Algorithmes et structures de données pour la vérification formelle

1. Les systèmes automatiques
2. Les BDDs (Binary Decision Diagrams)

Gérard Berry

Collège de France

Chaire Algorithmes, machines et langages

gerard.berry@college-de-france.fr

Cours 2, Paris, 9 mars 2016

Suivi du séminaire de J.C. Madre et P. Vuillod

COLLÈGE DE FRANCE 1530

Agenda

1. Présentation général du cours 2016
2. Rappels sur la complexité algorithmique
3. Le calcul Booléen
4. Les BDDs
5. Application des BDDs à la vérification
6. Les ZDDs
7. Conclusion

Merci à Jean-Christophe Madre pour son aide inestimable

Rappel : premier cours à Rennes !

Inria Rennes, 4 novembre 2015

Cours 1 :

L'importance des langages en informatique

Séminaire par Thomas Jensen (Inria Rennes) :

Intégration de la vérification formelle dans les langages de programmation

Vérification automatique : quelle est la question ?

- Dans beaucoup de domaines, il faut résoudre de nombreux problèmes combinatoires difficiles
  - conception assistée de circuits électroniques
  - optimisation et vérification de programmes ou de circuits
  - équivalence de programmes ou de circuits
  - programmation par contraintes, optimisation générale, jeux, etc.
- Ces problèmes prennent souvent la forme de formules mathématiques, avec deux grandes questions :
  - satisfiabilité : la formule peut-elle être rendue vraie en choisissant bien les valeurs des variables ?
  - validité : la formule est-elle vraie pour toutes les valeurs des variables?

L'expression des problèmes repose sur des théories variées:
logique, arithmétique, graphes, algèbre, etc.
qu'il faut aussi mélanger

Vérification automatique : quel est l'objectif ?

- Fournir des algorithmes automatiques pour résoudre les questions posées
  - algorithmes énumératifs : explorer explicitement l'espace pour trouver les solutions ou montrer leur absence (exhaustivement, aléatoirement, etc.)
  - algorithmes déductifs : s'appuyer sur la logique et les théories mathématiques décidables (ou semi-décidables), de complexité raisonnable, surtout dans les « cas pratiques ».
- Ne pas demander de compétence particulière à l'utilisateur
  - sinon celle de pouvoir exprimer son problème dans les formalismes fournis → ergonomie non triviale

Un objectif hélas intrinsèquement difficile : prévoir le coût !
Client : Quelle ressources dois-je allouer pour ce problème?
Fournisseur : Ahem... On ne le saura qu'après....

Vérification automatique : quels sont les moyens ?

- Une variété de théories décidables, avec des progrès majeurs
  - le calcul booléen, avec BDDs, ZDDs, et algorithmes SAT
  - la satisfaction modulo théories (SMT), mélange de logique booléenne et de théories décidables variées, rendues compatibles ente elles
  - les automates temporisés, pour les problèmes temps-réels
  - les explorateurs explicites (SPIN, CADP, etc.), bien adaptés aux algorithmes disribués et aux protocoles de communication
- Des communautés très actives
  - améliorations permanentes des algorithmes
  - constitutions de grandes librairies de benchmarks : difficiles, aléatoires, industriels,...
  - compétitions SAT, SMT, etc., majeures pour les progrès

Un merveilleux domaine de recherche,
où il faut avoir l'algorithme dans la peau !

Déroulé des cours

- Cours 2, 09/03/2016 : les BDDs (Binary Decision Diagrams)
  Séminaire par J-C Madre et P. Vuillod : les BDDs en CAO de circuits

- Cours 3, 16/03/2016 : SAT, la satisfaction Booléenne
  Séminaire par L. Simon : Victoires contre des problèmes difficiles

- Cours 4, 23/03/2016 : SMT, la satisfaction modulo théories
  Séminaire par S. Conchon : Le démonstrateur ALT-ERGO

- Cours 5, 30/03/2016 : La vérification de systèmes temporels
  Séminaire par K. Larsen : Real-time model checking of embedded systems

- Cours 6, 06/04/2016 : La vérification par énumération explicite
  Séminaire par S. Delaune : Vérifier les protocoles cryptographiques

- Cours 7, 13/04/2016 : Réponses aux questions de l'année
  Séminaire par C. Keller : Combiner assistants de preuve
  et démonstrateurs automatiques

Présents et internautes, envoyez vos questions par courriel à

gerard.berry@college-de-france.fr

même si vous étiez dans la salle.

Et, s'il vous plaît, n'attendez pas les derniers jours !

Agenda

1. Présentation général du cours 2016
2. Rappels sur la complexité algorithmique
3. Le calcul Booléen
4. Les BDDs
5. Application des BDDs à la vérification
6. Les ZDDs
7. Conclusion

Rappels sur la complexité algorithmique

Indécidable : il n'existe pas d'algorithme résolvant la question

Semi-décidable : il existe un algorithme répondant oui à la question si elle est vraie, et répondant non ou bouclant sinon – fréquent et bien utile !

Soit *n* la taille de l'énoncé du problème.
Coût du pire cas (temps, espace, énergie, etc.)

Double exponentiel : 2**(2**n)

Exemple : arithmétique de Presburger (+,−,<,=, etc.)

Exponentiel : 2^n

Exemples : énumération explicite de sous-ensembles
algorithmes connus pour le calcul booléen

Rappels sur la complexité algorithmique

P-Space complet : espace polynomial = exponentiel ?

Exemples : QBF = formules Booléennes quantifiées

vérification de formules LTL (linear temporal logic)

NP-complet : ???????

Vérifier qu'un candidat est bien une solution est polynomial

Mais en trouver une est pour l'instant exponentiel (pire cas)

Exemples : des milliers, et partout dans ce cours

***** SAT = satisfaction dans le calcul booléen

Polynomial (classe P) *:* n^2, n^3, ...

primalité d'un nombre !! Agrawal-Kayal-Saxena 2002

programmation linéaire !! Khachiyan 1979 (simplexe = 2^n)

log, linéaire ou mieux: $n \times \log(n)$, n, sublinéaire, constant.

tris, tables de hash, etc. Pas pour ce cours ☹

Deux grandes conjectures

Cook 1972 : P ≠ NP

NP ≠ P-Space

Mais « pire cas » ne veut pas dire toujours !

Complexité en moyenne : n'aide pas ici, généralement mauvaise

Complexité en pratique : de grosses surprises positives !

Il n'y a pas de théorie de la complexité des applications réelles

Les algorithmes marchent souvent bien en pratique, mais on ne sait pas toujours pourquoi

Ce mystère nous dépasse, faisons semblant de l'avoir organisé

Agenda

1. Présentation général du cours 2016
2. Rappels sur la complexité algorithmique
3. Le calcul Booléen
4. Les BDDs
5. Application des BDDs à la vérification
6. Les ZDDs
7. Conclusion

Le calcul booléen

- Calcul de vrai, faux, et, ou, non : le plus simple de tous ! mais encore très largement incompris....

$$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$$

- Après Boole : quatre visions algorithmiques différentes, mais complémentaires
  - systèmes de déduction logique : ex. résolution, cf. cours 3
  - formes normales uniques : ex. BDDs ou ZDDs, ce cours
  - exploration systématique des valeurs des variables : SAT, cours 3
  - mélange évaluation binaire / déduction : CDCL : SAT, cours 3

Déduction par *modus ponens* : A, A ⇒ B ⊢ B
Calcul Booléen : évaluation des formules par mise des variables à vrai (1) ou faux(0)
Algèbre de Boole : lois d'opérateurs

Lois de la conjonction et de la disjonction

vrai	*vrai*, 1, 1
faux	*faux*, 0, 0
variables	$x, y, x_1, y_1, \ldots$
formules	$A, B, C, \ldots$
conjonction (et)	$A \wedge B$
disjonction (ou)	$A \vee B$
associativité	$A \wedge (B \wedge C) = (A \wedge B) \wedge C$
	$A \vee (B \vee C) = (A \vee B) \vee C$
commutativité	$A \wedge B = B \wedge A$
	$A \vee B = B \vee A$
idempotence	$A \wedge A = A$
	$A \vee A = A$
éléments neutres	$A \wedge 1 = 1 \wedge A = A$
	$A \vee 0 = 0 \vee A = A$
éléments absorbants	$A \wedge 0 = 0 \wedge A = 0$
	$A \vee 1 = 1 \vee A = 1$
distributivité	$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
	$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

Lois de la négation

négation (non)	$\neg A$
inversion	$\neg 0 = 1$
	$\neg 1 = 0$
double négation	$\neg(\neg A) = A$
	$A \wedge \neg A = 0$
tiers exclu	$A \vee \neg A = 1$
de Morgan	$\neg(A \wedge B) = \neg A \vee \neg B$
	$\neg(A \vee B) = \neg A \wedge \neg B$

Rappel du cours du 26 mars 2014 :
logique classique : avec le tiers exclu
logique intuitionniste : sans tiers exclu, De Morgan partiel
$\Leftrightarrow$ stabilisation électrique des circuits

Forme normale négative (NNF)

littéral positif x, y, x_1, y_1
littéral négatif $\bar{x} = \neg x, \overline{x_1} = \neg x_1, \ldots$

formule en NNF les négations n'apparaissent que dans les littéraux
exemple $(\bar{x} \vee y) \wedge (\bar{y} \vee (z \wedge \bar{t}))$

mise en NNF simple application récursive de De Morgan

Forme normale disjonctive

DNF : Ou de Et de littéraux (chaque variable au plus une fois) $(\bar{x} \wedge \bar{y} \wedge z) \vee (z \wedge \bar{t})$

Notation polynomiale $\bar{x}\bar{y}z + z\bar{t}$

cubes

Exemple : le full adder
$s = x\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}\bar{y}z + xyz$
$c = xy + yz + zx$

La DNF est très utilisée pour les définitions de fonctions pour les circuits, sf. séminaire de J-C. Madre et P. Vuillod

Forme normale conjonctive (CNF)

- CNF : et de ou de littéraux $(\overline{x} \vee y \vee z) \wedge (z \vee \overline{t})$

clauses

- Vérification de circuits et programmes
- Programmation par contraintes,
- Algorithmes combinatoires,
- Représentation de connaissances,
- Etc.

