrée de cette étape une base globale qui contient forcément la base globale que nous aurions obtenue dans le cas où l'on aurait limité la génération de monômes premiers dans l'étape 1. Nous sommes donc sûrs d'obtenir alors (moyennant l'efficacité de notre heuristique) à la sortie de cette étape, une base irrédondante plus petite ou égale. Le résultat final après itération sur les étapes 3 et 5 ne pourra qu'être meilleur puisque l'itération aboutit si possible à un minimum local plus bas après chaque boucle. Il est donc préférable quand la complexité de l'exemple nous le permet de choisir dans la première étape l'option suivante : toutes les bases complètes locales. Tous les exemples traités ont confirmé cette proposition.

Notre méthode s'inspire d'un des premiers algorithmes d'extraction de base globale irrédondante minimale, celle de Quine et M^{c}Cluskey [MCC65]. Cette méthode a été reprise dernièrement dans [DAG85] où des optimisations ont été apportées pour pouvoir l'appliquer avec un temps CPU raisonnable sur des ensembles de fonctions de taille moyenne (nombre de var. $\leq$ 10, nombre de fct. $\leq$ 10). Nous avons repris ces améliorations et ajouté des heuristiques pour pouvoir traiter des exemples de complexité plus importante.

3.4.1. L'ALGORITHME DE QUINE ET MCCLUSKEY

Nous pouvons résumer l'algorithme de [MCC65] ainsi :

Il travaille sur trois listes :

IND : liste des monômes globaux dont on ne sait pas encore s'ils seront obligatoires ou supprimés.

OBL : liste des monômes globaux obligatoires.

PHI : liste des monômes globaux de la couverture à phi.

1- Initialisation :

Les monômes de la base de départ sont placés dans IND.
OBL est initialisé à vide.

2- Les monômes globaux M de IND qui sont obligatoires sont détectés, ils vérifient : M n'est pas inférieur à (IND - {M}) + PHI

(IND - {M} dénote l'ensemble IND duquel on a supprimé M)
Ils sont enlevés de IND pour être placés dans OBL.
S'il n'existe pas de monômes obligatoires, il faudra essayer tous les choix possibles à partir de ce point de l'algorithme, en prenant chacun des monômes comme "obligatoire" à tour de rôle. Nous obtenons ainsi un ensemble de bases irrédondantes, la base minimale est forcément dans cet ensemble.

3- Les monômes redondants de IND sont supprimés, ils vérifient :

soit M - (OBL + PHI) = 0 (la partie non à phi du monôme est entièrement couverte par les obligatoires)

soit il existe M' élément de IND différent de M tel que

$M' \geq M - (OBL + PHI)$.

4- Si IND = Ø alors OBL est une base irrédondante sinon on retourne en 2.
Le fait de supprimer des monômes de IND dans 3 fait qu'il peut alors y avoir de nouveaux monômes obligatoires dans 2 (obligatoire étant relatif aux monômes de IND).

3.4.2. UNE OPTIMISATION DE CET ALGORITHME

Cet algorithme tel que Quine et McCluskey l'utilisait, ne pouvait traiter que des fonctions de petites tailles. En effet les monômes étaient décomposés en points élémentaires et ainsi les propriétés 'obligatoire', 'redondant' étaient déterminées grâce à cette décomposition. Par exemple les monômes obligatoires étaient ceux qui étaient les seuls à couvrir au moins un point parmi les monômes présents.

Nous ne pouvons bien sûr nous permettre une telle décomposition, le nombre de points pour une fonction générale de n variables et k composantes étant de $2^n * k$.

Nous possédons dans les techniques de base du chapitre 2 tous les outils nécessaires pour y remédier. Les monômes obligatoires pourront ainsi être déterminés grâce au test d'inclusion (parag. 2.2.1), en testant l'inclusion des monômes dans (IND - {M}) + PHI. Les monômes redondants seront déterminés grâce au calcul de l'incouvert d'un monôme (parag. 2.2.4). La partie incouverte des monômes globaux (M - (OBL + PHI)) que nous avons besoin de calculer, sera évaluée grâce au calcul de l'incouvert PPMC (M - (OBL + PHI)).
Cela simplifie beaucoup les calculs à effectuer sans pour cela modifier la validité de l'algorithme comme nous le verrons plus loin (parag. 3.4.2.1, proposition 1).

De plus nous pouvons éviter beaucoup de travail inutile grâce à un marquage approprié des monômes de IND.

Nous voulons aussi parer au problème de la détection de toutes les bases irrédondantes, en effet une fonction générale (surtout si elle est de taille importante) peut posséder beaucoup de bases irrédondantes. Une heuristique sera donc employée dans l'étape 2 pour choisir le nouveau monôme obligatoire qui semble le plus approprié pour aboutir à la plus petite base irrédondante. Nous éviterons ainsi les différents essais qu'il faudrait effectuer dans le cas où il n'y aie pas de nouveaux obligatoires.

Si les monômes essentiels et stables ont été détectés (voir Etape 4 , (parag. 3.5) détection des monômes essentiels et stables), ils sont placés à l'initialisation dans OBL, évitant ainsi des tests d'inclusion inutiles dans leur cas.

3.4.2.1. UN ALGORITHME EFFICACE

Les monômes de IND seront marqués POSS-OBL ou POSS-SUP. Seuls les monômes marqués POSS-OBL peuvent devenir obligatoires dans l'étape 2. Seul les monômes marqués POSS-SUP peuvent être supprimés dans l'étape 3. NOUV-OBL est l'ensemble des nouveaux monômes obligatoires détectés pendant l'étape 2.

Etape 1 : Initialisation.

OBL est initialisé à Ø, NOUV-OBL est initialisé à l'ensemble des monômes globaux essentiels et stables.

IND est initialisée à l'ensemble des monômes ni essentiels ni stables. Tous les monômes de IND sont marqués POSS-OBL (ils peuvent tous devenir obligatoires).

L'incouvert de chaque monôme M, INC(M) est initialisé à PPMC (M-(NOUV-OBL + PHI)). Si INC(M) est différent de M, le monôme M est marqué POSS-SUP.

Etape 2 : Détection des nouveaux obligatoires.

début

Pour tout monôme M de IND *faire*

si il est marqué POSS-OBL *alors*

supprimer le marquage POSS-OBL de ce monôme;

si M est obligatoire *alors*

{ M n'est pas inférieur à (IND - {M}) + PHI }

M est enlevé de IND pour être placé dans NOUV-OBL

finsi

finsi

finpour

Si NOUV-OBL = Ø *alors*

Forcer un monôme M obligatoire d'àprès une heuristique (voir parag. 3.4.2.2) et le placer dans NOUV-OBL

finsi

Pour tout M élément de IND {restant} *faire*

Si INC(M) * NOUV-OBL ≠ 0 *alors*

INC(M) <- PPMC(M - (OBL + NOUV-OBL + PHI));

marquer M : POSS-SUP

finsi

finpour

OBL <- OBL + NOUV-OBL ;

NOUV-OBL <- Ø

fin

Etape 3 : Suppression des monômes redondants.

début

Pour tout M élément de IND *faire*

si M est marqué POSS-SUP *alors*

détruire le marquage POSS-SUP de M ;

si INC(M) = 0 ou (il existe M' élement de IND tel que M' ≥ INC(M)) *alors* supprimer M. {IND - M -> IND}

finsi

finsi

finpour

Pour tout M élément de IND {restant} *faire*

si INC(M) * SUPPRIME ≠ 0 *alors*

marquer M : POSS-OBL

finsi

finpour

fin

{SUPPRIME est l'ensemble des monômes supprimés pendant cette étape}

Etape 4 : test d'arrêt.

début

si IND est vide *alors*

OBL est une base irrédondante minimale "approchée"

sinon retour à l'étape 2

finsi

fin

On peut justifier cet algorithme par rapport à celui de Quine et M^{c}Cluskey par les propositions suivantes.

Proposition 1 :

Le fait de remplacer la partie incouverte de M (PI(M) = M-(OBL + PHI)) par l'incouvert de M (INC(M) = PPMC (M- (OBL + PHI))) ne change pas le résultat de l'algorithme.

Démonstration :

Un monôme M est supprimé si :

* PI(M) = 0 dans ce cas INC(M)= PPMC(0)= 0

* Il existe M' élément de IND tel que M' ≥ PI(M) , nous avons alors aussi M' ≥ PPMC (PI(M))

Proposition 2 :

Dans l'étape 2, seuls les monômes marqués POSS-OBL peuvent devenir obligatoires.

Démonstration :

Au départ ils sont tous marqués donc il n'y a pas de problème pour la première passe de l'étape 2.

Par la suite un monôme est marqué dans l'étape 3 seulement s'il existe un produit non nul entre son incouvert et au moins un des monômes supprimés pendant cette étape. A la passe précédente de l'étape 2, M n'était pas obligatoire. Il nous faut donc montrer:

((IND - {M}) + SUPPRIME + OBL + PHI) ≥ M

et INC(M) * SUPPRIME = 0

=> ((IND - {M}) + OBL + PHI) ≥ M

Nous avons

INC(M) * SUPPRIME = 0 => PI(M / OBL + PHI) * SUPPRIME = 0

puisque INC(M) ≥ PI(M / OBL + PHI) (PI : partie incouverte)

((IND - {M}) + SUPPRIME + OBL + PHI/) ≥ M

⇔ M - ((IND - {M}) + SUPPRIME + OBL + PHI) = 0

⇔ PI(M / OBL + PHI) - ((IND - {M}) + SUPPRIME) = 0

puisque PI(M / OBL + PHI) = M - (OBL + PHI)

Or PI(M / OBL + PHI) * SUPPRIME = 0 donc :

=> PI(M / OBL + PHI) - (IND - {M}) = 0

=> ((IND - {M}) + OBL + PHI) ≥ M

Proposition 3 :

Dans l'étape 3, seuls les monômes marqués POSS-SUP peuvent être supprimés.

Démonstration :

-A la première passe de l'étape 3 :

un monôme M a été marqué POSS-SUP que si INC(M) est différent de M.

si INC(M) = M, nous avons forcément INC(M) différent de 0 et il n'existe pas M' tel que M' ≥ INC(M) puisque M est premier. Donc M ne pourra être supprimé.

-Au cours des passes suivantes de l'étape 3 :

Si à la passe précédente un monôme M n'a pas été supprimé nous avons :

INC(M) ≠ 0 et il n'existe pas de M' élément de IND tel que M' ≥ INC(M)

Pour qu'au prochain passage dans l'étape 3, M puisse être supprimé il faut que son incouvert soit modifié. Il ne peut être modifié que s'il existe un produit non nul entre l'incouvert courant et les nouveaux obligatoires détectés à l'étape 2 précédente :

PPMC(M- (OBL + PHI)) * NOUV-OBL = 0

=> M- (OBL + PHI) * NOUV-OBL = 0

=> (M- (OBL + PHI)) * ¬ NOUV-OBL = M- (OBL + PHI)

=> (M- (OBL + PHI)) - NOUV-OBL = M- (OBL + PHI + NOUV-OBL) = M- (OBL + PHI)

=> PPMC(M- (OBL + PHI))= PPMC(M- (OBL + PHI+ NOUV-OBL))

Remarques :

Il est à souligner que le marquage POSS-SUP (resp. POSS-OBL) des monômes indécidés est supprimé à la sortie de l'étape 3 (resp.2). En effet les monômes ainsi marqués peuvent devenir à nouveau suppressibles (resp. obligatoires) seulement si de nouveaux monômes deviennent obligatoires (resp. sont supprimés) en respectant les conditions de marquage.

Dans l'étape 2, il pourrait sembler suffisant de mettre à jour l'incouvert des monômes en effectuant :

INC(M) <- PPMC(INC(M) - NOUV-OBL)

puisque

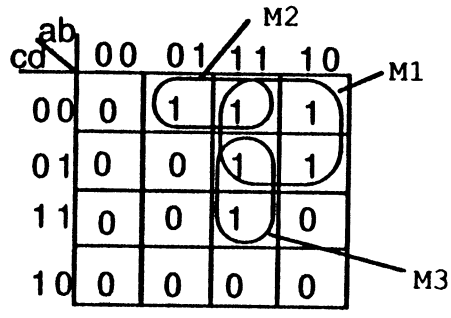
PI(M/(OBL+PHI+NOUV-OBL)) = PI (M/(OBL+PHI)) - NOUV-OBL

Mais :

PPMC(M-(OBL+PHI+NOUV-OBL)) est différent de PPMC(PPMC(M - (OBL + PHI)) - NOUV-OBL)

Exemple :

M1= a.¬c , M2= b.¬c.¬d , M3= a.b.d



INC(M1/ M2) = M1 , INC(INC(M1/ M2) / M3) = M1

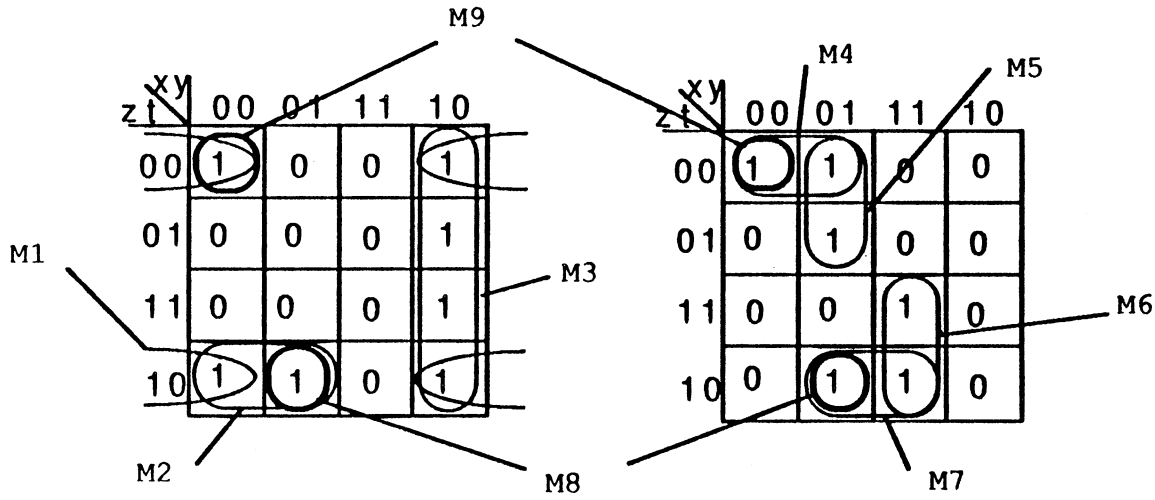
mais INC (M1/ (M2+M3))= a.¬b.¬c ≠ M1

Nous sommes donc obligés de refaire à chaque fois le calcul par rapport à tous les obligatoires (anciens et nouveaux) : INC(M) <- M - (OBL+ NOUV-OBL + PHI)

Au moment de la suppression des monômes redondants dans l'étape 3, l'ordre dans lequel on applique les essais de suppression sur les monômes indécidés, peut influer sur le résultat. En effet deux (ou plus) monômes peuvent avoir le même incouvert et donc, on va pouvoir supprimer soit l'un soit l'autre suivant l'ordre de suppression.

Il est à souligner que de toute façon, le nombre de monômes final ne changera pas suivant cet ordre puisque ces monômes "équivalents" couvrent les mêmes points encore incouverts. Par contre ce choix peut influer sur le deuxième critère de minimisation : le nombre de transistors du PLA final.

Exemple :



Au départ M3, M5, M6 sont obligatoires.

M4 et M7 sont alors supprimés puisque leurs incouverts sont respectivement inclus dans M9 et M8.

Au deuxième coup M8 et M9 deviennent ainsi obligatoires.

M1 et M2 ont alors le même incouvert ($\neg x.\neg y.z.\neg t$, [1,0]), on peut supprimer soit l'un , soit l'autre. Il vaut mieux supprimer M2 puisqu'il est le plus petit (gain en transistors).

Soit M= (m, I) un monôme global.

Nous définissons NT(M)= nombre de 1 dans I + nombre de variable de m.

NT(M) représente le nombre de transistors nécessaires pour définir M dans un PLA.

Nous classerons les monômes par ordre de NT décroissant avant l'étape de suppression afin de supprimer en premier lieu les monômes dont le nombre de transistors équivalents est maximal.

3.4.2.2. CHOIX D'UNE HEURISTIQUE

Dans l'étape 2 (détection des obligatoires), si aucun monôme n'est obligatoire, il faudrait forcer un par un chaque monôme indécidé à être obligatoire pour être sûr d'obtenir toutes les bases irrédondantes et ainsi choisir une des plus petites. Pour éviter cette explosion combinatoire, nous faisons un choix unique par heuristique. Cette heuristique va consister à considérer non pas le résultat final, mais seulement le résultat obtenu aprés la prochaine étape de suppression. Nous allons ainsi voir comment il est possible d'estimer pour chaque monôme, le nombre de monômes qui pourront être supprimés dans le cas où ce monôme est forcé "obligatoire". Il sera alors facile de choisir le monôme obligatoire de telle manière à minimiser le nombre de monômes obtenus aprés la prochaine étape de suppression.

Il est à remarquer que dans le cas présent, chaque monôme indécidé possède un incouvert qui n'est pas inférieur à un seul autre monôme indécidé, sinon il aurait été supprimé à l'étape précédente de suppression. D'autre part, aucun monôme indécidé n'étant obligatoire, leur incouvert est forcément couvert par d'autres monômes indécidés. Donc chaque monôme possède un incouvert couvert par au moins deux autres monômes indécidés.

Notre heuristique s'appuit sur la proposition suivante.

Proposition :

Supposons que nous sommes dans le cas où aucun monôme indécidé n'est obligatoire.

Si un monôme M indécidé possède un incouvert tel qu'il suffit de deux autres monômes indécidés M' et M" pour le couvrir alors, si M' ou M" est rendu obligatoire M sera forcément supprimé à la prochaine étape de suppression.

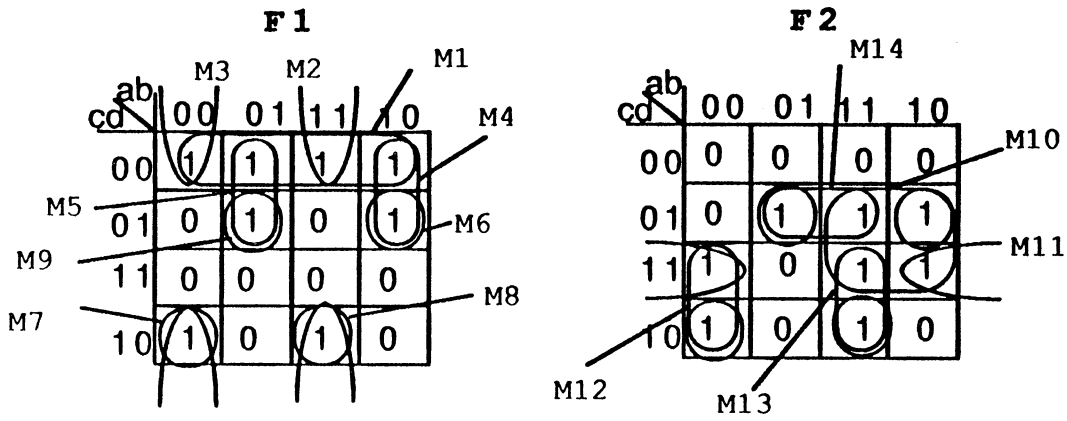
Démonstration :

Si M' (resp. M") est obligatoire l'incouvert de M devient inférieur à M" (resp. M'), il sera donc supprimé.

Une heuristique pourrait donc consister à choisir comme nouvel obligatoire le monôme qui possède une intersection non nulle avec le maximum de monômes indécidés vérifiant la proposition précédente. Nous serions ainsi sûr d'obtenir le minimum de monômes aprés l'étape de suppression suivante (parmi tous les choix possibles de nouvel obligatoire).

Exemple :

Il n'existe pas de monômes obligatoires parmi les 14 monômes M1, ..., M14 de la base représentée sur les deux tableaux suivants :



Si nous calculons pour chaque monôme M_i, le nombre de monômes qui respectent les conditions de la proposition et dont l'incouvert intersectent avec M_i (l'incouvert étant ici égal au monôme entier), nous trouvons :

M1 : 4 (M2, M3, M4, M5) ; M2 : 1 (M8) ; M3 : 1 (M7) ; M4 : 1 (M6)

M5 : 1 (M9) ; M6 : 2 (M4, M10) ; M7: 2 (M3, M12) ;

M8 : (2 (M2, M13) ; M9 : 2 (M5, M14) ; M10 : 4 (M6, M11, M13, M14)

M11: 1 (M12) ; M12 : 2 (M7 , M11) ; M13 : 1 (M8) ; M14 : 1 (M9)

Si l'on choisit M1 comme obligatoire alors M2, M3, M4 et M5 seront supprimés à la prochaine étape de suppression, par contre si l'on choisit M2 seulement M8 sera supprimé.
Si l'on choisit M1 ou M10 (qui vérifie tous les deux notre heuristique) comme nouvel obligatoire nous arrivons dans les deux cas directement (sans nouveau choix de nouvel obligatoire) à une base irrédondante de 7 monômes {M1, M6, M7, M8, M9, M10, M11} dans le premier cas , {M1, M3, M4, M8, M9, M10, M12} dans le second. Ces deux bases sont minimales.
En ne suivant pas l'heuristique proposée, il est possible d'obtenir dans le pire cas une base irrédondante de 8 monômes.

Par exemple,
on choisit : M5, M9 est supprimé,
M14 devient obligatoire, pas de nouveau supprimé.
Un nouveau choix s'impose : M2, M8 est supprimé,
M13 devient obligatoire, pas de nouveau supprimé.
On choisit M3, M7 est supprimé,
M12 devient obligatoire, M11 est supprimé
M10 devient obligatoire, M6 est supprimé, M4 devient obligatoire.
On obtient ainsi la base {M2, M3, M4, M5, M10, M12, M13, M14}.

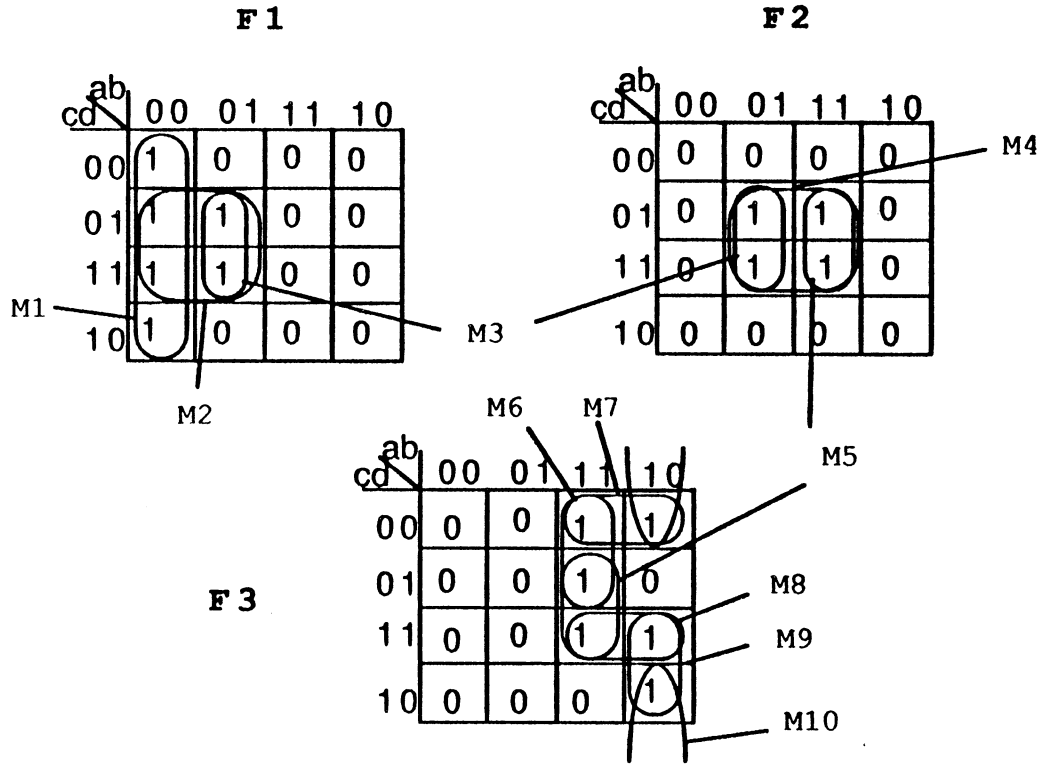
Malheureusement, une telle heuristique engendrerait des calculs trop importants pour les applications de grande complexité que nous voulons traitées. Nous avons opté pour une heuristique plus simple, mais tendant toujours à minimiser le nombre de monômes restant aprés la prochaine étape de suppression. Cette heuristique va consister à maximiser le nombre de monômes potentiellement suppressibles.
Ceci peut se faire trés simplement en calculant pour chaque monôme M indécidé, le nombre de monômes dont l'incouvert possède une intersection non nulle avec l'incouvert du monôme M. Le monôme choisi sera alors celui dont l'incouvert possède une intersection non nulle avec le plus grand nombre d'incouverts d'autres monômes

indécidés. Le nombre de monômes potentiellement suppressibles deviendra alors maximal. Le nombre de monômes supprimés tendra alors à être maximal.

Cette hypothèse s'est avérée dans la plupart des cas justes et vu les résultats finaux de notre minimiseur (voir chapitre 5), nous avons conclu que cette heuristique, tout en limitant énormément les calculs, est suffisante pour s'approcher au plus prés d'un minimum exact dans la plupart des cas.

3.4.2.3. APPLICATION SUR UN EXEMPLE

Nous allons dérouler l'algorithme sur un exemple simple. La base de départ de la fonction générale (F1, F2, F3) est composé des 10 monômes globaux (M1,...,M10). Elle est donnée sur les trois tableaux de Karnaugh suivants :



Passe1 :

Etape2 :

M1 est le seul obligatoire. OBL= {M1}.

M2 est le seul qui a une intersection non nulle avec M1, il est

marqué POSS-SUP.

Etape3 :

M2 est supprimé puisque son incouvert (¬a.b.d, [1,0,0]) est inclus dans M3= (¬a.b.d, [1,1,0]).

IND= {M3, M4, M5, M6, M7, M8, M9, M10}

M3 est le seul dont l'incouvert (= M3) possède une intersection non nulle avec M2, il est marqué POSS-OBL.

