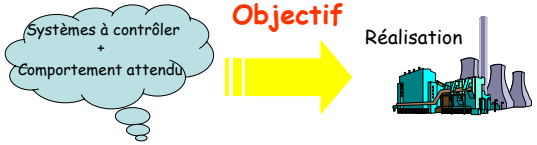
Systèmes à événements discrets

Partie I

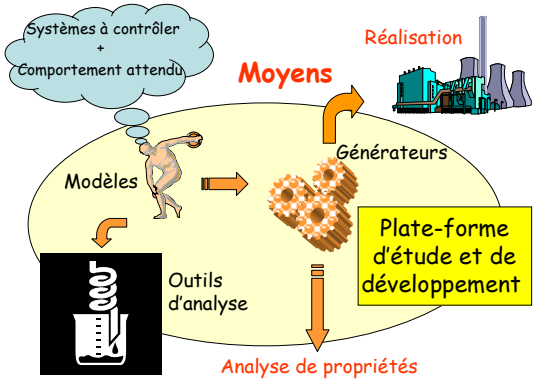
Systèmes Binaires

INFORMATIQUE INDUSTRIELLE



2

INFORMATIQUE INDUSTRIELLE



3

Syntaxe : Comment dire …

- Langages naturels
  - Français
  - Anglais
- Formalismes
  - Symboles
  - Graphiques
  - Schémas
- Langages formels
  - Langages de programmation
  - Mathématiques

4

Sémantique : Que dit-on ?

- Langages naturels
  - Très complexes
  - Beaucoup d'ambiguïtés
- Formalismes
  - Signification de chaque symbole ?
  - Signification de leurs associations ?
- Langages formels
  - Sémantique +/- comprise du langage
  - Mathématiques

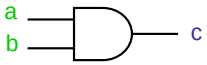
5

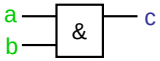
Logique Binaire

- Les mathématiques utiles sont simples
  - Algèbre binaire (c'est une alg. de Boole)
  - Automates et FSM
  - Logiques (ici ce peut être plus difficile)
- Les formalismes nombreux
  - Tables, tableaux (dont Karnaugh)
  - Graphes (dont graphes états-transitions)
  - Schémas logiques
- Langages
  - Expressions booléennes
  - HDL
  - Langages de programmation

6

Formalismes utilisés en logique (1)





a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

Table de vérité

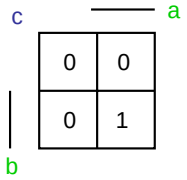


Tableau de Karnaugh

7

Formalismes utilisés en logique (2)

Formes textuelles

c = a . b

$c = a \wedge b$

c = a & b

c = **and**(a,b)

8

Formalismes utilisés en logique (3)

Formes canoniques

$$c = 0.\bar{a}.\bar{b} + 0.\bar{a}.b + 0.a.\bar{b} + 1.a.b$$

Pour l'ordre a,b :

$c = a.b = m_3$ 1° forme canonique ou Forme normale disjonctive

$$c = (a+b) \bullet (a+\bar{b}) \bullet (\bar{a}+\bar{b}) = M_0 \bullet M_1 \bullet M_2$$

2° forme canonique ou Forme normale conjonctive

Formalismes utilisés en logique (4)

pour $c = 0.\bar{a}.\bar{b} + 0.\bar{a}.b + 0.a.\bar{b} + 1.a.b$ ordre a,b

signature

$m_0 \quad m_1 \quad m_2 \quad m_3$

$F_c = (0,0,0,1)$

Sous-ensemble de B^n $\quad c \leftrightarrow \{(11)\} \subset B^2$

Cas particulier des fonctions symétriques

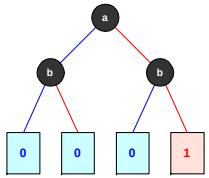
F est symétrique par rapport à xi et xj si $F(x_0,\cdots,x_i,\cdots,x_j,\cdots,x_{n-1}) = F(x_0,\cdots,x_j,\cdots,x_i,\cdots,x_{n-1})$

0 variable à 1
1 variable à 1
2 variables à 1

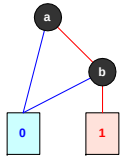
$S_c = (0,0,1)$

Formalismes utilisés en logique (5)

Formes graphiques



Arbre de décision de c



BDD de c

Binary Decision Diagrams

Les BDD sont des graphes orientés acycliques (DAG= Directed Acyclic Graphs) tels que

- le degré sortant des nœuds est au maximum 2,
- il existe un seul nœud racine (root) et deux nœuds pendants (terminals) 0 et 1.

Les arbres de décision (ou arbres de Shannon) sont des BDD.
Les plus utiles ne sont pas des arbres mais des DAGs.