La CNF est centrale pour les questions
de satisfiabilité et validité, cf prochain cours (SAT)

Opérateurs dérivés, décomposition de Shannon

Implication	$A \Rightarrow B$	$=_{def}$	$\neg A \vee B$
Equivalence, $=$	$A \Leftrightarrow B$	$=_{def}$	$(\neg A \vee B) \wedge (A \vee \neg B)$
Ou exclusif, $\neq$	$A \oplus B$	$=_{def}$	$(A \wedge \neg B) \vee (\neg A \wedge B)$
Choix sur variable	$mux(x, A, B)$	$=_{def}$	$(x \wedge A) \vee (\neg x \wedge B)$
Cofacteur-1	$A[x_1 \leftarrow 1]$	$=_{def}$	$A(1, x_2, \ldots, x_n)$
Cofacteur-0	$A[x_1 \leftarrow 0]$	$=_{def}$	$A(0, x_2, \ldots, x_n)$

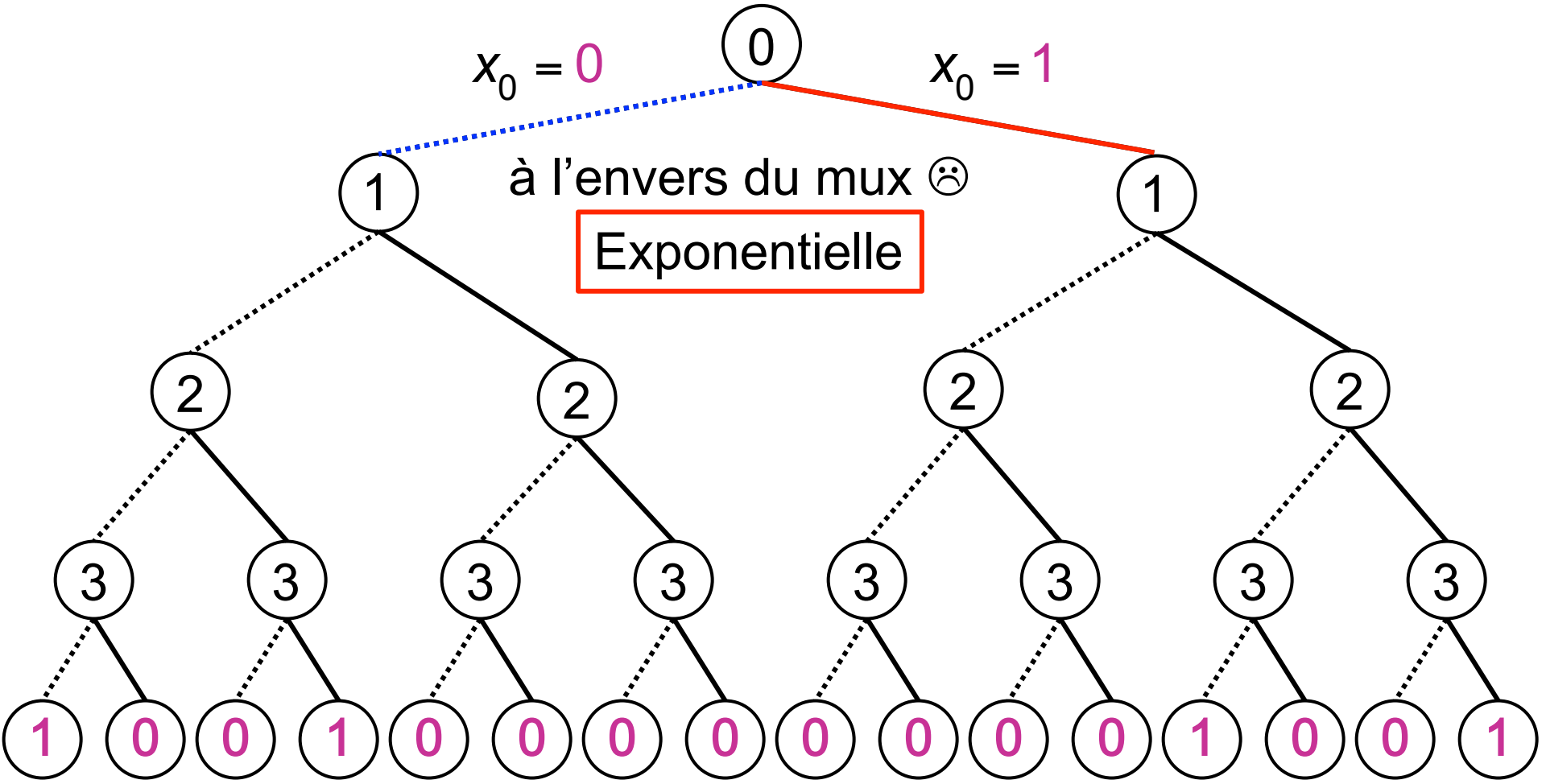
Décomposition de Shannon :

$$A = mux(x, A[x \leftarrow 1], A[x \leftarrow 0]) = (x \wedge A[x \leftarrow 1]) \vee (\overline{x} \wedge A[x \leftarrow 0])$$

Arbre de Shannon = forme normale unique

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

Ordre des variables fixé



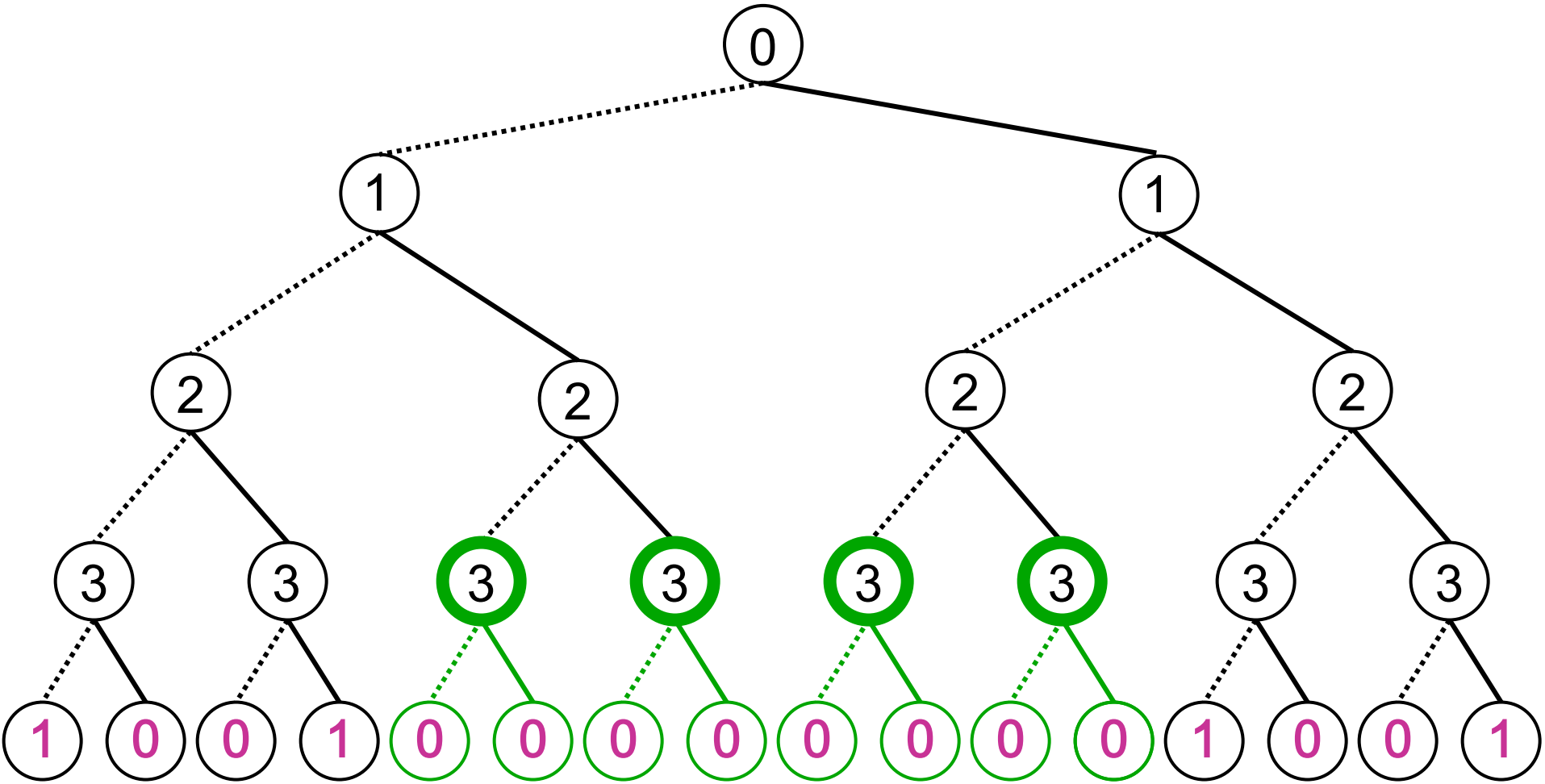
Agenda

1. Présentation général du cours 2016
2. Rappels sur la complexité algorithmique
3. Le calcul Booléen
4. Les BDDs
5. Exemple d'applications des BDDs
6. Les ZDDs
7. Conclusion

