

Diagrammes de Décision Binaires

Alban Linard

2008-2009

Menu du jour

- 1 Introduction
- 2 Principes
- 3 Représentation du système
- 4 Outils
- 5 Conclusion

Pourquoi un cours sur une structure de données ?

Commençons par un exemple **simple** :

Modèle le quel ?

Algorithme le quel ?

Pourquoi un cours sur une structure de données ?

Commençons par un exemple **simple** :

Modèle Ping, 20 clients

Algorithme lequel ?

Pourquoi un cours sur une structure de données ?

Commençons par un exemple **simple** :

Modèle Ping, 20 clients

Algorithme Accessibilité

Pourquoi un cours sur une structure de données ?

Commençons par un exemple **simple** :

Modèle Ping, 20 clients

Algorithme Accessibilité (aucun état ne doit vérifier une propriété p)

Pourquoi un cours sur une structure de données ?

Commençons par un exemple **simple** :

Modèle Ping, 20 clients

Algorithme Accessibilité (aucun état ne doit vérifier une propriété p)
Nécessite de parcourir tous les états

Pourquoi un cours sur une structure de données ?

Commençons par un exemple **simple** :

Modèle Ping, 20 clients

Algorithme Accessibilité (aucun état ne doit vérifier une propriété p)
Nécessite de parcourir tous les états

Quelques nombres intéressants :

États un peu moins de 10^{20}

Mémoire (1 état = 8 octet)

Temps (10^9 états/seconde)

Pourquoi un cours sur une structure de données ?

Commençons par un exemple **simple** :

Modèle Ping, 20 clients

Algorithme Accessibilité (aucun état ne doit vérifier une propriété p)
Nécessite de parcourir tous les états

Quelques nombres intéressants :

États un peu moins de 10^{20}

Mémoire (1 état = 8 octet) 800.000.000 To

Temps (10^9 états/seconde)

Pourquoi un cours sur une structure de données ?

Commençons par un exemple **simple** :

Modèle Ping, 20 clients

Algorithme Accessibilité (aucun état ne doit vérifier une propriété p)
Nécessite de parcourir tous les états

Quelques nombres intéressants :

États un peu moins de 10^{20}

Mémoire (1 état = 8 octet) 800.000.000 To

Temps (10^9 états/seconde) 3.170 ans

Facile à résoudre, ce problème !

On a l'habitude :

- d'accélérer des traitements :
 - caches
 - structures de données adaptées (tri, hash, ...)
 - ...
- de gagner de la mémoire :
 - compression
 - partage
 - ...

Facile à résoudre, ce problème !

On a l'habitude :

- d'accélérer des traitements :
 - caches
 - structures de données adaptées (tri, hash, ...)
 - ...
- de gagner de la mémoire :
 - compression
 - partage
 - ...

Les deux en même temps ?

Ah, pas vraiment, en fait...

Il faut à la fois :

- un gain mémoire de plusieurs ordres de grandeur
- un gain temps de plusieurs ordres de grandeur

C'est quoi la solution miracle?

Une structure de données : les Diagrammes de Décision (Binaires) !
Binary Decision Diagrams, BDD [5]

C'est quoi la solution miracle ?

Une structure de données : les Diagrammes de Décision (Binaires) !
Binary Decision Diagrams, BDD [5]

Ce cours présente

- la structure de données
- comment représenter des états
- comment représenter des transitions
- comment générer un espace d'états

Que représente-t-on ?

Ce que l'on souhaite représenter :

Ce qu'un BDD représente :

Problèmes (abordés plus loin) :

Que représente-t-on ?

Ce que l'on souhaite représenter :

- Les états (état initial, états accessibles)
- Les transitions entre états

Ce qu'un BDD représente :

Problèmes (abordés plus loin) :

Que représente-t-on ?

Ce que l'on souhaite représenter :

- Les états (état initial, états accessibles)
- Les transitions entre états

Ce qu'un BDD représente :

- une fonction $\mathbb{B}^n \rightarrow \mathbb{B}$ *explicitement*

Problèmes (abordés plus loin) :

Que représente-t-on ?

Ce que l'on souhaite représenter :

- Les états (état initial, états accessibles)
- Les transitions entre états

Ce qu'un BDD représente :

- une fonction $\mathbb{B}^n \rightarrow \mathbb{B}$ *explicitement*
- exemple 1 : $f(a, b, c) = \bar{c} \wedge (\bar{a} \wedge b)$
- exemple 2 : $g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$

Problèmes (abordés plus loin) :

Que représente-t-on ?

Ce que l'on souhaite représenter :

- Les états (état initial, états accessibles)
- Les transitions entre états

Ce qu'un BDD représente :

- une fonction $\mathbb{B}^n \rightarrow \mathbb{B}$ *explicitement*
- exemple 1 : $f(a, b, c) = \bar{c} \wedge (\bar{a} \wedge b)$
- exemple 2 : $g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$
- (j'avoue, f et g sont identiques)

Problèmes (abordés plus loin) :

- Caractéristiques des modèles \leftrightarrow utilisation des BDDs ?

Que représente-t-on ?

Ce que l'on souhaite représenter :

- Les états (état initial, états accessibles)
- Les transitions entre états

Ce qu'un BDD représente :

- une fonction $\mathbb{B}^n \rightarrow \mathbb{B}$ *explicitement*
- exemple 1 : $f(a, b, c) = \bar{c} \wedge (\bar{a} \wedge b)$
- exemple 2 : $g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$
- (j'avoue, f et g sont identiques)

Problèmes (abordés plus loin) :

- Caractéristiques des modèles \leftrightarrow utilisation des BDDs ?
- Traduction du modèle en BDDs ?

Comment représenter explicitement une fonction ?

Tables de vérité :

$$f(a, b, c) = \bar{c} \wedge \overline{(\bar{a} \wedge b)}$$

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

Remarque

- les variables sont ordonnées

Comment représenter explicitement une fonction ?

Tables de vérité :

$$f(a, b, c) = \bar{c} \wedge \overline{(a \wedge b)}$$

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

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

Remarque

- les variables sont ordonnées

Comment représenter explicitement une fonction ?

Tables de vérité :

$$f(a, b, c) = \bar{c} \wedge (\overline{\bar{a} \wedge b})$$

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

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

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

Remarque

- les variables sont ordonnées

Comment compressor ?

Certaines parties de la table de vérité sont inutiles :

$$f(a, b, c) = \bar{c} \wedge \overline{(\bar{a} \wedge b)}$$

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

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

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

Comment compresser ?

Certaines parties de la table de vérité sont inutiles :

- 1 les lignes dont le résultat est 0 (ou 1)

$$f(a, b, c) = \bar{c} \wedge \overline{(\bar{a} \wedge b)}$$

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

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

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

Comment compresser ?

Certaines parties de la table de vérité sont inutiles :

- 1 les lignes dont le résultat est 0 (ou 1)

$$f(a, b, c) = \bar{c} \wedge \overline{(a \wedge b)}$$

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

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

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

Comment compresser ?

Certaines parties de la table de vérité sont inutiles :

- 1 les lignes dont le résultat est 0 (ou 1)

$$f(a, b, c) = \bar{c} \wedge \overline{(\bar{a} \wedge b)}$$

<i>a</i>	<i>b</i>	<i>c</i>	<i>f</i>
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

<i>c</i>	<i>b</i>	<i>a</i>	<i>f</i>
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Comment compressor ?

Certaines parties de la table de vérité sont inutiles :

- 1 les lignes dont le résultat est 0 (ou 1)
- 2 les cellules sans influence sur le résultat

$$f(a, b, c) = \bar{c} \wedge (\bar{a} \wedge b)$$

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

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

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

Comment compresser ?

Certaines parties de la table de vérité sont inutiles :

- 1 les lignes dont le résultat est 0 (ou 1)
- 2 les cellules sans influence sur le résultat

$$f(a, b, c) = \bar{c} \wedge (\bar{a} \wedge b)$$

<i>a</i>	<i>b</i>	<i>c</i>	<i>f</i>
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

<i>c</i>	<i>b</i>	<i>a</i>	<i>f</i>
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Comment compresser ?

Certaines parties de la table de vérité sont inutiles :

- 1 les lignes dont le résultat est 0 (ou 1)
- 2 les cellules sans influence sur le résultat

$$f(a, b, c) = \bar{c} \wedge (\bar{a} \wedge b)$$

<i>a</i>	<i>b</i>	<i>c</i>	<i>f</i>
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

<i>c</i>	<i>b</i>	<i>a</i>	<i>f</i>
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Comment compresser ?

Certaines parties de la table de vérité sont inutiles :

- 1 les lignes dont le résultat est 0 (ou 1)
- 2 les cellules sans influence sur le résultat

$$f(a, b, c) = \bar{c} \wedge (\bar{a} \wedge b)$$

<i>a</i>	<i>b</i>	<i>c</i>	<i>f</i>
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$g(a, b, c) = \bar{c} \wedge (a \vee \bar{b})$$

<i>c</i>	<i>b</i>	<i>a</i>	<i>f</i>
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Peut-on compressor plus ?

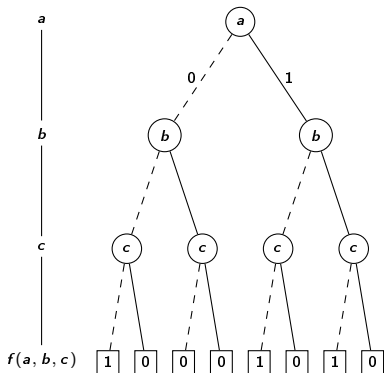
Curryfication

```
let f a b c = (not c) && not((not a) && b);;  
let f_a = f true;;  
let f_ab = f_a true;;  
let result = f_ab false;;
```

Peut-on compressor plus ?

