# Chapitre 3

# Graphes de décision binaires

Pour répondre au problème posé par l'accroissement de la taille des fonctions logiques à traiter (grand nombre de fonctions, de monômes et de variables), de nouvelles techniques de représentation des fonctions booléennes ont été développées. Elles sont généralement basées sur la récursivité du théorème de Shannon et non plus sur les théorèmes habituels du calcul booléen. Les Graphes de Décision Binaires (BDD pour Binary Decision Diagram) sont des modèles de représentation particulièrement efficaces pour représenter et manipuler de telles fonctions logiques. Ils ont été introduits par Akers en 1978. En 1986, Bryant propose les OBDD (Ordered Binary Decision Diagram), une représentation canonique des BDD ainsi que des algorithmes pour calculer efficacement les opérations booléennes sur ces structures de données.

## 3.1. Arbres de décision binaire

Les arbres de décision binaires constituent un modèle de représentation des fonctions booléennes. Un arbre de décision binaire est un arbre orienté composé d'une racine, de sommets intermédiaires et de sommets terminaux valant 0 ou 1. La racine et les sommets intermédiaires sont indexés et possèdent deux sommets "fils", un fils gauche et un fils droit. Le fils gauche est atteint en empruntant la branche "0", le fils droit en empruntant la branche 1. Un arbre de décision binaire est obtenu en appliquant récursivement la première forme du théorème de Shannon sur l'ensemble des variables de la fonction.

Théorème de Shannon (rappel):

$$f(x_1,x_2,...,x_i,...,x_n) = x_i' \cdot f(x_1,x_2,...,0,...,x_n) + x_i \cdot f(x_1,x_2,...,1,...,x_n)$$

Si ce théorème est appliqué à f pour $x_1$, puis aux deux sous-fonctions obtenues pour $x_2$, et ainsi de suite jusqu'à $x_n$, on peut réaliser un arbre de décision binaire.

Exemple : f(a,b,c) = a'.b.c + a.c

f(a,b,c) = a'.f(0,b.c) + a.f(1,b,c)
f(0,b,c) = b.c
f(1,b,c) = c

f(0,b,c) = b'.f(0,0.c) + b.f(0,1,c)
f(0,0,c) = 0
f(0,1,c) = c
f(0,0,c) = c'.f(0,0,0) + c.f(0,0,1)
f(0,0,0) = 0
f(0,0,1) = 0
f(0,1,c) = c.'f(0,1,0) + c.f(0,1,1)
f(0,1,0) = 0

f(0,1,1) = 1
f(1,b,c) = b'.f(1,0.c) + b.f(1,1,c)
f(1,0,c) = c
f(1,1,c) = c
f(1,0,c) = c'.f(1,0,0) + c.f(1,0,1)
f(1,0,0) = 0
f(1,0,1) = 1
f(1,1,c) = c.'f(1,1,0) + c.f(1,1,1)
f(1,1,0) = 0
f(1,1,1) = 1

Ces relations peuvent être exprimées sous forme d'arbre de décision binaire de la façon suivante :

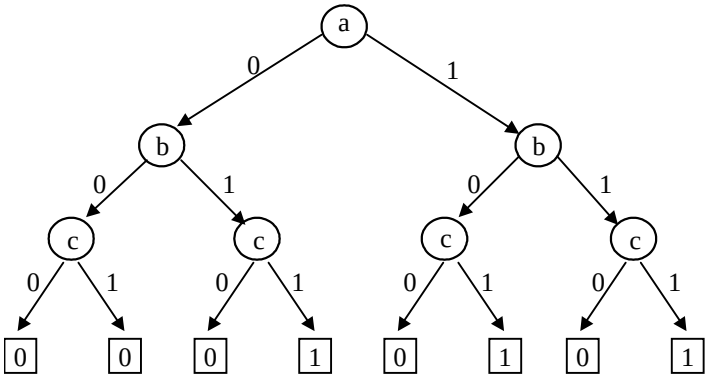



**Figure 3.1.** *Arbre de décision binaire associé à la fonction f = a'bc + ac*

Notons qu'il est possible de réduire la taille de l'arbre en s'arrêtant dans l'application du théorème de Shannon dès que l'on se trouve en présence d'une constante 0 ou 1. Ainsi, dans l'exemple précédent nous voyons que f(0,0,c)=0, c'est à dire que f=0 quelle que soit la valeur qui sera assignée à c. En s'arrêtant à ce niveau dans l'application du théorème de Shannon, l'arbre de décision binaire associé à la fonction f devient:

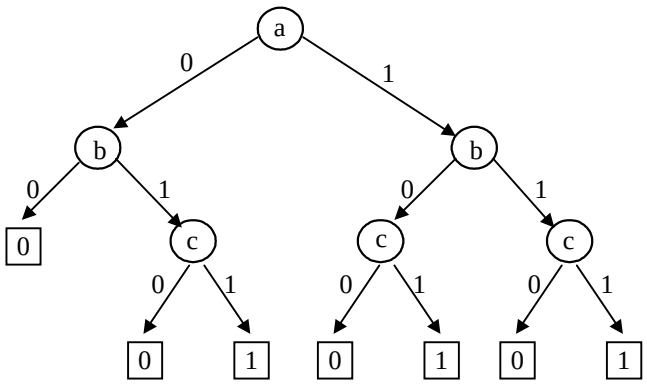



**Figure 3.2.** *Arbre de décision binaire simplifié associé à la fonction f = a'bc + ac*

### 3.2. Diagrammes de décision binaires (BDD)

Après sa construction, un arbre de décision binaire peut être réduit par transformation en un graphe acyclique orienté appelé graphe de décision binaire (BDD). En effet, nous avons pu remarquer qu'un arbre de décision

binaire peut posséder des feuilles, voir des sous-graphes identiques. La représentation peut être simplifiée en ne conservant qu'une seule représentation des feuilles et sous-graphes concernés, et en remplaçant les autres feuilles et sous-graphes par un arc vers le premiers. Le graphe de décision binaire ainsi obtenu peut également être réduit par élimination des sommets redondants, c'est à dire des sommets ayant un fils gauche et un fils droit identique.