Passe2 :

Etape2 :

M3 est maintenant obligatoire (il est le seul à couvrir les points (¬a.b.c.d, [1,0,0]) et (¬a.b.c.d, [1,0,0]) puisque M2 a été supprimé).

OBL= {M1, M3}

M4 est le seul tel que INC(M4) * M3 ≠ 0, il est marqué POSS-SUP.

Etape3 :

M4 est supprimé, son incouvert (a.b.d, [0,1,0]) est inclus dans M5= (a.b.d, [0,1,1]).

IND= {M5, M6, M7, M8, M9, M10}.

M5 est marqué POSS-OBL.

Passe3 :

Etape 2 :

M5 est obligatoire. OBL= {M1, M3, M5}

M6 et M8 sont marqués POSS-SUP.

Etape 3 :

M6 et M8 sont supprimés, leurs incouverts sont inclus respectivement dans M9 et M7.

IND= {M9, M7, M10}

M9 et M7 sont marqués POSS-OBL.

Passe 4 :

Etape 2 :

M9 et M7 sont obligatoires. OBL= {M1, M3, M5, M7, M9}

M10 est marqué POSS-SUP.

Etape 3 :

M10 est supprimé puisque son incouvert est vide.

Etape 4 :

IND est vide,

OBL= {M1, M3, M5, M7, M9} est la base irrédondante minimale.

3.5. ETAPE 4 : DETECTION DES MONOMES ESSENTIELS ET STABLES

3.5.1. DETECTION DES MONOMES ESSENTIELS

Nous employons l'algorithme donné au chapitre 2.3. La détection des monômes essentiels apporte, dans la plupart des cas, un gain de temps. En effet dans la boucle de l'algorithme de minimisation proposé (étapes 3, 4 et 5), à chaque passage de l'étape 3, il ne sera plus la peine de tester si les monômes essentiels sont obligatoires, puisque, par définition, un monôme essentiel est obligatoire dans toutes les bases.

D'autre part cette étape est primordiale pour la limitation du nombre de nouveaux monômes générés dans l'étape 5. De plus, comme nous le verrons au paragraphe 3.6, nous effectuons dans cette phase une grande partie du travail de l'étape de génération de nouveaux monômes.

Cette détection est faite après la première passe de l'étape 3 (extraction d'une base irrédondante). Nous pouvons remarquer que les monômes essentiels font forcément partie de l'ensemble des monômes obligatoires détectés au départ de l'extraction de base irrédondante, puisque les monômes essentiels sont obligatoires dans toutes bases.

Nous n'aurons donc pas besoin de tester "l'essentialité" des autres.

3.5.2. LES MONOMES STABLES

Nous allons définir une nouvelle propriété sur les monômes qui est trés utile pour l'étape suivante de notre algorithme (génération de nouveaux monômes).

Cette propriété peut s'assimiler à une 'pseudo-essentialité' dans la base enrichie des nouveaux monômes générés à l'étape suivante. C'est à dire qu'ils seront forcément dans la base irrédondante obtenue après adjonction de nouveaux monômes et extraction d'une base irrédondante.

De plus, nous verrons dans le chapitre suivant (3.6) comment cette propriété permet de limiter au même titre que les essentiels, le nombre de nouveaux monômes générés.

Définition :

Un monôme premier est stable pour une base B donnée si et seulement si :

M n'est pas inférieur à COUV-STAB (M)

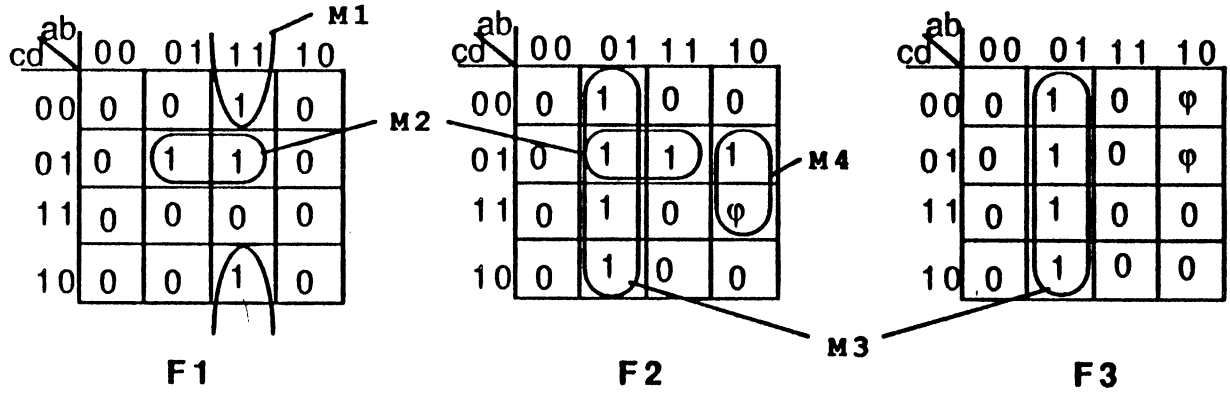
avec COUV-STAB(M)= { produit-simple(M, M'), produit-complexe(M, M'), M. M" tel que M' élément de B-{M} et M" élément de PHI }

PHI étant l'ensemble des monômes de la couverture à ϕ.

On peut d'ores et déjà observer que les monômes essentiels sont stables. Nous déterminerons donc à la première passe de cette étape 3, tout d'abord les monômes essentiels, puis, parmi ceux qui ne le sont pas les monômes stables.
Pour les passes suivantes (boucle étape 3, étape 4 , étape 5), nous déterminerons seulement les monômes stables; les essentiels étant fixés une fois pour toute contrairement aux stables qui dépendent de la base courante.

3.5.3. EXEMPLE DE DETECTION DES MONOMES STABLES ET ESSENTIELS.

Soient les trois fonctions phi-booléennes F1, F2, F3 définies sur les tableaux suivants :



Les monômes globaux M1, M2, M3 et M4 forment une base de ces fonctions.

-COUV-ESS(M1)= { consensus-global(M1, M2) = (a.b.¬c, [1,0,0]) }

M1 n'est pas inclus dans COUV-ESS(M1) donc M1 est essentiel.

-COUV-ESS(M2)= { consensus-global(M1, M2) = (a.b.¬c, [1,0,0]);
produit-complexe(M2, M3)= (¬a.b.¬c.d, [1,1,1]);
consensus-global(M4, M2) = (a.¬c.d, [0,1,0]) }

M2 est inclus dans COUV-ESS(M2), il n'est donc pas essentiel.

-COUV-STAB (M2)= { produit-complexe(M2, M3)= (¬a.b.¬c.d, [1,1,1])}

M2 n'est pas inclus dans COUV-STAB(M2), il est donc stable.

-COUV-ESS(M3)= { produit-complexe(M1, M3)= (¬a.b.¬c.d, [1,1,1]) }

M3 est donc essentiel.

-COUV-ESS(M4)= { consensus-global(M4, M2) = (a.¬c.d, [0,1,0]);
produit-simple (M4, Mj1)= (a.¬b.c.d, [0,1,0]);
produit-complexe (M4, Mj2)= (a.¬b.¬c.d, [0,1,1]) }

Les monômes Mj1= (a.¬b.c.d, [0,1,0]) et Mj2= (a.¬b.¬c, [0,0,1]) étant les monômes de PHI.

M4 est inclus dans COUV-ESS(M4), il n'est donc pas essentiel.

-COUV-STAB(M4)= { M4.Mj1 = Mj1 }

donc M4 est stable.

3.6. GENERATION DE NOUVEAUX MONOMES

Cette étape est la phase clé de notre algorithme. Quand nous arrivons à ce point du processus, nous avons une base irrédondante pseudo-minimale, nous avons donc atteint un minimum local. Cette étape va consister à ajouter de nouveaux monômes à la base obtenue afin de trouver une base plus petite après une nouvelle extraction de base irrédondante. On va donc augmenter le coût de la solution courante (minimum local), afin d'essayer de retomber dans un minimum plus bas.
Ces nouveaux monômes devront donc être déterminés et choisis avec soin, leur nombre devant être limité au maximum pour ne pas retomber dans un problème d'explosion combinatoire.

3.6.1. IDEE DE BASE

Nous voulons générer des monômes plus 'intéressants' que ceux obtenus jusque là. Ces nouveaux monômes devront rassembler des points appartenant à différents monômes existant, permettant éventuellement de les remplacer (en obtenant un gain).
Pour cela, nous allons appliquer à chaque couple de monômes présents dans la base irrédondante, une opération élémentaire : le produit généralisé.

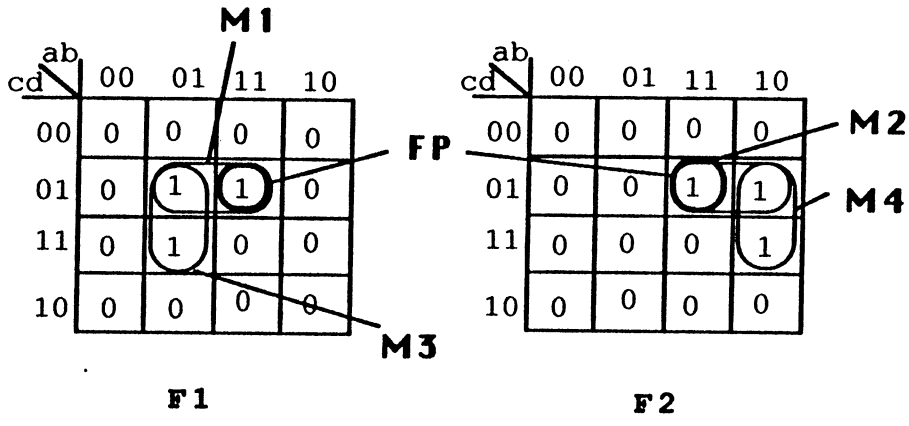
Définition 1 :

Le produit généralisé de deux monômes globaux $M= (m, l)$ et $M'= (m', l')$ existe si $m.\ m' \neq 0$, il est alors défini par :

$M \S M' = (m.\ m' ,\ l + l')$

Exemple :

Soit la base irrédondante {M1, M2, M3, M4} de (F1, F2) représentée sur les tableaux suivants.



M1 § M2 = (a.b.¬c.d, [1,1])

Il sera appelé fils produit (noté FP), M1 et M2 sont ses 'pères'.

On peut remarquer que {FP, M3, M4} forme encore une base de (F1,F2), FP a remplacé ses deux pères M1 et M2.

On regroupe ainsi des points n'appartenant pas aux deux pères à la fois, en un seul monôme. Au départ de notre algorithme de minimisation, nous avons fait une minimisation locale de chaque fonction. Nous avons ainsi obtenu par la suite des monômes globaux (m, I) tels que m est 'favorisé' par rapport au vecteur d'inclusion I. C'est à dire que les monômes globaux ainsi générés, sont tels que m possède un minimum de variables(grande taille) au détriment de I qui est calculé en fonction m.

Cette étape va consister à ajouter de nouveaux monômes globaux (m, I) tel que m sera plus petit ($m.m' \leq m$ et m') au profit du vecteur d'inclusion I ($I + I' \geq I$ et I').

Il est tout d'abord primordial que le fils produit de deux monômes d'une fonction soit inférieur à celle-ci.

Théorème :

Le fils produit de deux monômes inférieurs à une fonction est inférieur à cette fonction.

Démonstration :

Soit P=(p, IP) un point du fils-produit de (m,I) et (m',I').

Nous avons par définition du fils-produit :

$p \leq m.m'$ et $IP \leq l + l'$

donc $p \leq m$ et $p \leq m'$ et ($IP \leq l$ ou $IP \leq l'$)

donc $P \leq (m, l)$ ou $P \leq (m', l')$

Ceci étant valable pour tout point du fils-produit, celui-ci est inférieur à la fonction contenant (m, l) et (m', l').

Nous pouvons ajouter à la définition 1 des restrictions qui nous permettent d'éviter de calculer des fils-produits dont nous connaissons d'ores et déjà l'inutilité.

Proposition :

Soit FP le fils-produit de deux monômes premiers (m, l) et (m', l'), si $m < m'$ (resp. $m' < m$) alors FP= (m, l) (resp. = (m', l')).

Démonstration :

$m < m'$ => $l > l'$ car (m,l) est premier

=> $l + l' = l$

$m < m'$ => $m.\ m' = m$ donc $FP = (m.\ m', l + l') = (m, l)$

Donc dans ce cas le fils-produit est égal à l'un de ses pères, ce n'est donc pas la peine de le générer.

Proposition :

Soit FP le fils-produit de deux monômes premiers (m, l) et (m', l'),
si $l < l'$ (resp. $l' < l$) alors $FP \leq (m,l)$ (resp. $FP \leq (m',l')$).

Démonstration :

$l < l' \Rightarrow l + l' = l'$

Nous avons $m.m' \leq m'$

donc $FP = (m.m', l + l') \leq (m', l')$

Dans ce cas le fils-produit est inférieur a l'un de ses pères ce n'est donc pas la peine de le générer.

Proposition :

Soient deux monômes premiers distincts (m, l) et (m', l'), si l n'est pas inférieur à l' et inversement alors m n'est pas inférieur à m' et inversement.

Démonstration :

Supposons m < m' , nous avons forcément l > l' sinon m n'est pas premier (idem avec m' < m).

Il sera donc seulement nécessaire d'effectuer le test sur les vecteurs d'inclusions des pères pour savoir si le fils-produit sera utile.
Nous pouvons donc apporter une première restriction au produit généralisé:
Le fils-produit de deux monômes (m,l) et (m', l') existe si

l n'est pas inférieur à l' et inversement (ou l + l' > l et l + l' > l')

Malgré ces restrictions le nombre de monômes crées par produit généralisé peut devenir très grand. Sur la totalité des fils-produits ainsi générés, nous nous sommes rendu compte que peu d'entre eux étaient utiles (la plupart étant éliminés dans l'extraction de base irrédondante suivante). Nous avons donc essayé de trouver des critères de choix des fils-produits "utiles".

Définition :

Un fils-produit est inutile si l'on est certain qu'il sera éliminé lors de la prochaine extraction de base irrédondante.

3.6.2. PERES ESSENTIELS ET STABLES.

Nous allons montrer que si l'un des pères d'un fils-produit est essentiel ou stable alors ce fils-produit est inutile.

Théorème :

Un monôme stable dans une base B ne peut être supprimé par adjonction de fils-produits à B.

Démonstration :

Rappel : Un monôme M est stable s'il n'est pas inférieur à COUV-STAB(M) où COUV-STAB(M)= {produit-simple (M,M'), produit-complexe(M,M') ou M.M" avec M' élément de B-{M} et M" élément de PHI}.

D'après la définition du produit-complexe, l'ensemble des fils-produits de M est inclus dans l'ensemble des produit-complexe de M avec les autres monômes de B. L'ensemble de tous les fils-produits de M est donc inclus dans COUV-STAB(M).
Donc l'ensemble des produit-complexes et des produit-simples de M couvre les points de M qui sont couverts, par des monômes présents dans B et par tous les fils-produits de M.
L'ensemble des produits de M avec les monômes de PHI couvre tous les points de M qui n'appartiennent pas à la borne inférieure de la fonction.

Donc si M n'est pas couvert par COUV-STAB(M), il existe un point de la borne inférieure qui n'est couvert ni par un monôme de B ni par un fils-produit, donc M sera toujours obligatoire dans B $\cup$ { fils-produit (M, M')}, pour tout M'.

Un monôme stable sera donc au même titre que les monômes essentiels rangé d'office dans la liste des obligatoires au moment de la prochaine extraction de base irrédondante.

Nous allons montrer qu'il est inutile de générer des fils-produits de monômes essentiels ou stables.

Lemme 1 :

Le fils-produit FP de (m,I) et (m',I') est inférieur à la somme de ses pères (m,I) + (m',I') .

Démonstration :

Soit P= (p,IP) un point de FP.

$p \leq m.m'$ donc $p \leq m$ et $p \leq m'$

$IP \leq I + I'$ donc $IP \leq I$ ou $IP \leq I'$

donc $P \leq (m,I)$ ou $P \leq (m',I')$ donc $P \leq (m,I) + (m',I')$.

Lemme2 :

Si FP est le fils-produit de M1 et M2 et E une somme de monômes globaux alors l'incouvert de FP par rapport à E+M1 est inférieur à l'incouvert de M2 par rapport à E et réciproquement (l'incouvert de FP par rapport à E+M2 est inférieur à l'incouvert de M1 par rapport à E).

Démonstration :

INC (FP/E+M1)= PPMC (FP. ¬M1. ¬E) par définition de l'incouvert

FP ≤ M1 + M2 d'après lemme1

donc PPMC (FP. ¬M1. ¬E) ≤ PPMC((M1+M2). ¬M1. ¬E)

(M1+M2). ¬M1. ¬E = M2. ¬M1.¬E ≤ M2. ¬E

donc PPMC (FP. ¬M1. ¬E) ≤ PPMC(M2. ¬E) = INC(M2 / E)

Théorème :

Soit FP le fils-produit de M1 et M2, si M1 ou M2 est essentiel ou stable alors FP sera inutile.

Démonstration :

Supposons M1 essentiel ou stable.

a) Si M2 est essentiel ou stable :

FP sera alors supprimé dans l'étape 3 puisque M1 et M2 seront obligatoires et FP est inférieur à M1 + M2 (lemme1).

b) Sinon :

INC(FP/{essentiels} ∪ {stables}) ≤ INC(M2/{essentiels} ∪ {stables}) (lemme 2).

* si INC(FP/{essentiels} ∪ {stables}) < INC(M2/{essentiels} ∪ {stables}) alors FP sera supprimé d'après l'algorithme d'extraction de base irrédondante (voir paragraphe 3.4).

* si INC(FP/{essentiels} ∪ {stables}) = INC(M2/{essentiels} ∪ {stables}) alors soit FP soit M2 sera éliminé. Mais puisque FP possède plus de variables et de 1 dans son vecteur d'inclusion, c'est lui qui sera supprimé (voir ordre de suppression dans étape 3.4.2.1).

D'après ce théorème si l'un des pères d'un fils-produit est essentiel ou stable,

nous sommes sûrs qu'il sera éliminé. Nous limiterons donc la génération des nouveaux monômes par ce critère.

3.6.3. UNE HEURISTIQUE DE RESTRICTION

Le nombre de fils-produits étant encore trop élevé malgré ces restrictions, nous avons été obligés d'élaborer une heuristique pour limiter encore ce nombre. Cette heuristique est basée sur la notion d'incouvert (paragraphe 2.2.4).

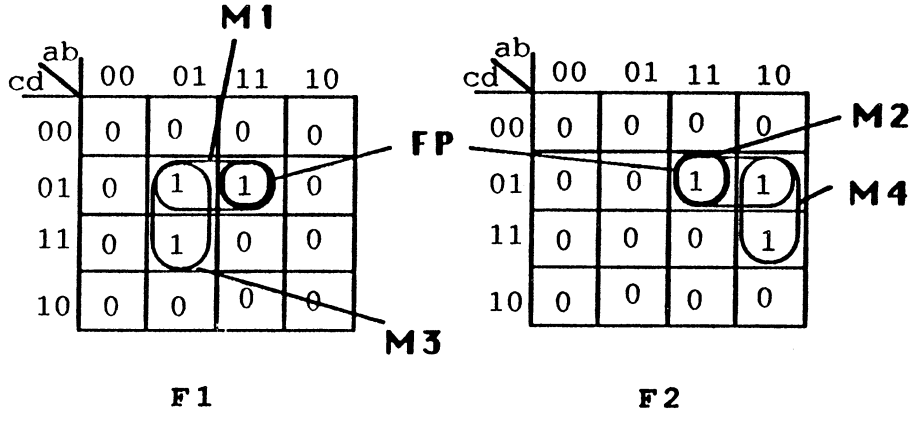
Pour qu'un fils produit puisse remplacer ces pères, il faut qu'il les rende non-obligatoires. En effet les fils-produits sont ajoutés à une base irrédondante, donc par définition tous les monômes de cette base sont obligatoires. On ne pourra supprimer des monômes de cette base que si les monômes que l'on ajoute rendent ceux-ci non-obligatoires. Pour qu'un père ne soit plus obligatoire aprés adjonction des fils-produits, il faut que son incouvert (par rapport au autres monômes de la base courante) soit entièrement couvert par l'ensemble des fils-produits.

3.6.3.1. UNE PREMIERE APPROCHE

Une première approche nous a conduit à regarder chaque fils-produit isolément par rapport à ses pères. Un fils-produit n'aura une chance de remplacer ses pères que s'il couvre l'incouvert de ceux-ci par rapport à la base de départ. Nous sommes alors sûr que les deux pères ne seront plus obligatoires puisque les points qu'ils étaient les seuls à couvrir (incouvert) seront alors aussi couverts par leur fils-produit.

Exemple :

Si l'on reprend l'exemple précédent :

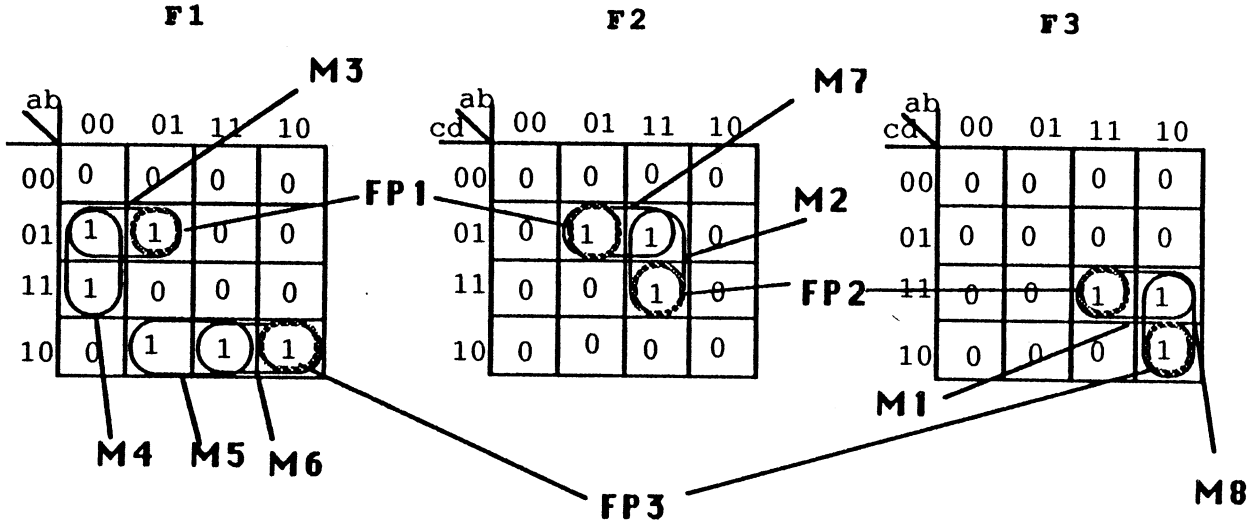


Nous avons :

INC (M1/ {M2,M3, M4})= (a.b.¬c.d, [1,0]) ≤ (a.b.¬c.d, [1,1])
INC (M2/ {M1,M3, M4})= (a.b.¬c.d, [0,1]) ≤ (a.b.¬c.d, [1,1])

Un fils-produit qui vérifie cette condition ne pourra pas forcément remplacer ses deux pères. En effet par suppression d'autres monômes par adjonction d'autres fils-produits, l'incouvert des pères d'un fils-produit peut être changé au cours de l'algorithme d'extraction de base irrédondante par suppression de monômes.

Exemple :



FP1 est le fils-produit de M3 et M7,
FP2 est le fils-produit de M1 et M2,
FP3 est le fils-produit de M6 et M8.
Ils vérifient tous les trois les conditions de notre heuristique.

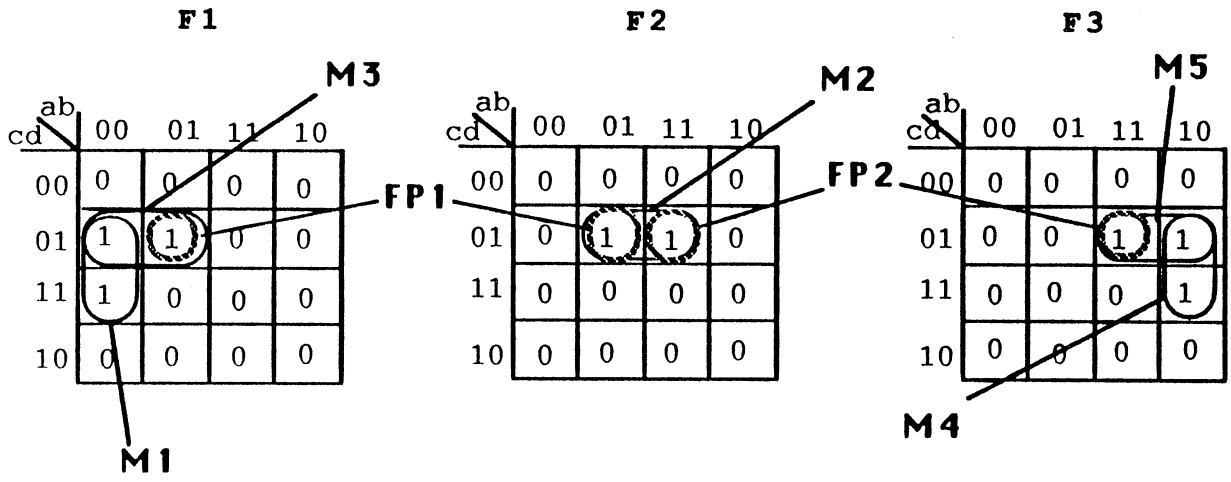
Pourtant si nous faisons tourner l'algorithme d'extraction de base irrédondante à partir de {Mi, FP1, FP2, FP3}, nous aboutissons à la base irrédondante {M4, M5, FP1, FP3, M1 , M2}. FP2 n'a pu remplacer M1 et M2, car M8 et M7 ont été suprimés.

Nous ne pourrons donc pas remplacer d'office les pères dont les fils-produits vérifient cette condition.

De plus il peut arriver que l'incouvert d'un père soit couvert par l'union de plusieurs de ses fils-produit. Dans l'exemple suivant aucun gain ne pourra être obtenu si l'on applique cette heuristique, pourtant il y a possibilité de gagner un monôme.

Exemple :

D'après cette première heuristique aucun fils-produit ne sera généré.