Binary Decision Diagrams

Ordered BDD (OBDD): les variables sont toujours dans le même ordre.

Reduced BDD (RBDD):

- tout nœud avec deux enfants identiques est supprimé,
- deux nœuds associés à des BDD isomorphes sont fusionnés.

Reduced Ordered BDD (ROBDD) : Reduced + Ordered, appelés simplement BDD dans la suite.

ROBDD avec inverseurs : très courants dans les logiciels.

Zero-Suppressed BDD (zBDD) : très économiques pour les fonctions creuses

ROBDD (1)

Règles de réduction (rappel) :

- si $f=c.g + c'.g$ alors éliminer le nœud correspondant
- si deux nœuds ont des graphes isomorphes alors fusionner ces nœuds.

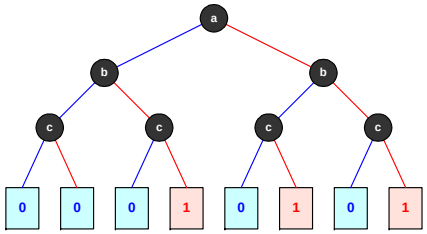
Conséquence: chaque nœud représente une fonction distincte

Théorème (Bryant 1986) : les ROBDD sont canoniques

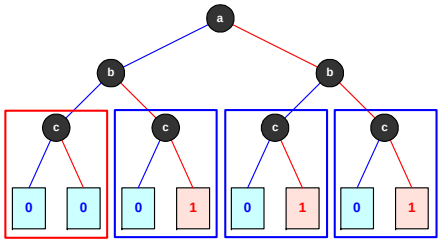
Dans un ROBDD il existe un ordre total $<$ sur les variables.
Si $v < w$ alors v apparaît plus « haut » que w dans le DAG.
On appelle Top variable d'une fonction f la variable associée à sa racine.

ROBDD (2)

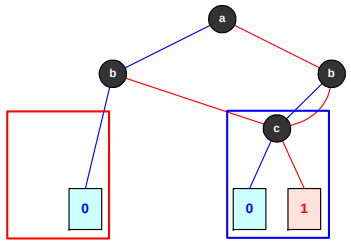
$$f(a,b,c) = b \bullet c + a \bullet c = m_3 + m_5 + m_7$$



ROBDD (3)

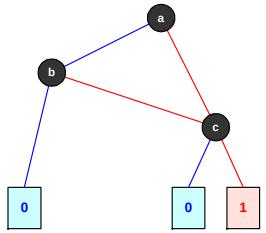


ROBDD (4)



17

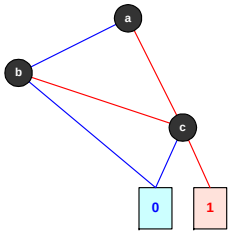
ROBDD (5)



18

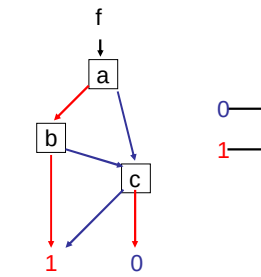
ROBDD (6)

$$f(a,b,c) = b \bullet c + a \bullet c = m_3 + m_5 + m_7$$

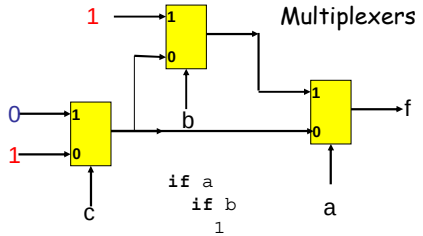


19

ROBDD (7)



Reduced Ordered BDD




```
if a
  if b
    1
  else if !b
    !c
  endif b
else if !a
  !c
endif a
```

20

Influence de l'ordre des variables

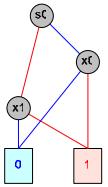
- Exemple d'un multiplexeur à k variables de sélection s_i et 2^k variables d'entrée x_j.
- Variables : $s_0, s_1, \ldots, s_{k-1}, x_0, x_1, \ldots, x_{2^k-1}$
- Soit $k+2^k$ variables (soit 20 pour k = 4)
- Arbre de Shannon

$1+2+\ldots+2^{k+2^k-1} = 2^{k+2^k}-1$ Sommets intermédiaires

Pour k=4 ceci donne 1 048 575

Influence de l'ordre des variables

- Exemple d'un multiplexeur à k variables de sélection s_i et 2^k variables d'entrée x_j .
- Ordre : $s_0, s_1, \ldots, s_{k-1}, x_0, x_1, \ldots, x_{2^k-1}$
- Complexité : $2^{k+1}+1$ sommets