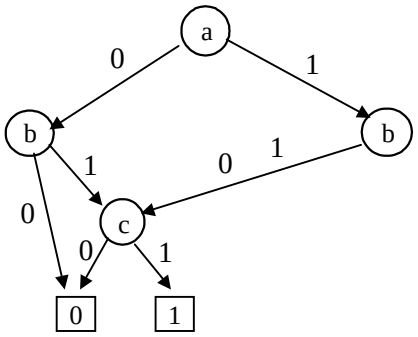



**Figure 3.3.** *Réduction par élimination des feuilles et sous-graphes identiques*

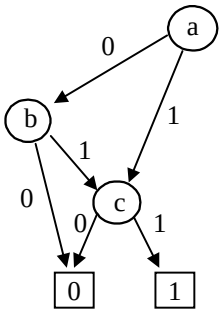



**Figure 3.4.** *Réduction par élimination sommets redondants*

En cherchant et remplaçant tous les sous-arbres équivalents d'un diagramme de décision binaire, on obtient le diagramme de décision binaire minimal pour l'ordre correspondant des variables (ROBDD pour Reduced Ordered BDD). Il est important de noter que pour un ordre donné de variables, le graphe de décision binaire minimal est **unique**. Cette unicité peut évidemment être utilisée pour montrer l'équivalence de deux expressions logiques.

Remarques :

- La complexité de la procédure de réduction d'un BDD est O(nlog(n)).
- La structure du BDD (et en particulier sa taille) dépend de l'ordre dans lequel sont considérées les variables. Ainsi, pour deux ordres différents, les BDD peuvent être de tailles très différentes.

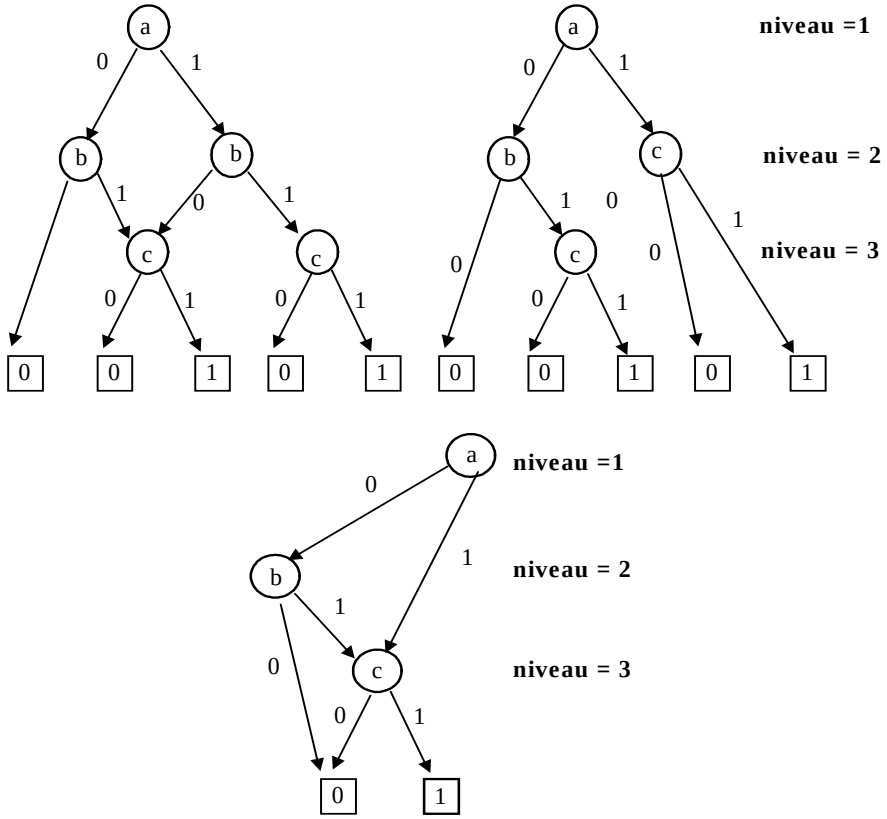



**Figure 3.5.** *OBDDs et ROBDD de la fonction f = (a+b).c*

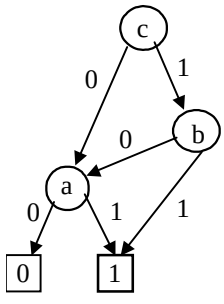



**Figure 3.6.** *ROBDD de la fonction f = (a+b).c sous l'ordre c, b, a.*

### 3.3. Construction des BDDs

Afin de garantir la canonicité de la représentation , les contraintes suivantes sont imposées :

- chaque variable ne peut apparaître qu'une fois au plus sur chaque chemin entre la racine et une feuille
- les variables sont ordonnées de telle façon que si un sommet de label $x_i$ a un fils de label $x_j$ alors ord($x_i$) < ord ($x_j$)

Il y a 2 stratégies principales de construction des BDDs :

- de la racine vers les feuilles (top-down). Cette méthode est utilisée lorsque on part d'une formule algébrique
- des feuilles vers la racine (bottom-upp) lorsque l'on part d'une description structurelle du circuit.

### *3.3.1. Méthode top-down*

On utilise la formule de Shannon.