Suppression des nœuds redondants

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

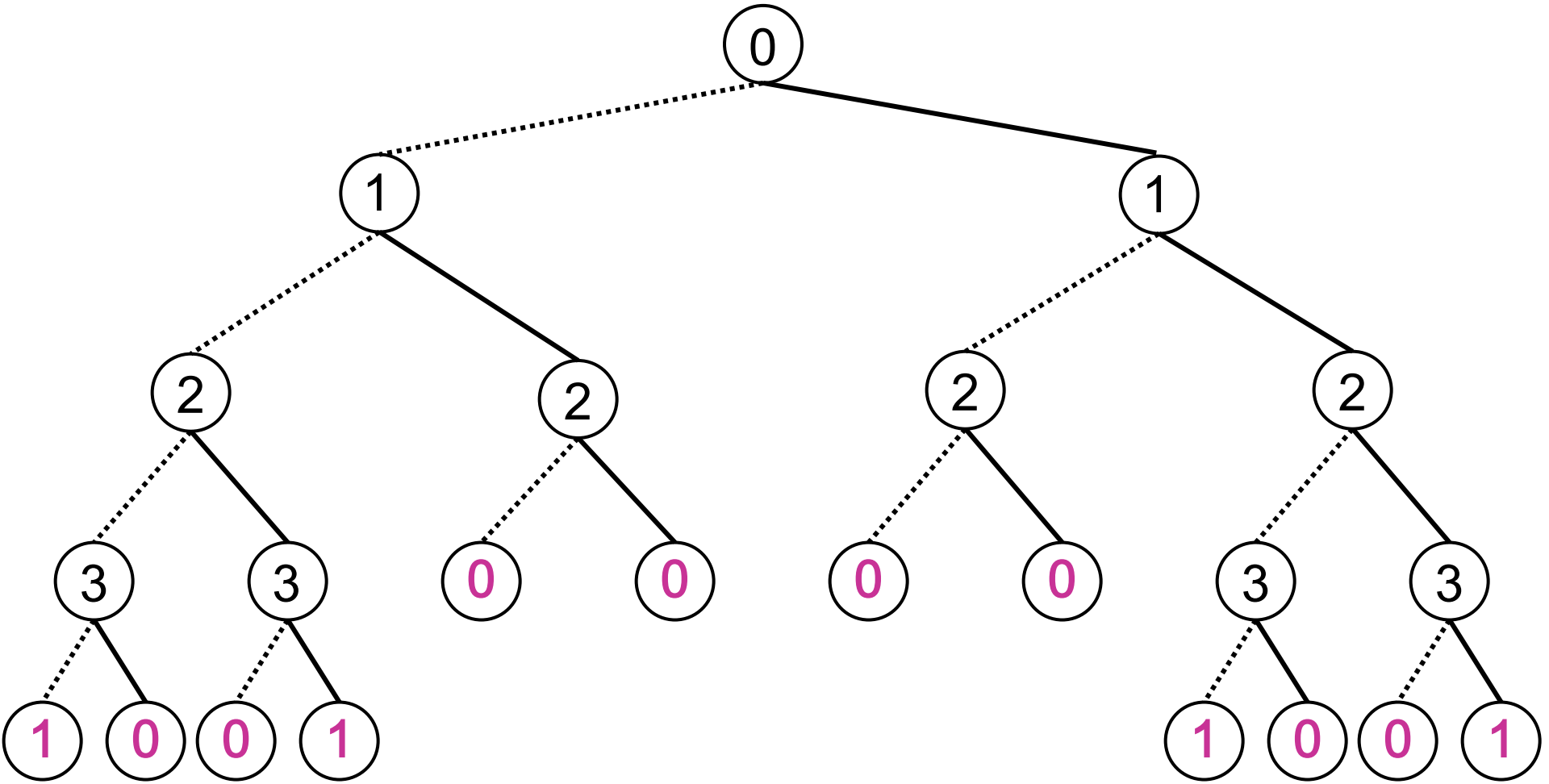
Ordre des variables fixé



Suppression des nœuds redondants

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

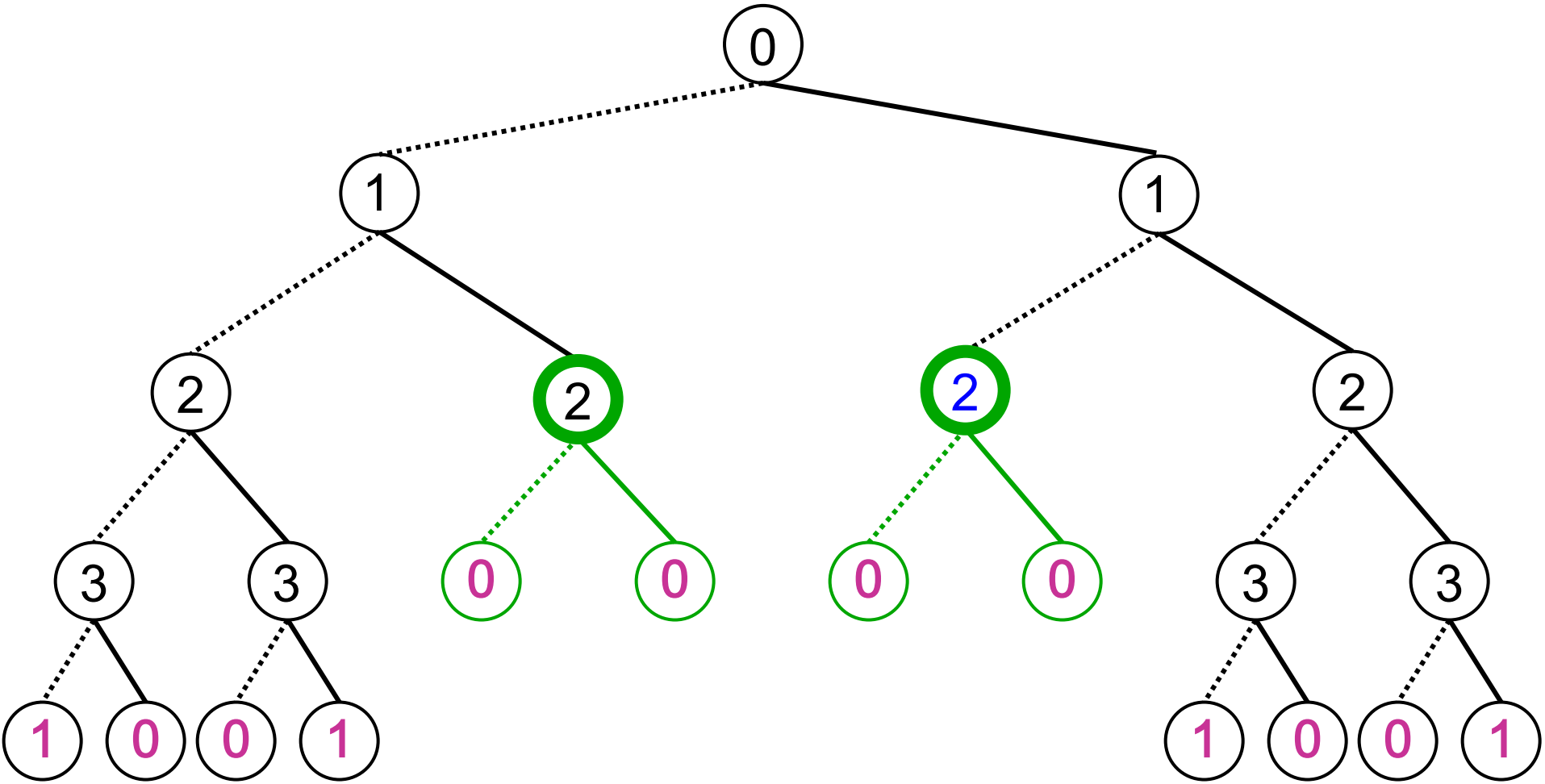
Ordre des variables fixé



Suppression des nœuds redondants

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

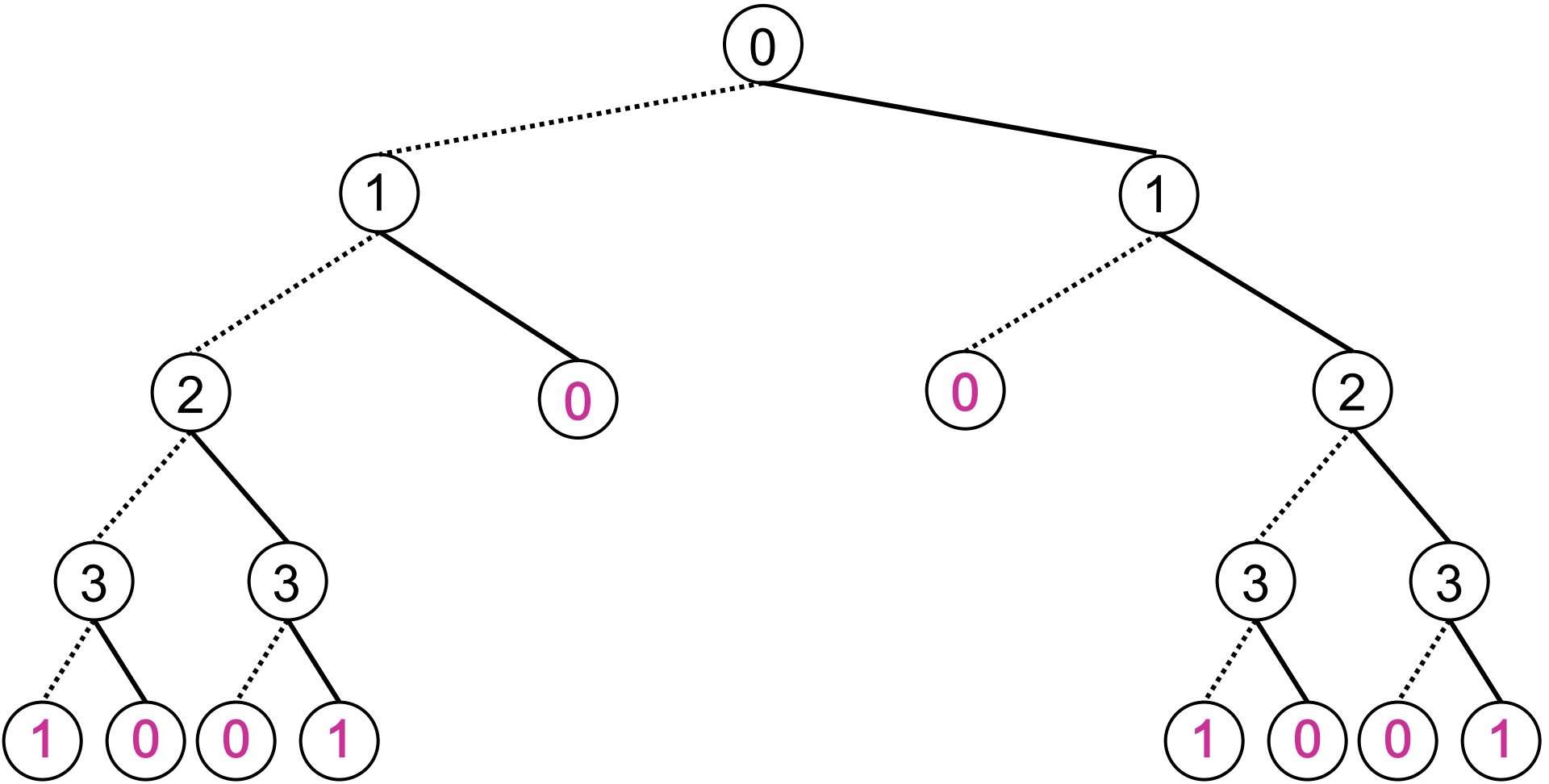
Ordre des variables fixé



Partage des sous-arbres identiques

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

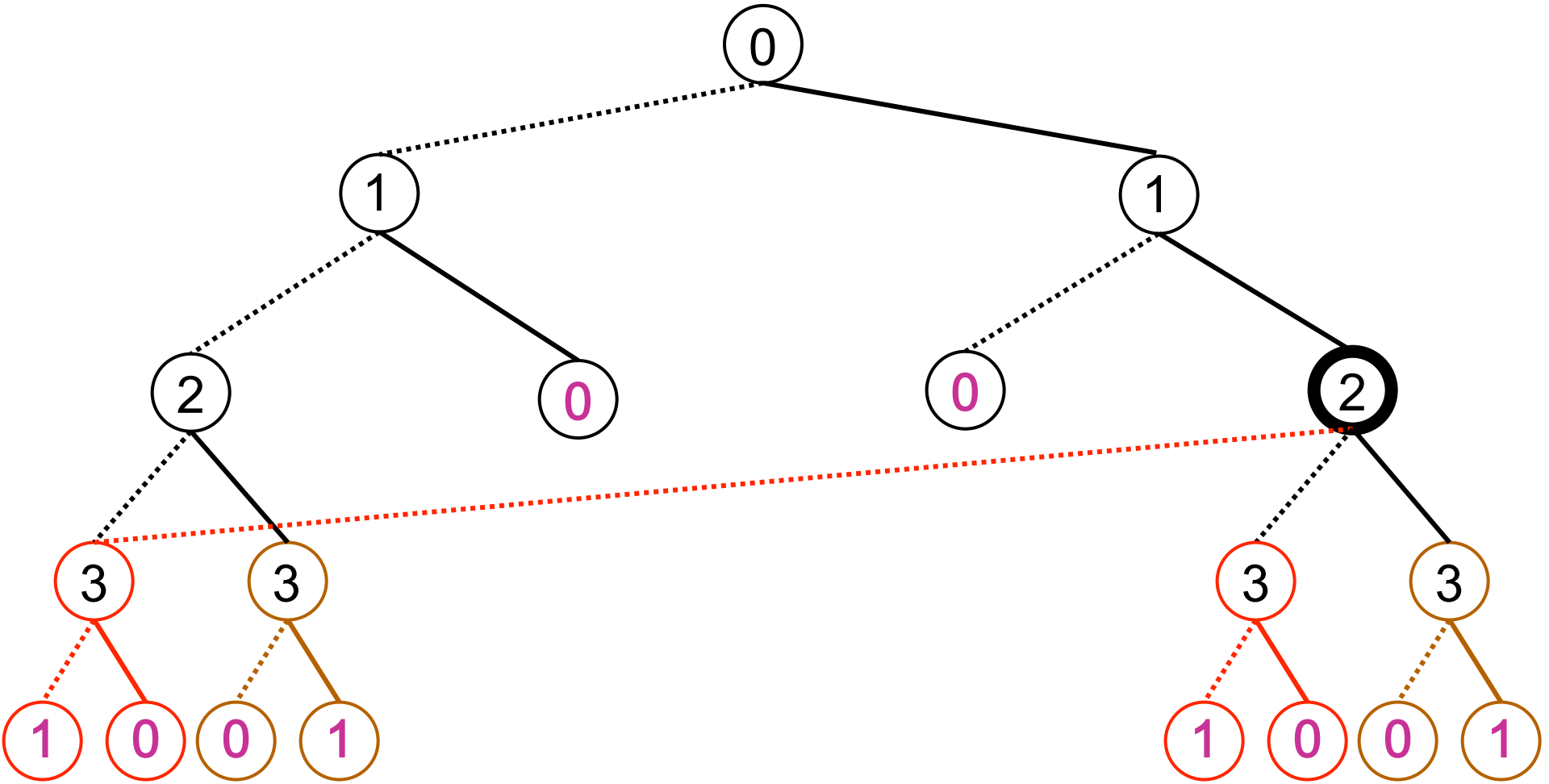
Ordre des variables fixé



Partage des sous-arbres identiques

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