Curryfication

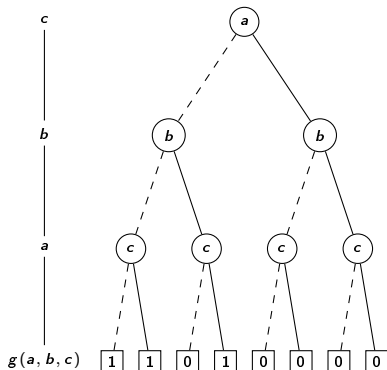
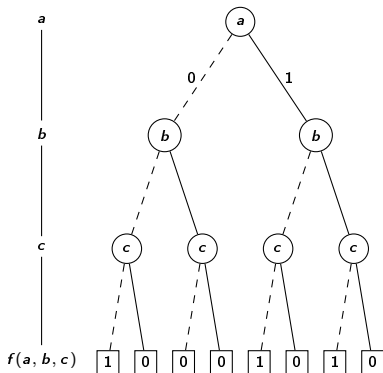
```
let f a b c = (not c) && not((not a) && b);;  
let f_a = f true;;  
let f_ab = f_a true;;  
let result = f_ab false;;
```



Peut-on compressor plus ?

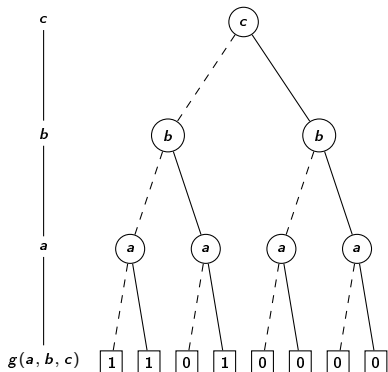
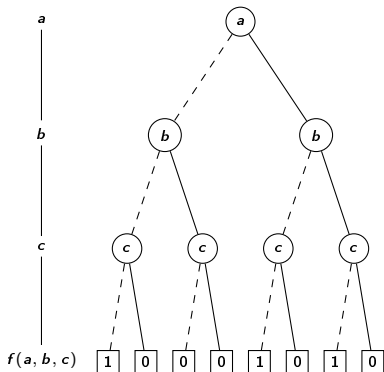
Curryfication

```
let f a b c = (not c) && not((not a) && b);;  
let f_a = f true;;  
let f_ab = f_a true;;  
let result = f_ab false;;
```



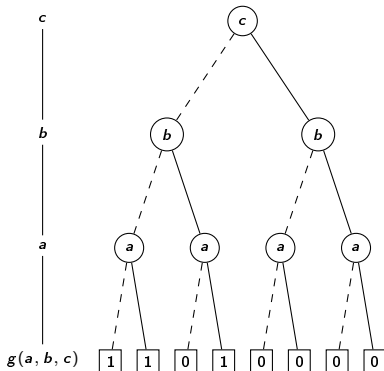
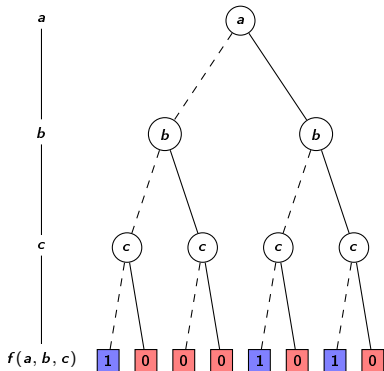
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



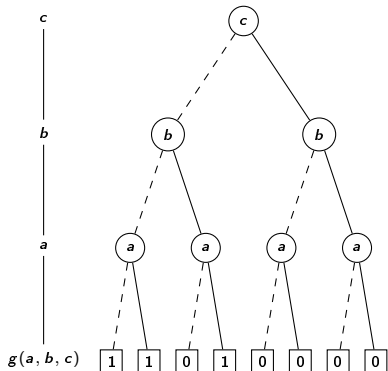
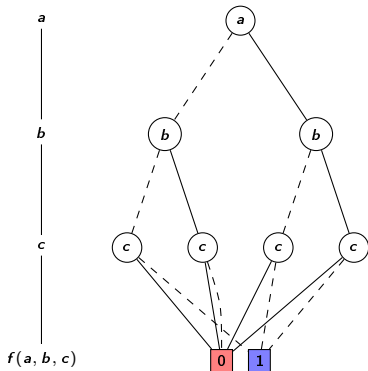
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



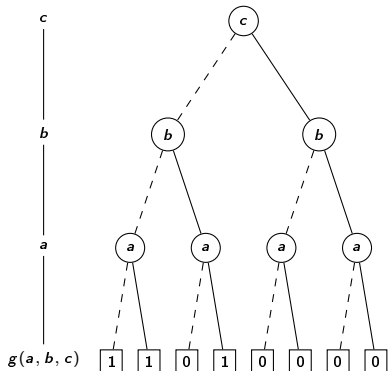
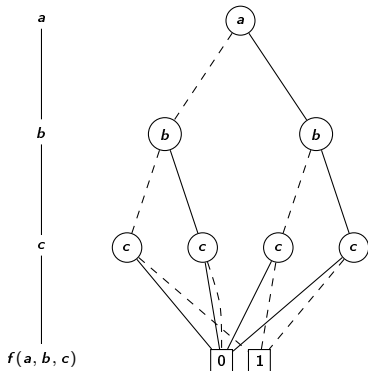
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



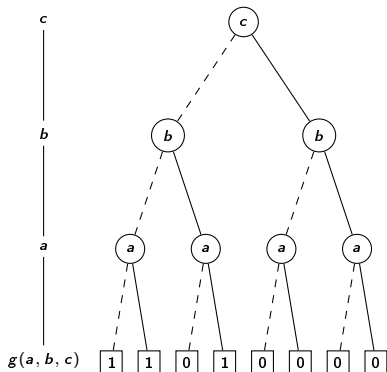
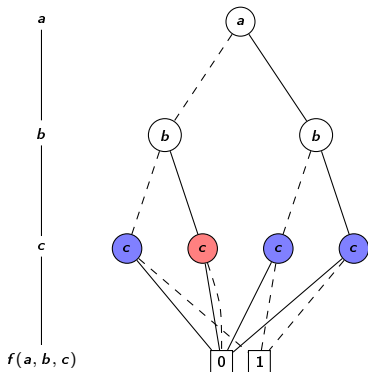
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



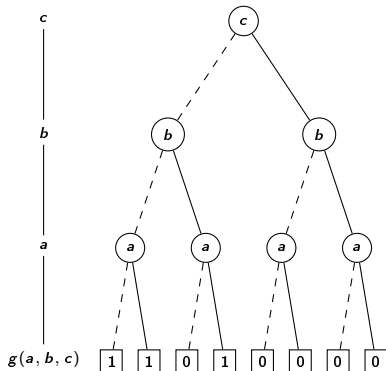
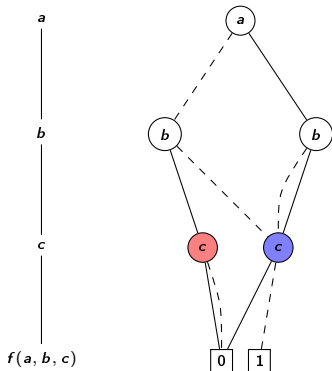
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



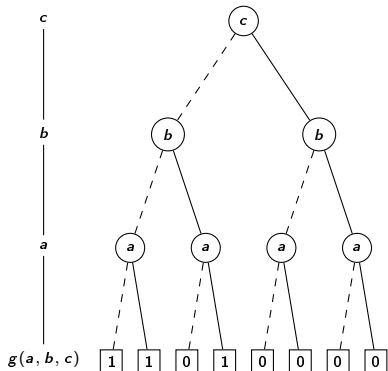
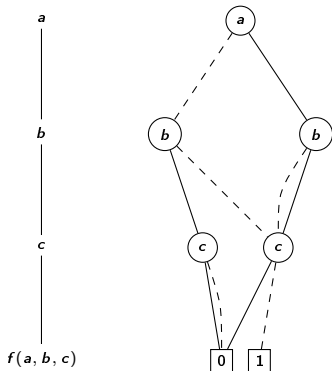
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



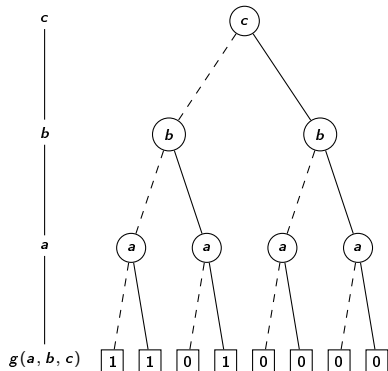
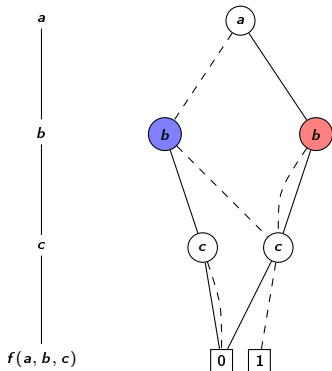
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



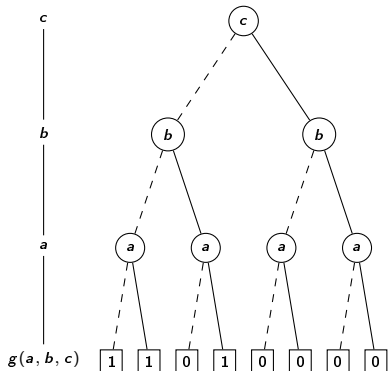
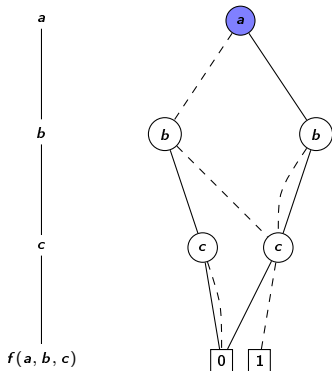
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