INC (M2/ {M1, M3, M4, M5} n'est pas inférieur à FP1 et INC (M2/ {M1, M3, M4, M5} n'est pas inférieur à FP2

Pourtant si FP1 et FP2 sont ajoutés à la base {M1, M2, M3, M4, M5}, la base irrédondante obtenue sera {M4, M1, FP1, FP2} (gain d'un monôme). FP1 et FP2 ont pu remplacer M2, M3, M5.

3.6.3.2. UNE HEURISTIQUE AFFINEE

Nous n'ajouterons un fils-produit que s'il couvre l'incouvert (par rapport à la base de départ) d'au moins un de ses deux pères.

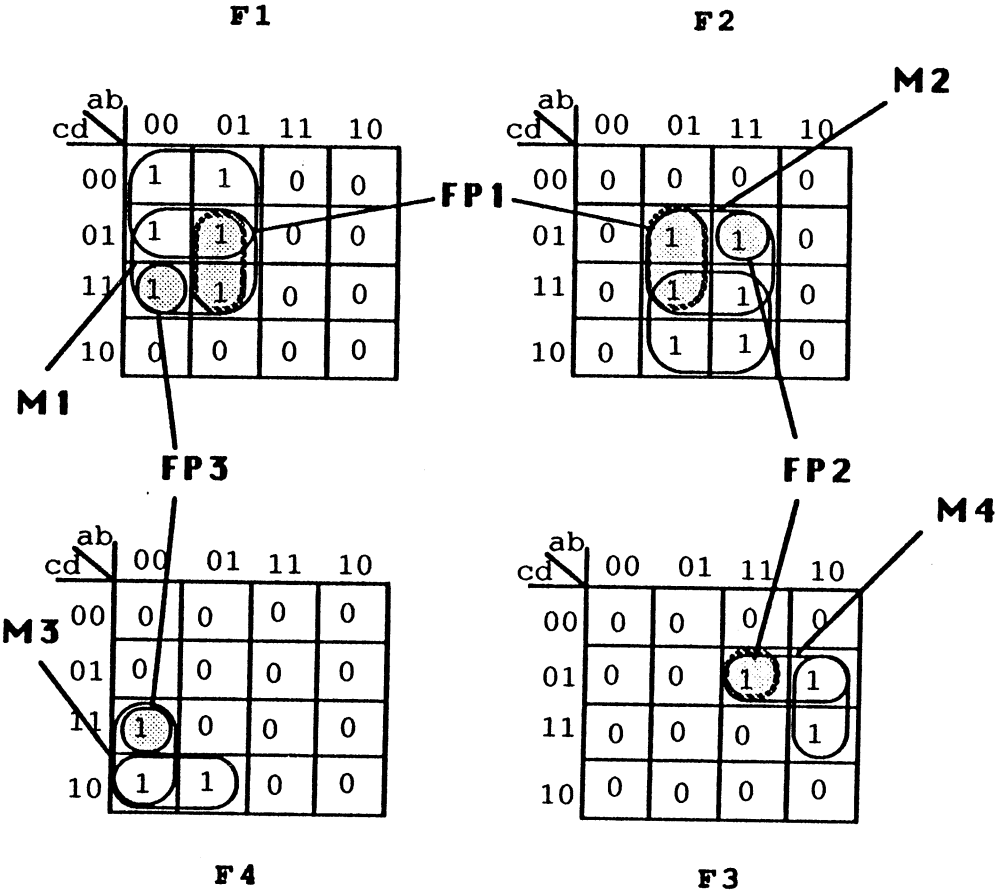
Dans l'exemple précédent les deux fils-produits FP1 et FP2 seront générés si l'on applique cette heuristique.

Cette heuristique permet ainsi de remplacer des pères dont l'incouvert est couvert par l'union de plusieurs de ses fils-produits, tout en garantissant que pour chaque fils-produit généré, au moins un monôme ne sera plus obligatoire (celui dont l'incouvert était entièrement couvert par ce fils). De plus il est à souligner que si le nombre de fils-produits nécessaires pour couvrir l'incouvert des pères est supérieur à leur nombre, on ne pourra pas obtenir de gain. Et c'est ce qui a de fortes chances d'arriver si il faut plus d'un fils-produit pour couvrir l'incouvert de chaque père.

On limite ainsi beaucoup le nombre de monôme ajoutés en estimant leur utilité potentielle pour un gain futur.

Il se peut pourtant, que cette heuristique ne permette pas de résoudre certains cas plus complexes, où un ensemble de pères ne vérifiant pas tous l'heuristique, puisse être remplacé par un ensemble de fils (de cardinalité inférieur).

Exemple :



D'après notre nouvelle heuristique seuls FP2 et FP3 seront générés. En effet FP1 ne couvre ni l'incouvert de M1, ni celui de M2.

INC(M1/B-{M1})= (¬a.c.d,[1,0,0,0]) n'est pas inférieur à FP1=(¬a.b.d,[1,1,0,0])
INC (M2/ B-{M2})= (b.c.d,[0,1,0,0]) n'est pas inférieur à FP1

Ces deux fils-produits ne pourront apporter de gain, pourtant si FP1 était généré aussi, on pourrait obtenir le gain d'un monôme (M1, M2, M3, M4 remplacés par FP1, FP2 et FP3).

Cette lacune est lieé au fait que l'incouvert d'un père est comparé individuellement à chacun de ses fils-produits et non à l'union de ceux-ci. Dans

l'exemple l'union des fils-produits de M1 couvre bien son incouvert (idem pour M2). Le temps de calcul qu'engendrerait la détection de tels cas n'est pas admissible par rapport à la proportion des situations de ce type.

De plus, nous pouvons essayer de parer à une partie de ces situations très simplement. Après l'adjonction de nouveaux monômes, au moment de la suppression des indécidés dans l'étape d'extraction de base irrédondante, dans le cas où un choix de suppression est possible (voir étape3, les incouverts sont égaux), nous supprimerons en priorité les pères par rapport au fils-produits. Nous laissons ainsi une chance aux nouveaux monômes, (même s'ils ne sont pas utiles sur le moment) d'être utiles à la prochaine boucle de l'algorithme général.

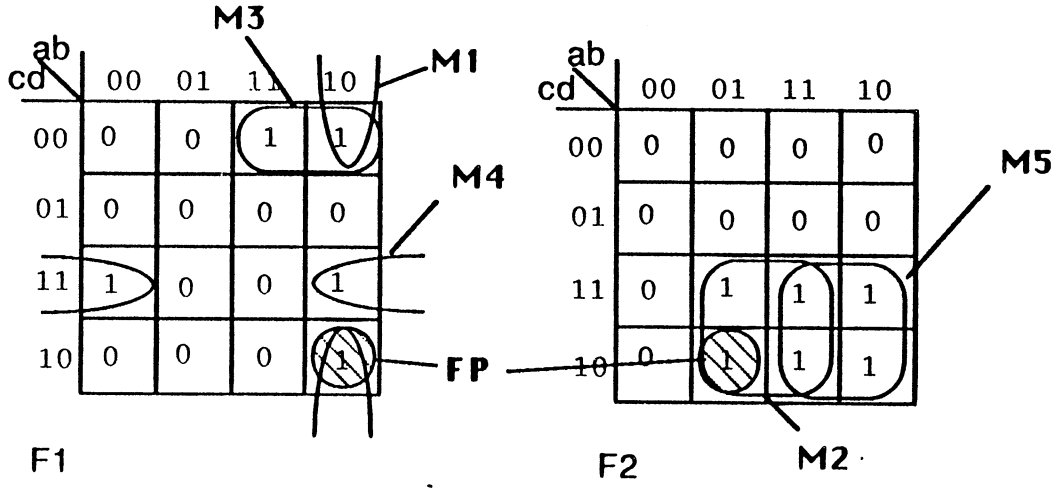
Dans l'exemple précédent M3 et M4 peuvent ainsi être remplacés par FP2 et FP3. Et à la prochaine intersection généralisée, cette fois-ci FP1 sera généré puisque les incouverts de M1 et M2 auront diminué.

Ainsi cette heuristique dont l'élaboration est trés simple, limite beaucoup le nombre de nouveaux monômes générés, tout en aboutissant à de trés bons résultats (voir résultats chap. 5).

3.6.4. DES FILS-PRODUITS PREMIERS

Les fils-produits comme ils ont été définis jusqu'ici ne sont pas forcément premiers.

Exemple :



Le fils-produit FP= (¬a.b.¬c.d, [1,1]) de M1 et M2 n'est pas premier puisque le monôme (¬a.b.d,[1,1]) est inférieur à (F1, F2).

Nous allons rendre les fils-produits premiers. Nous serons ainsi toujours en possession d'une base après l'adjonction de ces nouveaux monômes.
Le fait d'agrandir ces nouveaux monômes pour les rendre premiers va permettre dans certains cas d'apporter un gain que l'on n'aurait pas pu avoir.

Dans l'exemple précédent un seul fils-produit est généré (M3, M4, M5 sont essentiels), si on ne le rend pas premier, il n'y aura pas de gain. Par contre si nous l'agrandissons en (¬a.b.d,[1,1]), il remplacera M2 et M4.
L'agrandissement du fils-produit FP, lui a permis de couvrir l'incouvert de ses deux pères (alors qu'il n'en couvrait qu'un au départ).

Un monôme peut être étendu dans différentes directions et ainsi aboutir à différents monômes premiers.

D'aprés l'heuristique établie précédement, nous savons que chaque fils-produit obtenu couvre au moins l'incouvert d'un de ses deux pères, nous allons donc diriger leur expansion de telle façon à ce qu'ils couvrent l'incouvert de leur deuxième père (si ce n'est pas encore le cas).

Proposition :

Un fils-produit FP= (m, l) qui ne couvre pas l'incouvert d'un de ses deux pères M1, ne peut le couvrir que par expansion de m.

Démonstration :

Par construction le vecteur d'inclusion de FP est strictement plus grand que celui de M1, donc l'incouvert de M1 étant inférieur à M1, le vecteur d'inclusion de cet incouvert est forcément inférieur à celui de FP.

Notre expansion des fils-produits sera donc faite au niveau de la partie "monôme" de façon à essayer de couvrir la partie "monôme" de l'incouvert du père. Pour ce faire, on tentera de supprimer d'abord les variables qui n'apparaissent pas dans l'incouvert. Nous complèterons ensuite son vecteur d'inclusion pour qu'il devienne premier (voir paragraphe. 3.3). Il se peut que le monôme premier ainsi

obtenu soit déjà présent, soit dans la liste des monômes de la base de départ, soit dans la liste des monômes générés avant lui. Le premier cas est impossible.

Proposition :

Si un fils-produit couvre l'incouvert d'au moins un de ses deux pères alors il n'existe pas de monôme de la base de départ auquel il est inférieur.

Démonstration :

Soit FP le fils-produit de M1 et M2 et B la base de départ.
Supposons que FP couvre INC(M1 / (B - {M1}) + PHI).

a) Supposons qu'il existe M élément de B et différent de M1 et M2 tel que M ≥ FP. Par transitivité nous avons M ≥ INC (M1), ceci voudrait dire que M1 n'est pas obligatoire dans B. Donc que B n'est pas une base irrédondante, ce qui est contraire à nos hypothèses.

b) Par construction du fils-produit le vecteur d'inclusion de celui-ci est strictement plus grand que celui de ses deux pères, un fils-produit ne peut donc être inférieur à un de ses deux pères.

Donc en agrandissant un fils-produit au maximum pour le rendre premier, il est impossible d'obtenir un monôme de la base de départ, il suffit donc de tester si le monôme premier obtenu n'a pas été précédement généré pour être sûr de ne pas générer plusieurs monômes identiques.
Nous pouvons remarquer que pour éviter du travail inutile, il suffit d'effectuer ce test juste après l'expansion de la partie m du fils-produit. En effet deux monômes globaux premiers (m, I) et m', I') sont égaux si m et m' sont identiques. Ce n'est pas la peine d'agrandir des vecteurs d'inclusions inutilement.

3.6.5. RESUME DE L'ALGORITHME DE GENERATION DE NOUVEAUX MONOMES

Pour chaque père de la base de départ, l'ensemble de ses fils-produits avec les autres monômes est un sous-ensemble de ses produits complexes obtenus pendant l'étape de détection des monômes essentiels et stables (voir paragraphe 3.5). Il suffit donc dans cette étape de déterminer pour chaque père parmi ses produits complexes

quels sont les fils produits qui vérifient les conditions et l'heuristique précédemment établies. Ces fils-produits seront alors rendus premiers avant d'être ajoutés à la base.

Algorithme :

Soit LPC(M) l'ensemble des produits complexes de M, LFP sera la liste des fils-produits que l'on ajoutera finalement à la base de départ B.

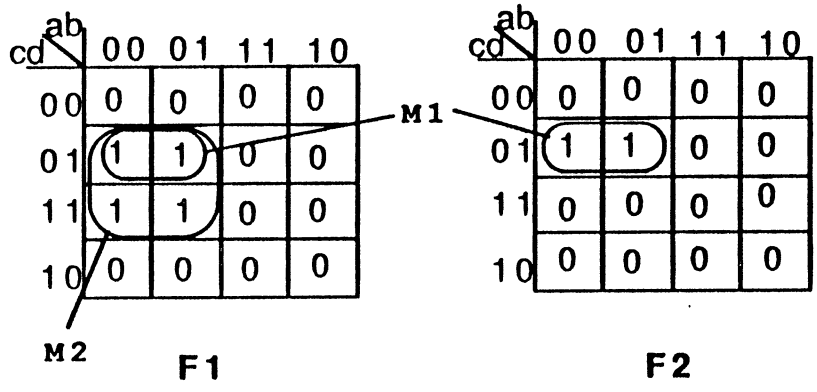
Début
Pour tout M élément de B *faire*
début
Si M n'est ni essentiel, ni stable *alors*
début
Pour tout MPC élément de LPC(M) *faire*
début
Si INC(M) ≤ MPC alors
début
Expansion dirigée de MPC en MPC-PREMIER ;
Si MPC-PREMIER n'est pas élément de LFP *alors*
LFP <- LFP ∪ MPC-PREMIER
finsi
finpour
finsi
finpour
B <- LFP ∪ B
fin

3.7. ETAPE FINALE :
RETOUR A DES ENSEMBLES DE MONOMES LOCALEMENT IRREDONDANTS

Quand nous arrivons à cette étape finale, nous sommes en possession d'une base irrédondante pseudo-minimale de monômes globaux. Nous avons obtenu un minimum approché en terme de nombre de monômes (taille du circuit final). Il est cependant encore possible d'obtenir un gain relatif aux performances du PLA (fonction du nombre de transistors du circuit). Un transistor est nécessaire pour chaque variable et chaque 1 du vecteur d'inclusion d'un monôme global.

Il est évident que l'on ne peut supprimer des variables dans les monômes de la base irrédondante obtenue, puisque ces monômes sont premiers. Par contre pour une fonction simple composant la fonction générale, il se peut que l'ensemble de monômes qui lui sont associés (composante du vecteur d'inclusion correspondant à cette fonction à 1) ne soit pas irrédondant. Et donc que certains 1 dans des vecteurs d'inclusions puissent être supprimés.

Exemple :



M1= (¬a.¬c.d, [1,1]) et M2= (¬a.d, [1,0]) forment une base irrédondante de (F1, F2).
L'ensemble des monômes définissant F1 : {¬a.¬c.d, ¬a.d} n'est pas un ensemble irrédondant puisque ¬a.¬c.d est redondant. On ne peut pas parler de base car ces monômes ne sont pas forcément premiers.
Nous pouvons donc remplacer M1 par M1'= (¬a.¬c.d, [0,1]) , {M1', M2} couvre toujours l'ensemble des points de (F1, F2).

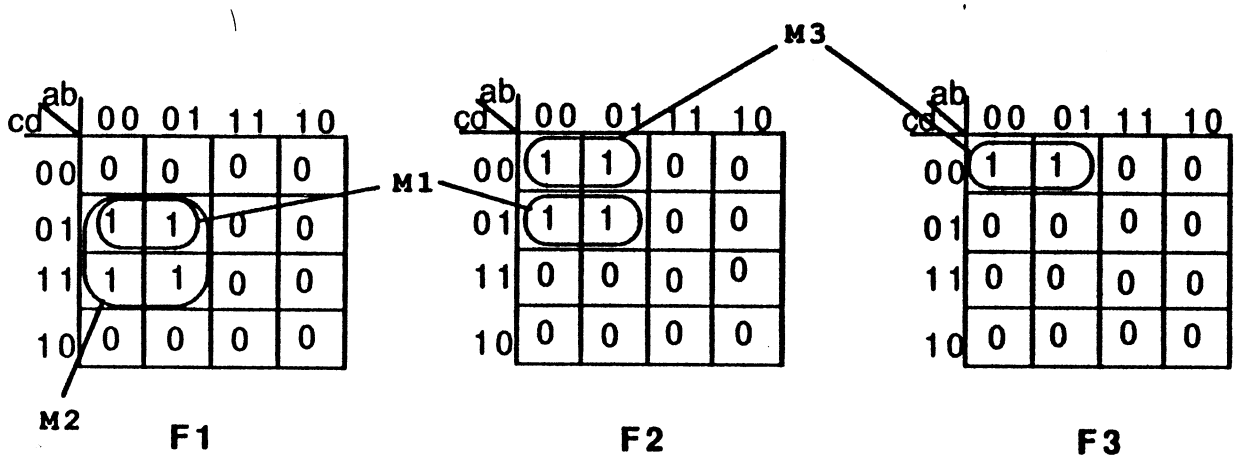
Pour effectuer cette transformation, il suffit donc, à partir de l'ensemble des monômes locaux associés à chaque fonction, d'extraire un ensemble de monômes

irrédondants. Ceci sera fait aisément grâce à l'algorithme défini dans l'étape d'extraction de base irrédondante (parag. 3.4.), à partir de l'ensemble des monômes locaux à chaque fonction simple et de la couverture à ϕ de ces fonctions. Pour une fonction donnée, les monômes locaux supprimés donneront une transformation d'un 1 en 0 de la composante (correspondant à la fonction) du vecteur d'inclusion du monôme global correspondant.

Une fois cette première phase effectuée, il se peut maintenant qu'il existe des monômes globaux dont la partie monôme puisse être étendue. En effet, les monômes globaux dont le vecteur d'inclusion a été modifié dans la première phase ne sont plus premiers. Le fait qu'ils ne soient plus nécessaires dans certaines fonctions va peut être permettre qu'on les étende dans les autres.

Exemple :

M1= (¬a.¬c.d, [1,1,0]) , M2= (¬a.d, [1,0,0]) et M3= (¬a.¬c.¬d, [0,1,1]) forment une base irrédondante de {F1, F2, F3}.



Dans la première phase M1 sera remplacé par M1'= (¬a.¬c.d, [0,1,0]).
Dans la deuxième phase M1' va pouvoir être agrandi en M1"= (¬a.¬c, [0,1,0]).

On remarquer que dans cette deuxième phase, il est inutile d'essayer d'étendre les monômes dont le vecteur d'inclusion n'a pas été modifié dans la phase précédente, ils sont toujours premiers.

Pour un monôme global (m, I), cette expansion peut se faire aisément par essais successifs de suppression des variables de m, suivi des tests d'inclusion dans les

bornes supérieures des fonctions correspondant au vecteur d'inclusion I (voir minimisation locale, parag. 3.2).

Le fait d'agrandir certains monômes dans cette deuxième phase peut occasionner l'apparition de nouveaux monômes redondants localement à certaines fonctions. Dans l'exemple précédent, une fois M1' remplacé par M1", on s'aperçcoit que M3= (¬a.¬c.¬d, [0,1,1]) peut être remplacé par M3'= (¬a.¬c.¬d, [0,0,1]). Nous bouclerons donc sur ces deux phases, tant qu'il y a au moins une transformation dans une des deux.

Dans la première phase (diminution des vecteurs d'inclusion) les seules fonctions qui peuvent contenir des monômes redondants sont celles qui contiennent au moins un monôme agrandi à la phase précédente. Pour les autres l'ensemble de monômes qui a été trouvé precédemment reste irrédondant, ce n'est donc pas la peine de réappliquer l'algorithme d'extraction de base irrédondante.

Résumé de l'agorithme :

Soit (F1, ..., Fm) la séquence des fonctions simples traitées, soit (PHI1, ..., PHIm) leurs couvertures à j respectives. Soit B la base irrédondante de départ. Soit E l'ensemble des monômes qui ne sont plus premiers, soit F-N-IRR le sous ensemble des formes non-irrédondantes des fonctions simples, soit ENS-MON-LOC(Fi) l'ensemble des monômes associés à la fonction Fi (Partie "monôme" des monômes globaux de B dont la $i^{ème}$ composante du vecteur d'inclusion est à 1).

1- Initialisation :

E <- Ø ;

F-N-IRR <- {F1, ..., Fm} ;

2- Phase 1 :

Pour tout Fi de F-N-IRR *faire*

Extraire un ensemble irrédondant de (ENS-MON-LOC(Fi), PHIi) grâce à l'algorithme donné au paragraphe 3.4. ;

Modifier les vecteurs d'inclusion des monômes globaux de B correspondant aux monômes locaux supprimés ;

E <- ensemble des monômes globaux précédement modifiés ;

F-N-IRR <- F-N-IRR - {Fi}

finpour

3- Phase 2 :

Pour tout monôme global (m, I) de E *faire*

Etendre au maximum m tel qu'il soit encore inclus dans les fonctions de I ;

Si m a été agrandi *alors*

mettre dans F-N-IRR les fonctions correspondant à I

finsi

finpour

{Test d'arrêt :}

Si F-N-IRR ≠ Ø *alors* retour en 2 (phase 1)

sinon arrêt.

3.8. EQUIVALENCE DE DEUX FORMES POLYNOMIALES

Dans un outil de CAO comme ASYL, il peut être utile de tester l'équivalence de deux expressions polynômiales. En particulier, il est vital pour un concepteur, de vérifier l'équivalence du résultat du minimiseur deux couches et du système de départ. Nous avons donc inclus, en fin de notre minimiseur, un test de conformité.

Définition :

Soit F une fonction phi-booléenne simple de borne supérieure BS et de borne inférieure BI, une somme de monômes E est une représentation de F (E définit correctement la fonction F) si et seulement si BS ≥ E ≥ BI.

A partir de cette définition, il est simple de vérifier que la forme polynômiale obtenue à la fin de la minimisation est une représentation de la fonction générale de départ. La fonction générale F= (f_1,...,f_k) est donnée par l'expression polynômiale de sa borne supérieure et de sa borne inférieure. Cette représentation peut être

décomposée en k expressions polynômiales simples (bs_1,...bs_k) des bornes supérieures et k expressions polynomiales (bi_1,...,bi_k) des bornes inférieures des fonctions simples (f_1,...,f_k). De même pour la forme polynômiale du résultat de la minimisation, soit (r_1,...,r_k) le vecteur des k expressions polynômiales simples équivalentes. Il ne reste plus qu'à vérifier que $bs_i \leq r_i \leq bi_i$ pour tout i entre 1 et k.

Puisque bs_i , r_i et bi_i sont des formes polynômiales, cela peut être vérifié facilement grâce au test d'inclusion (paragraphe 2.2.1). En effet, il suffit de tester que chaque monôme de r_i est inférieur à bs_i et que chaque monôme de bi_i est inférieur à r_i.

Si aucun test d'inclusion n'est réfuté (pour tout i entre 1 et k), nous sommes sûr que le système final définit correctement le système initial.

4. ENVIRONNEMENT ET PROGRAMMATION

4.1. LE SYSTEME ASYL

Le travail présenté ici, s'insère dans un sytème complet développé à l'URCSI (anciennement LCS) depuis déjà quelques années. Cet outil de CAO d'Aide à la SYnthèse Logique (ASYL) a été mis en oeuvre en 82 [HAN85], [SAU88]. Il permet, à partir d'une description comportementale, d'implémenter un contrôleur sur différents types de circuits de façon automatique (voir figure 1).

Les point principal d'entrée de ce système est une spécification comportementale (automate de Moore ou Mealy à commandes structurées). A un niveau plus haut, se trouve une description de type RTL (Register Transfer Language) [HART77] qui permet une simulation comportementale du circuit grâce à un outil développé également à l'URCSI : CADOC [CRAST85].

Il est effectué sur l'automate de départ des vérifications de conformité. Détection de blocage dans le cas où une configuration d'entrée dans un état donné n'aboutit à aucun état suivant ou détection de parallélisme dans le cas où une configuration d'entrée dans un état donné aboutit à plusieurs états suivants.

Une étape de codage est ensuite effectué. Elle consiste à coder les états du contrôleur sur un certain nombre de variables d'états. Ce codage peut être de différents types (compact, un parmi n, séquentiel ...), mais dans ASYL un effort a été apporté pour choisir de façon automatique un codage "optimisé" [POI88]. Le codage sera optimisé dans le sens où il permettra un gain de surface ou de boitiers suivant la cible choisie au moment de l'implémentation physique du contrôleur. Ce codage optimisé dépendra intrinsèquement du type de circuit choisi pour l'implémentation finale.

Une fois les codes des états choisis, les équations booléennes des variables d'états et des sorties du contrôleur peuvent être générées. Une étape de minimisation est alors ensuite effectuée. Il est évident que la minimisation est complètement liée à la cible choisie puisque les critères de minimalité et les moyens de les rejoindre varient complètement d'une cible à l'autre.