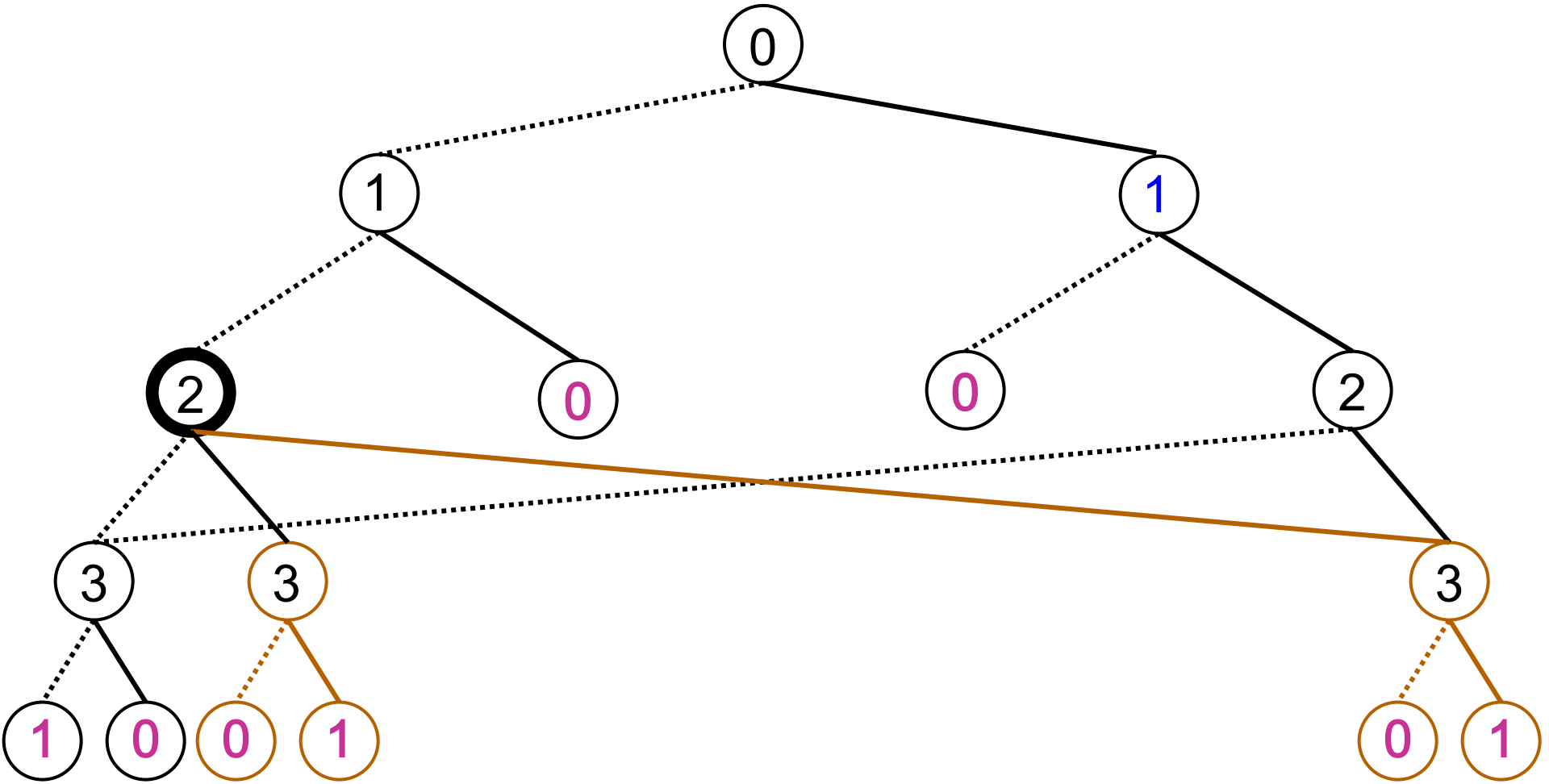
Ordre des variables fixé



Partage des sous-arbres identiques

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

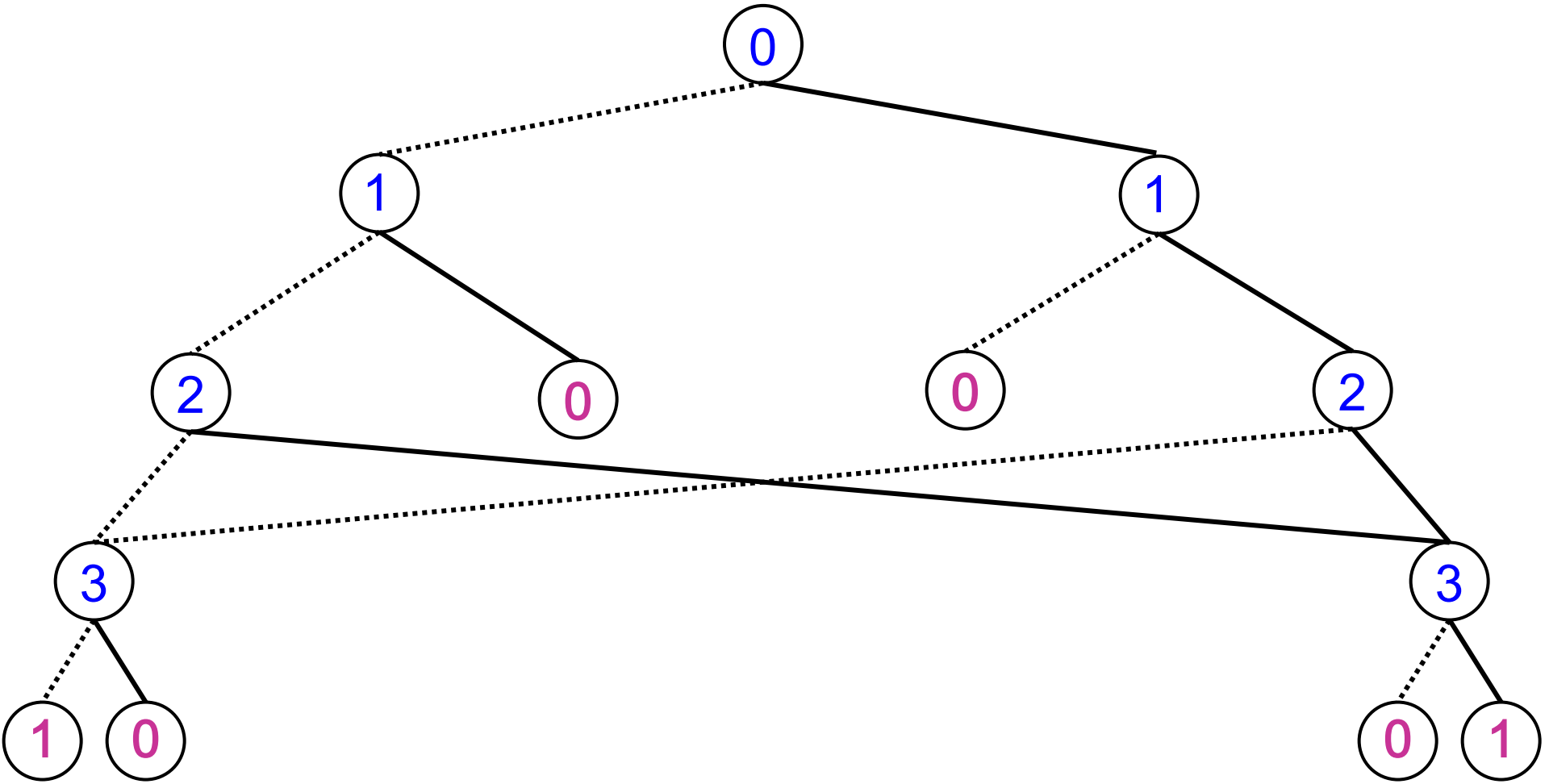
Ordre des variables fixé



Partage des sous-arbres identiques

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

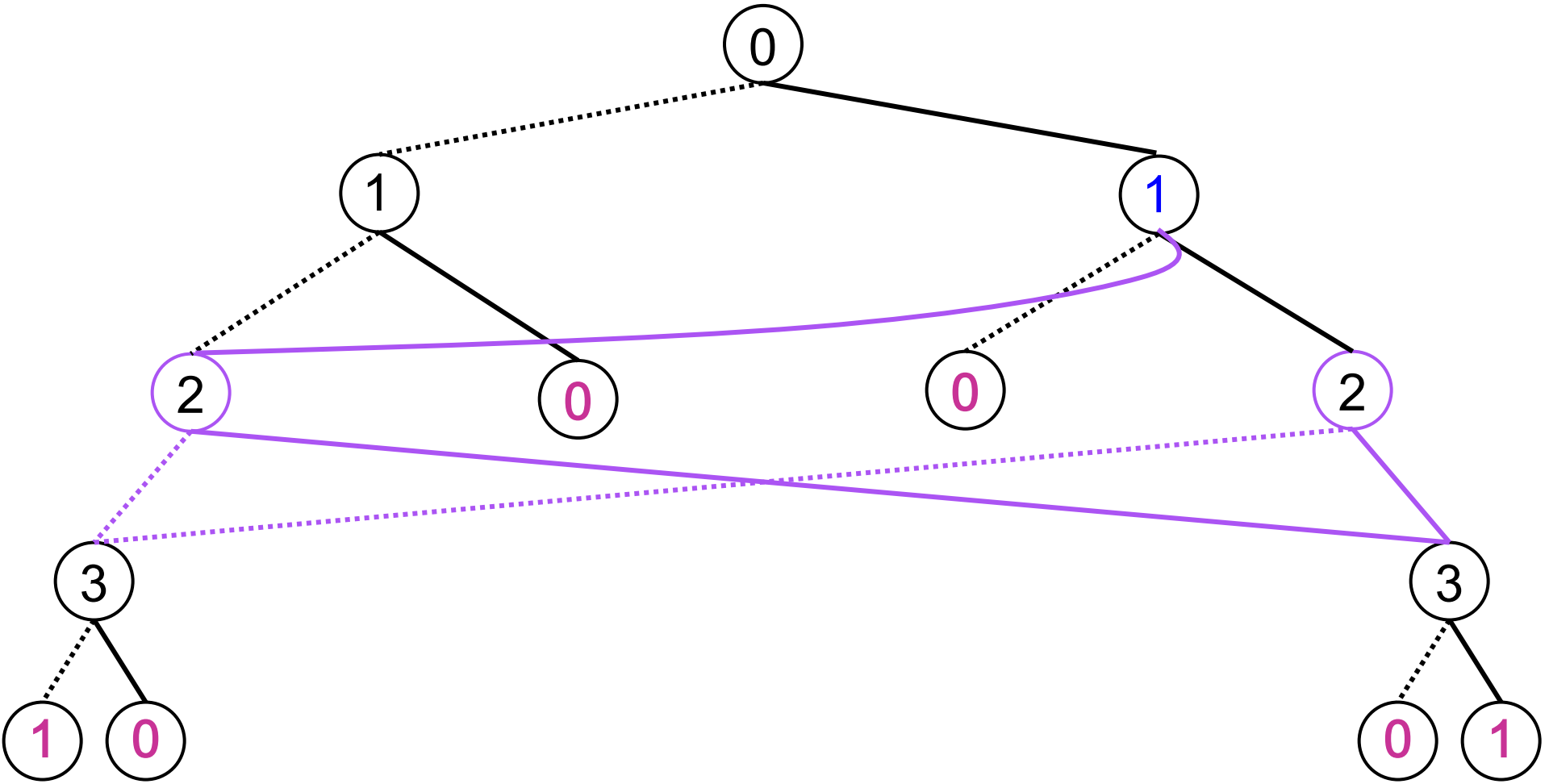
Ordre des variables fixé



Partage des sous-arbres identiques

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

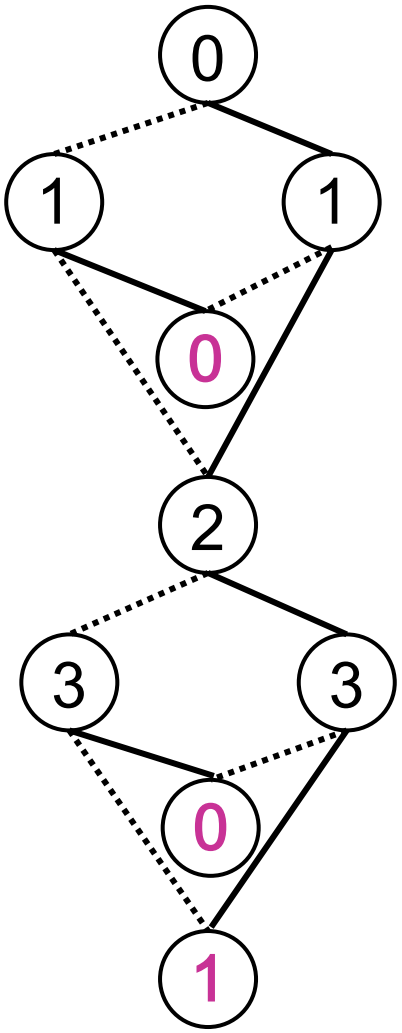
Ordre des variables fixé



ROBDD = Reduced Ordered Binary Decision Diagrams

$f(x_0, x_1, x_2, x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$

Ordre des variables fixé



forme canonique
pour un ordre de variables

test d'égalité en temps 1
(comparaison de pointeurs)

Satisfiabilité triviale
comptage des solutions linéaire
(les chemins vers 1)

Validité triviale
formule réduite à 1 !

donc difficile à calculer dans le pire cas....

L'invention des BDDs

Le calcul Booléen de grande échelle paraissait impossible en pratique jusqu'à la fin des années 1980...
...et l'invention des BDDs et leurs implémentations efficaces

- Premières idées : Shannon, Lee (1959), Akers (1978), Boute (1976)
- Vraie exploitation, structures de données : Bryant (1986), Billon, Billon & Madre (1988), Coudert & Madre (1990) Model-checking, sûreté de fonctionnement, etc.
- Systèmes : TiGeR (Madre, Coudert, Touati) → Esterel
  SMV (McMillan), CUDD (Somenzi), etc.