Exemple : soit f(a,b,c)=a'bc' + ac avec ord(a)<ord(b)<ord(c)

- Si l'on fait l'expansion par rapport à a on obtient la Fig.3.7.a
- En faisant l'expansion par rapport à b on obtient la Fig.3.7.b
- En faisant l'expansion par rapport à c on obtient la Fig.3.7.c

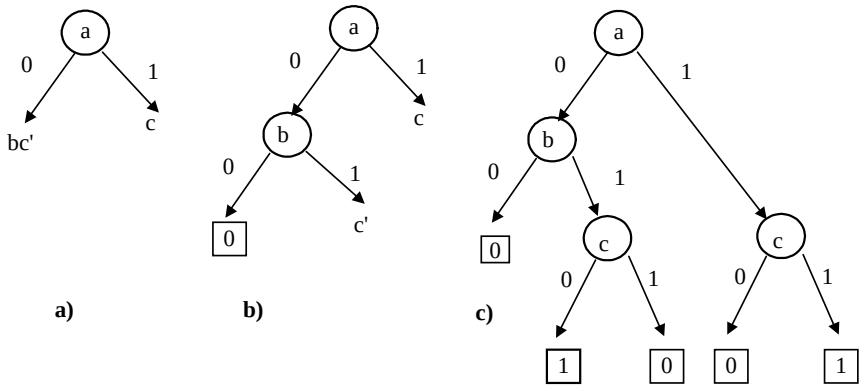



**Figure 3.7.** *Construction du BDD de la fonction f = a'bc' + ac*

### *3.3.2. Méthode bottom-up*

Le ***niveau*** d'une formule est défini par :

- les variables d'entrée sont de niveau 0
- chaque sous-formule f=g<Op>h a un niveau égal à Max (niveau(g), niveau(h)) + 1

Méthode pour construire le BDD d'une fonction f de n variables :

1. construire les BDD des variables
2. construire les BDDs des formules de niveau 1 à l'aide de la fonction : appliquer (f1 : BDD, f2 : BDD, opérateur <Op>) : BDD
3. répéter l'étape 2 jusqu'à ce que tous les niveaux soient considérés.
   Par exemple si f = xy + z : on construit les BDD de x , y et z puis celui de x.y puis celui de x.y + z.

## 3.4. Opérations logiques entre deux fonctions représentées par BDD

Les opérations logiques applicables sur les BDD on été définies par Bryant. Ces opérations sont les opérations de complémentation, de test d'implication, de ET logique, de OU logique, et de OU Exclusif. L'algorithme permettant de déterminer le BDD résultant d'une opération logique entre deux fonctions est basé sur l'application récursive du théorème de Shannon.

Etant donné deux fonctions f et g, un opérateur Op et un ordre des variables on a:

$f<Op>g = x_i.(f/x_i=0 <Op> g/x_i=0 ) + x_i.(f/x_i=1 <Op> g/x_i=1)$

Le BDD résultant d'une opération logique entre deux fonctions peut être conçu à partir des deux BDD originaux en appliquant la procédure suivante :

Considérer les sommets racines des BDD. En fonction de la nature de ces deux sommets, appliquer récursivement l'une des règles suivantes. Itérer le processus jusqu'à la génération des sommets terminaux.

**R1** : Si les deux sommets Sf et Sg sont des sommets terminaux alors le sommet résultant est un sommet terminal de valeur Valeur(Sf) <Op> Valeur(Sg).

**R2** : Si le sommet Sf est un sommet terminal mais pas le sommet Sg, créer le sommet Sg en lui associant comme fils droit le résultat de la comparaison (Sg, fils droit de Sf) et comme fils gauche le résultat de la comparaison (Sg, fils gauche de Sf).

**R3** : Si ni le sommet Sf ni le sommet Sg ne sont des sommet terminaux alors :

**R3.1** : Si Sf et Sg représentent la même variable, créer un sommet représentant la variable en lui associant comme fils droit le résultat de la comparaison (fils droit de Sg, fils droit de Sf) et comme fils gauche le résultat de la comparaison (fils gauche de Sg, fils gauche de Sf).

**R3.2** : Si Sf et Sg ne représentent pas la même variable, créer un sommet S représentant la variable intervenant la première dans l'ordonnancement considéré (S=min[ord(Sf), ord(Sg)] en lui associant comme fils droit (gauche) le résultat de la comparaison entre le fils droit (gauche) de min[ord(Sf), ord(Sg)] et l'autre sommet (max[ord(Sf), ord(Sg)]).

Remarque : Etant donné deux fonctions f et g dont les BDD respectifs ont n et m sommets. En termes d'appels récursif, la complexité de la procédure d'application d'une opération entre ces deux BDD est 2.n.m.