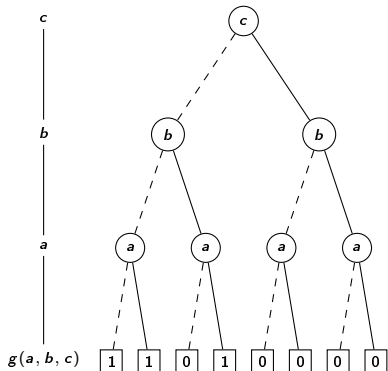
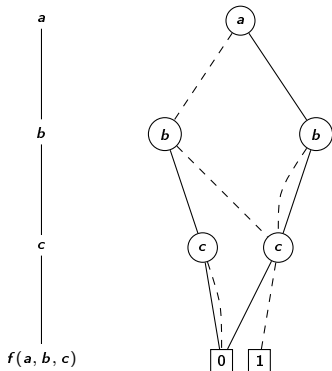
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



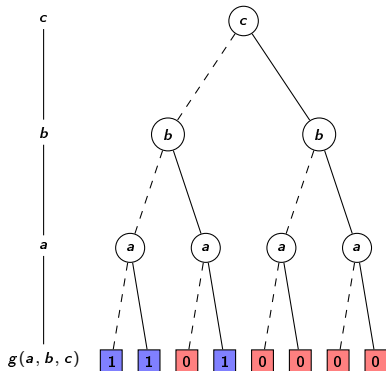
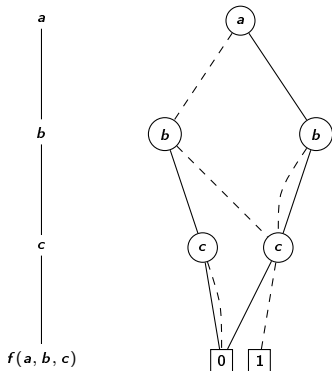
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



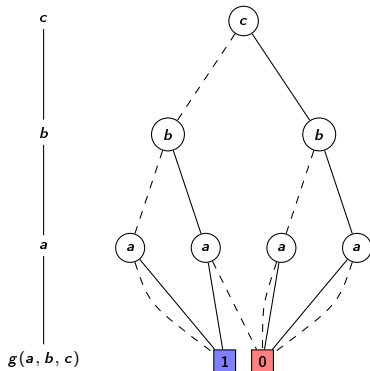
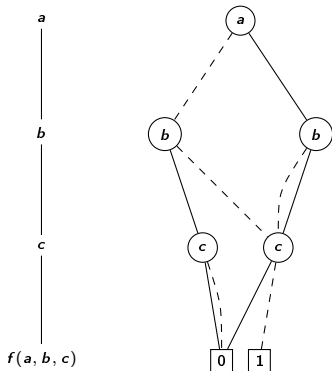
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



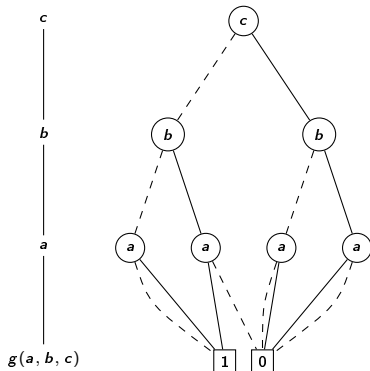
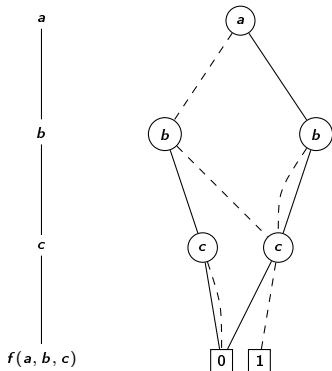
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



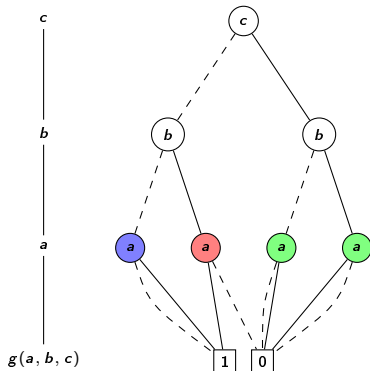
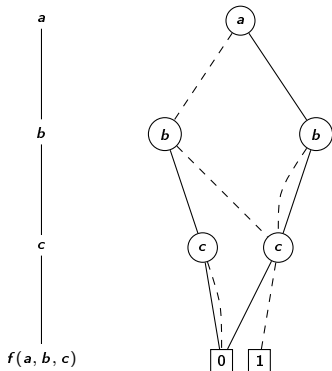
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



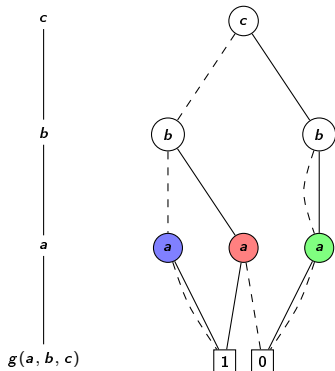
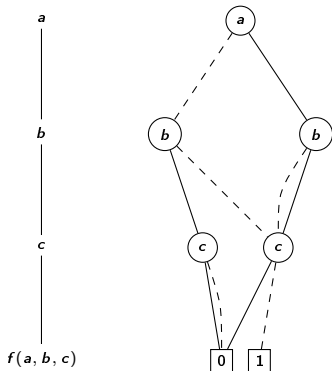
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



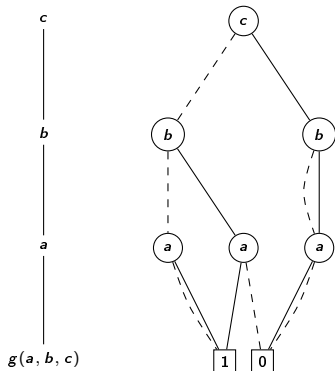
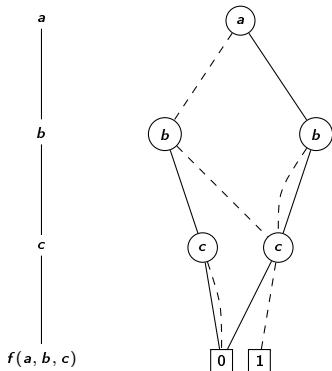
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



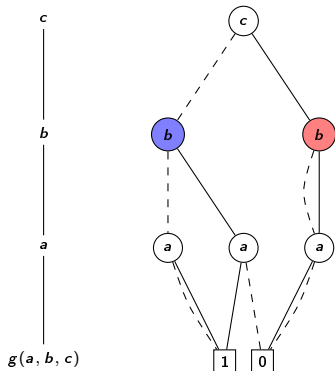
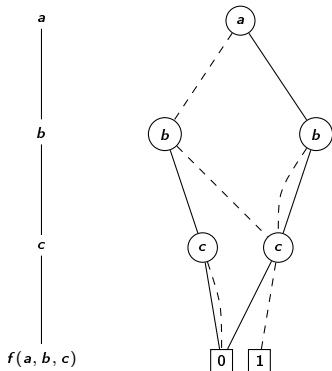
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



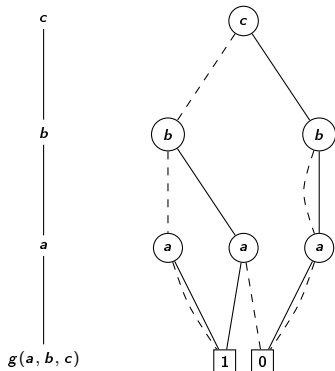
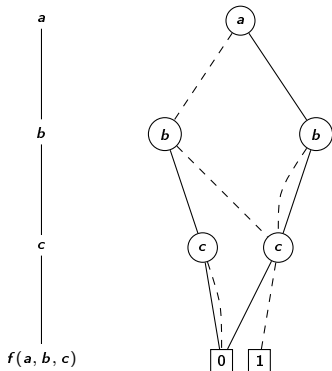
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



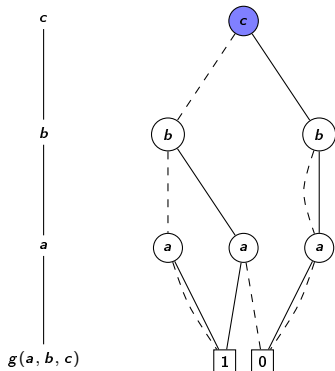
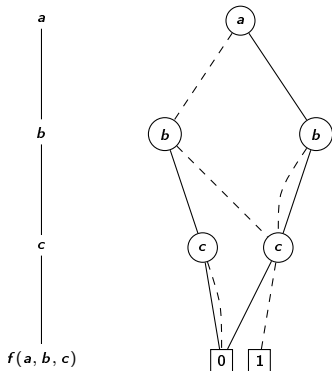
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut



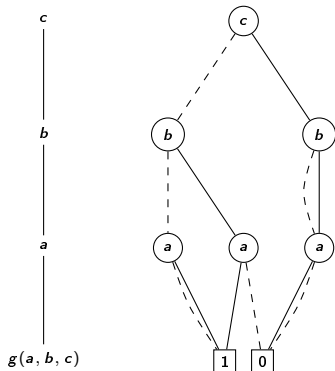
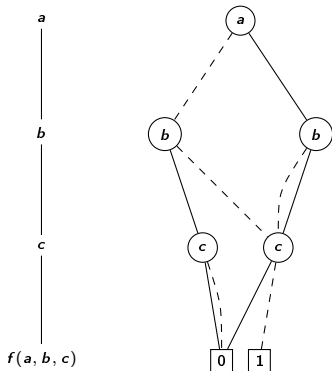
Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut

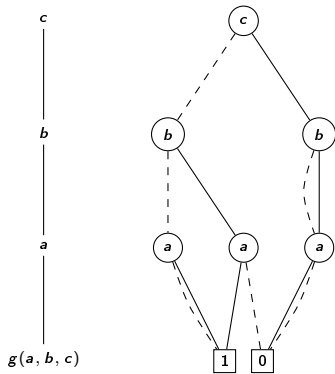
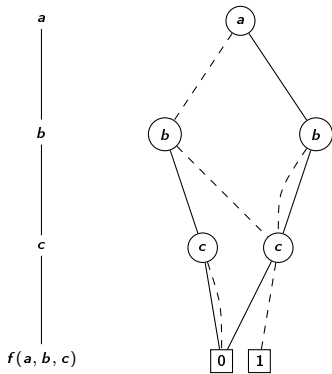


Encore plus ?