Référence moderne : D. Knuth vol. 4, voir bibliographie

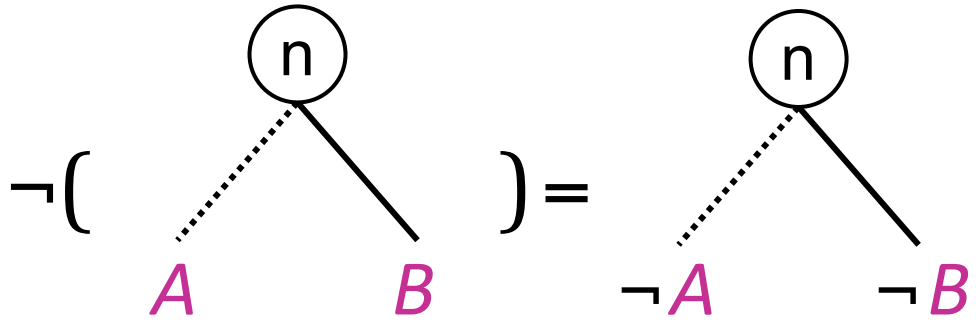
Eléments de complexité

- Le problème est toujours la taille des BDDs. Pour *n* variables
  - Fonction aléatoire : 2^n
  - Addition / soustraction binaires : *n* (avec le bon ordre)
  - Fonctions symétriques : n^2

    (invariantes par permutation des variables)
  - Multiplication n bits : 2^n (mais BMDs de Bryant n^2)
  - Vérification d'algorithmes asynchrones : difficile
  - Circuits et logiciels synchrones (Esterel) : souvent bien

Très sensible au nombre des variables
et à l'ordre des variables, discuté plus loin

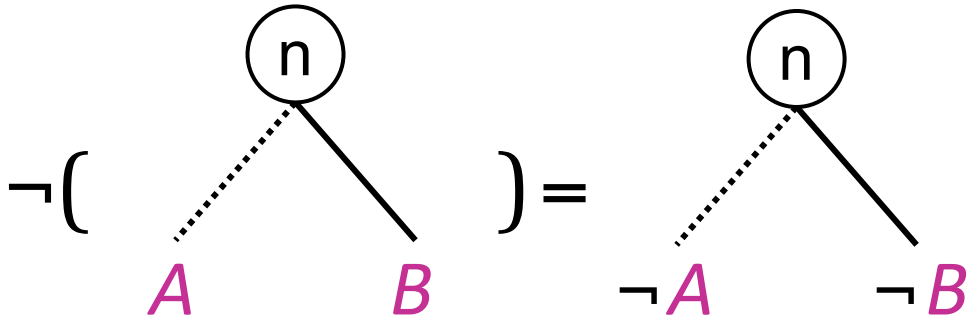
Négation récursive avec mémo-fonction



$$\neg mux(x_n, A, B)$$
$$= \neg((x_n \wedge A) \vee (\overline{x}_n \wedge B))$$
$$= \neg(x_n \wedge A) \wedge \neg(\overline{x}_n \wedge B)$$
$$= (\overline{x}_n \vee \neg A) \wedge (x_n \vee \neg B)$$
$$= (\overline{x}_n \wedge x_n) \vee (\overline{x}_n \wedge \neg B) \vee (x_n \wedge \neg A) \vee (\neg A \wedge \neg B)$$
$$= (x_n \wedge \neg A) \vee (\overline{x}_n \wedge \neg B)$$
$$= mux(x_n, \neg A, \neg B)$$

redondant !

Négation récursive avec mémo-fonction



Attention :

implémentation directe exponentielle

car exploration totale du graphe déplié !

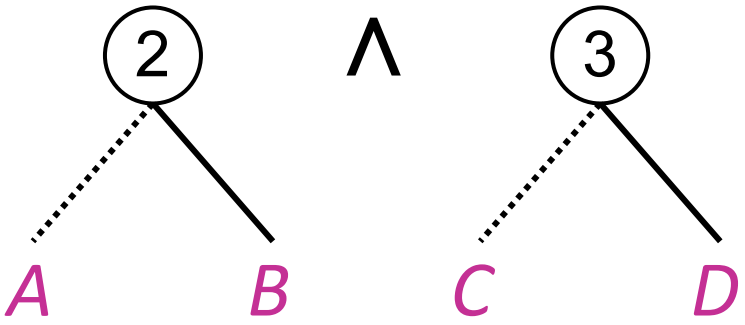
Solution :

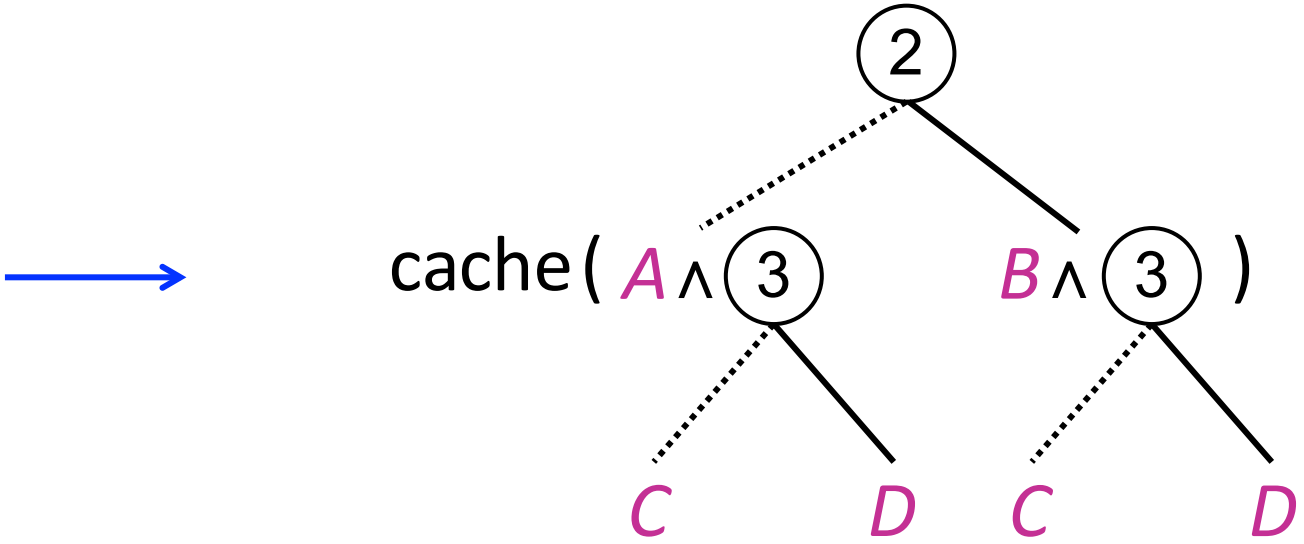
utiliser un cache des négations déjà calculées,

i.e., une table de $A \rightarrow BDD(\neg A)$

→ coût linéaire en temps

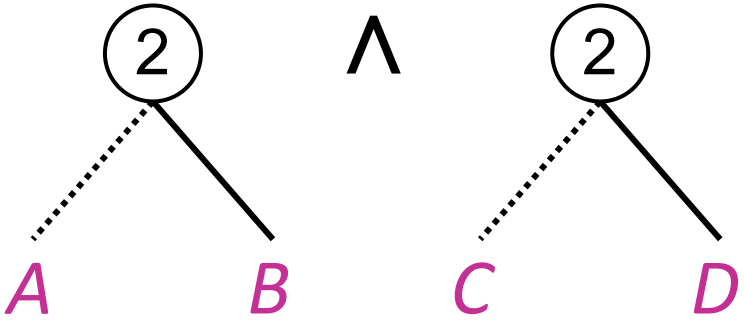
Opérations binaires récursives





$$mux(x_2, A, B) \wedge mux(x_3, C, D) = mux(x_2, A \wedge mux(x_3, C, D), B \wedge mux(x_3, C, D))$$

Opérations binaires récursives



$$mux(x_n, A, B) \wedge mux(x_n, C, D) = mux(x_n, A \wedge C, B \wedge D)$$

Coût n^2 grâce au cache
Idem pour ou, xor, mux, etc.

Structures de données pour le partage

- Implémentation de noeud et table de BDDs :

groupés dans un mot

→ 4 mots

```
class BDD_Node {
  int n ;  // numéro de la variable (fixe)
  int r ;   // rang dans l'ordre des variables (mobile)
  int refCount ; // compteur de références
                 // pour ramasse-miettes
  BDD_Node* f0 ;   // fils pour n = faux
  BDD_Node* f1 ;   // fils pour n = vrai
  BDD_Node* next ;  // suivant dans la table
};
class BDD_Table : HashTable(BDD_Node) { ... };
```