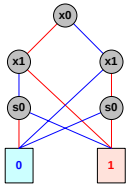
Soit

•5 nœuds pour k=1;

•33 pour k=4

Influence de l'ordre des variables

- Exemple d'un multiplexeur à k variables de sélection s_i et 2^k variables d'entrée x_j .
- Ordre : $x_0, x_1, \ldots, x_{2^k-1}, s_0, s_1, \ldots, s_{k-1}$
- Complexité : $2^{2^k+1}-1$ sommets



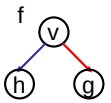
Soit

•7 nœuds pour k=1;

•131 071 pour k=4

Calculs sur BDD

Dans le cas général:

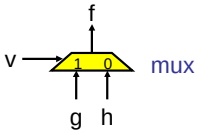


Écrit: (*v,g,h*)
Se lit: **si** *v* **alors** *g* **sinon** *h*

Opérateur ***ite*** (If Then Else)

$f = ite(v,g,h) = v.g + v'.h$

$g=f_v$ et $h=f_{v'}$ (cofacteurs de f)



Rappel:
ite peut réaliser n'importe quelle fonction de 2 variables

Construction de BDD (1)

Pour $F(x_1,x_2,\ldots,x_n)$ et pour l'ordre $x_1,x_2,\ldots,x_n$ et calculer `Build(F,1)`

```
function Build (F,k)
if F==0 then return Node0;
if F==1 then return Node1;
let v0 = Build(Fx_k',k+1);
let v1 = Build(Fx_k,k+1);
return CreateNode(x_k,v0,v1)
fin
```

25

Construction de BDD (2)

on suppose qu'il existe les fonctions:
`CheckExists()` et `Insert()`

```
function CreateNode(Var,F,G)
if F == G then return F;
if CheckExists(Var,F,G) then
     return the existing node
else Insert(Var,F,G);
      return the new node
endif
end
```

26

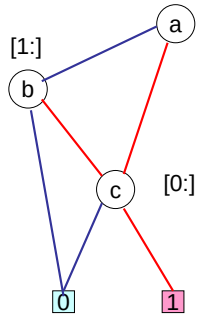
- `Build(b.c+a.c,1)`
  - `v0=Build(b.c,2)/*a=0*/`
    - `V0=Build(0,3) /*b=0*/` → Node 0
    - `V1=Build(c,3) /*b=1*/`
      - `V0=Build(0,4) /*c=0*/` → Node 0
      - `V1=Build(1,4) /*c=1*/` → Node 1
      - `CreateNode(c,0,1)` → Node [0:]
    - `CreateNode(b,0,[0:])` → Node[1:]
  - `v1=Build(b.c+c,2) /*a=1*/`
    - `V0=Build(c,3) /*b=0*/` → Node [0:]
    - `V1=Build(c+c,3) /*b=1*/` → Node [0:]
    - `CreateNode(b,[0:],[0:])` → Node [0:]
  - `CreateNode(a,[1:],[0:])`

27

Construction de BDD (4)

Création nœuds :

- Node 0
- Node 1
- Node [0:]
- Node [1:]



28

I.T.E

Formulation récursive de ***ite***

Soit v la variable Top pour f,g,h

ite (f,g,h) = ite(v,ite(f_v,g_v,h_v), ite($f_{v'}$,$g_{v'}$,$h_{v'}$))

Preuve :

ite(f,g,h) = f.g + f'.h

= v.(f.g + f'.h)$_v$ + v'.(f.g + f'.h)$_{v'}$

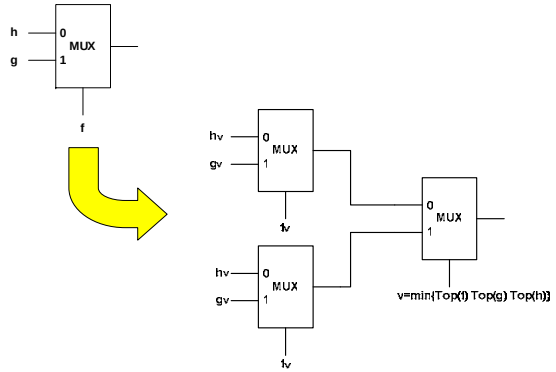
= v.(f_v .g_v+ f'_v.h_v) + v'.($f_{v'}$.$g_{v'}$ + $f'_{v'}$.$h_{v'}$)

= v.ite(f_v,g_v,h_v) + v'.ite($f_{v'}$,$g_{v'}$,$h_{v'}$)

= ite (v,ite(f_v,g_v,h_v), ite($f_{v'}$,$g_{v'}$,$h_{v'}$))