- Fusion des nœuds équivalents
- Récursivement, de bas en haut

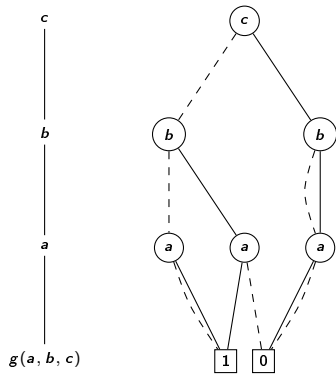
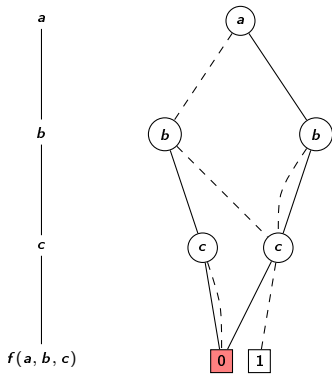


Et les compressions de la table ?



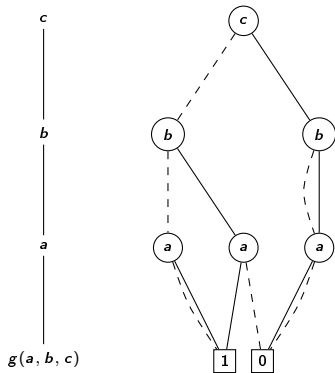
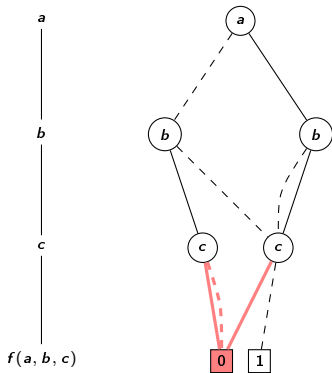
Et les compressions de la table ?

- Suppression des lignes donnant 0



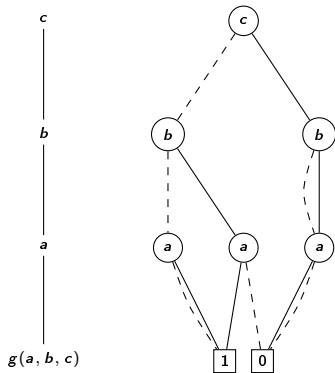
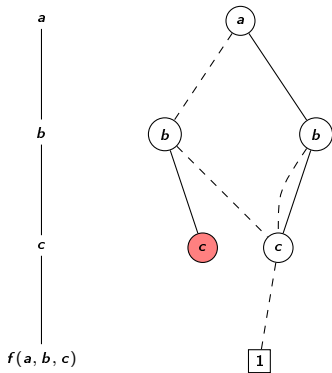
Et les compressions de la table ?

- Suppression des lignes donnant 0



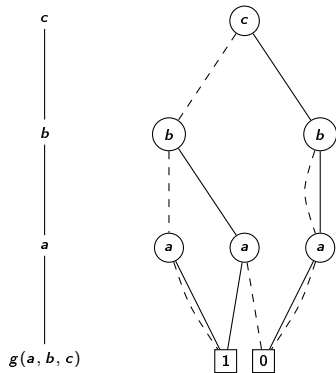
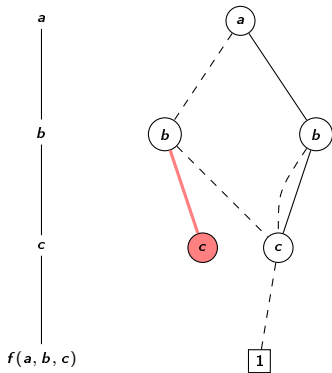
Et les compressions de la table ?

- Suppression des lignes donnant 0



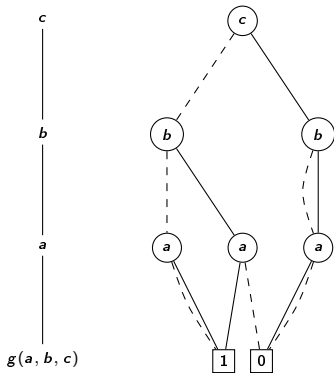
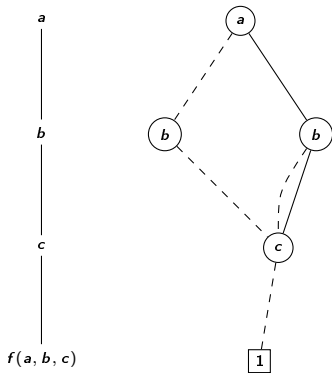
Et les compressions de la table ?

- Suppression des lignes donnant 0



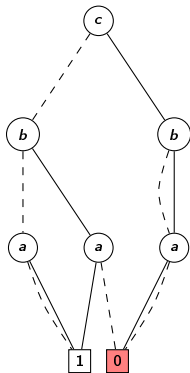
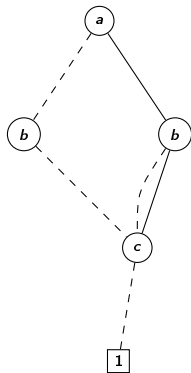
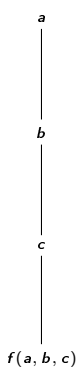
Et les compressions de la table ?

- Suppression des lignes donnant 0



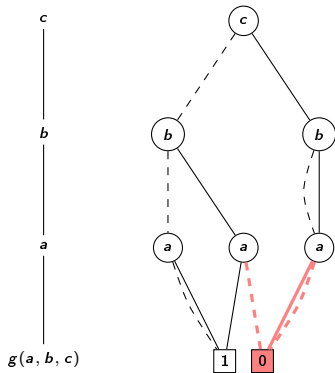
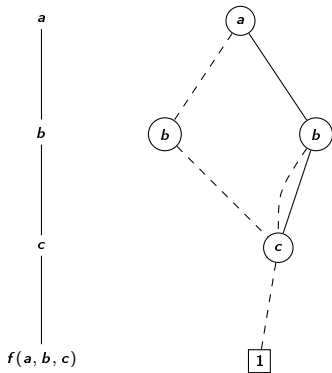
Et les compressions de la table ?

- Suppression des lignes donnant 0



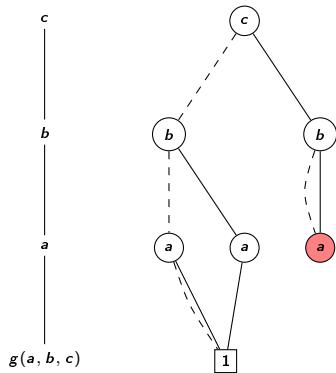
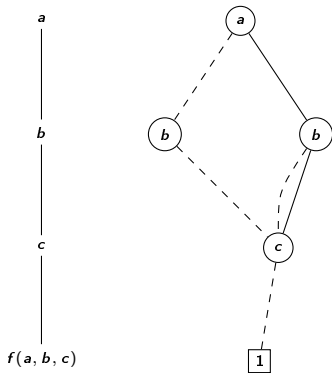
Et les compressions de la table ?

- Suppression des lignes donnant 0



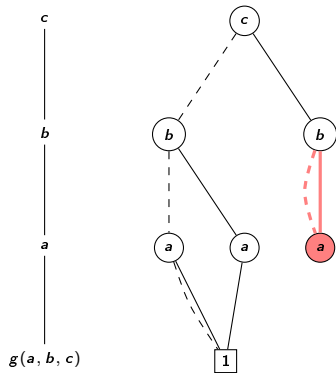
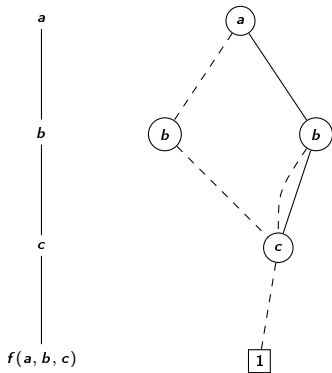
Et les compressions de la table?

- Suppression des lignes donnant 0



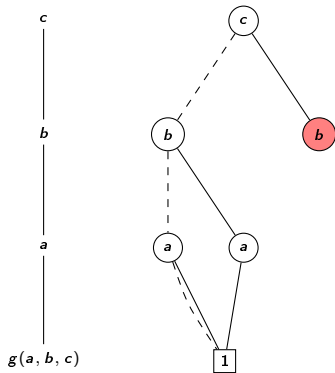
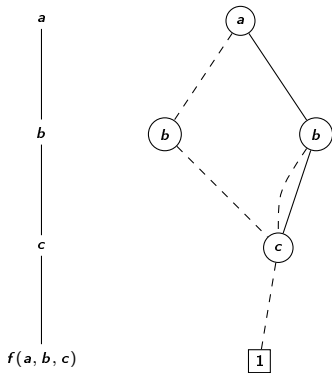
Et les compressions de la table ?

- Suppression des lignes donnant 0



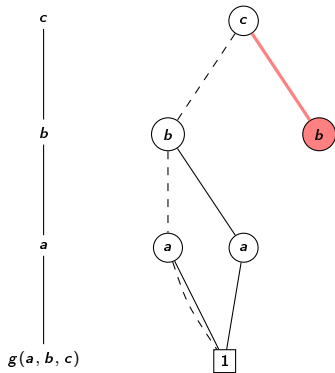
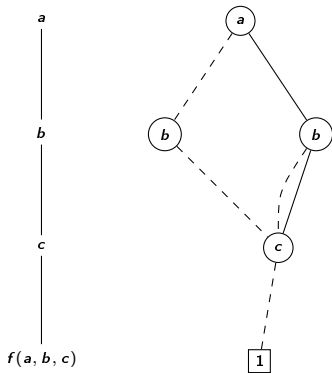
Et les compressions de la table ?