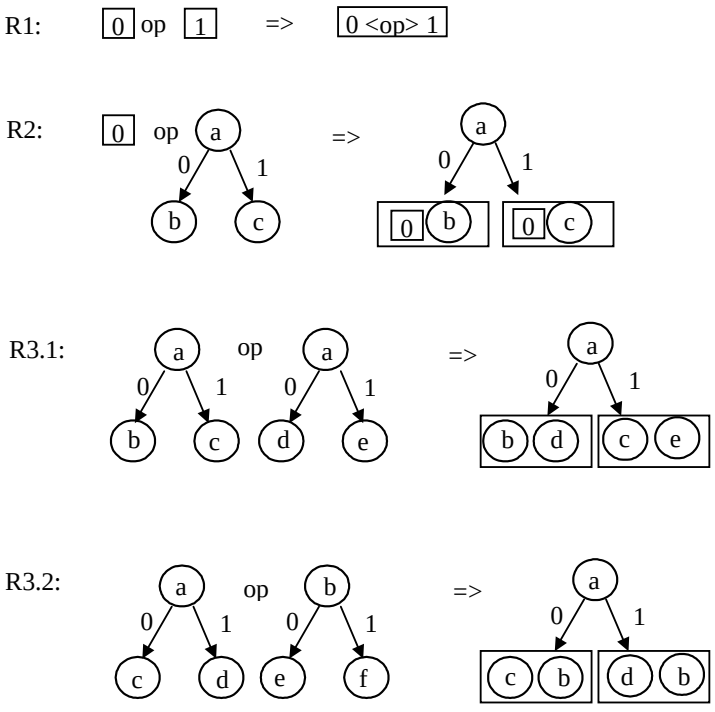


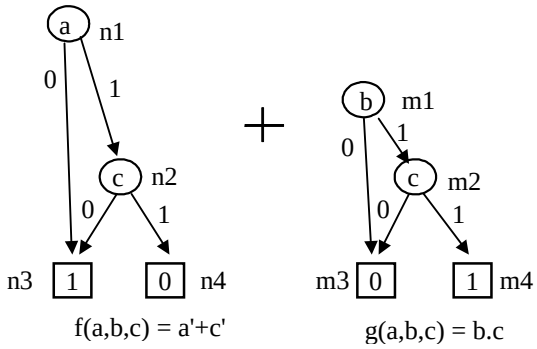

**Figure 3.8.** *Illustration des règles de combinaisons de BDD*

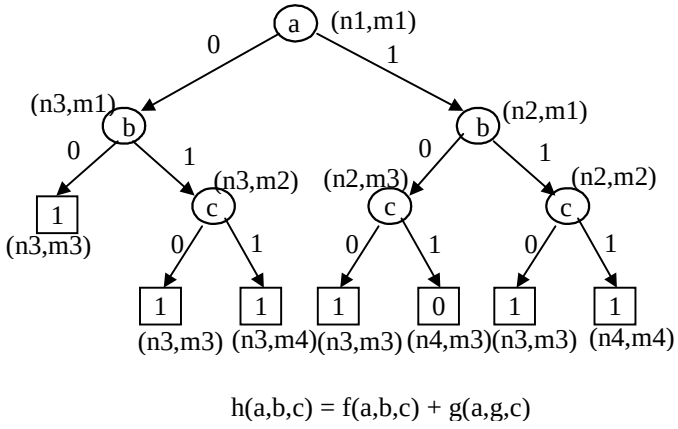
Remarque : Etant donné deux fonctions f et g dont les BDD respectifs ont n et m sommets. En termes d'appels récursifs, la complexité de la procédure d'application d'une opération entre ces deux BDD est 2.n.m.

Remarque **:** Cette méthode permet (outre les opérations) :

- de construire un BDD de façon ascendante,
- de calculer l'inverse d'une fonction en faisant "f ⊕ 1"
- de calculer une implication f1 => f2 <=> not ( f1 . not(f2))

Exemple : Soit les fonctions f(a,b,c) = a'+c' et g(a,b,c) = b.c représentées par leurs BDD. Construire le BDD de la fonction h(a,b,c) = f(a,b,c) + g(a,b,c).




h(a,b,c) = f(a,b,c) + g(a,g,c)

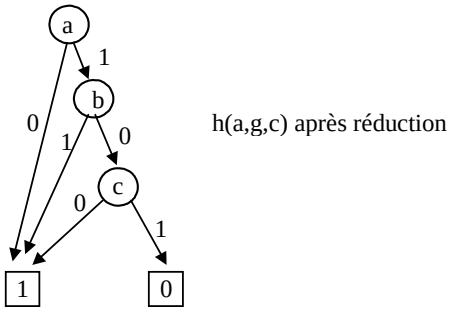



**Figure 3.9.** *Opération entre deux BDD*

### 3.5 Implantation informatique des BDD

La structure informatique permettant de représenter un BDD peut être un tableau tel que chaque ligne comporte 3 champs : un champs « Identificateur » permettant d'identifier le sommet dans le BDD, un champs Arc_0 donnant l'adresse du fils gauche du sommet en question et un champs Arc_1 donnant l'adresse du fils droit.

| Identificateur | Arc_0 | Arc_1 |
|---|---|---|
| 0 | - | - |
| 1 | - | - |
| Variable sommet i | Adresse sommet gauche | Adresse sommet droit |
| | | |

Exemple : soit une fonction f(a,b,c) = a' + c' + b.c

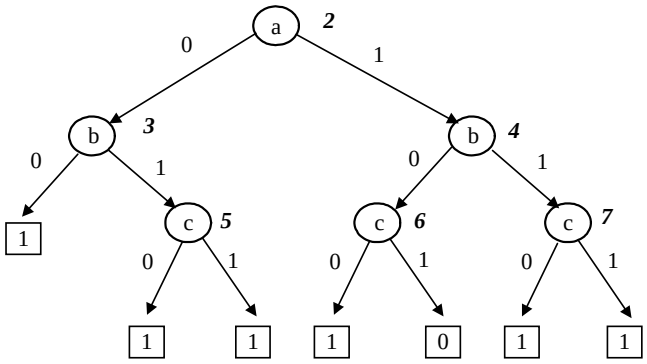



**Figure 3.10.** *BDD de la fonction f = a' + c' + b.c*

Le BDD de la fonction f peut se représenter informatiquement par le tableau suivant :

| | | | |
|---|---|---|---|
| 0 | 0 | - | - |
| 1 | 1 | - | - |
| 2 | a | 3 | 4 |
| 3 | b | 1 | 5 |
| 4 | b | 6 | 7 |
| 5 | c | 1 | 1 |
| 6 | c | 0 | 1 |
| 7 | c | 1 | 1 |

### 3.6. Algorithme de réduction des BDD

L'algorithme de réduction d'un BDD a pour objectif de supprimer les sommets redondants et les sous-graphes isomorphes. Le principe de algorithme est le suivant :

Faire id(v) = 0 pour chaque feuille v avec valeur (v) = 0 ;

Faire id(v) = 1 pour chaque feuille v avec valeur (v) = 1 ;
Initialiser le ROBDD avec 2 feuilles avec id =0 et id = 1 ;
Nextid = 1; /* Nextid = identificateur suivant disponible */
Pour (i = n to 1 avec i=i-1) {
V(i) = { v ∈ V / niveau(v) = i };
Pour chaque v ∈ V(i) {
if (id(gauche(v)) = id (droit(v)) { /* sommet redondant */
id(v) = id(gauche(v))
enlever v de V(i)}
else
cleF(v) = id (gauche(v)) , id (droit(v)) } /* la clef est définie comme la paire d'identificateurs des fils*/
ancienne_clef = 0 , 0;
Pour chaque v ∈ V(i) trié selon la clef {
if cleF(v) = ancienne_clef
id(v) = nextid /* le graphe de racine v est redondant */
else {
Nextid++;
id(v)=Nextid;
ancienne_clef = clef (v);
ajouter v au ROBDD avec des arcs allant aux sommets dont l'id est égale à ceux de gauche(v) et droit(v)) }}}

La figure 3.11 donne un exemple d'application de cet algorithme de réduction d'un BDD.