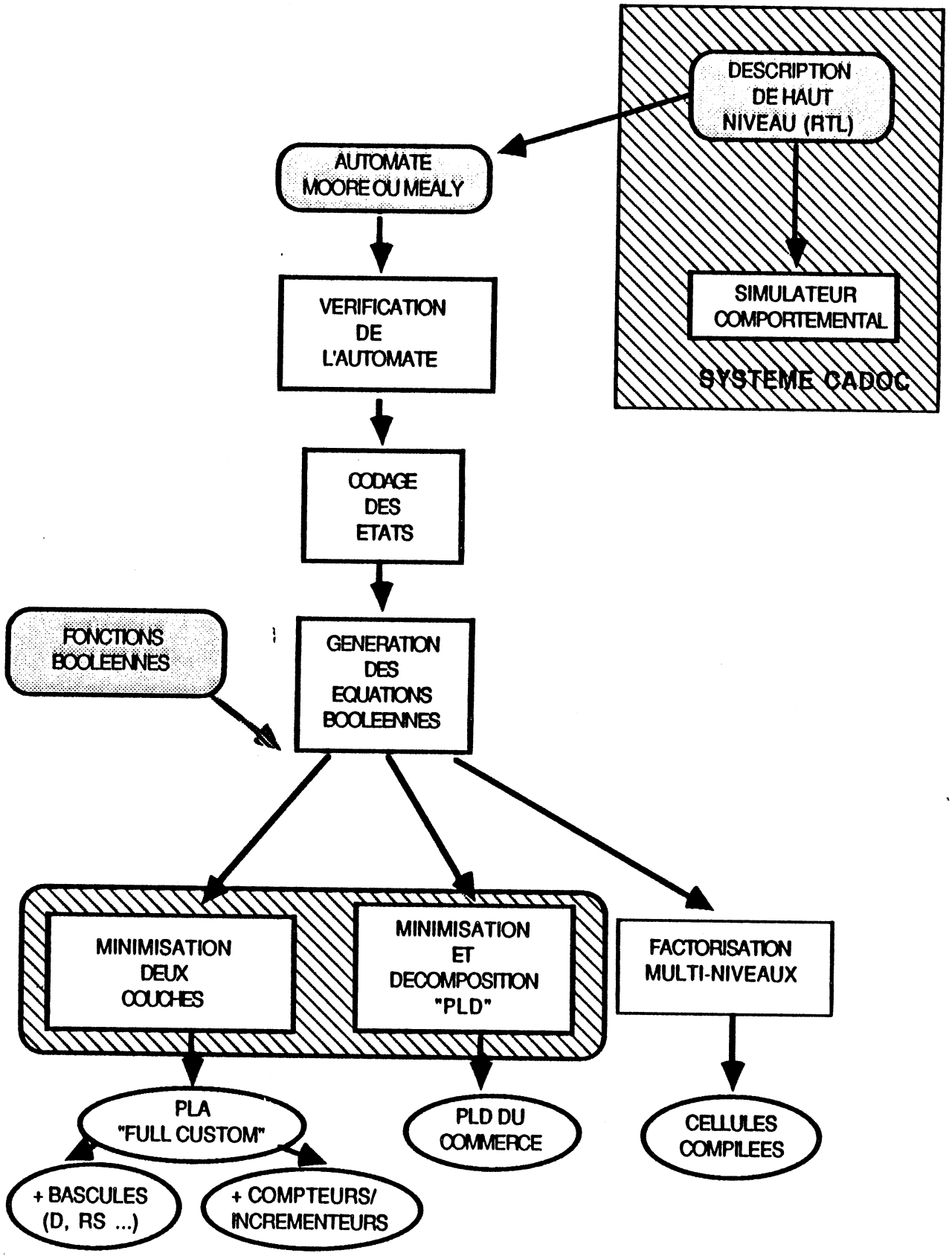


FIGURE 1 : LE SYTEME ASYL

Pour la cible 'cellules compilées', les fonctions seront décomposées en cellules (ou portes complexes) de type NMOS ou CMOS respectant différentes conditions :

- ordre lexicographique des variables respecté permettant une implémentation VLSI aisée.

- conditions de type électrique telles que le nombre de transistors en série et en parallèle.

La minimisation consistera alors à réduire la surface de silicium tout en optimisant la vitesse du circuit final. Elle se traduira donc par une factorisation de transistors (correspondant aux variables) et une décomposition intelligente en sous-fonctions [THU83], [LEV88].

4.2. UN PROTOTYPAGE EN PROLOG

Avant d'implémenter cet algorithme dans un langage de programmation performant, nous avons tout d'abord réalisé une maquette en PROLOG.

PROLOG (PROgrammation en LOGique) tire ses origines des travaux sur la logique formelle et plus particulièrement des prédicats du premier ordre. Il a été conçu pour s'adapter aux différents domaines dits de l'Intelligence Artificielle.

Avec la programmation classique dite procédurale ou impérative, toutes les étapes de l'algorithme permettant de résoudre un problème sont spécifiées. Avec la programmation en logique, par contre, on va chercher à spécifier le résultat attendu plutôt que la manière d'y parvenir.

Exemple :

Soit le problème :

X appartient-il à la liste L ?

Dans un langage impératif il faut :

* définir la structure de données représentant la liste
* manipuler explicitement cette structure
* parcourir la liste en cherchant la valeur X

En Prolog on part d'une définition du problème :

X appartient à la liste L si :

* soit X est le premier élément de L
* soit X appartient au reste de la liste

Ce qui s'écrit de façon naturelle :

appartient (X, X. L1) -> fini.
appartient (X, Y. L1) -> appartient (X, L1).

La philosophie de programmation est complètement différente. Pour écrire un programme PROLOG résolvant un problème, il suffit de bien définir le problème sous forme d'une liste de prédicats. Un prédicat peut être toujours vrai (axiome) ou dépendre d'autres prédicats. A partir de cette spécification du problème, l'écriture du programme PROLOG est une simple recopie. Pour de plus amples détails sur la programmation en PROLOG, on peut se référer à [CLO87], [COL83] et [COL84]

Ce langage s'est bien prêté aux problèmes de la minimisation. En faisant une maquette de notre minimiseur en PROLOG, nous avons pu de façon très rapide, essayer différentes techniques et heuristiques et ainsi faire des choix judicieux.

Un exemple, la preuve de tautologie :

En Prolog, un monôme sera une liste de variable, soit sous forme normale x_i, soit complémentée $\neg x_i$. Une fonction simple sera alors une liste de monômes. La fonction tautologie sera représentée par la liste [1] et la fonction toujours nulle par [0].
Le problème, comme il a été spécifié au paragraphe 2.2.2, peut être enoncé par:

une fonction f est une tautologie si et seulement si:

* soit f = 1
* soit f_x et $f_{\neg x}$ sont des tautologies, x étant une variable de f

Le programme Prolog est alors le suivant :

```
tautologie ( [0] ) -> faux et
                      fini.
tautologie ( [1] ) -> vrai et
```

```
                    fini .
tautologie ( F ) -> variable (F, X) et
                    cofacteur (F, X, FX) et
                    cofacteur (F, not X, FNOTX) et
                    tautologie (FX) et
                    tautologie (FNOTX).
```

Le prédicat variable (F, X) sera vrai si X est une variable de F. La résolution de ce prédicat pourra être plus ou moins compliqué suivant le choix de la variable de coupure. Le prédicat cofacteur (F, X, FX) renverra dans la variable FX le cofacteur de F par rapport à X.

En Prolog, un problème est ainsi décomposé en sous-problèmes de complexité moindre, on peut ainsi modifier certains prédicats sans toucher aux autres. De plus, on peut très facilement ajouter une variante à un tel prédicat sans modifier ce qui a déjà été écrit. Supposons que l'on veuille maintenant dans la preuve de tautologie, traiter le cas particulier où il existe une variable n'apparaissant que sous une seule forme dans F (Voir parag. 2.2.2.1). Il suffit pour, ce faire, de rajouter à la clause tautologie précédemment donnée :

```
tautologie ( F ) ->
        variable-monoforme ( F, X ) et
        cofacteur ( F, not X, FNOTX ) et
        tautologie ( FNOTX ).
```

Ce langage permet ainsi une programmation concise, claire et rapidement modifiable d'un problème.

Le minimiseur écrit en PROLOG comporte à peu près 500 lignes, alors qu'il a fallu plus de 5000 lignes pour l'implémenter en PASCAL. Ce programme Prolog n'est malheureusement resté qu'une maquette, la place mémoire et le temps d'éxecution en programmation logique étant encore très élevés par rapport à un langage impératif. Les exemples de forte complexité que nous voulions arriver à traiter ne pouvaient 'passer' avec une telle programmation.

4.3. UNE REALISATION EN PASCAL

Une fois les différentes techniques et heuristiques choisies grâce au prototypage PROLOG, nous avons implémenté le minimiseur d'ASYL dans un langage plus performant pour pouvoir appliquer notre minimisation à des exemples de tailles importantes en un temps et une place mémoire raisonnables. Le système ASYL étant en PASCAL, nous avons donc choisi ce langage pour fixer nos algorithmes.

Une expression polynômiale d'une fonction générale est définie par une matrice booléenne. Chaque ligne définit un monôme global (m,I), les premières colonnes définissant les variables de m, les dernières son vecteur d'inclusion I. A chaque colonne correspondante à m est associée une variable x_i. Si x_i apparaît sous forme complémentée (resp. directe) dans le monôme, alors on aura un "O" (resp. 1) dans la $i^{ème}$ colonne de ce monôme. Si x_i n'apparaît pas dans m, alors cette colonne est mise à la valeur indéfinie "X". De même pour le vecteur d'inclusion, chaque colonne correspond à une fonction simple composant la fonction générale.

exemple :

$F = (f1, f2) = (x.y\ ,\ [1,1]) + (x.\neg z\ ,\ [1,0]) + (\neg x\ ,\ [0,1])$

La représentation matricielle en est

x	y	z	f1	f2
1	1	X	1	1
1	X	0	1	0
0	X	X	0	1

Pour les fonctions phi-booléennes, nous aurons besoin de la borne inférieure et de la couverture à j . Nous les représenterons par deux matrices de ce type.

Chaque élément de la matrice est représenté sur un octet; si la fonction générale est définie de B^n dans B^k, nous aurons besoin de m*(n+k) octets pour représenter une expression polynômiale de m monômes.

Cette structure de données n'est pas des plus concises puisque deux bits suffiraient pour coder les trois valeurs (1,0,X) possibles pour les variables d'un monôme, et un seul bit suffirait pour chaque composante des vecteurs d'inclusions. De plus, il aurait

été intéressant de pouvoir utiliser des opérations 'machine' pour réaliser la plupart des opérations élémentaires que nous avons à effectuer sur les monômes. Nous pourrions passer ainsi d'une complèxité dépendant du nombre de variables à une complèxité constante par rapport à ce nombre. Une telle structure de donnée et des accés à des opérations machines ne sont pas possibles en Pascal standard. Nous avons fait une étude rapide sur la faisabilité d'une telle structure et les moyens d'utiliser des opérations machines pour réaliser les différentes manipulations sur les monômes. Les principaux résultats de cette étude sont décrits dans le paragraphe suivant. Un langage comme C pourrait alors trés bien s'adapter à une telle représentation.

4.4. STRUCTURE DE DONNEE AMELIOREE

Un monôme global (m, I) d'une fonction générale de B^n dans B^k sera représenté par deux vecteurs de n bits (VB1, VB2) pour m, et un vecteur de k bits VI pour I. Il faut trouver le codage des valeurs (1, 0, X) des variables d'un monôme, tel que les manipulations de base sur les monômes (produit, consensus, comparaison ...) puissent être réalisées par des opérations 'machine' sur VB1, VB2, VI.

Le codage suivant donne des propriétés intéressantes :

1 sera codé (1,1)

0 sera codé (0,0)

X sera codé (0,1)

Le code restant (1,0) servira à définir la configuration 'impossible'.

Exemple :

le monôme (a.¬b, [1,0,1]) pour une fonction dépendant de trois variables a, b et c sera représenté par :

(VB1, VB2, VI) = ((1,0,0), (1,0,1), (1,0,1))

Au niveau des vecteurs d'inclusion, les opérations de base tel que produit, somme pourront être réalisées de façon trés simple. Le produit de I1 et I2 sera réalisé par un ET logique entre VI1 et VI2, la somme par un OU logique.

Il en est de même pour les opérations sur les parties monômes. Nous voulons par exemple, réaliser le produit de m (représenté par (VB1,VB2)) et de m' (représenté par (VB1',VB2')). Avec le codage donné précédemment ceci peut être réalisé simplement en deux étapes :

*Calcul de P1= VB1 + VB1' et P2 = VB2 . VB2'

*Si ($P1_i$,$P2_i$) est différent de (1,0) pour tout i de 1 à n alors (P1,P2) est une représentation correcte du produit m.m'

sinon m.m' = 0 et une représentaion de m. m' est ((0,...,0), (0,...,0)).

Pour réaliser la première étape, il suffit d'effectuer le ET et OU logique. Pour la deuxième étape, il suffit d'effectuer le ET logique entre P1 et ¬P2 et de tester si P1.¬P2 = (0,...,0). Si ce n'est pas le cas, le produit m.m' est nul.

Ainsi, avec une telle structure de données, on peut se ramener pour la plupart des manipulations sur les monômes à des séries d'opérations 'machine' élémentaires. La complexité d'une grande partie des procédures de l'algorithme de minimisation ne dépendra alors plus du nombre de variables et du nombre de composantes des vecteurs d'inclusion.

5. RESULTATS ET CONCLUSION

L'ensemble d'exemples que nous avons utilisé pour montrer la validité de notre logiciel a voulu être des plus significatifs.
Ils sont trés variés par leurs tailles :

- nombre de variables de 4 à 26,
- nombre de fonctions simples de 3 à 39,
- nombre de monômes initiaux de 12 à 1848.

De plus ils sont issus de différents milieux et représentent différents types d'application :

1-Benchmarks officiels tirés du DAC86.

NOM	V	F	MI
F2	4	4	12
MISEX1	8	7	32
MISEX2	25	18	29
RD53	5	3	32
5XP1	7	10	75
DUKE2	22	29	87
MISEX3	14	14	1848

2-Fonctions mathématiques :

NOM	V	F	MI
MULT3	6	6	64
DIST1	8	5	256
DIST2	8	5	256
CARRE6	6	12	64

-MULT3 : multiplication 3 * 3 bits.

-DIST1 et DIST2 deux fonctions arrondies de racine carrée

de $x^2 + y^2$ (x et y sur 4 bits).

-CARRE6 : x^2 avec x sur 6 bits.

3-Exemples industriels fournis par l'Electronique Serge Dassault.

NOM	V	F	MI
ESD1	7	5	27
ESD2	16	13	104
ESD3	12	11	43
ESD4	10	11	39

4-Contrôleurs réalisés à l'URCSI ou benchmarks de contrôleurs officiels.

NOM	V	F	MI
STYR *	14	15	134
CSE *	11	11	58
PLANET*	13	25	131
JAY *	10	39	179
INTVAR	26	7	503

V : nombre de variables

F : nombre de fonctions simples

MI : nombre de monômes initial

Les exemples marqués d'une étoile sont des fonctions phi-booléennes, le nombre de monômes initial est alors celui de la borne supérieure.

Dans un premier temps nous montrerons le comportement de notre algorithme de minimisation :

* nombre de boucles et évolution du nombre de monômes au cours de l'algorithme (Tableau 1).

* nombre de monômes essentiels et évolution du nombre de monômes stables, et de fils produits générés (Tableau 2).

* différences de résultats suivant les deux stratégies possibles de l'étape 1 (parag. 3.2) de notre algorithme (Tableau 3).

*Gain obtenu dans la dernière étape réduisant le nombre de transistors dans le PLA final (Tableau 4).

Nous comparerons ensuite ces résultats à ceux d'ESPRESSO-II [BRA84] qui est à l'heure actuelle un des minimiseurs les plus performants.
Ceci pour les trois critères :

-nombre de monômes résultant de la minimisation

-nombre de points (nombre de variables et de 1 dans les vecteurs d'inclusions de l'ensemble des monômes)

-temps d'exécution de la minimisation

Pour conclure, nous comparerons nos résultats à ceux de minimiseurs exacts.

5.1. COMPORTEMENT DE L'ALGORITHME

Le tableau 1 montre l'évolution du nombre de monômes au cours de notre algorithme. La première colonne donne le nombre de monômes globaux au départ. Pour les fonctions phi-booléennes, les deux nombres donnés sont le nombre de monômes de la borne inférieure et celui de la couverture à j. La deuxième colonne donne le nombre de monômes dans la première base obtenue à partir des bases locales de chaque fonction simple. Les bases locales n'étant ici pas complètes. Les colonnes suivantes présentent les bases irrédondantes obtenues par la suite (boucle extraction de base irrédondante, génération de nouveaux monômes).

Le nombre de boucles dans notre algorithme est trés faible, il dépasse rarement deux, excepté pour les très gros exemples comme MISEX3 (1848 monômes, 14 variables et 14 fonctions simples), PLANET (131 m., 13 v. et 25 f.) ou INTVAR (503 m., 26 v. et 7 f.) pour lesquels 4, 3 et 3 boucles sont nécessaires.

NOM	initial	première base	base irréd. 1	base irréd. 2	base irréd. 3	base irréd. 4
F2	12	12	12	8	8	
MISEX1	32	18	13	12		
MISEX2	29	29	29	28		
RD53	32	32	31			
5XP1	75	70	67	63	63	
DUKE2	87	124	95	87	86	
MISEX3	1848	1201	789	731	725	718,715
MULT3	64	38	35	31	31	
DIST1	256	157	116	107	107	
DIST2	256	195	150	139	139	
CARRE6	64	62	55	51	51	
ESD1	27	22	21			
ESD2	104	117	83	76	76	
ESD3	43	42	36	34		
ESD4	39	90	40	33		
STYR*	131, 3	209	114	100	100	
CSE*	54, 4	95	52			
PLANET*	115, 16	274	136	104	102	100
JAY*	179, 19	236	135	113	113	
INTVAR	503	503	351	228	222	222

TABLEAU 1 : Evolution du nombre de monômes

L'algorithme sort de la boucle (extraction de base irrédondante, génération de nouveau monôme), soit parce que nous sommes arrivés à une solution stable (plus de gain), soit parce qu'aucun nouveau monôme n'est généré (voir tableau 2). Il est difficile de donner une explication générale à ces évolutions trés différentes. Elles dépendent de la structure même de chaque exemple (nombres de variables et de fonctions simples) et surtout de sa forme d'entrée (nombre et structures des monômes initiaux).

Le nombre de monômes de la première base peut être plus grand que le nombre initial (par exemple ESD4 où l'on passe de 39 à 90), en effet on n'augmente pas le

nombre de monômes dans chaque base locale des fonctions simples mais on peut très bien augmenter le nombre de monômes de la base globale résultante (voir parag. 3.3).

On peut remarquer que les exemples qui vérifient ce cas, possèdent en général beaucoup de variables et de fonctions simples par rapport au nombre de monômes initiaux (exemple Duke2 : V= 22, F= 29, MI=87 ou Planet : V=13, F= 25, MI=131). Ceci peut s'expliquer par le fait que la minimisation locale de chaque fonction simple (nombreuses) aura tendance à modifier les monômes de façon différentes (beaucoup de variables) sans pour autant en supprimer beaucoup (peu de monômes au départ). On aboutit alors pour chaque fonction simple à des monômes différents, on augmente ainsi le nombre de monômes de la première base globale à partir des monômes initiaux.

Mais il est dans tous les cas nettement inférieur (surtout pour les gros exemples) au nombre de monômes de la base complète (voir tableau 6, parag. 5.3) et diminue trés rapidement (dés la première extraction de base irrédondante) .

Le tableau 2 donne le nombre de monômes essentiels, et le nombre de monômes stables et de nouveaux monômes (fils produit) générés dans les boucles successives de notre algorithme.

La proportion de monômes essentiels dans la première base globale est très variables suivant les exemples. Ce nombre ainsi que le nombre de monômes stables influence beaucoup le nombre de nouveaux monômes générés. Ainsi pour 5XP1, il n'y a que 8 monômes essentiels et pas de stable dans la première base irrédondante (67), le nombre de fils-produit est alors de 51. Par contre, dans la première base irrédondante de RD53, tous les monômes sont soit essentiels, soit stables, aucun fils-produits ne sera généré.

Le nombre de monômes stables croît très vite dans la plupart des exemples. Ainsi grâce à la restriction basée sur les monômes stables et essentiels plus l'heuristique basée sur l'incouvert des pères (parag.3.6.2 et 3.6.3), le nombre de nouveaux monômes devient très faible dès la deuxième boucle.

NOM	Essentiels	stabl.,fils 1		stabl.,fils 2		stabl.,fils 3		stabl.,fils 4	
F2	0	0	6	8	0				
MISEX1	10	0	1	1	0				
MISEX2	26	0	1	1	0				
RD53	21	10	0						
5XP1	8	0	51	4	17				
DUKE2	51	6	14	15	1	16	0		
MISEX3	97	235	383	328	85	361	33	374	18
MULT3	3	2	24	17	2				
DIST1	16	31	21	48	2				
DIST2	22	37	58	68	4				
CARRE6	3	6	34	15	9				
ESD1	13	8	0						
ESD2	48	12	12	21	2				
ESD3	25	3	2	5	0				
ESD4	22	0	9	9	0				
STYR	31	37	28	53	1				
CSE	36	12	3	15	0				
PLANET	25	14	86	49	11	56	5	59	2
JAY	32	13	64	41	12				
INTVAR	41	39	242	147	15	161	2		

TABLEAU 2 : Monômes essentiels, stables et fils produit

Ainsi tout au long de l'algorithme, le nombre de monômes reste trés petit par rapport au nombre de monômes que l'on pourrait engendrer par l'obtention d'une base globale complète (voir tableau 6, paragraphe 5.3).

Le tableau suivant montre les différences que l'on peut obtenir suivant le choix des bases locales dans la première étape (parag. 3.2), soit quelconques (Min. Loc. 1), soit complètes (Min. Loc. 2.). Les deux premières colonnes donnent les résultats obtenus en nombre de monômes, les deux suivantes les temps d'exécution en secondes.

Les résultats pour MISEX3 et INTVAR ne sont pas donnés, les monômes des bases locales complètes de ces exemples n'ayant pu être calculé dans un temps raisonnable. Les temps de calculs augmentent considérablement (facteurs > 3 dans certains cas) si l'on choisit de générer les bases complètes locales (Tps ML 2), le nombre de monômes de la première base globale étant alors considérablement augmenté. On passe par exemple respectivement pour DUKE2, STYR et JAY de 124, 209 et 236 (Tableau 1) à 255, 498 et 510 monômes dans la première base globale dans le cas où les bases locales sont complètes.

NOM	Min. Loc. 1	Min. Loc. 2	Tps ML 1	Tps ML 2
F2	8	8	1,6	1,6
MISEX1	12	12	3,2	4,5
MISEX2	28	28	8,1	9,2
RD53	31	31	3,5	6,2
5XP1	63	63	44	50,6
DUKE2	86	86	166,9	379
MISEX3				
MULT3	31	31	12,5	15,3
DIST1	107	107	94	147
DIST2	139	138	147,9	290
CARRE6	51	48	39,5	48,6
ESD1	21	21	2,9	5,4
ESD2	76	75	69,3	187,1
ESD3	34	34	11,1	23,2
ESD4	33	33	42	46,1
STYR	99	99	202,1	746,2
CSE	52	52	32,9	66
PLANET	100	100	696	1635,5
JAY	113	111	544,9	1486,6
INTVAR				

TABLEAU 3 : Alternative sur les bases locales

Sur les 20 exemples traités, cette alternative n'apporte un gain que pour 4 de ceux-ci. De plus ce gain n'est pas des plus significatifs, au maximum 3 sur 50 pour CARRE6. Cette alternative pourra être envisagé pour des exemples de taille moyenne (nb. var. ≤ 10, nb. fct. ≤ 10, nb. mon ≤ 200), si on ne veut pas trop augmenter le temps d'exécution (facteur < 2).

Le tableau 4 donne le gain obtenu dans la dernière étape de notre algorithme (parag. 3.7) dans le cas de bases locales quelconques. Cette étape est effectuée pour minimiser le nombre de variables et de 1 dans les vecteurs d'inclusion de l'ensemble des monômes (correspond au nombre de transistors du PLA final dont dépend les performances du circuit).

NOM	avant	après
F2	40	40
MISEX1	99	96
MISEX2	224	213
RD53	182	172
5XP1	346	325
DUKE2	1105	994
MISEX3	9081	8029
MULT3	189	169
DIST1	795	747
DIST2	1082	1002
CARRE6	362	257
ESD1	129	129
ESD2	736	663
ESD3	238	238
ESD4	406	384
STYR	1050	962
CSE	493	484
PLANET	1324	1160
JAY	1056	935
INTVAR	2812	2692

TABLEAU 4 : Gain de transistors dans l'étape finale

Ce gain varie entre 0 (ESD1 ou ESD3) et plus de10 pour 100 (PLANET ou JAY), il n'est donc pas négligeable.

Cette dernière étape est effectuée pour obtenir un gain de transistors mais il peut arriver qu'elle aboutisse au gain de quelques monômes. C'est le cas pour STYR et PLANET où un monôme est gagné dans cette étape (voir dernière base irrédondante dans tableau 1 et résultat final dans tableau 5).

5.2. COMPARAISON AVEC ESPRESSOII

Le tableau 5 donne la taille respective de chaque exemple (V: nombre de variable, F: nombre de fonctions simples, MI: nombre de monômes initiaux), puis le nombre de monôme, le nombre de transistors et le temps en secondes auxquels aboutissent respectivement le minimiseur d'Asyl et Espresso2.

L'option de notre minimiseur dans la première étape (bases locales) est la plus rapide (bases quelconques parag. 3.2.1).

Les temps CPU donnés ont été établis pour Espresso2 sur un VAX 785 UNIX, pour Asyl sur un MICRO-VAX2 VMS (à peu prés deux fois moins puissant).

Le nombre de monômes entre Espresso2 et notre minimiseur est, pour la plupart des exemples, identique ou ne diffère que de 1 à 3 pour 100, dans un sens ou dans l'autre. Il en est de même pour le nombre de transistors.