- Suppression des lignes donnant 0



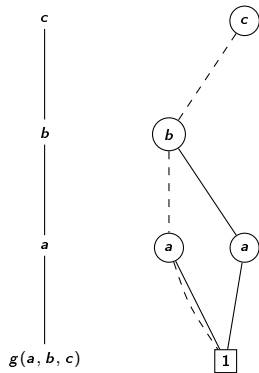
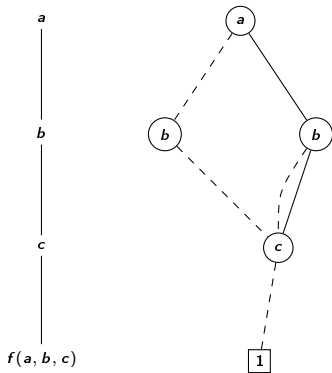
Et les compressions de la table ?

- Suppression des lignes donnant 0



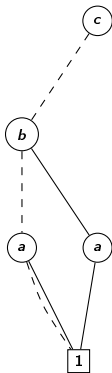
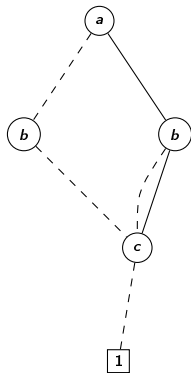
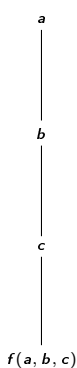
Et les compressions de la table ?

- Suppression des lignes donnant 0



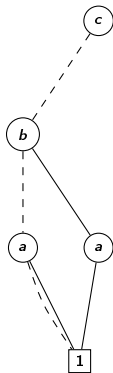
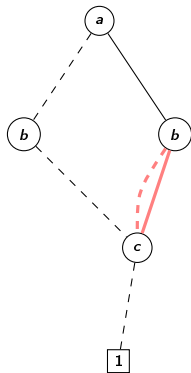
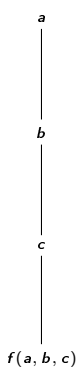
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)



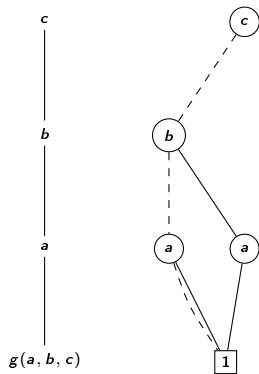
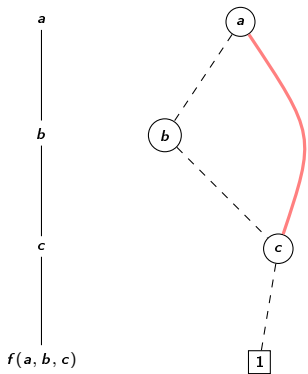
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)



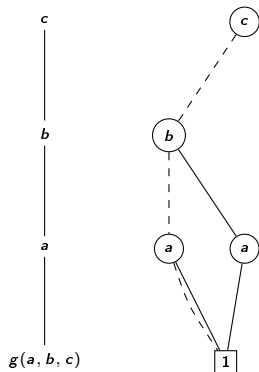
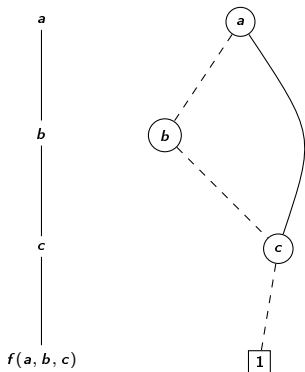
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)



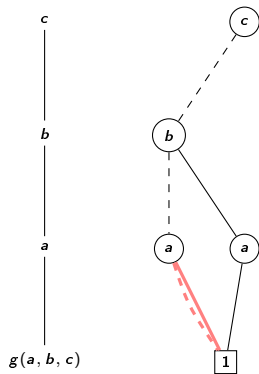
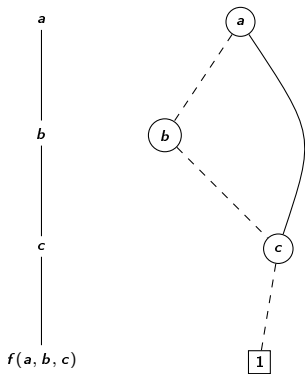
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)



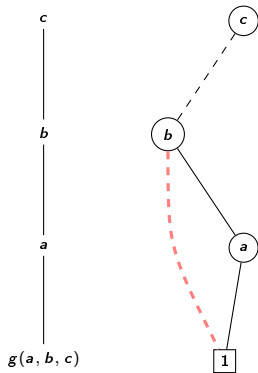
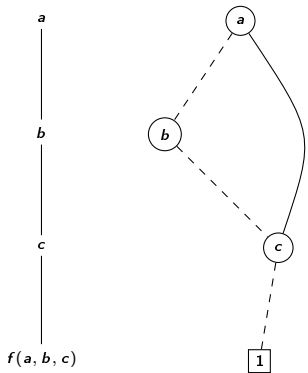
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)



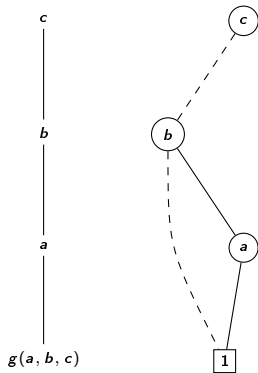
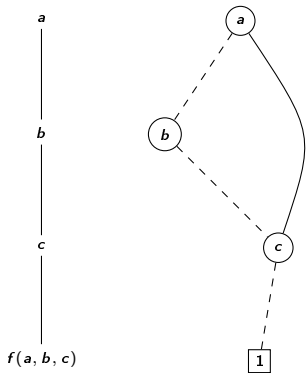
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)



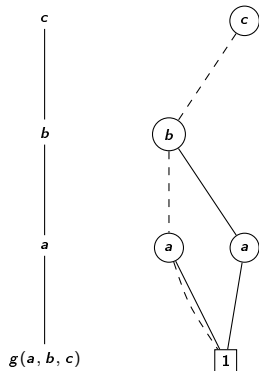
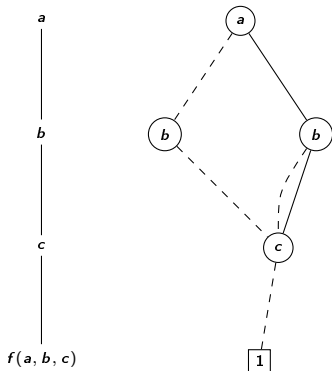
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)



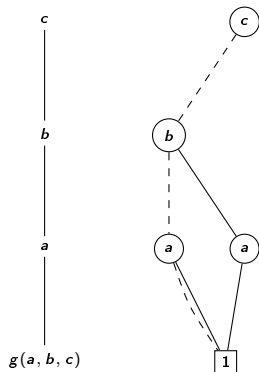
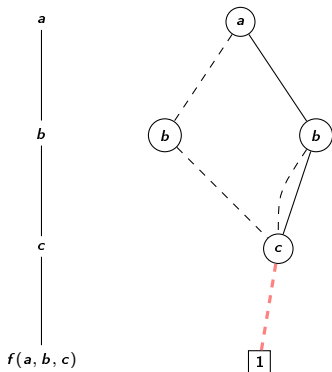
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)
- Suppression des cellules inutiles (« Zero-Suppressed », ZDD)



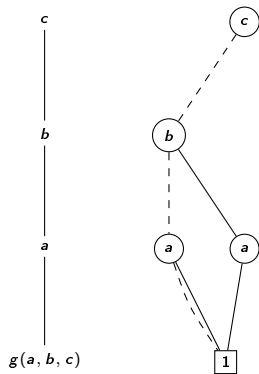
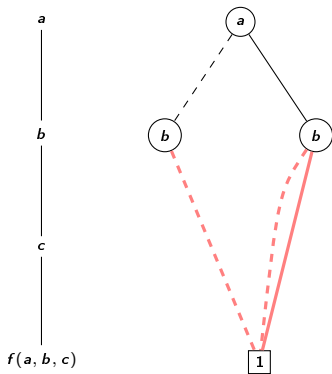
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)
- Suppression des cellules inutiles (« Zero-Suppressed », ZDD)



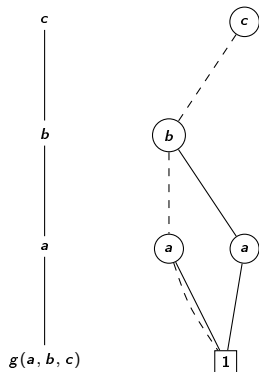
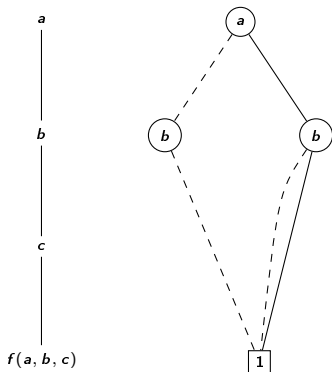
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)
- Suppression des cellules inutiles (« Zero-Suppressed », ZDD)



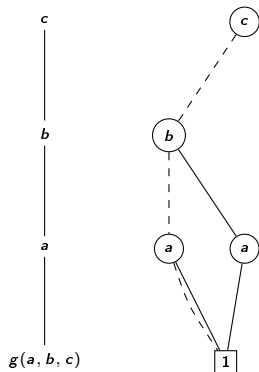
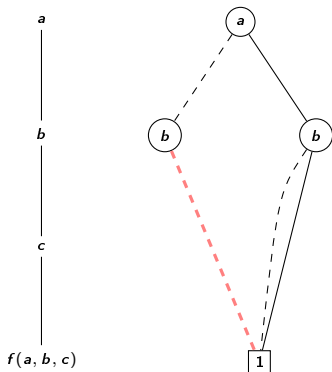
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)
- Suppression des cellules inutiles (« Zero-Suppressed », ZDD)