Application: comment calculer (récursivement) une opération entre deux BDD

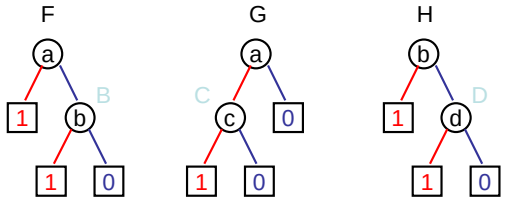
ITE (2)



Apply (1)

Soit F = a + b; G = a.c; H = b + d

Construire I = ite(F,G,H)



B, C, D sont des pointeurs intermédiaires

Apply (2)

I = ite(F,G,H)

= ite(a, ite(F_a, G_a, H_a), ite($F_{a'}$, $G_{a'}$, $H_{a'}$))

= ite(a, ite(1,C,H), ite(B,0,H))

= ite(a, C, ite(b, ite(B_b, 0_b, H_b), ite($B_{b'}$, $0_{b'}$, $H_{b'}$)))

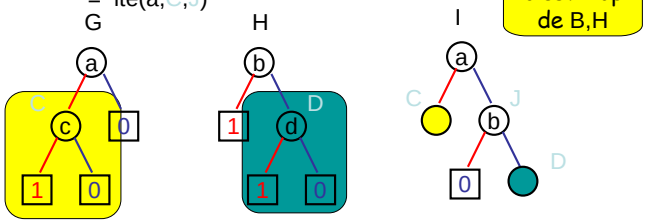
= ite(a, C, ite(b, ite(1,0,1), ite(0,0,D)))

= ite(a, C, ite(b, 0, D))

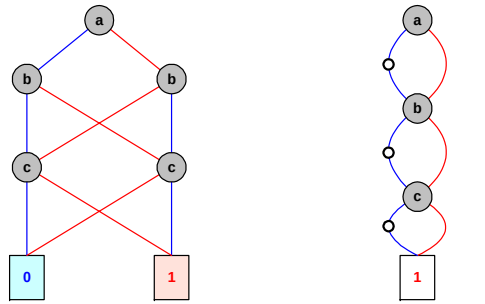
= ite(a,C,J)

a est Top de F,G,H

b est Top de B,H

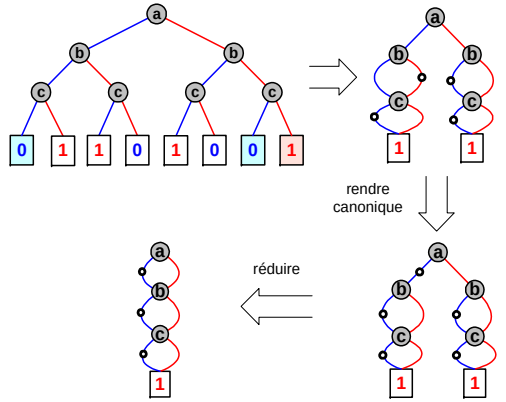


BDD à arcs complémentés



33

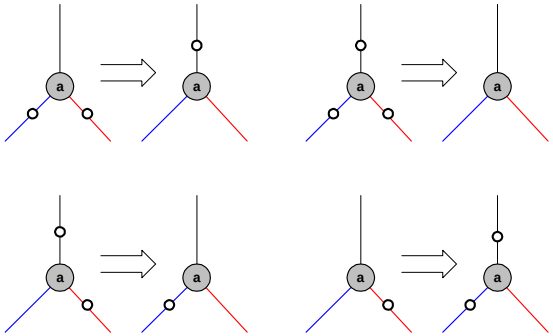
BDD à arcs complémentés (2)



34

BDD à arcs complémentés (3)

Pour rendre canonique



35

Zero Suppress BDD

1°) Éliminer tous les nœuds tels que la branche « then » pointe sur 0

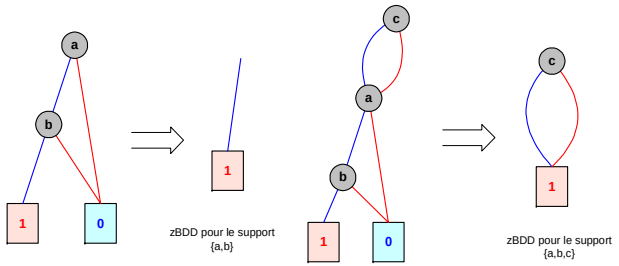
Connecter alors les arcs entrants au nœud « else »

2°) Partager les nœuds identiques

Pour être canonique pour un ordre des variables donné, il faut préciser l'ensemble support

36

Z-BDD : exemples



37

Symboles : Quiz

∅ Φ ϕ ϕ

0 1 0 0 ϕ -

Indifférent (don't care)

le plus simple !

38

ϕ

Signature de Kih-Oskh

Couverture de l'album Tintin "Les cigares du pharaon"

39