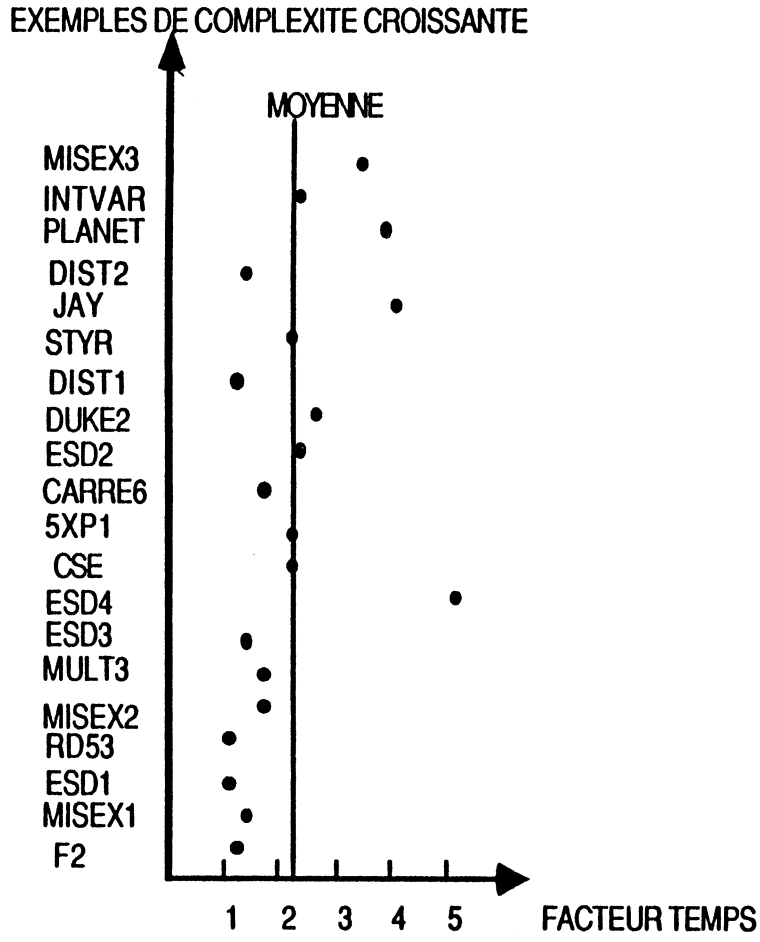
Au niveau des temps d'exécution, on peut estimer que l'on peut se ramener à des temps d'exécution sur machines équivalentes en divisant par deux les temps de notre minimiseur. Nous pouvons alors constater qu'il y a un facteur de 1 (RD53) à 5 (ESD4) entre les temps d'exécution d'Espresso2 et ceux de notre minimiseur.

NOM	V	F	MI	MONOMES		TRANSISTORS		TEMPS	
				ASYL	ESPR.	ASYL	ESPR.	ASYL	ESPR.
F2	4	4	12	8	8	40	40	1,6	0, 6
MISEX1	8	7	32	12	12	96	96	3, 2	1, 1
MISEX2	25	18	29	28	28	213	213	8, 1	2, 2
RD53	5	3	32	31	31	172	175	3, 5	1, 6
5XP1	7	10	75	63	64	359	353	44	10,1
DUKE2	22	29	87	86	86	994	990	146,9	32, 5
MISEX3	14	14	1848	715	690	8029	7836	10983	1545
MULT3	6	6	64	31	33	169	186	12,5	3,3
DIST1	8	5	256	107	109	747	769	94	34,5
DIST2	8	5	256	139	141	1002	1012	147,7	47,2
CARRE6	6	12	64	51	50	262	271	39,5	10,9
ESD1	7	5	27	21	21	129	129	2,9	1, 4
ESD2	16	13	104	76	75	663	658	69,3	15, 8
ESD3	12	11	43	34	34	238	238	11,1	3,9
ESD4	10	11	39	33	33	384	384	42,3	4,1
STYR	14	15	134	99	100	962	967	201,9	44, 7
CSE	11	11	58	52	51	484	476	32,9	7, 2
PLANET	13	25	131	99	99	1160	1189	696	85, 2
JAY	10	39	179	113	111	1056	946	544,9	60
INTVAR	26	7	503	222	224	2692	2645	1397	294

TABLEAU 5 : Comparaison Espresso2

La figure suivante montre les facteurs temps à machines équivalentes entre notre minimiseur et Espresso2. Les exemples sont classés par complexité croissante (fonction des temps d'éxecution des deux minimiseurs).

La moyenne obtenue sur ces 20 exemples est de 2,3. Ce facteur temps n'est pas fonction de la complexité des exemples mais plutôt de leurs caractéristiques propres. Les méthodes employées dans les deux minimiseurs étant complètement différentes, chacune est plus ou moins bien adaptée aux caractéristiques d'un exemple donné. Ainsi par exemple nous avons pour ESD4 un facteur de 5,1 alors que pour INTVAR, il est de 2,3.



Pour expliquer un tel rapport entre les temps d'Asyl et d'Espresso2, il nous faut regarder d'un peu plus près la méthode employée dans Espresso2.

Résumé de la méthode d'Espresso2 :

1- Eclatement de chaque monôme global (m,I) en x monômes (m,J) où J ne contient qu'un seul des 1 apparaissant dans I. Le nombre de monômes peut être considérablement augmenté.

2- Calcul du complément de chaque fonction simple. Ces compléments vont être nécessaire dans l'étape d'expansion.

3- Expansion de chaque monôme, de tel manière à ce qu'il couvre le plus possible d'autres monômes, et suppression des monômes couverts.

Ceci est réalisé grâce aux calculs de deux matrices pour chaque monôme à agrandir:

-une matrice de blocage de taille : (nombre de variables + nombre de fonctions simples) * nombre de monômes résultants de la complémentation des fonctions simples) permettant de limiter l'expansion du monôme, afin qu'il soit encore inférieur à la fonction général de départ.

-une matrice de couverture de taille : (nombre de variables + nombre de fonctions simples) * (nombre de monômes obtenus en 1 - nombre de monômes déjà supprimés)) permettant de diriger l'expansion, de telle manière à couvrir le plus possible d'autres monômes.

A partir de ces deux matrices, un algorithme complexe est développé pour agrandir le monôme courant (heuristique pour l'ordre d'expansion).

4- Extraction d'une base irrédondante à base d'heuristiques et de calculs matriciels complexes.

5- si première passe : détection des monômes essentiels qui sont mis à part une fois pour toute.

6- Réduction des monômes les uns aprés les autres à leur incouvert minimal (par rapport à tout les autres monômes encore non réduits). Heuristique pour l'ordre de réduction.

7- Boucle sur les étapes 6, 3 et 4 jusqu'à ce qu'il n'y est plus de gain.

8- Essai d'une autre stratégie:

Calcul de nouveaux monômes qui ont une forte chance d'aboutir à un gain. Pour cela: réduction maximale des monômes présents (incouverts par rapport à tous les autres monômes) puis expansion de ces monômes.

Si les monômes étendus qui couvrent plus d'un monôme réduit précédement, isont ajoutés à la base obtenue en 7, et retour en 4.

9- Minimisation du nombre de transistors par retour à des ensembles localement irrédondants.

Si nous regardons l'évolution de la méthode employée dans Espresso2 sur les exemples traités, nous constatons :

- Le nombre de monômes est considérablement augmenté dans la première étape.

Par exemple :

Le nombre de monômes de Duke2, Dist1 et Planet passe respectivement de 87 à 242, 256 à 602 et 131 à 884 dans cette étape.

- Les matrices calculées dans l'étape d'expansion sont de trés grandes tailles (d'autant plus que le nombre de monômes ait été augmenté dans la première étape).

Par exemple :

Pour Duke2, dans la première étape d'expansion pour chaque monôme qui n'est pas supprimé, une matrice de dimension (22+29)*318 = 16218 (318 étant le nombre de monômes des compléments) et une matrice de taille variant entre 51*242 et 51*90 sont calculées.

Ou encore pour misex3, la taille des matrices calculées lors de la première étape d'expansion sont de 28*702 ou varient entre 28*1848 et 28*1200 et cela pour les 1200 monômes résultant de cette étape.

- Dans Espresso2, le nombre de boucle (étapes d'expansion, de réduction et d'extraction de base irrédondante ou dernière stratégie) pour arriver à une solution stable s'échelonne de 3 (pour misex1, misex2 ou mult3 par exemple) à 15 (pour misex3) en passant par 6 (pour dist2) ou 9 (pour intvar).

Il faut noté que lorsqu'un gain est obtenu dans la dernière stratégie (étape 8), la boucle principale (étape 3, 5, 6) est de nouveau effectuée.

Nous avons constaté que l'extraction de base irrédondante employée dans Espresso2 donne des résultats équivalents à celle employée dans Asyl.

Les calculs effectués dans la méthode d'Espresso2 sont loin d'être de faibles complexités.

Le nombre de boucles pour arriver à une solution stable est sur l'ensemble des

exemples traités, nettement supérieur à celui d'Asyl (pour les 20 exemples traités, 89 extraction de bases irrédondantes pour Espresso et 54 pour Asyl).
Dans Espresso2, le nombre de monômes générés au cours de la méthode peut être considérablement augmenté par rapport au nombre de monômes initiaux contrairement au minimiseur que nous proposons.

Devant de telles remarques, la rapidité d'espresso2 par rapport à Asyl ne peut s'expliquer que par une programmation en C basée sur une structure de données permettant des calculs optimisés, chose que nous n'avons pu obtenir par une programmation en PASCAL standard (voir parag.4).

Pour vérifier cette proposition, nous avons comparé notre version en PASCAL de la procédure de détection des monômes essentiels et celle en C d'Espresso2 sur un ensemble d'exemples. L'algorithme de cette procédure est à peu prés le même dans Asyl et Espresso2 (si l'on suprime la détection des monômes stables). De plus cette procédure est une bonne représentation de l'ensemble de notre algorithme puisqu'elle contient, des calculs de produits et de consensus entre monômes, et des tests d'inclusions. Nous arrivons à des rapports variant entre 2 et 3 entre les temps de la version en PASCAL et en C. Nous estimons que l'on pourrait gagner au moins le même rapport sur l'ensemble de notre algorithme par une programmation en C et donc, arriver à des temps inférieurs à ceux d'Espresso2 sur l'ensemble de la minimisation.

Ainsi le démarche originale que nous proposons, tout en étant plus simple et homogène aboutit à des résultats comparables à ceux d'Espresso2. Le nombre de monômes générés au cours de notre méthode est, la plupart du temps, largement inférieur à celui que peut obtenir Espresso2. Notre méthode converge plus rapidement vers un minimum stable. La différence des temps d'exécution est dûe à une utilisation de langages de programmation différents dans les deux minimiseurs.

5.3. COMPARAISON AVEC LE MINIMUM EXACT

Nous avons voulu comparer nos résultats à ceux que pourrait donner un minimiseur exact. Pour cela nous avions à notre disposition un minimiseur basé sur la génération de tous les monômes premiers globaux et extraction de toutes les bases

irrédondantes à partir de l'algorithme de Quine et Mc Cluskey. Nous avons testé ce minimiseur sur les 20 exemples traités. Les résultats obtenus sont donnés dans le tableau 6. Ce tableau donne le nombre de monômes de la base complète et du minimum exact quand ils ont pu être obtenu grâce au minimiseur exact. Les deux colonnes suivantes donne le nombre maximal de monômes générés au cours de la méthode employée dans Asyl et le nombre de monômes finaux que nous obtenons.

NOM	BASE COMPLETE	EXACT	NBMAX ASYL	ASYL
F2	18	8	18	8
MISEX1	28	12	32	12
MISEX2	42	28	30	28
RD53	51	31	32	31
5XP1	390	???	118	63
DUKE2	1044	???	124	86
MISEX3	???	???	1848	715
MULT3	90	30	64	31
DIST1	339	???	256	107
DIST2	444	???	256	139
CARRE6	205	47	89	51
ESD1	38	21	27	21
ESD2	300	???	117	76
ESD3	87	34	43	34
ESD4	177	33	90	33
STYR	619	???	209	99
CSE	160	???	95	52
PLANET	896	???	274	99
JAY	925	???	236	113
INTVAR	???	???	503	222

TABLEAU 6 : Comparaison minimum exact

Pour certains exemples, les temps de minimisation exacte sont du même ordre que ceux du minimiseur proposé. C'est le cas pour F2, MISEX1 et MISEX2 qui

possèdent des bases complètes de petites cardinalités, mais surtout qui possèdent peu de bases irrédondantes. Dès que le nombre de monômes premiers devient un peu plus grand, le nombre de bases irrédondantes est considérablement augmenté et on arrive vite à des temps d'exécution trés grand. Par exemple, le temps pour trouver le minimum de RD53 est de l'ordre de 7 minutes (sur un VAX-780 Unix) alors que sa base complète ne possède que 51 monômes.

Sur ces 20 exemples, le minimum exact n'a pu être trouver que pour 9 d'entre eux (dans un temps raisonnable). Asyl donne le minimum exact pour 7 exemples parmi ces 12. Dans le cas contraire Asyl est trés proche du minimum, respectivement à 1 et 4 monômes pour Mult3 et Carre6. On peut noter que la deuxième alternative (de la minimisation locale, voir tableau 3) donne pour Carre6 48 (à 1 monôme du minimum exact) dans un temps tout à fait raisonnable.
De plus, nous constatons que le nombre de monômes générés au cours de notre méthode reste trés petit par rapport au nombre de monômes de la base complète pour les exemples de fortes complexités.

Dernièrement des tentatives de version exacte de minimisation ont été élaborées dans [DAG85] et [RUD86]. elles sont toujours basées sur une génération de tous les monômes premiers globaux et sur l'extraction de toutes les bases irrédondantes (grâce à l'algorithme de Quine et Mc Cluskey). La génération de la base complète a été optimisée grâce à une limitation du nombre de consensus à effectuer. La génération d'une base irrédondante minimale est une version optimisée de l'algorithme de Quine et Mc Cluskey. Le nombre de bases irrédondantes générées est limité par, premièrement une heuristique pour trouver dés le premier essai une base proche du minimum, et deuxièmement, par un retour arrière limité par le coût de la solution précédente.

Des résultats sont donnés dans [RUD86]. Sur 134 exemples 30 n'ont pu être résolus par cette méthode, parce qu'ils possèdent, soit trop de monômes premiers, soit trop de bases irrédondantes. Nous n'avons pu malheureusement pas pu obtenir ces exemples et donc, comparer ces résultats exact à ceux de notre minimiseur.
Par contre, dans cet article, une comparaison avec Espresso2 est faite pour les 104 résultats exacts obtenus. Le gain obtenu entre la version exacte et Espresso2 est d'à

peu prés 1 pour 1000 monômes alors que le temps d'exécution a été multiplié par un facteur avoisinant 17. Ces versions exactes sont plus des exercices de style que des outils réalistes puisqu'elles ne permettent toujours pas de traiter des exemples de complexité importante et aboutissent à des temps d'exécution beaucoup plus longs.

Mais cela nous permet de comprendre pourquoi nos résultats et ceux d'Espresso sont si proches. Et nous pouvons conclure que comme Espresso2, nous sommes très proche du minimum exact.

L'efficacité de notre minimiseur, par rapport à une méthode exacte, s'explique par un choix judicieux des monômes à hauts potentiels de gain (parmi tous les monômes premiers) dans une approche progressive qui converge ainsi rapidement vers un minimum exact.

DEUXIEME PARTIE

SYNTHESE MULTICOUCHES SUR RESEAUX PROGRAMMABLES FIGES

INTRODUCTION

Nous nous intéressons dans cette deuxième partie à la synthèse de fonctions booléennes sur plusieurs couches de réseaux programmables 'figés', existants dans des bibliothèques du commerce.

L'implémentation de fonctions booléennes, contrôleurs ou logique variée sur réseaux programmables (PLD: Programmable Logic Device) a pris ces dernières années un essor considérable. Pourtant peu d'outils de CAO réalisant une synthèse automatique existe.

Ces circuits sont programmables, c'est à dire qu'ils sont configurables suivant l'application donnée. Cette configuration peut être, suivant la technologie, définitive (fusibles) ou effaçable (EPLD, par ultra-violet par exemple). C'est ce qui fait leur grande souplesse d'utilisation. On peut réaliser à partir d'un ensemble de boîtiers de même type, une large gamme d'applications.
Ces circuits contiennent une partie dite combinatoire (ou logique) et, pour certains, des points de mémorisation.

La partie combinatoire peut être de différents types :

*PLA : l'étage OU et l'étage ET sont programmables (figure 1), mais contrairement à la première partie de cette thèse, leurs dimensions sont figées.

*PAL : ce sont des PLA à étage OU figé (figure 2).

*LCA (Logic Cell Array) qui se présentent comme une matrice de cellules identiques pouvant réaliser de 1 ou 2 fonctions logiques élémentaires (de 3 à 5 variables). Les cellules sont interconnectées par des canaux de routage (figure 3).

Les points de mémorisations sont des bascules (D, RS, ...) activées par une entrée du circuit. Ces circuits rassemblent ainsi toutes les caractéristiques pour la réalisation complète d'un contrôleur tout en permettant d'autres types d'implémentation.

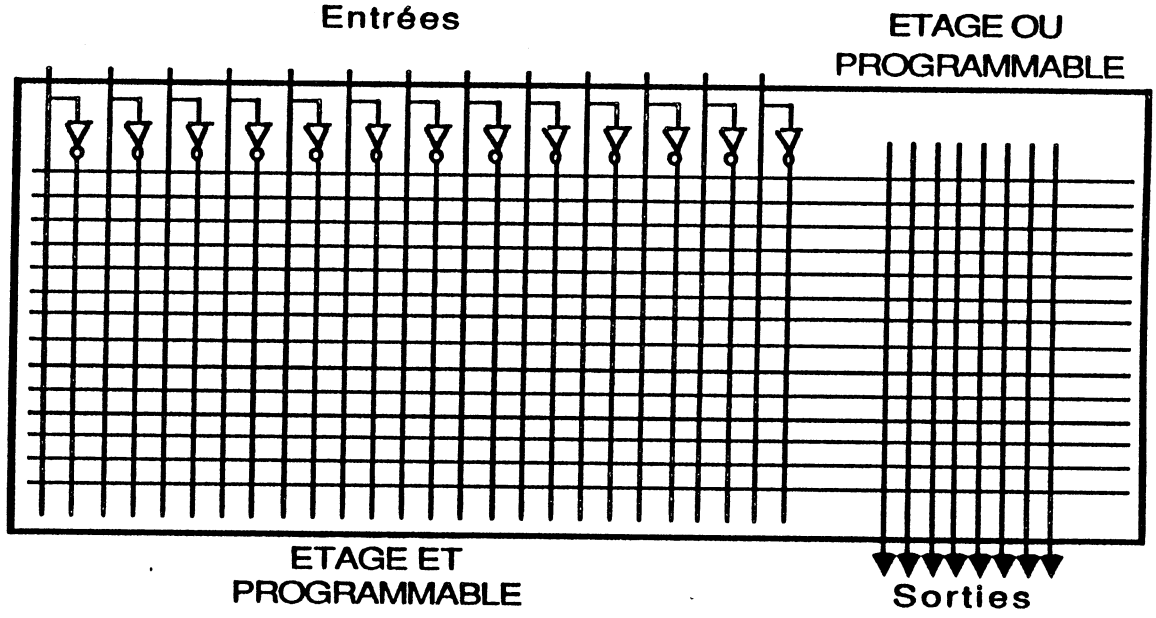


Figure 1 : PLA

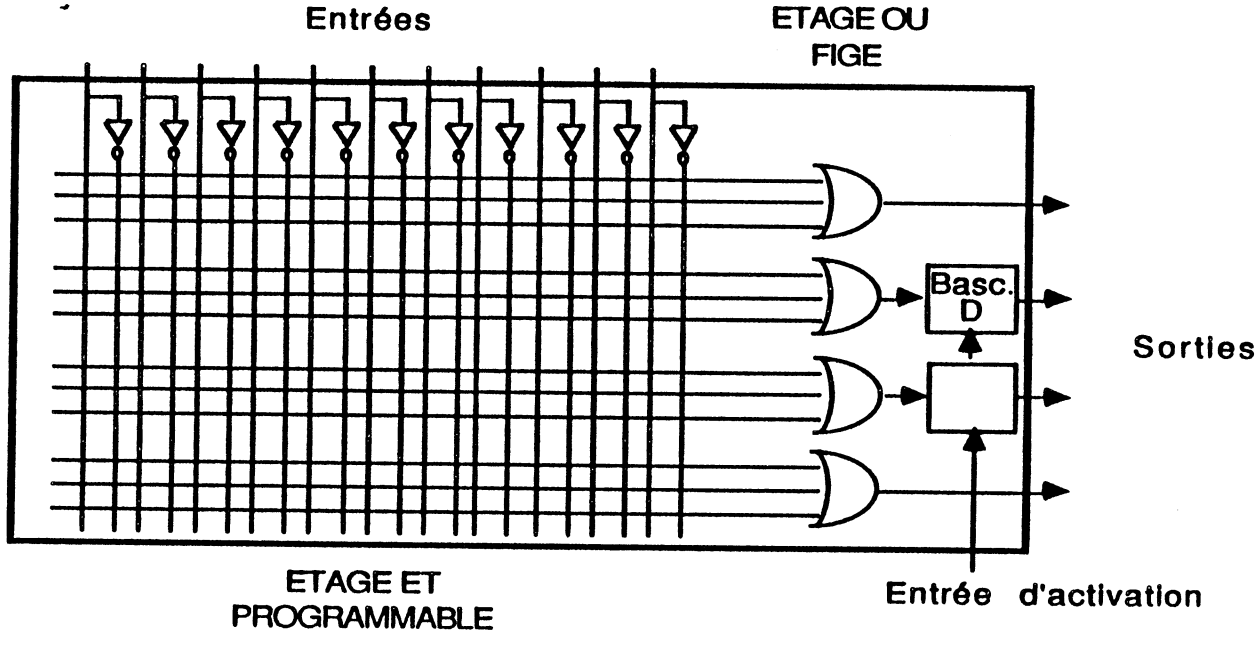


Figure 2 : PAL

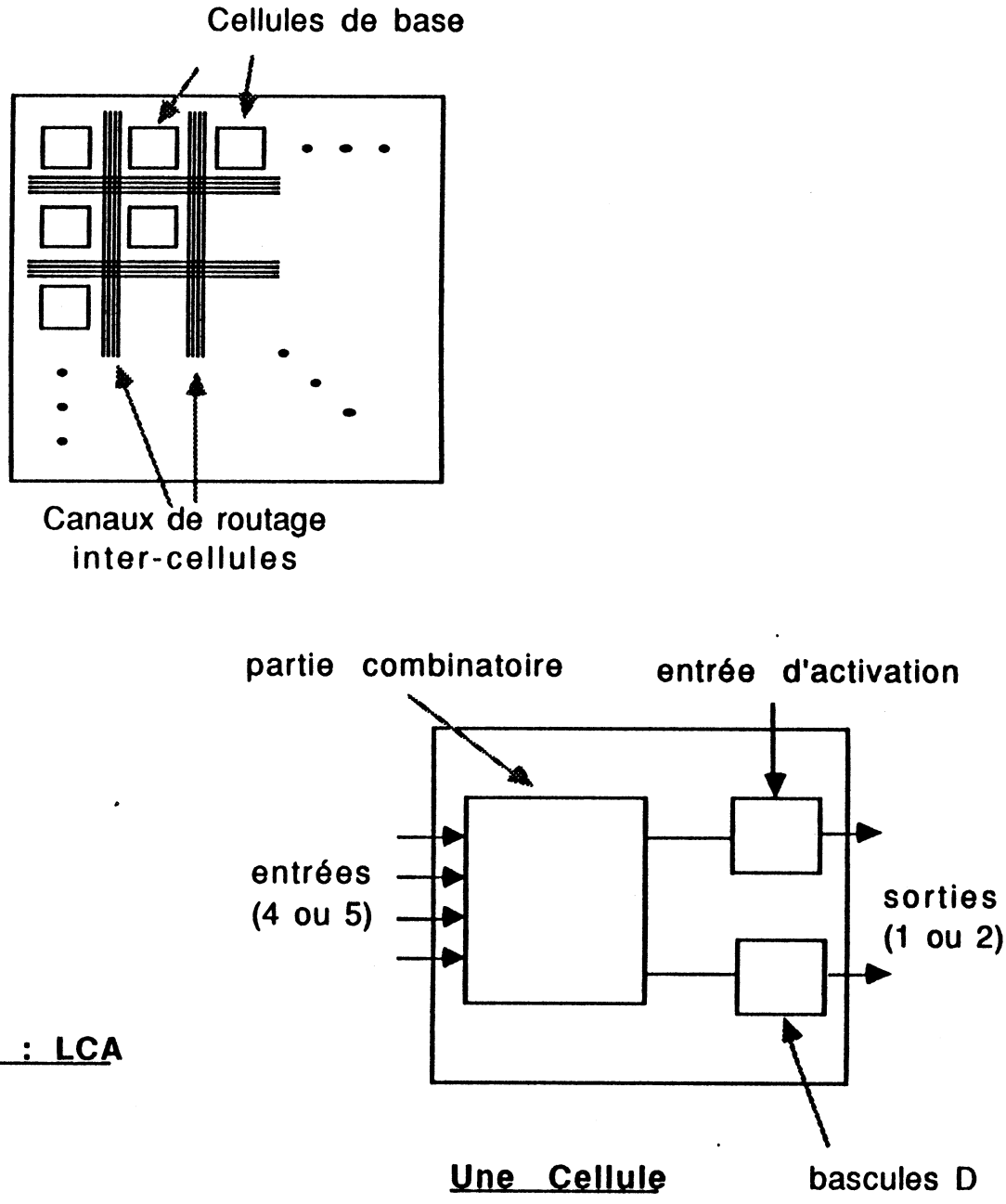


Figure 3 : LCA

Nous nous intéresserons dans la deuxième partie de cette thèse à la synthèse sur PAL.

Nous donnerons tout d'abord les caractéristiques variées de ce type de circuit. Nous définirons ensuite la problématique liée à la synthèse de fonctions booléennes sur cette cible. Puis nous donnerons la méthode générale employée, et détaillerons les différentes étapes de synthèse.

1. CARACTERISTIQUES DES PALS

Nous nous sommes basés sur une des familles les plus utilisées de ce type de circuit : la bibliothèque de [MMI81]. Elle contient une grande variété de boitiers et regroupe à peu prés toutes les caractéristiques que l'on peut trouver chez d'autres constructeurs.

Ces circuits ont tous comme point commun un étage OU figé, chaque ligne de l'étage ET (monôme) étant rattachée à une seule sortie (fonction booléenne simple). Le nombre d'entrées, de sorties, de monômes par sortie varient suivant les boitiers, respectivement de 8 à 20, de 1 à 10, et de 2 à 16.

Les entrées peuvent être sous forme normale ou complémentée.

Les sorties peuvent être (figure 4) :

- mémorisées ou "latchées" en partie ou en totalité.
- rebouclées comme une nouvelle entrée.
- de différentes polarités :
  - polarité normale : la sortie est définie par la somme des monômes programmés sur l'étage ET.
  - polarité complémentée : la sortie est définie par le complément de la somme des monômes programmés sur l'étage ET.
  - polarité complémentée : une des deux formes de polarité précitées peut être choisie.