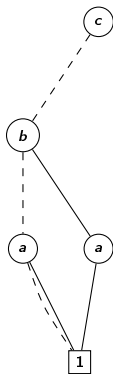
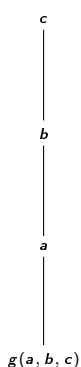
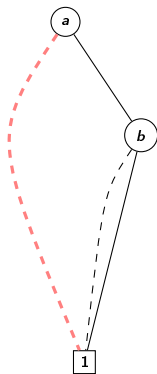
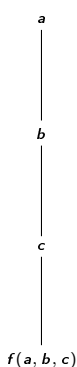
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)
- Suppression des cellules inutiles (« Zero-Suppressed », ZDD)



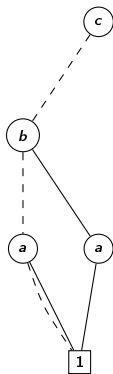
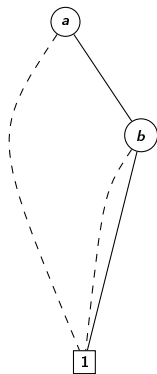
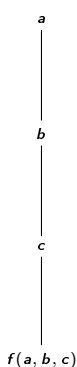
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)
- Suppression des cellules inutiles (« Zero-Suppressed », ZDD)



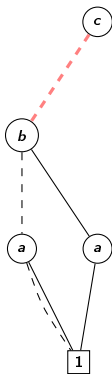
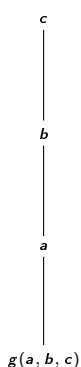
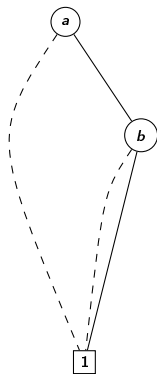
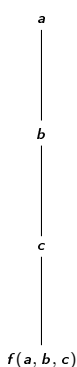
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)
- Suppression des cellules inutiles (« Zero-Suppressed », ZDD)



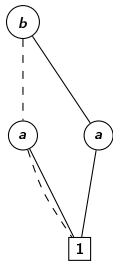
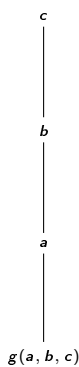
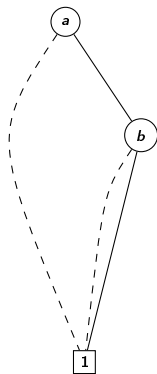
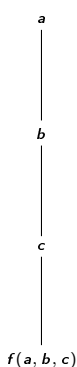
Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)
- Suppression des cellules inutiles (« Zero-Suppressed », ZDD)



Et les compressions de la table ?

- Suppression des lignes donnant 0
- Suppression des cellules inutiles (« Don't care », BDD)
- Suppression des cellules inutiles (« Zero-Suppressed », ZDD)



Canonicité

Étant donné un ordre sur les variables, toute fonction est représentée par un unique BDD (ou ZDD) [13, 1, 5]

Conséquence :

- pour un même ordre de variables, égalité de deux fonctions en temps constant : $O(1)$

Problème : Ordre des variables

Nous avons vu :

- que le nombre de nœuds dépend de l'ordre des variables
- que les compressions dépendent de l'ordre des variables

À retenir

En fixant les compressions (BDD ou ZDD), trouver le meilleur ordre est un problème NP-complet ! [11, 16, 3, 7]

Comment choisir un ordre ?

Problème : Ordre des variables

Nous avons vu :

- que le nombre de nœuds dépend de l'ordre des variables
- que les compressions dépendent de l'ordre des variables

À retenir

En fixant les compressions (BDD ou ZDD), trouver le meilleur ordre est un problème NP-complet ! [11, 16, 3, 7]

Comment choisir un ordre ?

- **Au hasard !** (la plupart du temps)
- Connaissance du modèle [12, 2, 15]
- Modification et test dynamique [10]

- Nous savons créer un BDD à la main...
- Quelles sont les manipulations possibles ?

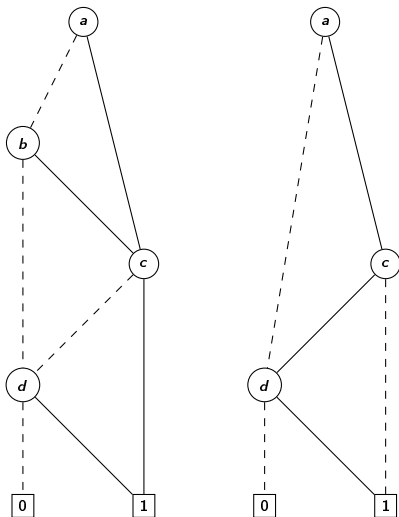
Opérations ensemblistes : APPLY

- $\cup, \cap, \setminus, \dots$
- Opérandes sur les mêmes variables
- Opérateur sur le terminal

```
APPLY(lhs, rhs,  $\odot$ ) = switch do
| case lhs  $\in$  {0,1}  $\wedge$  rhs  $\in$  {0,1}
|   result  $\leftarrow$  lhs  $\odot$  rhs
| case lhs.var = rhs.var
|   result.var  $\leftarrow$  lhs.var;
|   result.lhs  $\leftarrow$  APPLY(lhs.false, rhs.false,  $\odot$ );
|   result.rhs  $\leftarrow$  APPLY(lhs.true, rhs.true,  $\odot$ );
| case lhs.var < rhs.var
|   result.var  $\leftarrow$  lhs.var;
|   result.lhs  $\leftarrow$  APPLY(lhs.false, rhs,  $\odot$ );
|   result.rhs  $\leftarrow$  APPLY(lhs.true, rhs,  $\odot$ );
| case lhs.var > rhs.var
|   result.var  $\leftarrow$  rhs.var;
|   result.lhs  $\leftarrow$  APPLY(lhs, rhs.false,  $\odot$ );
|   result.rhs  $\leftarrow$  APPLY(lhs, rhs.true,  $\odot$ );
return result ;
```

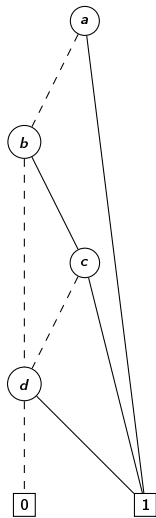
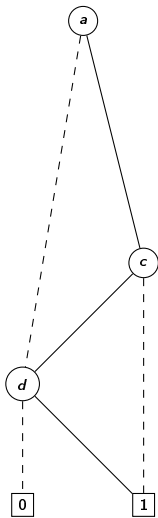
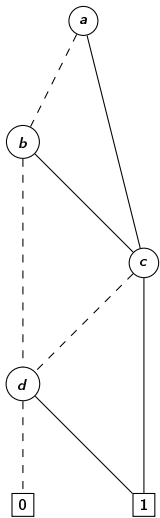
Fonction APPLY

APPLY (Exemple $f \vee g$)

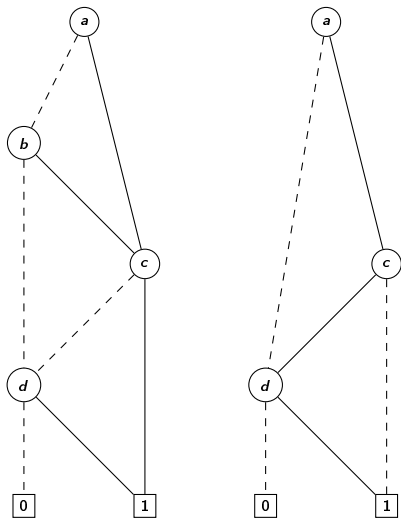


```
case  $lhs \in \{0, 1\} \wedge rhs \in \{0, 1\}$ 
|  $result \leftarrow lhs \odot rhs$ 
case  $lhs.var = rhs.var$ 
|  $result.var \leftarrow lhs.var;$ 
|  $result.lhs \leftarrow APPLY(lhs.false, rhs.false);$ 
|  $result.rhs \leftarrow APPLY(lhs.true, rhs.true);$ 
case  $lhs.var < rhs.var$ 
|  $result.var \leftarrow lhs.var;$ 
|  $result.lhs \leftarrow APPLY(lhs.false, rhs);$ 
|  $result.rhs \leftarrow APPLY(lhs.true, rhs);$ 
case  $lhs.var > rhs.var$ 
|  $result.var \leftarrow rhs.var;$ 
|  $result.lhs \leftarrow APPLY(lhs, rhs.false);$ 
|  $result.rhs \leftarrow APPLY(lhs, rhs.true);$ 
```

APPLY (Exemple $f \vee g$)



APPLY (Exemple $f \vee g$)



- Nécessité d'un cache : on ne recalcule pas une opération déjà effectuée

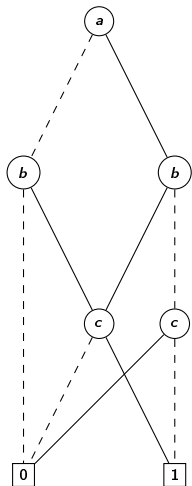
Calcul de résultat : RESTRICT

- Calculer la fonction $g = f(x \Rightarrow v)$
- Peut s'appliquer itérativement jusqu'à ce que toutes les variables soient liées

```
RESTRICT(bdd, variable, value) = switch do  
  case bdd.var = variable  $\wedge$  !value  
  | result  $\leftarrow$  bdd.false  
  case bdd.var = variable  $\wedge$  value  
  | result  $\leftarrow$  bdd.true  
  case bdd.var < variable  
  | result.var  $\leftarrow$  variable ;  
  | result.false  $\leftarrow$  RESTRICT(bdd.false, variable, value);  
  | result.true  $\leftarrow$  RESTRICT(bdd.true, variable, value);  
  case bdd.var > variable  
  | result.var  $\leftarrow$  bdd  
  
return result ;
```

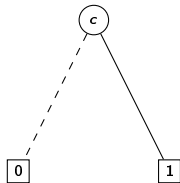
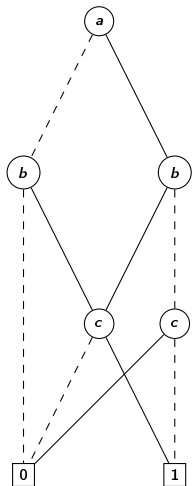
Fonction RESTRICT

RESTRICT (Exemple $f|b$)



```
case  $bdd.var = variable \wedge \neg value$ 
|  $result \leftarrow bdd.false$ 
case  $bdd.var = variable \wedge value$ 
|  $result \leftarrow bdd.true$ 
case  $bdd.var < variable$ 
|  $result.var \leftarrow variable ;$ 
|  $result.false \leftarrow$ 
|  $RESTRICT(bdd.false, variable, value)$ ;
|  $result.true \leftarrow$ 
|  $RESTRICT(bdd.true, variable, value)$ ;
case  $bdd.var > variable$ 
|  $result.var \leftarrow bdd$ 
```

RESTRICT (Exemple $f|b$)



Ca marche vraiment ?