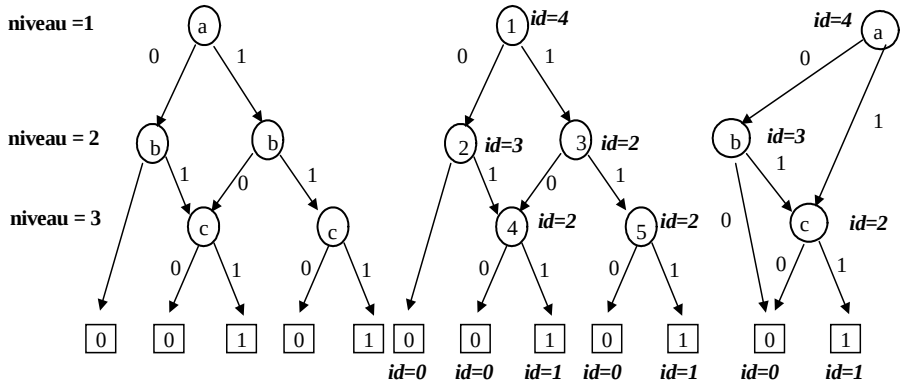



**Figure 3.11.** *Construction du ROBDD de la fonction f = (a+b)c*

## 3.7. Applications directes des BDD

### *3.7.1. Evaluation d'une fonction représentée sous forme de BDD*

Etant donnée une assignation des variables d'entrées d'une fonction f, la valeur de la fonction f peut être trouvée en traversant le BDD depuis la racine jusqu'à une feuille en empruntant la branche gauche ou droite de

chaque sommet selon la valeur de la variable d'entrée correspondant. Dans le pire cas, le BDD correspondant à une fonction f de n variables est composé de $2^n$-1 sommets. De plus, le nombre de branches entre la racine et une feuille est au plus égal à n. La complexité d'une procédure d'évaluation d'une fonction, c'est à dire de traversée d'un BDD depuis sa racine jusqu'à une feuille est O(n).

La représentation sous forme de BDD d'une fonction f permet de trouver rapidement une combinaison des entrées telles que f=0 ou telle que f=1. La procédure permettant de trouver une solution x de f(x) = a (a = 0 ou 1) est en O(n).

### *3.7.2. Equivalence de 2 fonctions*

Puisque pour un ordre donné le ROBDD d'une fonction est unique, pour que 2 fonctions soit identiques, il suffit que leurs ROBBDs respectifs soient isomorphes.

### *3.7.3. Cofacteurs*

Pour obtenir le cofacteur par rapport à $x_i$ il suffit de supprimer le sommet $x_i$ et de lier les sommets pointant sur $x_i$ au fils droit de $x_i$.

Pour obtenir le cofacteur par rapport à $x_i$ il suffit de supprimer le sommet $x_i$' et de lier les sommets pointant sur $x_i$ au fils gauche de $x_i$

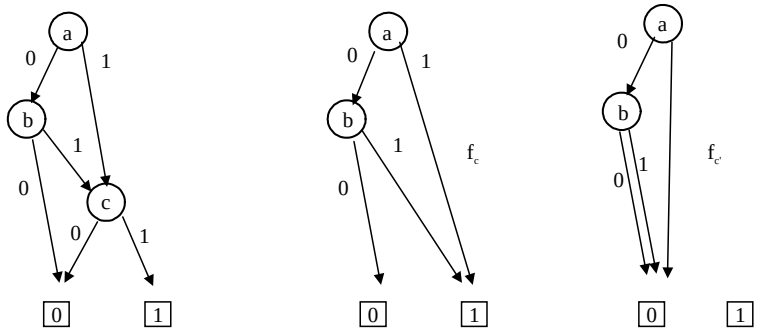



**Figure 3.12.** *BDD des cofacteurs*

### *3.7.4. Preuve de tautologie*

Une fonction simple est une tautologie si elle est toujours égale à 1.

Démontrer qu'une fonction est une tautologie est relativement simple en élaborant l'arbre de décision binaire de la fonction. En effet, une condition nécessaire et suffisante pour qu'une fonction soit une tautologie est que tous les sommets terminaux (feuilles) de l'arbre de décision binaire vaillent 1 (aucun sommet terminal à 0).

Exemple : La fonction f(a,b,c) = a'.b.c + a.b.c + b.c' + b' est une tautologie car tous les sommets terminaux de l'arbre de décision binaire sont à 1.

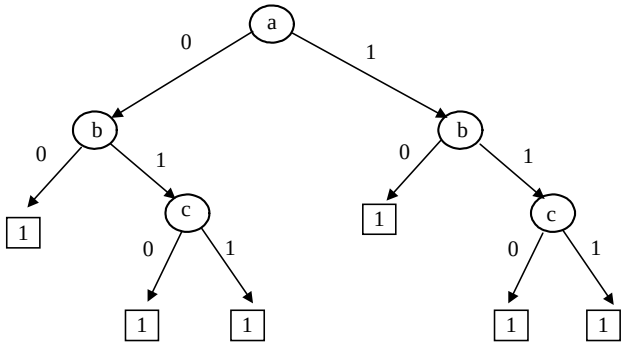



**Figure 3.13.** *Arbre de décision binaire de la fonction f = a.b.c + a.b.c + b.c + b*

Pour démontrer qu'une fonction est ou n'est pas une tautologie, on peut raisonner de la manière suivante:

Si l'un des cas suivants est détecté, il est possible d'arrêter le traitement, la fonction n'est pas une tautologie.