Ces différents types de sorties peuvent se conjuguer, et il existe par exemple des PALs à sorties sur bascule, à polarité programmable, et rebouclée après la bascule en entrée.

Un boîtier ne possède qu'une entrée pour l'activation de toutes les bascules qu'il contient. La polarité est la même pour toutes les fonctions d'un boîtier. Le nombre de monômes de chaque sortie est en général le même (ou varie très peu : de 1 à 2 monômes) dans un PAL donné.

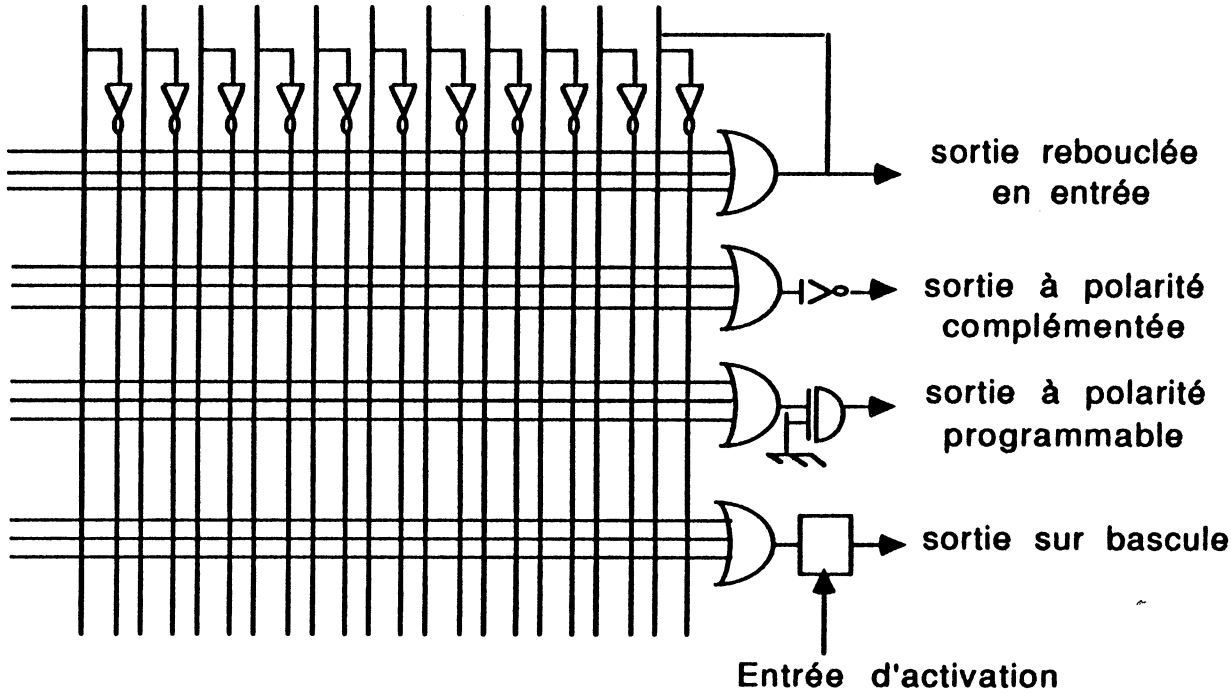


Figure 4 : types de sortie dans les PAL

2. PROBLEMATIQUE

Nous avons à notre disposition une bibliothèque de boîtiers. Chaque boîtier possède des caractéristiques particulières :

- nombre d'entrées
- nombres de sorties avec ou sans bascule, rebouclées ou non en entrée
- polarité des sorties
- nombre de monômes pour chaque sortie

Nous devons réaliser un ensemble de fonctions booléennes simples grâce aux différents types de boîtiers de la bibliothèque. Ces fonctions seront données sous une forme polynômiale. De plus leur spécification sera enrichie par des données concernant leur synchronisation :

-sortie sur bascule ou non

-nom de l'entrée d'activation de chaque bascule.

A partir de ces spécifications le problème consiste à trouver un ensemble minimal de boîtiers de la bibliothèque, permettant l'implémentation physique de

l'ensemble des fonctions données. Un poids pourra éventuellement être associé à chaque boîtier de la bibliothèque (par exemple : nombre de broches, prix...) et un critère secondaire d'optimalité sera alors la somme des poids associés aux boîtiers choisis.

Pour chaque boîtier choisi, devra être donnée la forme polynômiale de chaque fonction réalisée par ce boîtier, ainsi que le brochage (noms affectés aux broches de celui-ci) pour pouvoir effectuer par la suite la programmation de chacun des circuits, et réaliser les interconnexions de ceux-ci sur la carte finale.

3. METHODE GENERALE

Une forme polynômiale d'une fonction f est "plaçable" sur une sortie d'un boîtier B si :

- le nombre de variables apparaissant dans f est inférieur ou égal au nombre d'entrées de B
- il existe une sortie de B telle que le nombre de monômes de cette sortie soit supérieur ou égal au nombre de monômes de f.

Une fonction dont la forme polynômiale est f, et dont la forme polynômiale de son complément est $\neg f$, est réalisable dans un boîtier B si :

- il existe dans B une sortie S possédant la même caractéristique de synchronisation que f (avec ou sans bascule), de polarité normale (resp. complémentée) tel que f (resp. $\neg f$) soit plaçable sur S.

Une fonction peut n'être réalisable sur aucun des boîtiers de la bibliothèque (si le nombre de ses variables est trop grand par exemple), il sera alors nécessaire de la décomposer en sous-fonctions plus petites. Ceci sera fait de telle manière que la nouvelle forme de f et de ses sous-fonctions soient réalisables. C'est ce que nous appellerons la décomposition.

Il en résultera une forme polynômiale de chaque fonction et des sous-fonctions créées (et de leurs compléments). Une fois que toutes les fonctions auront été rendues réalisables dans au moins un boîtier de la bibliothèque, il faudra choisir un ensemble minimal de boîtiers permettant de réaliser toutes les fonctions et donner pour chacun le brochage et les formes polynômiales devant être programmées. C'est ce que nous

appellerons le placement.

Notre méthode se décompose donc en trois étapes successives :

1- Minimisation du nombre de monômes des formes polynômiales de chaque fonction et de son complément.

2- Décomposition des fonctions en sous-fonctions réalisables si nécessaire.

3- Placement définitif des fonctions et sous-fonctions dans un ensemble de boîtiers pris dans la bibliothèque.

4. PREMIERE ETAPE : COMPLEMENTS ET MINIMISATION

Nous voulons obtenir une forme polynômiale 'minimisée' pour chaque fonction et son complément.

4.1. COMPLEMENTATION

Les fonctions que nous voulons complémenter peuvent être phi-booléennes. Soit f une fonction simple phi-booléenne de couverture à 1: f_1, et de couverture à ϕ f_ϕ. Pour obtenir le complément de f, nous calculerons le complément de f_1 ($\neg$ f_1) grâce à la méthode décrite dans la première partie au chapitre 2.2.3. Le complément de f sera alors la fonction phi-booléenne de couverture à 1: $\neg$ f_1, et de couverture à ϕ f_ϕ.

4.2. MINIMISATION

Une fois le calcul du complément de chaque fonction effectué, il faut trouver pour chacune d'elles et leurs compléments, une forme polynômiale telle que la fonction ait le plus de chance d'être réalisable. Il faudra pour cela trouver une forme polynômiale possédant un nombre minimal de monômes et de variables.

Dans un PAL, une ligne de l'étage ET (correspondant à un monôme) ne peut être commune à plusieurs fonctions, c'est à dire que si un monôme apparaît dans plusieurs fonctions, même si ces fonctions sont placées sur le même boîtier, le monôme devra être répliqué.

Nous obtiendrons donc indépendamment pour chaque fonction (et son complément) une base irrédondante minimale (cf. première partie). On peut démontrer

que toutes les bases d'une fonction possédent un nombre minimal de variables.

Pour ce faire nous trouverons une base quelconque en appliquant sur la borne supérieure de la fonction, les méthodes décrites dans la première partie au chapitre 3.2.1 . Nous extrairons ensuite une base irrédondante s'approchant du minimum grâce à l'algorithme donné dans la première partie au chapitre 3.4.2. Cette méthode à base d'heuristiques est largement suffisante par rapport à une méthode exacte qui consisterait à trouver la base complète (première partie chapitre 3.2.2.) et en extraire toutes les bases irrédondantes. La méthode exacte augmente considérablement le temps de calcul pour un gain de monômes inexistant ou minime. De plus, elle a rarement une répercussion sur le nombre de PAL qu'il faudra finalement utiliser.

5. DEUXIEME ETAPE : LA DECOMPOSITION

La décomposition va consister à redéfinir les fonctions qui ne peuvent être réalisées directement grâce aux boîtiers de la bibliothèque donnée. Ces fonctions devront être décomposées en sous-fonctions plus petites qui pourront être réalisées. La nouvelle forme de la fonction et de son complément, qui dépendra alors de ces sous-fonctions, devra alors être réalisable par au moins un élément de la bibliothèque.

Une fonction ne peut être réalisée par aucun des boîtiers d'une bibliothèque dans les cas suivants :

- aucun boîtier ne possède les caractéristiques de synchronisation (bascule) convenant à la fonction. Dans ce cas on ne pourra réaliser la fonction quelle que soit sa transformation, il faudra enrichir la bibliothèque.

- aucun boîtier ne possède, pour une polarité normale (resp. complémentée), une sortie (dont la spécification de synchronisation correspond à celui de la fonction) qui comporte un nombre de monômes supérieur ou égal à celui de la fonction (resp. de son complément).

- aucun boîtier ne possède, pour une polarité normale (resp. complémentée), un nombre d'entrées supérieur ou égal au nombre de variables de la fonction (resp. de son complément).

Dans ces deux derniers cas, il est possible de décomposer la fonction (ou son complément) en sous-fonctions possédant soit moins de variables, soit moins de

monômes (soit les deux), et pouvant ainsi être réalisées.

Exemple :

Soit une fonction f de 5 variables et de forme minimisée à 10 monômes. Supposons que la bibliothèque ne possède qu'un PAL à 10 entrées et 3 sorties de 6 monômes et de polarité normale. Ce boîtier possède assez d'entrées mais ses sorties possèdent un nombre insuffisant de monômes.

Soit $f = m_1 + m_2 + \ldots + m_{10}$

f peut être décomposée en 2 sous-fonctions sf1 et sf2 :

$f = sf_1 + sf_2$

$sf_1 = m_1 + \ldots + m_6$

$sf_2 = m_7 + \ldots + m_{10}$

f, sf_1 et sf_2 peuvent maintenant être placées sur le boîtier.

Nous pouvons remarquer que pour toutes les sous-fonctions créées, on pourra réaliser au choix, la fonction ou son complément puisque les sous-fonctions seront des entrées de la fonction 'mère' (les entrées des boitiers peuvent être sous forme normale ou complémentée). Les sous-fonctions seront non mémorisées (sans bascule), la fonction 'mère' gardera ses spécifications initiales (avec ou sans bascule).

Il existe beaucoup de façons pour décomposer une fonction. Il faut dans cette décomposition tenir compte d'un critère important : le temps de réponse des sorties. Celui-ci est défini par le temps maximum qui peut s'écouler entre le moment où les entrées de la sortie changent d'états ,et le moment où l'on est certain que la sortie est stabilisée.
Pour chaque boîtier ce temps est fixe et donné par le constructeur.
En général, pour un concepteur de carte, le chemin critique est un critère primordial. Nous essairons donc de minimiser le chemin critique de l'ensemble des fonctions, c'est à dire le temps de réponse de la fonction la plus lente.

Quand une fonction est décomposée en sous-fonctions le temps de réponse de la sortie augmente et peut varier suivant la décomposition. Pour cette raison, nous ne décomposerons que les fonctions qui ne peuvent être réalisées sans décomposition et

nous tiendrons compte de ce critère important quand la décomposition est obligatoire.

On peut pourtant trouver des exemples où une décomposition qui n'est pas nécessaire apporte un gain de boîtier.

Exemple :

Soient deux fonctions de 5 mêmes variables :

f_1 : 4 monômes,

f_2 : 8 monômes.

Supposons que la bibliothèque possède deux types de PAL de polarité normale :

Type 1 : 10 entrées et 4 sorties de 4 monômes,

Type 2 : 10 entrées et 2 sorties de 8 monômes.

f_1 peut être réalisée par un boîtier de type 1.

f_2 peut être réalisée par un boîtier de type 2.

Si on décompose f2 :

$f_2 = sf_1 + sf_2$

$sf_1 = m_1 + \ldots + m_4$

$sf_2 = m_5 + \ldots + m_8$

Les 4 fonctions f_1, f_2, sf_1, sf_2 peuvent être réalisées sur un unique boîtier de type 1.

Mais le temps de réponse de f_2 a été multiplié par deux.

Nous supposerons par la suite que tous les PALs ont le même temps de réponse T.

5.1. APPROCHE FONCTION PAR FONCTION

Nous allons donner une première approche du problème qui consisterait à effectuer une décomposition de chaque fonction indépendament des autres. Cette approche, prototypée en Prolog laisse espérer de bons résultats dans des temps raisonnables.

Notre méthode est composée de deux étapes:

* Dans la première étape, nous réécrivons les fonctions non-réalisables, sous

une forme dépendant de sous-fonctions "plus petites". Nous travaillons de façon descendante et réappliquons le processus si nécessaire sur les sous-fonctions, jusqu'à obtenir un ensemble de fonctions toutes réalisables. Nous distinguons certains cas particuliers qui peuvent être traités très simplement de manière efficace :

- problème de polarité (seulement ¬f réalisable)
- problème de bascule (seulement f sans bascule réalisable)
- diminution du nombre de monômes seulement

* Dans une deuxième étape, nous ferons un bilan plus global des décompositions nécessaires, et optimiserons le nombre de sous-fonctions en pratiquant la "réinjection". Celle-ci consiste à examiner de façon ascendante l'arbre de décomposition, et à en minimiser la profondeur en remplaçant des sous-fonctions à un certain niveau par leur expression donnée dans les couches inférieures. Nous minimiserons ainsi le nombre de fonctions créées et le temps de réponse des fonctions.

5.1.1. PROBLEME DE POLARITE

Considérons le cas où une fonction f n'est réalisable sur aucun des boîtiers d'une bibliothèque mais où ¬f est réalisable. Ceci peut arriver si la forme polynômiale de ¬ f (resp. f) est 'plaçable' seulement sur un boîtier de polarité normale (resp. complémentée).

Pour réaliser f, il suffira simplement de réaliser ¬f et de réecrire f = ¬f' avec f' = ¬f.

f possède alors un seul monôme d'une variable, cette nouvelle écriture de f est donc forcément réalisable. On se sert d'un inverseur d'entrée pour inverser ¬f. Le temps de réponse de f devient alors égal à 2 * T.

Cette méthode très simple permet de réaliser une fonction en minimisant, le nombre de nouvelles fonctions (une seule) et son temps de réponse.

Algorithme :

Décomp-polarité (f) ;

début

retourner f = ¬f' et ¬f = f'

retourner f'= ancienne forme de ¬f.

fin.

5.1.2. PROBLEME DE BASCULE

Considérons le cas où une fonction f qui doit être mémorisée sur une bascule, n'est réalisable que sans bascule. C'est à dire qu'il existe au moins un boîtier vérifiant toutes les spécifications de la fonction, sauf celle concernant la synchronisation (sans bascule). Dans ce cas, il suffit de réaliser f' (sans bascule) correspondant à la forme polynômiale de f, et de réecrire f = f'.
f devient ainsi réalisable sur un boîtier avec bascule puisqu'elle ne comporte plus qu'un monôme à une variable.

Le temps de réponse de f devient égal à 2 * T. Comme dans le cas précédent (problème de polarité), cette méthode minimise le nombre de nouvelles fonctions créées et le temps de réponse de la fonction.

Algorithme :

Décomp-bascule (f) ;
début
retourner f = f' et ¬f = ¬f'
retourner f' = ancienne forme de f et ¬f' = ancienne forme de ¬f
(avec spécification de f': sans bascule)
fin.

5.1.3. DIMINUTION DU NOMBRE DE MONOMES

Nous allons traiter le cas où une fonction n'est pas réalisable, parce que le nombre de monômes de sa forme polynômiale et de celle de son complément sont trop grand. En revanche, le nombre de variable ne l'est pas, il existe au moins un boitier comportant assez d'entrées. Comme nous allons le voir, ce cas peut être résolu simplement et efficacement. De plus, ce cas est très fréquent dans la pratique, le nombre d'entrées des boitiers (variant entre 8 et 20) étant souvent suffisant par rapport au nombre de monômes (variant entre 2 et 16).

Soit f une fonction telle qu'il n'existe aucun boîtier de la bibliothèque possédant une sortie ayant un nombre de monômes suffisant pour réaliser f (ou son complément). Mais supposons qu'il existe un boîtier comportant un nombre d'entrées supérieur au nombre de variables de f. Le problème consiste à trouver une décomposition de f, de

manière à utiliser un nombre minimum de boîtiers pour sa réalisation et à minimiser le temps de réponse de la sortie correspondante.

Nous ne pouvons évidemment pas essayer toutes les décompositions possibles et trouver les placements optimaux qui en découleraient. Nous avons donc élaboré une méthode qui essaye de s'approcher d'une solution minimale sans pour cela tomber dans une trop grande complexité.

Méthode :

1- Nous choisissons entre la fonction f et son complément, celle qui possède le moins de monômes. Soit ~f cette fonction.

2- Nous déterminons le boîtier qui possède le plus de monômes sur l'ensemble de ses sorties, tout en ayant un nombre de variables suffisant pour ~f et tel que ses sorties ne soient pas sur des bascules. Supposons pour simplifier que le nombre de monômes pour chaque sortie de ce boîtier soit égal à t.

3- Décomposer ~f :

Soit $\sim f = m_1 + \ldots + m_k$

~f devient $\sim f = sf_1 + \ldots + sf_j$

avec $sf_1 = m_1 + \ldots + m_t$

$sf_2 = m_{t+1} + \ldots + m_{2*t}$

...

$sf_j = m_l + \ldots + m_k$ avec $k - l + 1 \leq t$

Nous avons alors le complément de ~f qui se réecrit :

$\neg(\sim f) = (\neg\, sf_1) . \ldots . (\neg\, sf_j)$

Nous savons que les sous-fonctions sf_i sont réalisables.

4. Si f est réalisable directement grâce à la nouvelle forme de ~f.

Supposons ~f = f , il existe alors soit un boîtier de polarité normale ayant au moins j entrées et possédant une sortie ayant au moins j monômes, soit un boîtier de polarité complémentée ayant au moins j variables (dans les deux cas respectant la

caractéristique de synchronisation de f). Le cas est similaire avec ~f = ¬f.
f et les sf_i sont alors toutes réalisables grâce aux boîtiers de la bibliothèque, nous avons fini.

Sinon deux cas se présentent :

- soit nous pouvons réaliser f en utilisant une des deux méthodes données précedemment (5.1.1. et 5.1.2).
- soit il n'existe pas de boîtier ayant au moins j entrées. Ce cas sera traiter dans le chapitre suivant.

Nous réappliquerons donc la décomposition (traitant tout les cas) sur la nouvelle écriture de ~f.

Exemple :

Soit une fonction f et son complément de 5 variables respectivement de 11 monômes et 14 monômes. Soit une bibliothèque possédant un seul type de boîtier à 10 variables, et 10 sorties de 3 monômes à polarité normale.
f sera décomposé ainsi :

$$f = sf_1 + sf_2 + sf_3 + sf4$$

$$sf_1 = m_1 + m_2 + m_3$$

$$sf_2 = m_4 + m_5 + m_6$$

$$sf_3 = m_7 + m_8 + m_9$$

$$sf_4 = m_{10} + m_{11}$$

et $\neg f = \neg sf_1 \,.\, \neg sf_2 \,.\, \neg sf_3 \,.\, \neg sf4$

$sf_1, \ldots, sf_4$ sont réalisables sur le boîtier donné.

Par contre f n'est pas encore réalisable puisqu'elle possède 4 monômes. Nous réalisons ¬f, qui ne possède qu'un monôme et 4 variables, et pour pouvoir réaliser f nous réecrivons : $f = \neg f'$ avec $f' = \neg f$.
Ainsi f ne possède maintenant plus qu'un monôme et une variable.

Justification :

Le découpage reste très simple. Le nombre de boîtiers utilisés grâce à un tel découpage sera minimal puisque toutes les sous-fonctions peuvent être réalisées grâce à un même type de boîtier possédant le plus possible de monômes. Les sous-fonctions pourront être regroupées au maximun dans ce type de boitier puisqu'elles dépendent toutes d'un sous-ensemble des variables de la fonction de départ.

Le temps de réponse de la sortie correspondant à f est minimisé, il sera dans le pire cas égal à 3 * T (f n'est pas encore réalisable aprés cet décomposition), et dans le meilleur des cas à 2 * T (traversée du boitier réalisant f et du ou des boitiers réalisant les sous-fonctions (en parallèle)).

Algorithme :

Decomp-monômes (f) ;
début
déterminer (~f) ;
déterminer (x) ;
retourner sf_i ;
réecrire $f= \Sigma sf_i$ et $\neg f = \prod \neg sf_i$;
Décomposition (f);
{ Décomposition est la procédure générale de décompositon qui regroupe les différents cas de décomposition, elle sera explicitée plus loin}
fin.

5.1.4. DIMINUTION DU NOMBRE DE VARIABLES

Nous allons traiter le cas général où une fonction n'est pas réalisable parce que sa forme polynômiale et celle de son complément dépendent de trop de variables et éventuellement de trop de monômes. C'est à dire qu'il n'existe aucun boîtier dans la bibliothèque possédant un nombre suffisant d'entrées et éventuellement de monômes par sortie, pour que la fonction ou son complément puisse y être placé.

Nous allons pour cela nous servir des notions établies dans [BRA82] de division

algébrique et de noyaux d'une fonction booléenne. Ces notions sont à la base des problèmes de synthèse logique multicouches [BRA84b], [BRA87]. Elles s'adaptent très bien à notre problème.

Nous allons rappeler les principales définitions liées à la factorisation.

5.1.4.1. DIVISION ALGEBRIQUE ET NOYAUX

Soient f et g les expressions polynômiales de deux fonctions booléennes.

Définition du produit algébrique:

Le produit algébrique f.g existe et il est unique si f et g dépendent d'ensembles de variables disjoints. Il est défini par :

$f.g = \sum m_i . m_j$ pour $i = 1, \ldots, n$ et $j = 1, \ldots, m$

avec $f = \sum m_i$ pour $i = 1, \ldots, n$ et $g = \sum m_j$ pour $j = 1, \ldots, m$

Exemple :

$f = a + b$

$g = c + \neg d.e$

$f.g = a.c + b.c + a.\neg d.e + b.\neg d.e$

Définition de la division algébrique:

g est un diviseur algébrique de f si :

$f = g. h + r$

où g. h est un produit algébrique,

h est la plus grande expression possible non nulle, c'est à dire qu'il n'existe pas h' tel que $h < h'$ et $f = g. h' + r$

et r est une expression polynômiale.

Le quotient h (noté aussi f/g) et le reste de cette division sont uniques.

Si $f = g. h$ avec les mêmes conditions g est un facteur algébrique de f.

Exemple :

f = a.b + a.d + c.b + d.c + e

g = a + c

g est un diviseur de f, le quotient est f/g = b + d et le reste est r = e.

Définition d'une expression libre:

Une expression polynômiale f est libre s'il n'existe pas de monôme m (avec m différent de 1) qui soit un facteur algébrique de f.
C'est à dire f = m. g .

Exemple :

a.b + a.c n'est pas libre,

a.b.c n'est pas libre,

a.b + c est libre.

Définition d'un noyau:

k est un noyau d'une expression polynômiale f si k = f/m, où m est un monôme et k est une expression polynômiale libre. m est appelé le co-noyau de k.

Remarque : un noyau comporte forcément plus d'un monôme puisque un monôme n'est pas une expression libre. Si f est une expression libre, f est un noyau d'elle-même : f = f/1.
Les noyaux d'une expression polynômiale représentent toutes les factorisations maximales (en nombre de variables) possibles.

Ces notions vont nous permettre de décomposer une fonction de manière simple et efficace en sous-fonctions possédant moins de variables et moins de monômes.
En effet, soit la division d'une expression polynômiale f par un de ses noyaux k :

f = q. k + r

Le noyau k et le quotient q possèdent moins de variables que f.
Le reste r, le noyau k et le quotient q possèdent moins de monômes que f puisque NBMON(f) = NBMON(q) * NBMON(k) + NBMON(r) où NBMON est le nombre de monômes d'une expression polynômiale.

Le nombre de variables de r n'est pas forcément plus petit que celui de f, mais il aura tendance à diminuer avec le nombre de monômes.

Les algorithmes de division algébrique et de recherche de noyaux [BRA87] suivants se sont avérés performants. L'algorithme de division algébrique est de l'ordre de n * log(n) (où n est la somme du nombre de monômes de l'expression et de celui du diviseur), et le nombre de noyaux d'une expression est souvent petit.

Définition d'un litéral :
Un litéral est une variable booléenne simple ou son complément.
a et ¬a sont deux litéraux distincts.

Algorithme de division algébrique :

```
alg-div ( f , g , q, r) ;
début
    U <- restriction de f aux litéraux de g;
    V <- restriction de f aux litéraux qui n'apparaissent pas dans g;
                        { u_j.v_j est le j^ieme monôme de f}
    V_i <- {v_j élement de V tel que u_j = g_i} ;
    q <- ∩ V_i ;   { q est le quotient de la division}
    r <- f - (g. q) ;        { r est le reste de la division}
fin
```

Algorithme de recherche des noyaux d'une expression :

```
noyaux ( f );
début
    m <- le plus gros monôme facteur algébrique de f;
    noyaux ( f ) <- noyaux1 ( 0, f/m );
    Si m = 1 alors                    {f est libre}
                noyaux(f) <- noyaux(f) ∪ {f}
    finsi
fin.
```

noyaux1(j, f);
début
    noyaux1(j, f) <- {f};
    pour i= j+1 à n faire
        début
            si l_i apparait dans plus d'un monôme de f alors
                début
                m <- plus gros monôme facteur algébrique de g/l_i;
                si $l_K \notin$ m pour tout k ≤ i alors
                début
                noyaux1(j,f) <- noyaux1(j,f) ∪ noyaux1(i, g/(l_i.m)
                fin
                fin
        fin
fin

l_i est un litéral de f.