Oui ! Et depuis longtemps :

1990 [4]

1992 [6] : « 10^{20} states and beyond... »

Ca marche vraiment ?

Oui ! Et depuis longtemps :

1990 [4]

1992 [6] : « 10^{20} states and beyond... »

Un prochain cours vous donnera des ordres de grandeur encore plus impressionnants.

Pourquoi ça fonctionne ?

Mémoire :

- Canonicité \implies fonctions identiques représentées qu'une seule fois
- Curryfication \implies partage de fonctions
- Compressions \implies gain variable selon certaines caractéristiques de la fonctions

Temps :

- Caches de calcul pour chaque nœud \implies parcours de graphe, pas d'arbre

Model checking

- Ensemble d'états

Que signifie le terminal ?

Quelles sont les variables ?

Model checking

- Ensemble d'états

Que signifie le terminal ?

true l'état est dans
l'ensemble

false l'état est hors de
l'ensemble



Quelles sont les variables ?

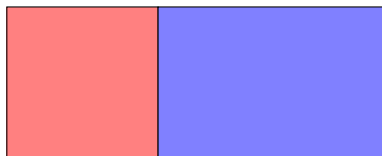
Model checking

- Ensemble d'états

Que signifie le terminal ?

true l'état est dans
l'ensemble

false l'état est hors de
l'ensemble



Quelles sont les variables ?

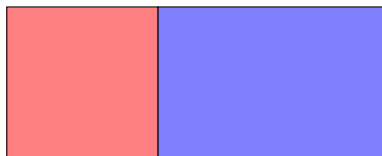
Model checking

- Ensemble d'états

Que signifie le terminal ?

true l'état est dans
l'ensemble

false l'état est hors de
l'ensemble



Quelles sont les variables ?

- Encodage binaire d'un état

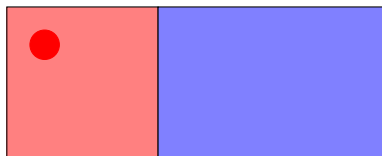
Model checking

- Ensemble d'états

Que signifie le terminal ?

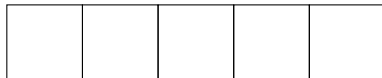
true l'état est dans
l'ensemble

false l'état est hors de
l'ensemble



Quelles sont les variables ?

- Encodage binaire d'un état



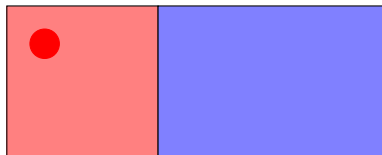
Model checking

- Ensemble d'états

Que signifie le terminal ?

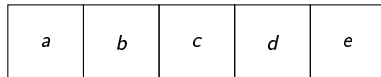
true l'état est dans
l'ensemble

false l'état est hors de
l'ensemble



Quelles sont les variables ?

- Encodage binaire d'un état



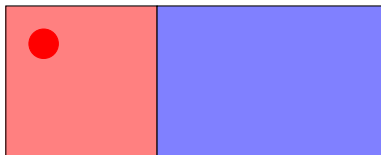
Model checking

- Ensemble d'états

Que signifie le terminal ?

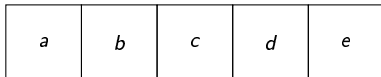
true l'état est dans
l'ensemble

false l'état est hors de
l'ensemble



Quelles sont les variables ?

- Encodage binaire d'un état
- Il doit y avoir un nombre fixé de bits manipulés



Quelles sont les variables du système ?

Tout dépend du formalisme !

Automate

Réseau de Petri

Programme

Quelles sont les variables du système ?

Tout dépend du formalisme !

Automate

Identifiant de l'état

Réseau de Petri

Programme

Quelles sont les variables du système ?

Tout dépend du formalisme !

Automate

Identifiant de l'état

Réseau de Petri

Marquages des places

Programme

Quelles sont les variables du système ?

Tout dépend du formalisme !

Automate

Identifiant de l'état

Réseau de Petri

Marquages des places

Programme

Variables globales, état de la pile, état du tas

Quelles sont les variables du système ?

Tout dépend du formalisme !

Automate

Identifiant de l'état

Réseau de Petri

Marquages des places

Programme

Variables globales, état de la pile, état du tas

Les variables sont les bits de la représentation binaire d'un état !

Quelques exemples

Que faire si on manipule des entiers ?

Borne

Il est nécessaire que les bornes soient .

Que faire si on ne connaît pas les bornes ?

Que faire si on manipule des entiers ?

Borne

Il est nécessaire que les bornes soient connues.

À partir des bornes, on détermine le nombre de bits de la représentation, donc le nombre de variables.

Que faire si on ne connaît pas les bornes ?

Que faire si on manipule des entiers ?

Borne

Il est nécessaire que les bornes soient connues.

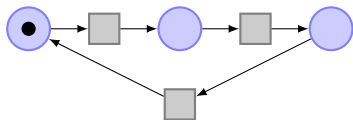
À partir des bornes, on détermine le nombre de bits de la représentation, donc le nombre de variables.

Que faire si on ne connaît pas les bornes ?

Utiliser d'autres Diagrammes de Décision : DDDs [8] et SDDs [9]

Que faire de variables redondantes ?

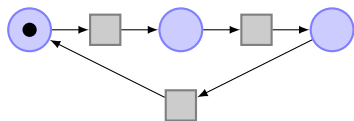
Les marquages des places sont dépendants les uns des autres.



Que faire de variables redondantes ?

Les marquages des places sont dépendants les uns des autres.
Faut-il :

- conserver 1 bit par place ?
- coder la position du jeton ?

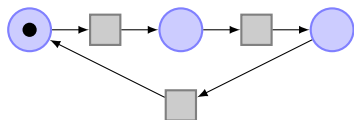


Que faire de variables redondantes ?

Les marquages des places sont dépendants les uns des autres.
Faut-il :

- conserver 1 bit par place ?
- coder la position du jeton ?

On ne sait pas. Il faut tester !



Adéquation des optimisations ?

Quelles compressions conviennent au modèle ?

Supprimer le terminal *true*

Supprimer le terminal *false*

« Don't care »

« Zero-Suppressed »

Adéquation des optimisations ?

Quelles compressions conviennent au modèle ?

Supprimer le terminal *true*

S'il y a plus d'états accessibles que non accessibles
(aucun package de BDD ne le fait)

Supprimer le terminal *false*

« Don't care »

« Zero-Suppressed »

Adéquation des optimisations ?

Quelles compressions conviennent au modèle ?

Supprimer le terminal *true*

S'il y a plus d'états accessibles que non accessibles
(aucun package de BDD ne le fait)

Supprimer le terminal *false*

S'il y a plus d'états non accessibles qu'accessibles

« Don't care »

« Zero-Suppressed »

Adéquation des optimisations ?

Quelles compressions conviennent au modèle ?

Supprimer le terminal *true*

S'il y a plus d'états accessibles que non accessibles
(aucun package de BDD ne le fait)

Supprimer le terminal *false*

S'il y a plus d'états non accessibles qu'accessibles

« Don't care »

Si le modèle est plutôt synchrone (BDD)

« Zero-Suppressed »

Adéquation des optimisations ?

Quelles compressions conviennent au modèle ?

Supprimer le terminal *true*

S'il y a plus d'états accessibles que non accessibles
(aucun package de BDD ne le fait)

Supprimer le terminal *false*

S'il y a plus d'états non accessibles qu'accessibles

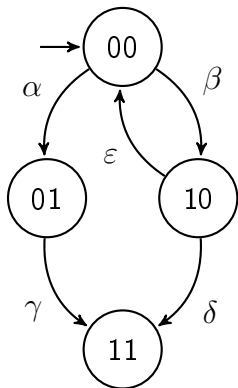
« Don't care »

Si le modèle est plutôt synchrone (BDD)

« Zero-Suppressed »

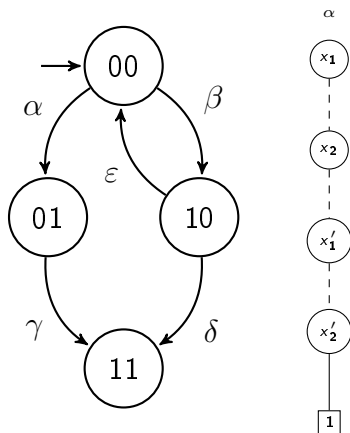
Si le modèle est plutôt asynchrone [14] (ZDD)

Qu'est-ce qu'une transition ?



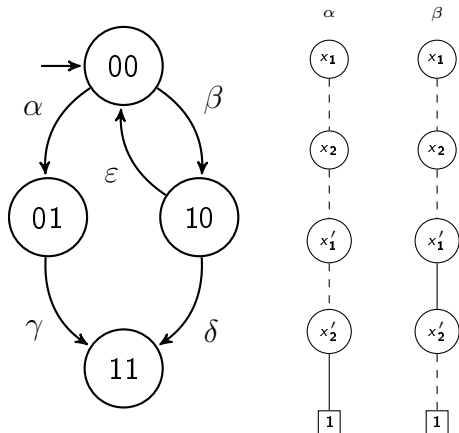
- Une transition est une relation entre deux états
- Codée par l'état de départ et l'état d'arrivée
- Si un état est codé par n variables, une transition est codée par $2n$ variables

Qu'est-ce qu'une transition ?