Le hachage sur (f0,f1) garantit l'unicité

Implémentation d'un opérateur binaire

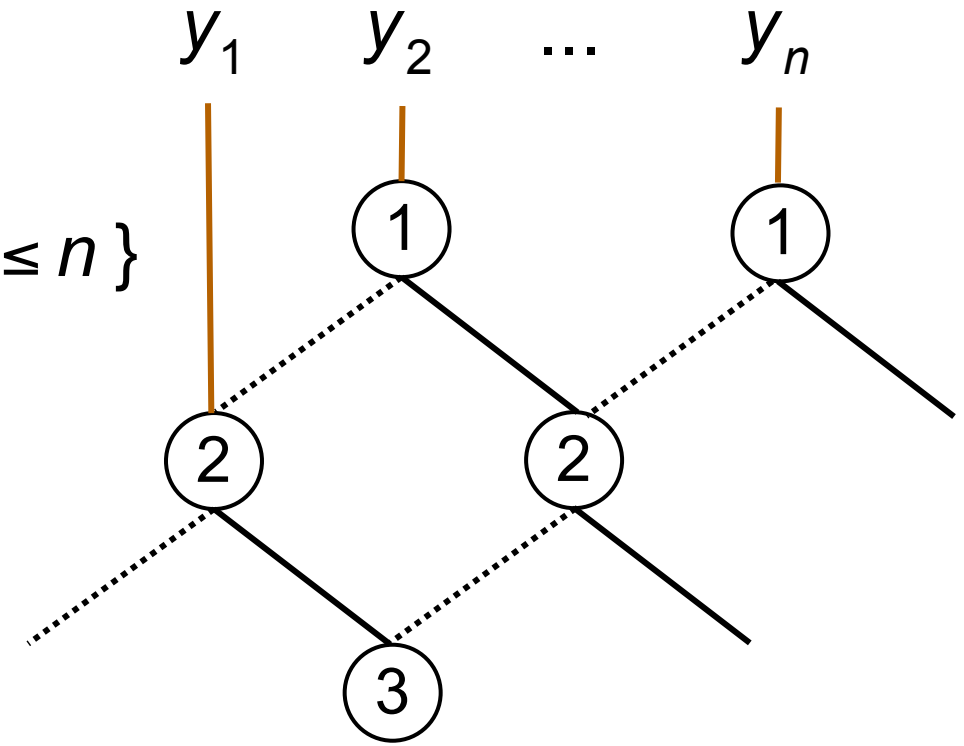
```
BddAnd(node1, node2) :
    if (node1 == Zero) or (node2 == Zero) return Zero
    if (node1 == One) return node2
    if (node2 == One) return node1
    result = AndCacheGet(node1, node2)
    if (result != Null) return result ;
    if (node1->r < node2->r)
        result = NodeCreate(node1->n,
                            node1->r,
                            BddAnd(node1->f0, node2),
                            BddAnd(node1->f1, node2))
    else if (node1->r == node2->r)
        result = ...
    else if (node1->r > node2->r)
        result = ...
    AndCacheAdd(node1, node2, result)
    return result
```

Le partage est généralisé à tous les BDDs

- Les BDDs pour une fonction partagent au maximum

$$(y_1,\ldots,y_n) = f(x_1,\ldots,x_m)$$

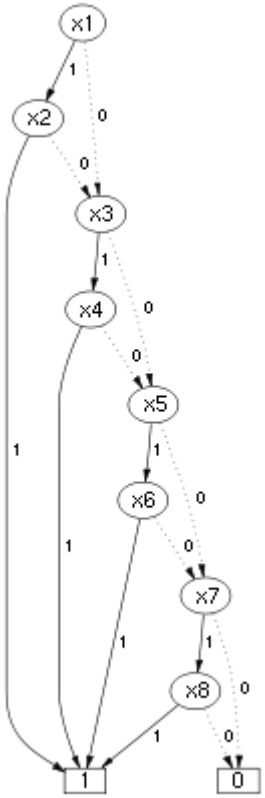
$$\{\, y_i = f_i(x_1,\ldots,x_m) \mid 1 \le i \le n \,\}$$

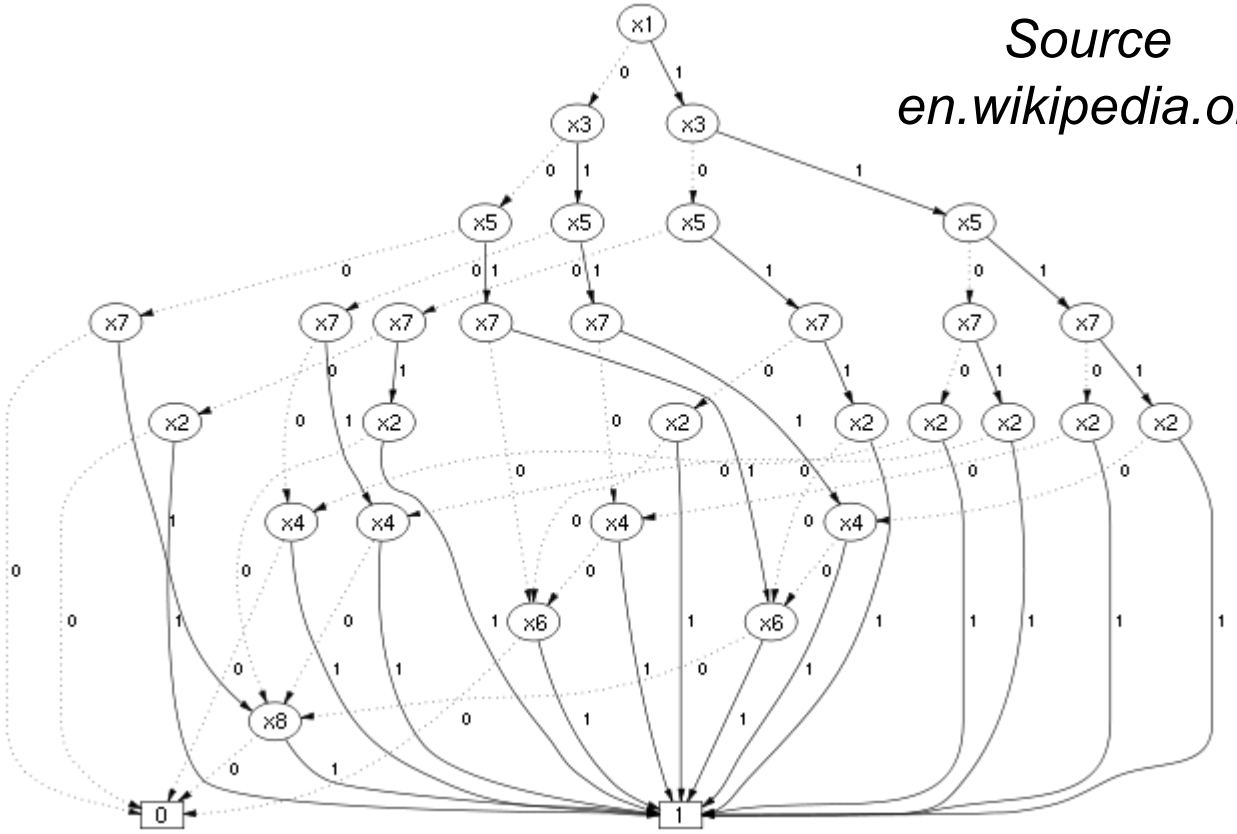


- Et même choses entre fonctions

Equivalence de formules ⇔ égalité de fonctions
⇔ égalité de pointeurs

Mais forte dépendance à l'ordre des variables





Source en.wikipedia.org

$$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6) \wedge (x_7 \vee x_8)$$

Méthodes de choix d'ordre / réordonnancement

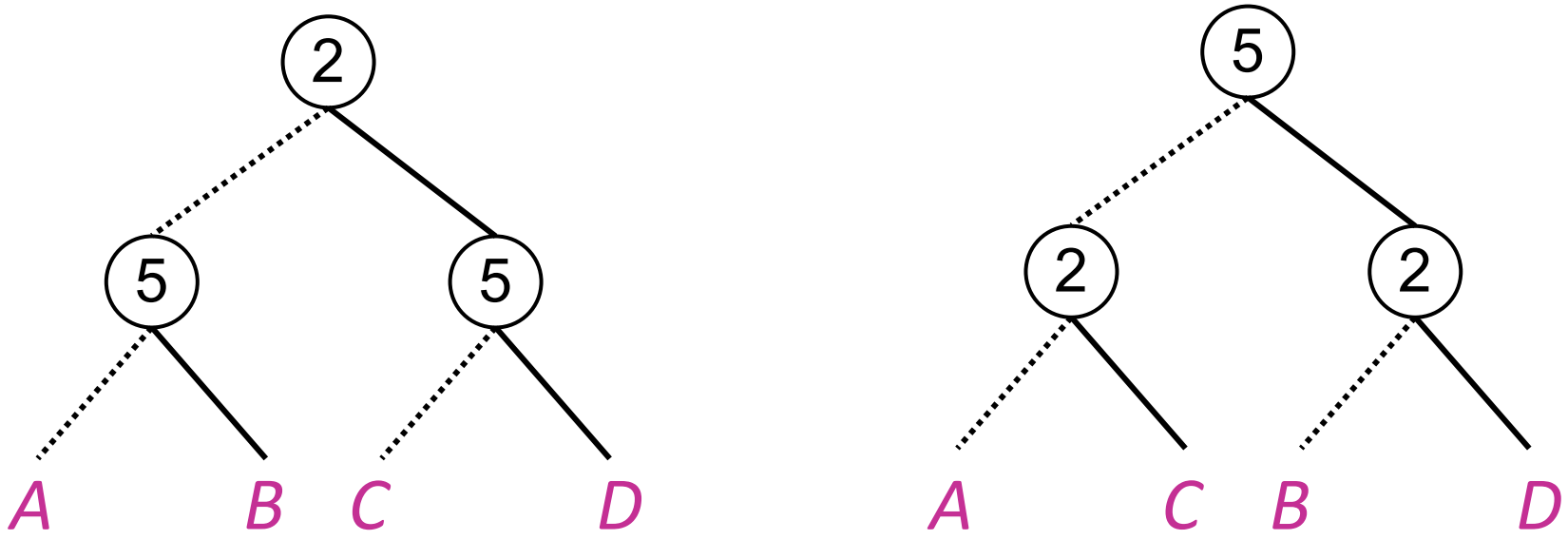
Trouver le bon ordre est NP-complet : $n!$ ordres → heuristiques