A partir de ces deux procédures de base, nous avons élaboré une méthode de décomposition permettant de résoudre notre problème.

5.1.4.2. METHODE DE DECOMPOSITION

Nous choisissons entre f et ¬f, la forme ~f qui possède le moins de monômes, le nombre de variables étant en général le même. Nous calculons ensuite tous les noyaux de ~f.

Deux cas se présentent alors :

A) Il existe au moins un noyau différent de ~f, il y a donc dans ~f une factorisation possible.

Nous choisissons alors le noyau le plus "intéressant" afin de diviser ~f.
~f sera alors réécrite q. k + r, où q est le quotient (~f/k) et r le reste de la division

algébrique de ~f par k.

Un noyau sera intéressant s'il permet la réalisation de q (quotient), k et r (reste) le plus rapidement possible, c'est à dire en ayant besoin d'effectuer un minimum de nouvelles décompositions. On limitera ainsi le nombre de sous-fonctions à créer pour la réalisation de q, k et r.

Nous voulons aussi minimiser le temps de réponse de la fonction, c'est à dire la longueur de la branche la plus longue dans l'arbre de décomposition. Nous choisirons donc k, de telle façon à minimiser le maximum du nombre de variables de k , de q et de r. Le nombre de monômes des sous-fonctions sera un critère secondaire puisque nous avons vu que la diminution du nombre de monômes était un cas rapidement réalisable (cf. 5.1.3).

On recommence sur q, k et r la décomposition si nécessaire.

Exemple :

f = v1.v2.¬v3 + v4.¬v5.v6 + v1.v2.¬v4.v6 + v6.v7.¬v8.v9.¬v10

Les noyaux de f sont : ¬v3 + ¬v4.v6 , v4.¬v5 + v1.v2.¬v4 + v7.¬v8.v9.¬v10, f.

On choisit : k = ¬v3 + ¬v4.v6 puisque c'est lui qui minimise le maximum du nombre de variables du quotient (2) et du reste (7).

f = q. k + r

q = v1.v2

r = v4.¬v5.v6 + v6.v7.¬v8.v9.¬v10

Algorithme :

```
factor(f, k); {décomposition par division algébrique de f par un noyau k}
début
    Alg-div (f, k, q, r);
    retourner ~f = q. k + r
    retourner ¬~f = ¬q.¬r + ¬k.¬r
    decomposition ( k );
    decompostion ( q );
    decomposition ( r );
fin.
```

B) Il n'y a pas de factorisation possible dans ~f .

Chaque variable de ~f apparaît au maximum deux fois : une fois sous forme normale, une fois sous forme complémentée (puisqu'il n'y a pas de factorisation possible).

Nous dirons qu'une variable est monoforme dans une fonction si elle apparaît dans cette fonction sous une seule forme, sinon elle est dite biforme.

Dans le cas présent une variable monoforme apparaît dans un seul monôme de ~f.

Nous distinguons deux cas :

1) ~f est composée d'un seul monôme :

Si x est le nombre maximum d'entrées parmi les boitiers de la bibliothèque (possédant une sortie sans bascule), on décompose ~f en j sous-fonctions possédant au maximum x variables.

Exemple :

~f = v1.v2.¬v3.¬v4.v5.v6.v7

Supposons x = 4, ~f devient :

~f = sf1. sf2

sf1 = v1.v2.¬v3.¬v4 et sf2 = v5.v6.v7

Algorithme :

Décomp-un-monôme (f);

début

déterminer (x);

retourner (sf_i);

réecrire f = $\prod sf_i$;

Décomposition (f);

fin.

2) ~f contient plus d'un monôme :

Nous allons décomposer la fonction en deux sous-fonctions de telle manière qu'elles soient le plus rapidement réalisables. La diminution du nombre de monômes étant un cas facile à traiter (sans beaucoup de décompositions successives, cf. 5.1.3), nous essaierons de minimiser le maximum du nombre de variables des deux sous-fonctions, le nombre de monômes étant un critère secondaire.

Nous allons diviser l'ensemble des monômes de ~f en deux sous-ensembles de manière à minimiser en premier lieu le maximum du nombres de variables de ceux-ci, et en second lieu le maximum du nombres de monômes. Nous réappliquons la décomposition si néccessaire sur ces deux sous-fonctions.

a) Toutes les variables de f sont monoformes. Chaque variable apparaît dans un seul monôme de f.

L'algorithme suivant aboutit au résultat escompté.

Algorithme :

Classer dans f les monômes par ordre de nombre de variables décroissant.

```
Decomp-monoforme (f, sf1, sf2);
début
Si f ≠ Ø alors
    début
        prendre premier monôme de f;
        si NB-VAR (sf1) > NB-VAR (sf2) alors
            sf2 <- sf2 ∪ {m}
        sinon si NB-VAR (sf2) > NB-VAR (sf1) alors
                sf1 <- sf1 ∪ {m}
            sinon si NB-MON (sf1) > NB-MON (sf2) alors
                    sf2 <- sf2 ∪ {m}
                sinon sf1 <- sf1 ∪ {m};
        decomp-monoforme (f - {m}, sf1, sf2)
    fin
sinon
    retourner sf1 et sf2
fin.
```

b) Il existe au moins une variable biforme dans f.

Il est à remarquer que les variables biformes de f apparaissent exactement dans deux monômes de f. Il nous faut donc regrouper, dans chacun des deux

sous-ensembles, des monômes comportant le plus possible les mêmes variables biformes.

Pour cela, nous calculons pour chaque couple de monômes ou apparait au moins une même variable biforme, le nombre de mêmes variables biformes dans ce couple. Nous classons l'ensemble de tous ces couples et tous les monômes par ordre de nombre de même variables biformes décroissants (nul pour les monômes), puis par nombre de variables total décroissant.

Nous appliquons l'algorithme précédent jusqu'à ce que tous les monômes de f ait été mis dans sf1 ou sf2. On pourra mettre soit un couple de monômes, soit un seul monôme dans une sous-fonction, on mettra à jour la liste des couples et monômes à ce moment là. Nous favorisons ainsi les couples qui possèdent beaucoup de mêmes variables biformes.

Exemple :

~f = ¬v1.¬v2.v3.v4 + v1.v2.¬v3.v5.v6 + ¬v6.v7.v8 + ¬v7.¬v8 + ¬v5.v9

Les couples de monômes ou monômes avec leur nombre de variables biformes identiques et leur nombre de variable total respectifs sont :

1- (¬v1.¬v2.v3.v4 , v1.v2.¬v3.v5.v6) : 3, 6

2- (¬v6.v7.v8 , ¬v7.¬v8) : 2, 3

3- (v1.v2.¬v3.v5.v6 , ¬v6.v7.v8) : 1, 7

4- (v1.v2.¬v3.v5.v6 , ¬v5.v9) : 1, 6

5- v1.v2.¬v3.v5.v6 : 0, 5

6- ¬v1.¬v2.v3.v4 : 0, 4

7- ¬v6.v7.v8 : 0, 3

8- ¬v7.¬v8 : 0, 2

9- ¬v5.v9 : 0, 2

L'ajgorithme se déroule alors ainsi :

sf1 <- ¬v1.¬v2.v3.v4 + v1.v2.¬v3.v5.v6

1, 3, 4, 5 et 6 sont supprimés.

sf2 <- ¬v6.v7.v8 + ¬v7.¬v8

2, 4, 7 et 8 sont supprimés.

puis sf2 <- v7.v8 + ¬v7.¬v8 + v9.¬v5

Cet algorithme, tout en restant de faible complexité, donne de bons résultats. Dans la pratique, le nombre de monômes ainsi que le nombre de couples contenant des variables biformes de ~f sont souvent petits.

Algorithme :

Decomp-biforme (f);

début

Classer les couple de monômes, contenant au moins une même variable biforme, et les monômes, suivant nombre de même variables biformes puis nombre de variables total décroissant;

Décomp-monoforme (f, sf1, sf2);

{en tenant compte de l'ordre précédement défini et en mettant à jour la liste des couples et des monômes}

Décomposition (sf1);

Décomposition (sf2);

fin.

Résumé de l'algorithme du cas général :

Décomp-cas-général(f);

début

si NB-MON(f) > NB-MON(¬f) alors ~f <- ¬f sinon ~f <- f ;

Choisir-noyau (~f, k); { choisi le noyau le plus intéressant }

si k≠~f alors

factor(~f, k)

sinon { il n'y a pas de factorisation }

si ~f = $\prod v_i$ alors { f est réduit à un monôme }

Décomp-un-monôme (~f)

sinon Décomp-biforme (~f);

fin.

On peut montrer que si f est une forme polynômiale minimisée (base irrédondante) alors un noyau k de f , le reste et le quotient de la division de f par k sont encore des bases irrédondantes. Il sera donc inutile d'essayer de minimiser les sous-fonctions résultant d'une telle décomposition.

Exemple d'une décomposition complète :

$f = v1.v2.\neg v3 + v4.\neg v5.v6 + v1.v2.\neg v4.v6 + v6.v7.\neg v8.v9.\neg v10$

Supposons que le nombre maximum d'entrée et de monômes par sortie dans la bibliothèque soient respectivement égal à 3 et 2.

Nous sommes dans le cas général :

Les noyaux de f sont :

$\neg v3 + \neg v4.v6$, $v4.\neg v5 + v1.v2.\neg v4 + v7.\neg v8.v9.\neg v10$, f.

On choisit :

$k = \neg v3 + \neg v4.v6$

$f = q.k + r$

$q = v1.v2$

$r = v4.\neg v5.v6 + v6.v7.\neg v8.v9.\neg v10$

f, k et q sont réalisables.

On recommence sur r.

r possède un seul noyau : $k' = v4.\neg v5 + v7.\neg v8.v9.\neg v10$

$r = q'. k'$

$q' = v6$

k' n'est toujours pas réalisable, et il n'y a plus de factorisation possible.

On applique Décomp-biforme sur k' :

$k' = sf1 + sf2$

$sf1 = v7.\neg v8.v9.\neg v10$

$sf2 = v4.\neg v5$

sf1 n'est pas réalisable, on applique Décomp-un-monôme:

on réecrit $sf1 = sf3. sf4$

sf3 = v7.¬v8.v9

sf4 = ¬v10

Toutes les fonctions et sous-fonctions sont maintenant réalisables.

Résumé de l'algorithme de décomposition :

Décomposition (f);

début

si f est réalisable alors retourner f

sinon si seulement ¬f est réalisable alors

Décomp-polarité (f)

sinon si seulement f sans bascule est réalisable alors

Décomp-bascule (f)

sinon si il existe un boitier possédant un nombre d'entrée suffisant pour f alors

Décomp-monômes (f)

sinon Décomp-général (f)

fin.

5.1.5. LA REINJECTION

La méthode précedente peut introduire des sous-fonctions inutiles. Dans l'exemple précédent q', sf2 et sf4 sont inutiles :

r = v6.k', k'= sf1 + v4.¬v5, sf1 = sf3. ¬v10 sont réalisables.

Nous essaierons donc une fois la décomposition réalisée de supprimer un maximum de sous-fonctions en faisant réapparaitre leurs formes dans les fonctions de niveau supérieur.

Cette réinjection peut-être plus moins efficace suivant l'ordre dans laquelle on la fait, et les choix successifs de décomposition.

5.1.5.1. HEURISTIQUE EN CAS DE FACTORISATION

Dans le cas de décomposition par factorisation, nous avons établi une

heuristique pour faciliter au maximum cette réinjection.

Nous avons souvent constaté que le quotient de la division algébrique par un noyau était réduit à un seul monôme : le co-noyau de ce noyau quand il est unique. Il est donc, souvent et facilement réinjectable. De plus, souvent le noyau est vite réalisable (il possède moins de variables que la fonction de départ), alors que le reste a besoin d'une nouvelle décomposition.

Notre heuristique consiste donc à favoriser la réinjection du reste, et si possible, la réinjection du quotient.

Pour cela nous allons favoriser, dans les décompositions successives des restes, les noyaux dont les co-noyaux dépendent le plus possible des mêmes variables. Cela peut s'obtenir très simplement en classant les variables suivant leur occurence dans les co-noyaux des précédentes décompositions, et en effectuant la nouvelle factorisation suivant cet ordre.

Exemple :

f = v1.v2.v3 + v1.v2.v4.v5 + ¬v1.v6 + ¬v1.¬v7.¬v8.¬v9 + v6.¬v7.v8.v9

Supposons que la bibliothèque ne contienne qu'un boîtier à 4 entrées et des sorties à 3 monômes.

les noyaux de f (nombre de variables du noyau et du reste correspondant) sont :

v2.v3 + v4.v5 ; (3, 5)

v6 + ¬v7.¬v8.¬v9 ; (4, 8)

¬v1 + ¬v7.v8.v9 ; (4, 8)

On choisira k = v2.v3 + v4.v5 puisque c'est lui qui minimise le maximum du nombre de variables du noyau et du reste. Son co-noyau est v1.v2.

On commencera donc pour les prochaines décompositions à factoriser v1 et v2 comme co-noyau (ordre de factorisation dans la recherche des noyaux).

f= q. k + r

q = v1.v2

r = ¬v1.v6 + ¬v1.¬v7.¬v8.¬v9 + v6.¬v7.v8.v9

r a deux noyaux :

v6 + ¬v7.¬v8.¬v9 ; (4, 4)

¬v1 + ¬v7.v8.v9 ; (4, 4)

on réecrit r = q'. k' + r'

avec k'= v6 + ¬v7.¬v8.¬v9 puisque on a commencé à factoriser v1.

q'= ¬v1

r'= v6.¬v7.v8.v9

On peut réinjecter q' dans r : r = ¬v1. k' + r'

et on peut réinjecter r et q dans f : f = v1.k + ¬v1. k' + r'

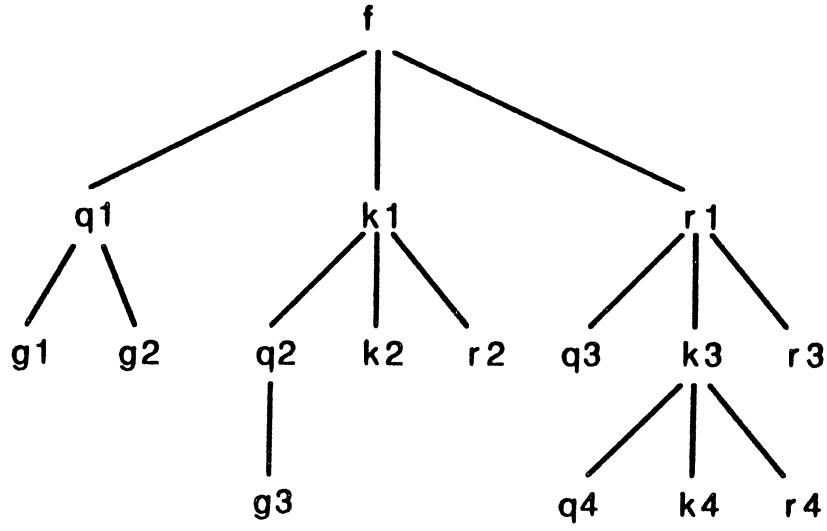
Si l'on avait choisi au deuxième niveau ¬v1 + ¬v7.v8.v9 comme noyau, cette deuxième réinjection n'aurait pas été possible.

5.1.5.2. ORDRE DE REINJECTION

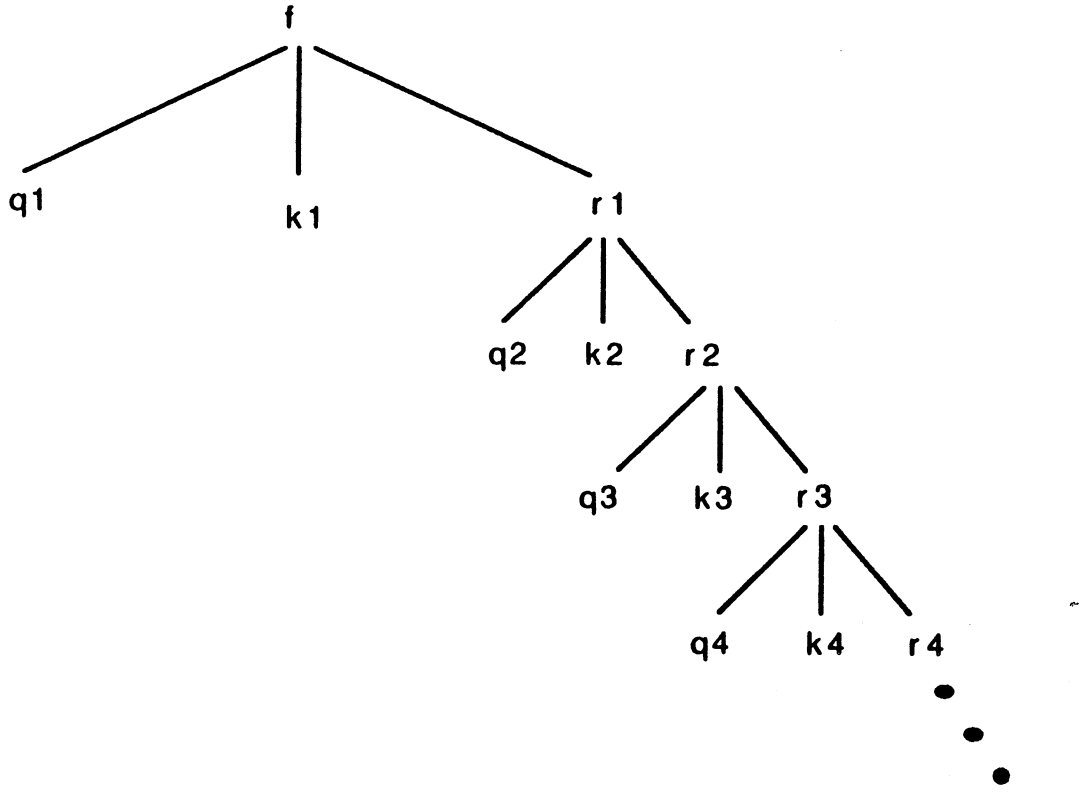
L'ordre des réinjections successives peut influer sur le résultat. Nous avons déterminé une heuristique pour cette ordre de réinjection.

Dans l'arbre de décomposition, on essaiera de raccourcir au maximum la branche la plus longue, puis de raccourcir au maximum toutes les autres branches. En effet, le temps de réponse de la fonction est égal au nombre de noeuds dans la branche la plus longue, et le nombre de sous-fonctions est égal au nombre de noeud dans l'arbre. On tendra ainsi à minimiser le temps de réponse de la fonction, tout en diminuant le nombre de sous-fonctions.

Exemple d'arbre de décomposition :



La plupart du temps, l'arbre de récursion a tendance à prendre la forme générale suivante (malgré l'effort d'égalisation des branches qui est effectué en choisissant le noyau qui minimise le maximum du nombre de variables de lui-même et du reste.



En effet, q et k sont vite réduits et réalisés, puisque qu'ils possèdent toujours moins de variables que f, et c'est donc souvent r qu'il faut à nouveau décomposer.

On partira du bout de la branche la plus longue, en la racourcissant au maximum par des réinjections successives en bout de branche, puis si ce n'est pas possible, dans des noeuds qui ne sont pas des feuilles.

Dans le cas où plusieurs branches sont de longueur maximale, on réinjectera d'abord la sous-fonction qui possède le moins de variables et de monômes, pour limiter la forme de la fonction mère aprés réinjection, permettant ainsi la réinjection potentielle de celle-ci.

Cette technique et ces heuristiques permettent ainsi de diminuer le nombre de sous-fonctions tout en limitant le chemin critique de la fonction décomposée. Il est à

noté qu'il est difficile d'estimer à priori, quelles sont les sous-fonctions les plus vite réalisables, afin de ne pratiquer qu'une passe descendante dans l'arbre de décomposition. La réinjection permet de choisir d'une façon simple les sous-fonctions qu'il faut générer, afin de limiter leur nombre ainsi que le temps de réponse.

Le nombre de variables diminue rapidement dans l'arbre de décomposition. De plus, on remarque que cette méthode amène, tout en diminuant le nombre de variables, aussi à une diminution du nombre de monômes des fonctions et sous-fonctions. Le nombre d'entrées des PAL variant entre 8 et 20, on arrive très vite à des sous-fonctions possédant un nombre de variables de cet ordre. Dans la plupart des cas le nombre d'étages de l'arbre de décomposition ne dépasse pas 3 (voir tableau et exemple suivant).

NOM	V	M	M-MAX	V-MAX	NB-SS-FCT	NB-ETAGE
F1	14	5	7	10	2	2
F2	25	10	14	20	3	2
F2	25	10	8	16	3	3
F3	18	10	7	10	4	2
F4	16	30	7	8	5	2
F5	18	40	7	10	7	2

M-MAX est le nombre maximum de monômes pour une sortie parmi tous les boitiers de la bibliothèque sélectionnée.

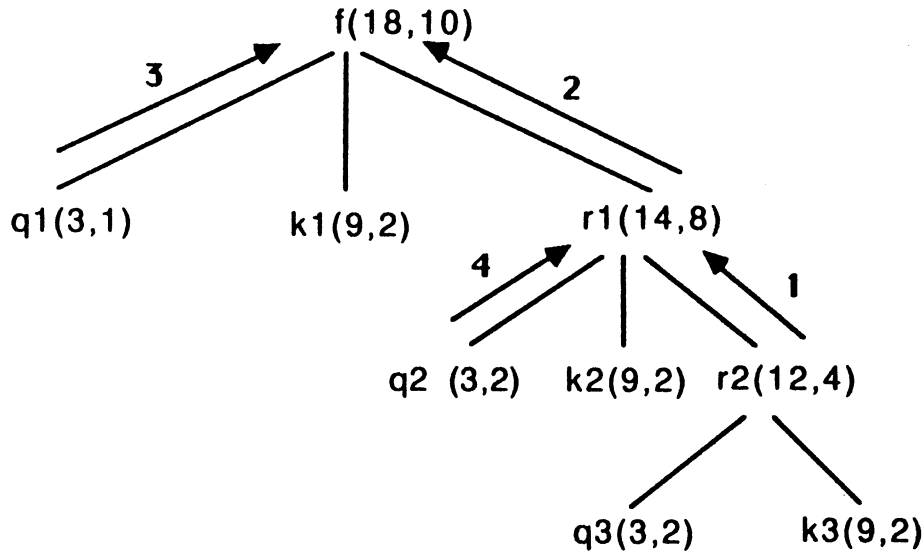
V-MAX est le nombre maximum de variables parmi tous les boitiers de cette bibliothèque

NB-SS-FCT est le nombre de sous-fonctions obtenues aprés la décomposition et la réinjection.

NB-ETAGE est le nombre d'étage de l'arbre de décomposition aprés réinjection, il correspond donc au temps de réponse de la fonction (nombre de boitiers à traverser).

Exemple:

Arbre de décomposition et réinjections obtenus pour la fonction F3:



Le couple associé à chaque sous-fonction correspond à son nombre de variables et son nombre de monômes. Les réinjections sont numérotées suivant l'ordre dans lequel elles sont effectuées.

Nous obtenons ainsi 2 étages et 4 sous-fonctions : k1, k2, q3, k3.

5.2. PERSPECTIVE POUR UNE APPROCHE MULTI-FONCTIONS

Cette approche n'a pas été implémentée, nous allons donner ses principales lignes directrices. Elle laisse espérer un gain significatif par rapport à l'approche précédente, mais malheureusement un coût beaucoup plus important.

Dans cette approche, nous regardons au moment de la décomposition, les fonctions dans leur ensemble. Nous essaierons, à tout moment de la décomposition, de trouver des sous-fonctions communes à plusieurs fonctions, pour obtenir ainsi aprés décomposition, moins de fonctions et donc moins de boîtiers.

5.2.1. DIMINUTION DU NOMBRE DE MONOMES

Nous supposons ici que le nombre de variables n'est pas trop grand et que seul le nombre de monômes importe. Nous voulons diminuer le nombre de monômes pour plusieurs fonctions. Nous allons employer la méthode décrite au chapitre (5.1.2), mais

en essayant de trouver des sous-fonctions identiques pour plusieurs fonctions.
Les formes polynômiales que nous avons obtenues aprés une minimisation indépendante de chaque fonction (chapitre 4.2) possèdent chacune un nombre minimal de monômes, mais on retrouve rarement des monômes communs puisque rien n'a été fait pour cela.

Nous devons donc trouver une forme pour chaque fonction f1, ..., fn telles qu'elles contiennent le plus possible de monômes communs. Pour cela, nous effectuerons une minimisation de la fonction générale (f1,...,fn),(voir première partie de cette thèse) nous obtiendrons ainsi un nombre de monômes minimum pour l'ensemble des formes polynômiales de f1, ..., fn, et donc, un maximum de monômes identiques dans l'ensemble de ces formes.

Exemple :
Soient les formes polynômiales minimisées indépendamment de 2 fonctions f1 et f2 :

$f1 = \neg a.\neg d + \neg a.b.\neg c + \neg b.\neg c.\neg d + \neg a.\neg b.c$

$f2 = b.\neg c.d + c.\neg d + \neg b.c + a.c$

Supposons que le nombre maximum de monômes par fonction soit égal à 2.
Si nous appliquons la méthode du chapitre 4.2 sur ces trois formes, nous obtenons 4 sous-fonctions supplémentaires.
Aucune sous-fonction commune à plusieurs fonctions n'existe.

Si nous effectuons une minimisation de la fonction générale (f1, f2) nous obtenons :
$(\neg a.\neg b.c, [1,1]) + (\neg a.c\neg d, [1,1]) + (\neg b.\neg c.\neg d, [1,0]) +$
$(\neg a.b.\neg c, [1,0]) + (b.\neg c.d, [0,1]) + (a.c, [0,1])$
Nous avons alors la sous-fonction commune à f1 et f2 :
$sf1 = \neg a.\neg b.c + \neg a.c\neg d$
Et nous pouvons décomposer f1 et f2 :
$f1 = sf1 + sf2$
$f2 = sf1 + sf3$
$sf2 = \neg b.\neg c.\neg d + \neg a.b.\neg c$ et $sf3 = b.\neg c.d + a.c$

5.2.2. CAS GENERAL

De la même façon, dans la décomposition basée sur la décomposition

algébrique, il faut essayer de trouver des diviseurs communs à plusieurs fonctions. Nous essaierons tout d'abord de trouver des diviseurs communs composés de plusieurs monômes.
Pour cela nous pouvons nous appuyer sur le théorème suivant [BRA82].