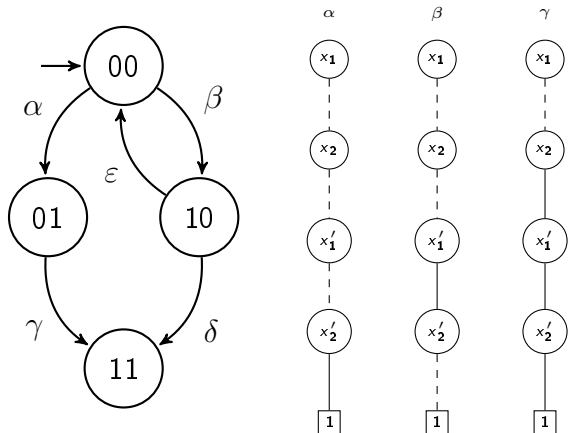
- Une transition est une relation entre deux états
- Codée par l'état de départ et l'état d'arrivée
- Si un état est codé par n variables, une transition est codée par $2n$ variables

Qu'est-ce qu'une transition ?



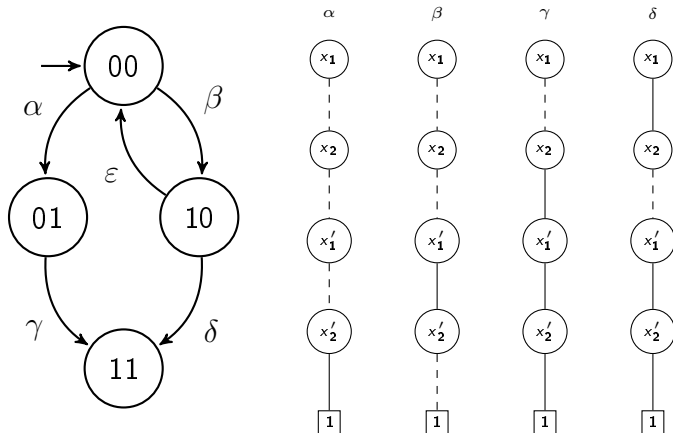
- Une transition est une relation entre deux états
- Codée par l'état de départ et l'état d'arrivée
- Si un état est codé par n variables, une transition est codée par $2n$ variables

Qu'est-ce qu'une transition ?



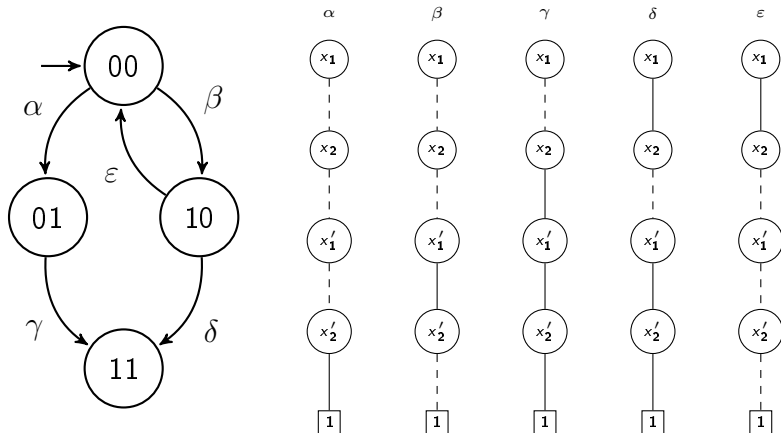
- Une transition est une relation entre deux états
- Codée par l'état de départ et l'état d'arrivée
- Si un état est codé par n variables, une transition est codée par $2n$ variables

Qu'est-ce qu'une transition ?



- Une transition est une relation entre deux états
- Codée par l'état de départ et l'état d'arrivée
- Si un état est codé par n variables, une transition est codée par $2n$ variables

Qu'est-ce qu'une transition ?



- Une transition est une relation entre deux états
- Codée par l'état de départ et l'état d'arrivée
- Si un état est codé par n variables, une transition est codée par $2n$ variables

Transition

- Relation entre v_1, \dots, v_n variables d'entrée et v'_1, \dots, v'_n variables de sortie

Ensemble de transitions

- On manipule des ensembles d'états

Transition

- Relation entre v_1, \dots, v_n variables d'entrée et v'_1, \dots, v'_n variables de sortie

Ensemble de transitions

- On manipule des ensembles d'états
- On applique des ensembles de transitions !

Transition

- Relation entre v_1, \dots, v_n variables d'entrée et v'_1, \dots, v'_n variables de sortie

Ensemble de transitions

- On manipule des ensembles d'états
- On applique des ensembles de transitions !
- $T = \sum_i t_i$

- CUDD "Le meilleur" (à ma connaissance)
- BuDDy Utilisé souvent
- JavaBDD Interface Java sur plusieurs bibliothèques de BDD
- libsdd Prochain cours !

Bibliothèques de BDDs : CUDD

```
DdManager *manager;  
DdNode *f, *var, *tmp;  
int i;  
...  
f = Cudd_ReadOne(manager);  
Cudd_Ref(f);  
for (i = 3; i >= 0; i--) {  
    var = Cudd_bddIthVar(manager, i);  
    tmp = Cudd_bddAnd(manager, Cudd_Not(var), f);  
    Cudd_Ref(tmp);  
    Cudd_RecursiveDeref(manager, f);  
    f = tmp;  
}
```

Bibliothèques de BDDs : BuDDy

```
bdd x, y, z;  
  
bdd_init(1000, 100);  
bdd_setvarnum(5);  
  
x = bdd_ithvar(0);  
y = bdd_ithvar(1);  
z = bdd_addrf(bdd_apply(x, y, bddop_and));  
  
bdd_printtable(z);  
bdd_delref(z);  
bdd_done();
```

Bibliothèques de BDDs : libsdd

```
class Inc(Inductive):  
    def __init__(self, var, max):  
        super(Inc, self).__init__(var)  
        self.max = max  
  
    @propagate(False)  
    def values(self, var, val):  
        return [ v + 1 for v in val if v < self.max ]  
  
def main():  
    d = node("a", 0, node("b", 0, node("b", 0, ONE)))  
    print "Originaluu:" , d  
    events = fixpoint( add([ ID , Inc("b", 3) ]) )  
    final = events(d)  
    print "Resultuuuu:" , final
```




- CrocoPat




```
Male("John");  
Female("Alice");  
ParentOf("John", "Alice");  
FatherOf(x,y) := ParentOf(x,y) & Male(x);  
MotherOf(x,y) := ParentOf(x,y) & Female(x);  
ParentOf(x,y) := MotherOf(x,y) | FatherOf(x,y);  
Parent(x) := EX(y, ParentOf(x,y));  
Childless(x) := FA(y, !ParentOf(x,y));
```

- Jedd




Exemple trop gros pour montrer...



- Structure de données efficace en temps et mémoire
- Adapté à la représentation d'ensembles d'états
- Manipulation peu aisée
 - Bibliothèques adaptées aux formalismes
 - Langages de manipulation

-  S. B. Akers.
Binary decision diagrams, June 1978.
-  Beate Bollig, Martin Lobbing, and Ingo Wegener.
Simulated annealing to improve variable orderings for obdds.
[In In Int'l Workshop on Logic Synth](#), pages 5–5, 1995.
-  Beate Bollig and Ingo Wegener.
Improving the variable ordering of obdds is np-complete.
[IEEE Trans. Comput.](#), 45(9) :993–1002, 1996.

-  Karl S. Brace, Richard L. Rudell, and Randal E. Bryant.
Efficient implementation of a bdd package.
In *DAC '90 : Proceedings of the 27th ACM/IEEE conference on Design automation*, pages 40–45, New York, NY, USA, 1990.
ACM.
-  Randal E. Bryant.
Graph-based algorithms for boolean function manipulation.
IEEE Transactions on Computers, 35(8) :677–691, 1986.
-  Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang.
Symbolic model checking : $10^{**}20$ states and beyond.
Inf. Comput., 98(2) :142–170, 1992.

Bibliographie III

-  G. Cabodi, S. Quer, Ch. Meinel, H. Sack, A. Slobodova, and C. Stangier.
Binary decision diagrams and the multiple variable order problem, 1998.
-  Jean-Michel Couvreur, Emmanuelle Encrenaz, Emmanuel Paviot-Adet, Denis Poitrenaud, and Pierre-André Wacrenier.
Data decision diagrams for petri net analysis.
Lecture Notes in Computer Science (LNCS), pages 101–120, Adelaide, Australia, 2002. Springer-Verlag.
-  Jean-Michel Couvreur and Yann Thierry-Mieg.
Hierarchical decision diagrams to exploit model structure.
Lecture Notes in Computer Science (LNCS), pages 443–457, Taipei, Taiwan, 2005. Springer-Verlag.

-  E. Felt, G. York, R. Brayton, and A. Sangiovanni-Vincentelli.
Dynamic variable reordering for bdd minimization.
[Design Automation Conference, 1993, with EURO-VHDL '93. Proceedings EURO-DAC '93. European, pages 130–135, Sep 1993.](#)
-  S. J. Friedman and K. J. Supowit.
Finding the optimal variable ordering for binary decision diagrams.
[IEEE Trans. Comput., 39\(5\) :710–713, 1990.](#)



Hiroshige Fujii, Goichi Ootomo, and Chikahiro Hori.

Interleaving based variable ordering methods for ordered binary decision diagrams.

In ICCAD '93 : Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design, pages 38–41, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.



C.Y. Lee.

Representation of switching circuits by binary-decision programs, July 1959.



Shin-ichi Minato.

Zero-suppressed bdds for set manipulation in combinatorial problems.

In [DAC '93 : Proceedings of the 30th international conference on Design automation](#), pages 272–277, New York, NY, USA, 1993. ACM.



C. Scholl, B. Becker, and A. Brogle.

Solving the multiple variable order problem for binary decision diagrams by use of dynamic reordering techniques.

Technical report, 1999.



Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima.
The complexity of the optimal variable ordering problem of
shared binary decision diagrams.
In ISAAC'93, volume 762, 1993.