- Toutes les variables de F sont monoformes,
- Une variable monoforme est présente dans tous les monômes,
- La somme des tailles (dans le sens nombre de points couverts) des monômes est inférieure à 2n. (rappel : un monôme de p termes couvre 2(n-p) points)

Si aucune des conditions précédentes n'est remplie, réaliser l'arbre de décision binaire. Pour simplifier le traitement, on pourra appliquer le théorème suivant :

Théorème : Soit f une couverture monoforme par rapport à une variable x (f=x.g + h) , h le sous-ensemble de f ne dépendant pas de x. f est une tautologie si et seulement si h est une tautologie.

Preuve : Soit f monoforme en x: f=x.g + h

(1) : h = 1 => f = 1 donc h tautologie => f tautologie

(2) : f tautologie => f = 1 quelque soit la valeur de x

Si x=0, f = h => h =1

Ainsi, on pourra également considérer les variables de la manière suivante:

- Si des monômes sont composés d'une seule variable prendre ces variables en premier,
- Si une variable x apparaît toujours sous sa forme directe, ne pas continuer le graphe sur la branche 1 ($f = 1 <=> f_{x'} = 1$) puisque $f = x.f_x + f_{x'}$ avec $f_{x'}$ indépendant de x.
- Si une variable x apparaît toujours sous sa forme complémentée, ne pas continuer le graphe sur la branche 0 ($f = 1 <=> f_x = 1$) puisque $f = f_x + x'.f_{x'}$ avec $f_x$ indépendant de x.

Exemple1 : Soit la fonction f(a,b,c) = a'.b.c + a.b.c + b.c' + b'.

- Il existe des variables biformes
- Aucune variable monoforme n'est présente dans tous les monômes
- La somme des tailles des monômes (1+1+2+4=8) n'est pas inférieure à $2^n$.

L'arbre de décision binaire doit être réalisé. Un monôme est constitué d'une seule variable (b') => b sera pris en premier. Aucune variable n'apparaît uniquement sous sa forme normale ou uniquement sous sa forme complémentée.

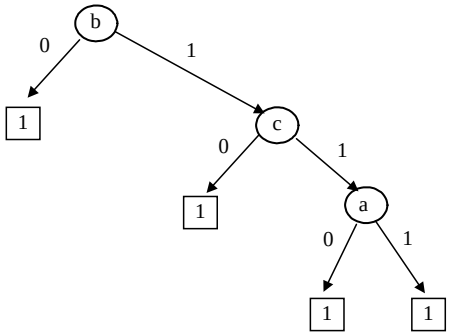



**Figure 3.14.** *Arbre de décision binaire de la fonction f = a'.b.c + a.b.c + b.c' + b'*

Exemple2 : Soit la fonction f(a,b,c) = a.b' + b.c + a.b.c' + a.c

- Il existe des variables biformes
- Aucune variable monoforme n'est présente dans tous les monômes
- La somme des taille des monômes (2+2+2+2=8) n'est pas inférieure à $2^n$.

L'arbre de décision binaire doit être réalisé. La variable a est monoforme directe, ne pas développer suivant la branche 1.

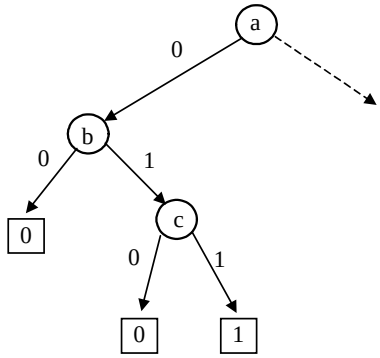



**Figure 3.15.** *Arbre de décision binaire la fonction f = a.b' + b.c + a.b.c' + a.c*

Exemple3 : Soit la fonction f(a,b,c) = a.b' + b' + b.c + b.c'

- Il existe des variables biformes.
- Aucune variable monoforme n'est présente dans tous les monômes.
- La somme des taille des monômes (1+1+2+4=8) n'est pas inférieure à $2^n$.

L'arbre de décision binaire doit être réalisé. La variable a est monoforme directe, ne pas développer suivant la branche 1.

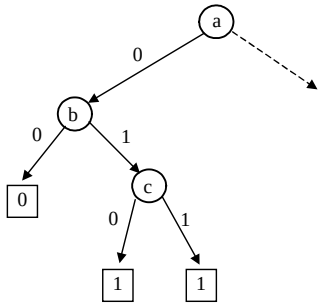



**Figure 3.16.** *Arbre de décision binaire de la fonction f = a.b' + b' + b.c + b.c'*

### *3.7.5. Test d'inclusion*

Savoir si un monôme est inclus dans l'expression de la fonction, c'est à dire couvert par une fonction, peut être un problème complexe. Le théorème d'inclusion permet de ramener ce problème à une problème de preuve de tautologie sur des cofacteurs.