Théorème :

Deux expressions polynômiales f et g ont un diviseur algébrique commun de plus d'un monôme d s'il existe un noyau k_f de f et un noyau k_g de g tel que :

d est égal à la somme des monômes communs de k_f et k_g .

Aprés le calcul de l'ensemble des noyaux de chaque fonction, il sera ainsi possible de trouver des diviseurs communs à plusieurs fonctions par intersection sur des noyaux de différentes fonctions.

Exemple :

f = a.e + b.e + c.d.e

g = a.d + a.e + b.d + b.e + c

noyaux de f : { a + b + c.d }
noyaux de g : { a + b , d + e }

a + b est un diviseur commun de f et g :

f = e. (a + b) + c.d.e et g = (d + e). (a + b) + c

En choisissant ces diviseurs communs en priorité, nous limiterons le nombre de sous-fonctions créées. Il nous faudra trouver des heuristiques concernant le choix de ces diviseurs multi-monômes communs, puis le choix de sous-monômes communs ou de noyaux propres à une seule fonction. Le problème n'est pas trivial et un prototypage en Prolog s'avèrera une étape bénéfique dans la recherche de ces méthodes.

6. LE PLACEMENT

Quand nous arrivons à cette étape de l'algorithme, toutes les fonctions sont réalisables à l'aide des boitiers de la bibliothèque donnée.
Une fonction peut être réalisée par le placement de sa forme polynômiale ou de celle de son complément obtenues dans les étapes précédentes (minimisation et décomposition). "Placer" effectivement des fonctions sur un boitier va consister à:

- premièrement, affecter aux broches du boitier, les noms des variables des fonctions placées (pour les pattes d'entrées), les noms des fonctions placées (pour les pattes de sorties), le nom de l'entrée d'activation des bascules (si le boitier en possède),
- deuxièmement, donner pour chaque fonction, la forme polynômiale qui devra être programmée sur le boitier pour la réaliser.

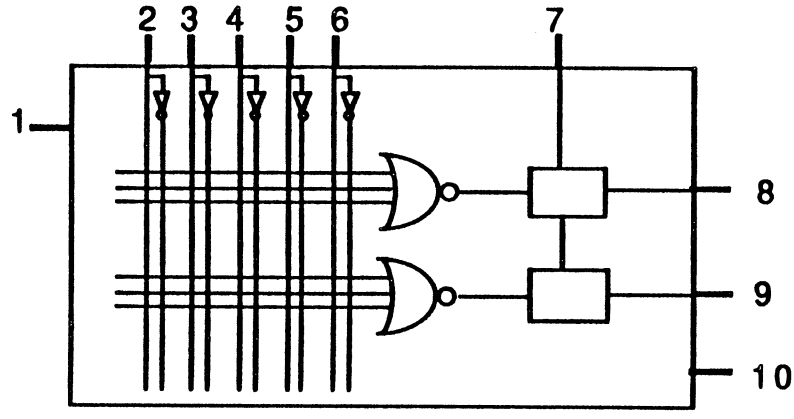
Exemple :
Réalisation de f1 et f2 dans un boitier B :

$f1 = a.b.d$, $\neg f1 = \neg a + \neg b + \neg d$

$f2 = \neg c + a.\neg b$, $\neg f2 = \neg a.c + b.c$

f1 et f2 doivent être mémorisées avec une entrée d'activation des bascules de nom 'Horl'.

Description du boitier B :



Aprés placement, on obtient la liste des noms associés aux broches (dans l'ordre

de leur numérotation) :

(gnd, a, b, c, d, non-connectée, Horl, f1, f2, vcc)

gnd et vcc correspondent à l'alimentation électrique du circuit.

La réalisation de f1 et f2 s'exprime par le placement de leurs compléments respectifs puisque les sorties de B sont réalisées par un NOR de trois monômes (polarité complémentée).
Les monômes à programmer sur l'étage ET seront donc :

$\neg f1 = \neg a + \neg b + \neg d$

$\neg f2 = \neg a.c + b.c$

Ces spécifications permettront la programmation automatique de chaque boitier, ainsi que la réalisation des interconnexions entre les différents boitiers sur la carte.

Le but de cette étape est de choisir un ensemble minimal de boitiers (dont la somme des poids est minimale), grâce auquels pourront être réalisées toutes les fonctions. La difficulté réside dans le fait que nous pouvons avoir le choix entre différents types de boitiers pour réaliser chaque fonction. De plus, les boitiers pouvant réaliser pour la plupart, plusieurs sorties, il faut essayer de regrouper au maximum les fonctions pour utiliser un nombre minimum de boitiers.

6.1. UNE METHODE EXHAUSTIVE

Nous nous sommes basés sur une méthode consistant à essayer tous les placements possibles de chaque fonction, en essayant de regrouper au maximum plusieurs fonctions dans les boitiers multi-sorties.

Cette méthode exhaustive consistera à essayer successivement pour chaque fonction, tous ses placements possibles dans chacun des boitiers de la bibliothèque. Pour regrouper au maximum les fonctions dans les boitiers multi-sorties, nous essairons de placer une fonction donnée, d'abord dans les boitiers utilisés pour la réalisation des fonctions précédement placées.

La réalisation d'une fonction sera possible dans un boitier déjà utilisé, d'une part si les spécifications initiales du boitier (polarité, bascule, nombre d'entrées, nombre de monômes par sortie) conviennent à celles de la fonction ,et d'autre part si l'affectation

d'une partie des broches du boitier permet encore le placement de la fonction (ou de son complément).

Par une telle méthode, nous sommes sûrs d'arriver à la solution optimale, mais l'arbre de placement (représentant toutes les possibilités de placements successifs de l'ensemble des fonctions) peut être trés grand, et la recherche de la solution optimale trés coûteuse.

Nous pouvons réduire de façon importante cette recherche en arrêtant la descente dans une branche de l'arbre de placement, dès que le coût de la solution courante devient supérieur ou égal au coût (nombre de boitiers et somme des poids de ces boitiers) de la solution précédement trouvée ("branch and bound").

Exemple :

Les fonctions à réaliser sont les suivantes :

f1 : 3 monômes, $\neg$f1 : 2 monômes, 4 variables (v1, v2, v3, v4)

f2 : 2 monômes, $\neg$f2 : 4 monômes, 6 variables (v1, v2, v5, v6, v7, v8)

f3 : 10 monômes, $\neg$f3 : 3 monômes, 4 variables (v1, v2, v9, v10)

f4 : 5 monômes, $\neg$f4 : 4 monômes, 4 variables (v3, v4, v9, v10)

La bibliothèque contient 3 boitiers :

B1 : polarité normale, 2 sorties à 4 monômes, 8 entrées, poids=2

B2 : polarité complémentée, 4 sorties à 4 monômes, 6 entrées, poids=2

B3 : polarité normale, 1 sorties à 2 monômes, 10 entrées, poids=1

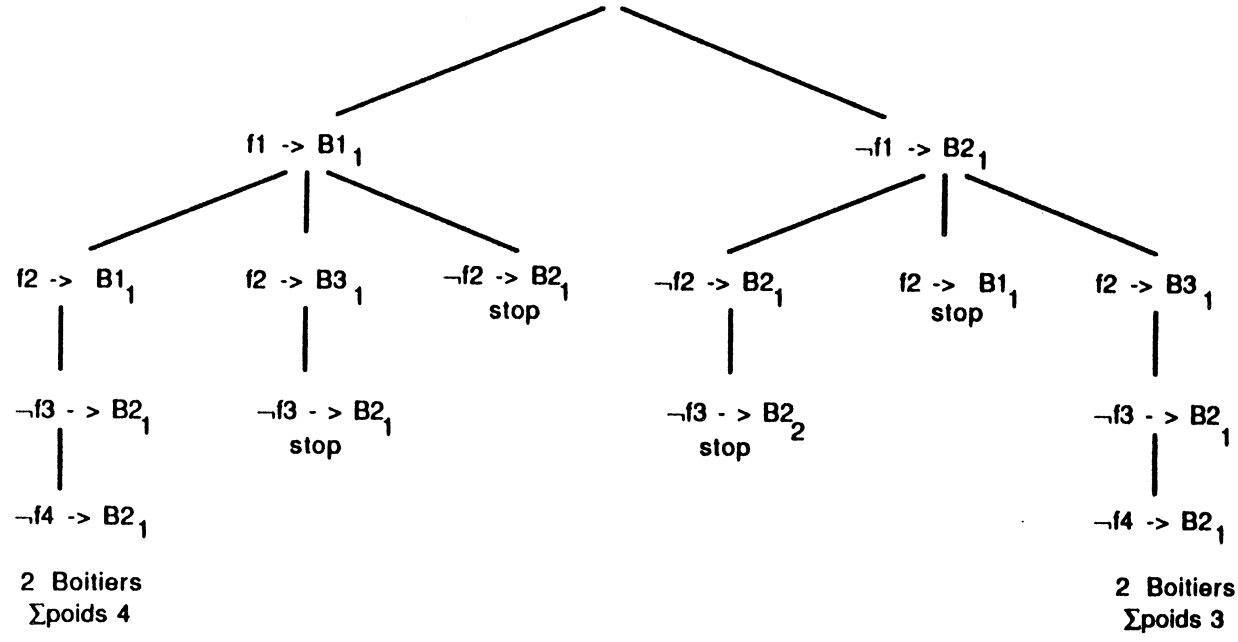
Supposons que l'on place les fonctions suivant l'ordre f1, f2, f3, f4.

Supposons que l'on essaie de placer toujours f_i avant $\neg f_i$.

Supposons que l'on essaie de placer une fonction sur les boitiers suivant l'ordre : B1, B2, B3.

Ces différents ordres sont aléatoires.

Nous obtenons alors l'arbre de placement suivant :



Nous pouvons remarquer que même si une fonction peut être réalisée dans un boitier utilisé, il faudra essayer tous ses autres placements dans les autres boitiers (déjà utilisés ou non), pour être certain d'arriver à la solution optimale. Dans l'exemple précédent, f2 peut être réalisée dans $B2_1$ (où f1 a été placé), mais il faudra la placer dans $B3_1$ (boitier vierge) pour arriver à la solution optimale.

Ces restrictions ne se sont pas avérées suffisantes. Nous avons donc établi une heuristique trés simple qui permet de réduire la taille de l'arbre de placement et d'arriver trés rapidement à une solution quasi-otimale, réduisant ainsi considérablement le coût de la recherche de la solution minimale.

6.2. UNE HEURISTIQUE DE PLACEMENT

La taille de l'arbre de placement dépend :

-de l'ordre des boitiers sur lequels on va essayer successivement de placer les fonctions

- de l'ordre dans lequel ces fonctions vont être placées.

Pour limiter la taille de l'arbre de placement, il faut essayer de réaliser d'abord les fonctions qui possèdent un minimum de choix potentiels pour leur placement sur les différents boitiers.

D'autre part, pour arriver à une bonne solution des les premiers essais, il faut

essayer de placer les fonctions sur les boitiers possédant un maximum de sorties, espérant ainsi regrouper un maximum de fonctions par boitier.
Si les fonctions possédant les caractéristiques les plus contraignantes (nombre de variable ou de monômes maximum) sont placées sur les boitiers comportant le plus possibles de sorties, les fonctions suivantes auront de fortes chances de pouvoir être placées sur les mêmes boitiers.

Notre heuristique va donc consister à effectuer avant le placement, les classements suivants.

Soit NB_{fi} (resp. $NB_{\neg fi}$) le nombre de boitiers où la fonction fi (resp. son complément) est plaçable.

Les fonctions sont ordonnées selon les critères suivant, par ordre de priorité décroissant :

- $NB_{fi} + NB_{\neg fi}$ croissant
- minimum du nombre de monômes de la fonction et de son complément, décroissant
- nombre de variables décroissant

Pour chaque fonction, on classera le couple (fi, ¬fi) selon les deux critères par ordre de priorité décroissant :

-$NB_{\sim fi}$ maximum d'abord,

-nombre de monômes de ~fi minimum.

Le nombre de variables étant en général le même pour les deux, on commencera par celle qui est plaçable sur le plus possible de boitiers, augmentant ainsi ses chances de réalisation sur un boitier déjà utilisé.

Les boitiers seront ordonnés selon les critères suivants, par ordre de priorité décroissant :

- nombre de sorties décroissant
- poids croissants
- nombre de variables décroissant
- nombre de monômes par sortie décroissant

- polarité programmable

Ce classement sera préservé dans la liste des boitiers utilisés au cours du placement.

En résumé, cette heuristique va consister à placer les fonctions les plus contraignantes, dans les boitiers possédant le plus possible de sorties, de poids minimum et possédant les caractéristiques qui faciliterons au maximum le placement des fonctions suivantes (nombre de variables et monômes maximum, polarité programmable). Les fonctions suivantes étant moins contraignantes, elles auront toutes les chances de pouvoir être placées dans les mêmes boitiers. Il faut rappeler que les boitiers ont un nombre de monômes identique (ou trés proche) pour toutes leurs sorties.

Nous limitons ainsi beaucoup le nombre d'essai qu'il faut effectuer avant d'arriver à une solution optimale.

Si l'on reprend l'exemple précédent:

Nous avons :

$NB_{f1} = 1$ et $NB_{\neg f1} = 1$,

$NB_{f2} = 2$ et $NB_{\neg f2} = 1$,

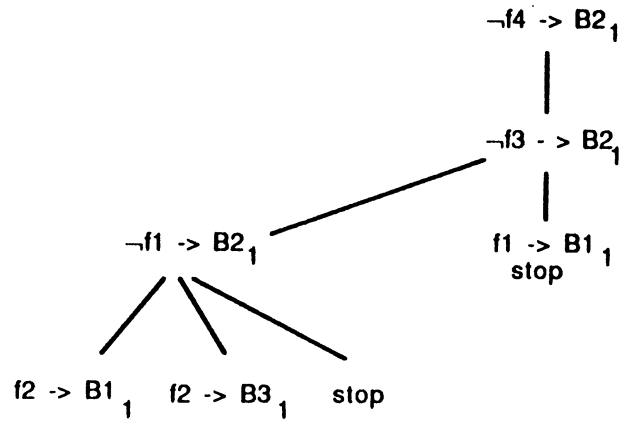
$NB_{f3} = 0$ et $NB_{\neg f3} = 1$,

$NB_{f4} = 0$ et $NB_{\neg f4} = 1$.

On classera les fonctions dans l'ordre : ($\neg$f4, f4), (f3,$\neg$f3), ($\neg$f1, f1), (f2 , $\neg$f2).

On classera les boitiers dans l'ordre : B2, B1, B3.

On obtient l'arbre de placement nettement moins important :



Cette heuristique s'est avérée dans la pratique très efficace, on s'approche prés du minimum, la plupart du temps, dès le premier essai, la recherche par retour arrière est alors très limitée et aboutit rapidement.

Le tableau suivant donne les résultats obtenus pour le placement sur un échantillon varié d'exemples.

Le langage de programmation est Prolog et les temps donnés ont été établis sur un MICRO-VAX2, VMS.

La bibliothèque uitlisée ici est composée de 32 boitiers différents.

NOM	V	F	SOL. 1, TPS	SOL. 2, TPS	SOL. 3, TPS
LION9	6	5	2, 14s	2, 6s	NON, 90s
BBSSE	11	11	2, 20s	NON, 80s	
DK14	6	8	1, 15s	NON, 40s	
5XP1	7	10	3, 29s	3, 8s	NON, 105s
RD53	7	2	2, 12s	2, 27s	NON, 70s
LAPP	16	14	2, 25s	2, 80s	NON, 130s

V: nombre de variables

F: nombre de fonctions

SOLi, tps : nombre de boitiers de la $i^{ème}$ solution et temps de placement en secondes.

CONCLUSION

La synthèse logique semble être une discipline en perpétuelle évolution car elle suit de très près l'évolution technologique et l'informatique en général.
Dans la première partie de cette thèse, nous avons été amenés à revoir les fondements du calcul booléen dans une optique d'efficacité. Il s'agit d'utiliser au mieux les moyens de calcul actuels pour arriver à traiter des applications réelles devenues très complexes. Ces progrès touchent certainement de nombreux secteurs de l'informatique qui utilisent le calcul booléen.
A partir de ces améliorations de base, deux cibles ont été considérées.

La synthèse sur "PLA sur mesure", qui nous a amené à proposer une théorie algébrique de la minimisation deux couches et qui a été exhaustivement explorée.
Il semble que la minimisation deux-couches soit arriver à son apogée, les résultats obtenus, dans des temps trés limités, étant trés proche du minimum exact et cela, quelle que soit la complexité des applications qui sont actuellement réalisables sur PLA.

La synthèse multicouches sur réseaux figés du type PAL, qui nous a amenés à une solution originale apportant des ouvertures tant théoriques que pratiques. La méthode proposée, encore à l'état de prototype, laisse espérer un dévellopement d'outils performants pour des conceptions de cartes à base de réseaux programmables.
Devant l'ampleur de l'essor de ce type de circuit, il semblait important d'apporter au concepteur une C.A.O. adaptée à leurs besoins. Vu la diversité de ces réseaux programmables, il faudra adapter nos méthodes de synthèses aux différentes caractéristiques actuelles (PAL, LCA, PLA ...) ou futures. La réalisation d'outils de CAO de ce type devrait ouvrir une ère nouvelle dans le domaine de la conception de carte, négligé jusqu'à présent.

Dans les deux cas, il apparaît clairement que les recherches en synthèse logique doivent se poursuivre inlassablement avec méthode, rigueur et efficacité pour réellement accompagner les évolutions technologiques.

REFERENCES BIBLIOGRAPHIQUES

[AGR85] P. Agrawal, V. D. Agrawal
Multiple output minimization
22nd DAC, 1985, pp. 674, 680

[ARE78] Z. Arevelo and J.G. Bredeson
A method to simplify a boolean function into a near minimal sum-of-products for programmable logic arrays
IEEE Trans. on Comp., Vol. C27, N°11,pp 1028-1039, Nov. 78

[BAH81] R.J. Bahnsen
Essential prime implicant tester
IBM Tech. Disclosure Bulletin, Vol. 24, N°5, pp. 2344, Oct. 1981

[BRA82] R. K. Brayton and C.T. McMullen.
The decomposition and factorization of booleans expressions.
ISCAS Proceedings pp 49-54, Mai 82.

[BRA84] R. K. Brayton, G. D. Hachtel, C.T. McMullen,
A. L. Sangiovanni Vincentelli.
Logic Minimization Algorithms for VLSI Synthesis.
Kluwer Academic Publishers, 1984.

[BRA84b] R. K. Brayton and C.T. McMullen.
Synthesis and optimization of multistage logic.
Proceedings ICCD, 1984, pp 23-28.

[BRA86] R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips,
R. Rudell, R. Segal, A. Wang, R. Yung, A. L. Sangiovanni Vincentelli.
Multiple-Level Logic Optimization System.
IEEE, ICCAD, Nov. 1986, pp. 356, 359.

[BRA87] R. K. Brayton, R. Rudell, A. L. Sangiovanni Vincentelli, A. R. Wang.
MIS, a multi-level logic optimization system
IEEE Trans. on Comp., Vol.CAD 6, N° 6, Nov. 87

[BRO81] D.W. Brown
A State-Machine Synthetizer - SMS
Proc. 18th Design Automation Conference, pp. 301-304,
Nashville, Juin 1981

[CARV85] M. Carvallo
Sur la minimisation d'une expression représentant une famille de fonctions booléennes.
RAIRO Informatique Théorique, Vol. 19, n°4, 1985, pp. 331/336

[CLO87] W.F. Clocksin, C.S. Mellish
Programmer en PROLOG
Eyrolles, 1987

[COL83] A. Colmerauer, H. Kanoui, M. Van Caneghem
PROLOG : bases théoriques et développement actuels
TSI, Vol. 2, N°4, Juillet 83

[COL84] A. Colmerauer
Prolog, langage de l'intelligence artificielle
La recherche, N°158, p. 1104-1114

[CRAS85] M. Crastes de Paulet
Spécification et simulations fonctionnelles de circuits complexes : le sytème CADOC
Thèse de Docteur Ingénieur, INP Grenoble, France, Nov. 85

[DAG85] M.R. Dagenais, V.K. Agarwal, N.C. Rumin
The McBoole Logic Minimizer
22nd Design Automation Conference, , pp. 667-673, 1985

[DAN86] A. Dandache
Conception de PLA CMOS
Thèse de Docteur de L'INP Grenoble, France, Juillet 86

[DEM83] G. De Micheli, A. Sangiovanni-Vincentelli
Computer-aided Synthesis of Pla-based finite state machines
ICCAD 1983, pp. 154, 156

[DEM84] G. De Micheli, A. Sangiovanni-Vincentelli, K. Brayton
Optimal Encoding of Control Logic
IEEE ICCD, New York, Oct. 84, pp. 16, 22

[GUR87] B. Gurunath and Nripendra, N. Biswas
An algorithm for multiple output minimization
IEEE, ICCAD, Nov. 1987, pp. 74, 77.

[GRE86] D. Gregory, K. Bartlett, Aart de Geus, G. Hachtel
SOCRATES: A system automatically synthesizing and optimizing combinational logic
23rd DAC, 1986, pp. 79, 85.

[HAC88] G. D. Hachtel, R. M. Jacoby
Verification Algotithms for VLSI Synthesis
IEEE, ICCAD, Vol. 7, No 5 May 1988, pp. 616, 640

[HAN85] S. Hanriat
Synthèse logique à base de règles pour les compilateurs de Silicium
Thèse de 3eme cycle, INP Grenoble, France, Juin 85

[HART77] Reiner W. HARTENSTEIN
Fundamentals of Structured Hardware Design
North-Holland 1977

[HON72] S.J. Hong and D.L. Ostapko
On complementation of Booleans functions
IEEE Trans. on Comp., Vol. C21, p. 1072, 1972

[HON74] S.J. Hong, R.G. Cain and D.L. Ostapko
MINI : A heuristic approach for logic minimization
IBM J. of Res. and Dev. Vol. 18, pp. 443-458, Sept. 1974

[KAR53] M. Karnaugh
The map method for synthesis of combinational logic circuits
AIEE Transactions, Part 1, Nov. 1953, pp 593-598

[KIN87] C. Kingsley
The implementation of a state machine compiler
24th DAC, 1987, pp. 580, 583

[KUN68] J. Kuntzmann
Algèbre de Boole
Edition Dunod, Paris, 1968

[LAA85] P. J. M. Van Laarhoven, E. H. L. Aarts, M. Davio
PHIPLA : A new algorithm for logic minimization
22nd DAC, 1985, pp. 739-743

[LEV88] G. Saucier, R. Leveugle
Highly wireable multi-level synthesis with compiled cells
International Workshop on logic and architecture synthesis for silicon compilers, INP Grenoble, Mai 88

[MCC65] E.J. McCluskey, Jr.,
Introduction to the theory of Switching Circuit
McGraw-Hill, 1965

[MMI81] Designing with Programmable Array Logic
The Technical Staff of Monolithic Memories Inc.
M^{C} Graw Hill, 1981

[MOR70] E. Morreale.
Recursive operators for prime implicant and irredundant normal form determination.
IEEE Trans. on Comp., Vol. C-19, p. 504, 1970.

[MUN86] T. Munakata
Procedurally Oriented Programming Techniques in Prolog
IEEE, Expert, Summer 1986, pp. 41, 47.

[NGU87] L. B. Nguyen, M. A. Perkowski, N. B. Goldstein
Palmini - Fast boolean minimizer for personal computers
24th DAC 1987, pp. 615, 621

[POI88] G. Saucier, P. Sicard, C. Duff, F. Poirot
Des contrôleurs sur PAL, PLA, EPLD et Logique Aléatoire
Colloque National, Conception de circuits à la demande.
Grenoble, Janvier 88, Papier B2.

[RHY77] V.T. Rhyne, P.S. Noe, M.N. McKinny, and U. W.Pooch
A new technique for the fast minimization of switching function
IEEE Trans. on Comp., Vol. C26, N°8, pp 757-764, August 1977

[ROT80] J.P. Roth
Computer Logic, Testing and Verification
Computer Science Press, 1980

[RUD86] R. Rudell, A. Sangiovanni-Vincentelli
Exact Minimization of Multiple-Valued Functions for PLA Optimization
IEEE Trans. on Comp., pp 352-355 1986

[SAS88] T. Sasao
Multiple-valued Logic and Optimization of Programmable Logic Arrays
IEEE, Computer, April 1988, pp. 71, 80.

[SAU87] G. Saucier, M. Crastes, P. Sicard
ASYL : A rule-based system for controlleur synthesis
IEEE Trans. on comp., Vol.CAD 6, N° 6, Nov. 87

[SHA48] C. E. Shannon.
The synthesis of two-terminal switching circuits.
Bell Sys. Tech. J., 1948

[SIC88] P. Sicard, C. Duff
An alternative to Espresso 2
International Workshop on logic and architecture synthesis for silicon compilers, INP Grenoble, Mai 88

[SIC88b] P. Sicard, G. Saucier
Logic Synthesis on PALs
International Workshop on logic and architecture synthesis for silicon compilers, INP Grenoble, Mai 88.

[THU83] G. Thuau
Conception logique et topologique en technologie MOS
Thèse de 3eme cycle, INP Grenoble, France, Dec 83

[TIS65] P. Tison
Théorie des consensus
Thèse de l'Université des sciences de Grenoble, 1965.

[TIS67] P. Tison
Generalisation of consensus theory and application to the minimization of Boolean functions
IEEE Trans. on Elec. Comp., Vol. EC16, 1967, pp : 446.456

TABLE DES MATIERES

PARTIE 2 : SYNTHESE MULTICOUCHES SUR RESEAUX PROGRAMMABLES FIGES

département des études doctorales

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 15 Titre III de l'arrêté du 5 juillet 1984 relatif aux études doctorales

VU les rapports de présentation de

Messieurs A. COSTES
N. XUONG

Monsieur SICARD Pascal

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité INFORMATIQUE

Fait à Grenoble, le 16 Août 1988

Le Président
de l'Institut National Polytechnique
de Grenoble