Rappel du théorème d'inclusion : Une expression F contient un monôme m si et seulement si le cofacteur de F par rapport à m ($F_m$) est une tautologie.

$$m \subset F \quad \Leftrightarrow \quad F_m = 1$$

Exemple : Soit la fonction f(a,b,c) = a.b + a.c + a'

Cherchons à savoir si le monôme "bc" est couvert par cette expression. La construction du BDD de la fonction $f_{bc}$ représenté sur la figure 3.17 nous permet de conclure immédiatement à l'inclusion (comme confirmé par lz table de Karnaugh)

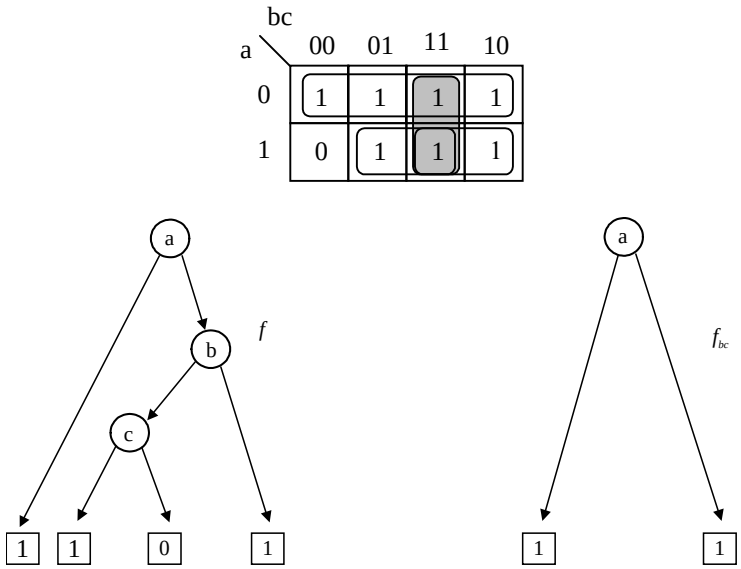



**Figure 3.17.** *Arbre de décision binaire de la fonction $f_{bc}$*

### 3.8. Représentation des multi-fonctions

La représentation des multi-fonctions peut se faire soit en réalisant un BDD par fonction, soit on partageant des sous-arbres. Cette dernière possibilité peut conduire à un gain de place conséquent.

Exemple :

f1(a,b) = a. b'

f2(a,b) = a ⊕ b

f3(a,b) = b'

f4(a,b) = a + b'

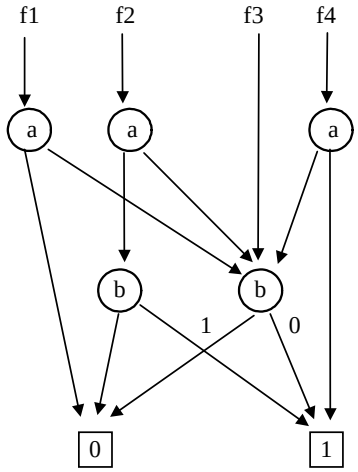



**Figure 3.18.** *BDD d'une multi-fonction*

La représentation informatique d'un tel BDD peut être :

| | | | | |
|---|---|---|---|---|
| | 0 | - | - | 0 |
| | 1 | - | - | 1 |
| 2 | b | 0 | 1 | |
| 3 | b | 1 | 0 | f3 |
| 4 | a | 0 | 3 | f1 |
| 5 | a | 2 | 3 | f2 |
| 6 | a | 3 | 1 | f4 |

### 3.9. Manipulation des ROBDDs : fonction ITE

On peut manipuler simplement les ROBDD à l'aide de la fonction ITE (f,g,h) (if then else) définie par :

ITE (f,g,h) = f.g + f'.h (i.e. If f Then g Else h)

Soit x la première variable (i.e. Ord(x) = 1) de f,g,h.

Soit z = ITE (f,g,h). La fonction z est associée au sommet de variable x et dont les fils implantent :

ITE($f_x$, $g_x$, $h_x$) et ITE ($f_{x'}$, $g_{x'}$, $h_{x'}$).

On a : $z = x.z_x + x'.z_{x'}$

$= x.(f .g + f' h)_x + x'.(f.g + f'h)_{x'}$

$= x.(f_x. g_x + f'_x h_x) + x' (f_{x'}.g_{x'} + f'_{x'}.h_{x'})$

$= ITE (x, ITE (f_x, g_x, h_x), ITE (f_{x'}, g_{x'}, h_{x'}))$

Identités remarquables :

ITE (f,1,0) = f

ITE (1,g,h) = g

ITE (0,g,h) = h

ITE (f,g,g) = g

ITE (f,0,1) = f'

De plus toutes les opérations habituelles peuvent être traduites en terme d'opérateur ITE :

| | | |
|---|---|---|
| 0 | => | 0 |
| 1 | => | 1 |
| f | => | f |
| f' | => | ITE (f,0,1) |
| f.g | => | ITE (f,g,0) |
| f.g' | => | ITE (f,g',0) |
| f'g | => | ITE (f,0,g) |
| f ⊕ g | => | ITE (f,g',g) |
| f + g | => | ITE (f,1,g) |
| (f+g)' | => | ITE (f,0,g') |
| f + g' | => | ITE (f,1,g') |
| f' + g | => | ITE (f,g,1) |
| (f.g)' | => | ITE (f,g',1) |

Exemple : Construction du ROBDD à l'aide de la fonction ITE.

Soit f(a,b,c) = a.c + b.c et l'ordre (a,b,c). Soit les ROBDDs de a , b et c sur la figure suivante 3.19.

Le ROBDD de a.c est calculé comme ITE (a , c, 0).

Le ROBDD de b.c est obtenu par ITE (b, c, 0)

Le ROBDD de f est obtenu par :

f = ITE (ac, 1, bc)

= ITE (a, ITE(c, 1, b.c), ITE (0,1,b.c))

= ITE (a, c, b.c).

La variable "top" est a , la branche gauche est le ROBDD de b.c et la branche droite est le ROBDD de c. De plus puisque il y a dans le ROBBD de b.c un sommet c, ce sommet est utilisé pour noter la branche gauche. Le résultat est décrit sur la figure 3.19.

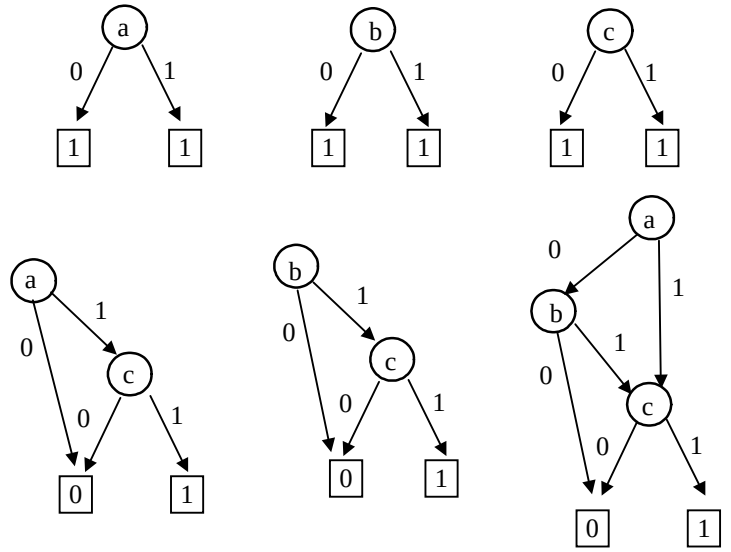



**Figure 3.19.** *ROBDDs de a , b et c. ROBDDs de a.c, de b.c et de f = a.c + b.c*

Algorithme :

```
ITE(f,g,h) {
  if (cas terminal)  /* c-àd 0 ou 1 figurant dans la table calculée */
        return ( r = résultat trivial)
  else    {
        if ( la table-calculee a une entrée {(f,g,h),r} )   /* déjà calculé */
              return (r à partir de la table-calculée)
        else {
              x = variable top de f, g, h ;
              t = ITE (fx, gx, hx);
              e = ITE (fx', gx', hx');
              if ( t == e)          /* fils gauche et droit isomorphes */
                    return (t);
              r = trouve-ou-ajoute-dans-Table (x, t, e); /* ajout r à table*/
              mise-a-jour-Table-calculee avec {(f,g,h), r};
              return (r);
              }
        }
  }
}
```

Cet algorithme utilise deux tables : Table et Table_calculee. Table est un tableau dont chaque ligne représente un sommet (c-à-d le tableau précédent). Ses colonnes sont l'identificateur du sommet, le nom de la variable associée , l'identificateur du fils gauche et l'identificateur du fils droit. Table-calculée est utilisée pour accélérer l'algorithme. Elle mémorise pour chaque triplet (f,g,h) le sommet implantant ITE (f,g,h).

Exemple : la Table correspondant à l'exemple de la Figure 3.20. est la suivante :

| | Clef | | |
|---|---|---|---|
| Identificateur | Variable | Fils gauche | Fils droit |
| 4 | a | 3 | 2 |
| 3 | b | 0 | 2 |
| 2 | c | 0 | 1 |

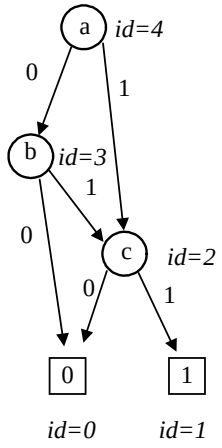



**Figure 3.20.** *ROBDD et table-unique*

### 3.10. Ordre des variables

L'ordre peut beaucoup influer sur le taille (nombre de nœuds) du BDD.

La recherche de l'ordre optimal est un problème NP-complet. En pratique on peut rechercher l'ordre optimal jusqu'à 20 variables. Pour plus de variables on utilise des heuristiques

Règles empiriques : Prendre en priorité

1. Les variables qui contrôlent le plus la fonction

- Variables apparaissant sous la même forme dans tous les monômes
- Variables constituant un monôme à elles seules
- Variables d'occurrence maximale

2. les groupes de variables ayant une relation "proche".

Exemple : $f = x_1x_2 + x_3x_4 + x_5x_6 + x_7x_8$

Si on prend l'ordre $x_1, x_2, x_3, ..., x_8$ => 8 sommets pour 8 variables

Si on prend l'ordre $x_1, x_8, x_2, x_7, ....$ => 30 ($2^{n/2+1}-2$) sommets

Exemple : un multiplexeur 8 voies (a0,...,a7) avec 3 bits de contrôle (c0,c1,c2).

Ordre c0,c1,c2,a0,a1, ..., a7 => 7+8 sommets

Ordre a0,a1,...,a7,c0,c1,c2 => $2^{11}$ sommets

### 3.11. Comparaison avec d'autres représentations

Le tableau suivant donne la complexité de certains algorithmes en fonction du type de représentation des fonctions logiques.

| | *SIGMA - PI* | *Galois* | *BDD* |
|---|---|---|---|
| *non F* | $O( \|f\|^{\|f\|1/2+1})$ | O (1) | O (1) |
| *f => g* | $O( (\|f\|+\|g\|)^{(\|f\|+\|g\|)\ 1/2+1})$ | O (\|f\|*\|g\|) log(\|f\|*\|g\|)) | O (\|f\|*\|g\|) |
| *f <=> g* | $O( (\|f\|+\|g\|)^{(\|f\|+\|g\|)\ 1/2+1})$ | O (\|f\|+\|g\|) log(\|f\|+\|g\|)) | O (\|f\|*\|g\|) |
| *f et g* | O (\|f\|*\|g\|) | O (\|f\|*\|g\|) log(\|f\|*\|g\|)) | O (\|f\|*\|g\|) |
| *f ou g* | O (\|f\|+\|g\|) | O (\|f\|*\|g\|) log(\|f\|*\|g\|)) | O (\|f\|*\|g\|) |
| Satisfaction | *NP-complet* | O (n) | O (n) |
| *Tautologie* | *NP-complet* | O (1) | O (1) |

Dans le cas d'une représentation par BDD, dans la majorité des cas, la borne supérieure du nombre de nœuds n'est pas atteinte. Ceci justifie en fait l'efficacité des BDD comparé à d'autres représentations.