- Ordonnancement statique par analyse d'un circuit
  - mettre en avant les variables apparemment influentes
  - topologie : en premier les entrées qui sont loin des sorties
    ou qui influencent beaucoup d'autres variables
    \+ ensemble des variables topologiquement proches
  - mettre en premier les variables qui simplifient le circuit quand on fixe leur valeur à 0 ou 1
  - ...
- Ordonnancement dynamique (Fujita, Rudell)
  - changer l'ordre dès que la taille croît trop

  essentiel en pratique

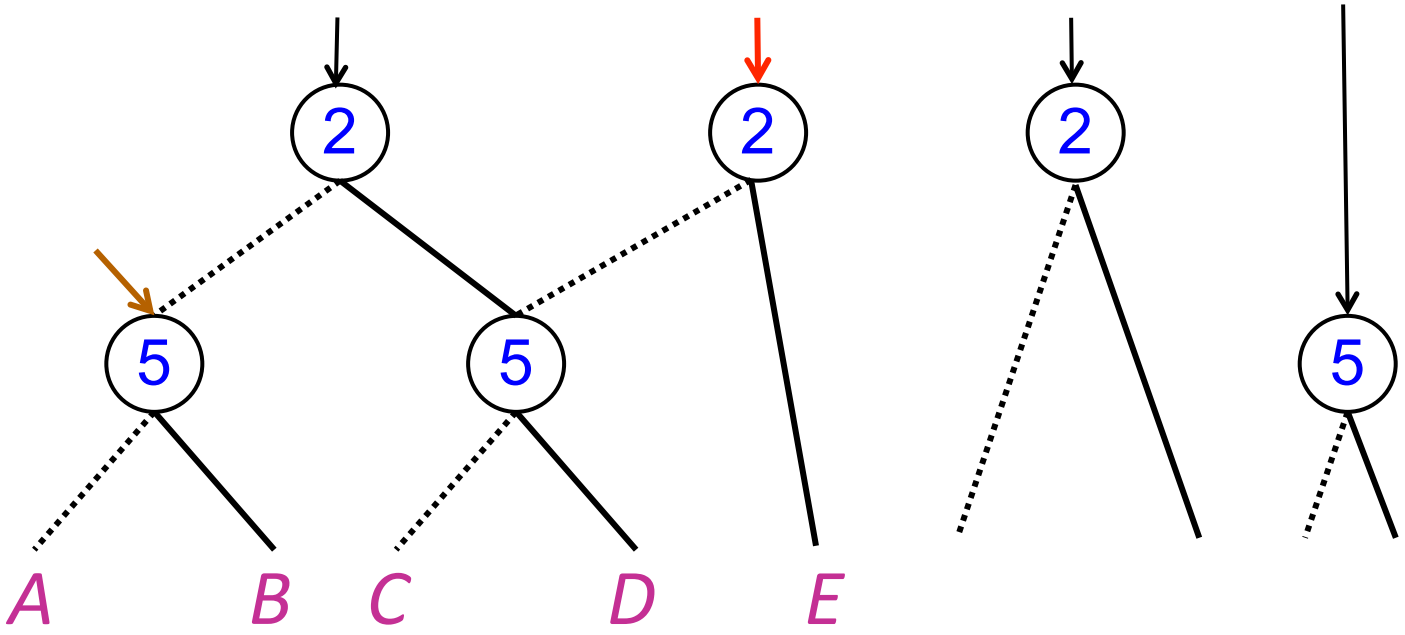
Réordonnancement dynamique

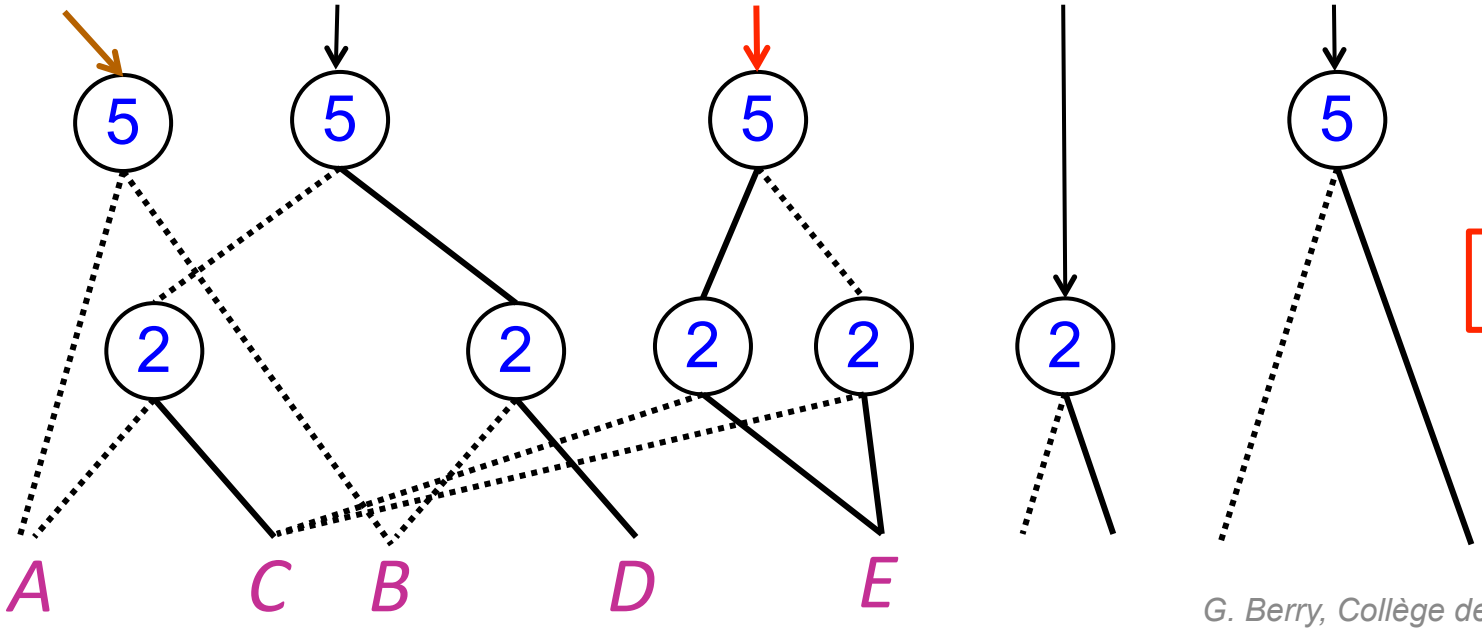
Idée : il est facile de transposer deux variables adjacentes



se fait en modifiant des valeurs et pointeurs en place

Mais il faut aussi créer / supprimer des nœuds !





plus grand

plus petit

réversible !

Deux méthodes dynamiques

La transposition ne fait que des modifications locales

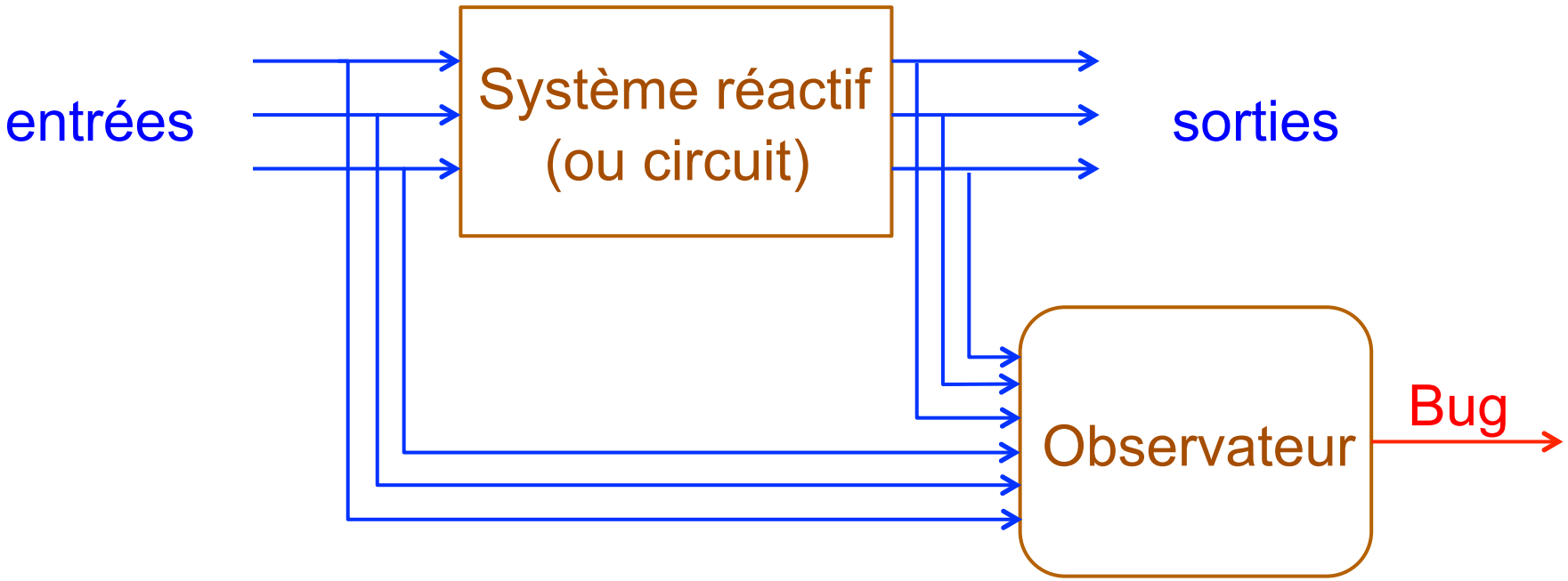
1. Window-permutation, Fujita *et.al.*, 1991
   - choisir une variable x_i , par exemple là où la largeur est grande
     - choisir une fenêtre de 4-5 variables autour de x_i
     - en combinant des transpositions, essayer toutes les permutations de la fenêtre, choisir celle qui minimise le nombre de noeuds
   - itérer ces opérations sur toutes les variables (ou jusqu'à plus soif)
   - méthode assez rapide, mais trop locale

2. Sifting (criblage), R. Rudell, 1993
   - criblage d'une variable x_i par transpositions successives : la descendre tout en bas puis la monter tout en haut des BDDs.
   - garder la position qui a minimisé le nombre de noeuds
   - faire ça successivement pour toutes les variables
   - plus lent, n^2 transpositions si n variables, mais meilleurs résultats

Agenda

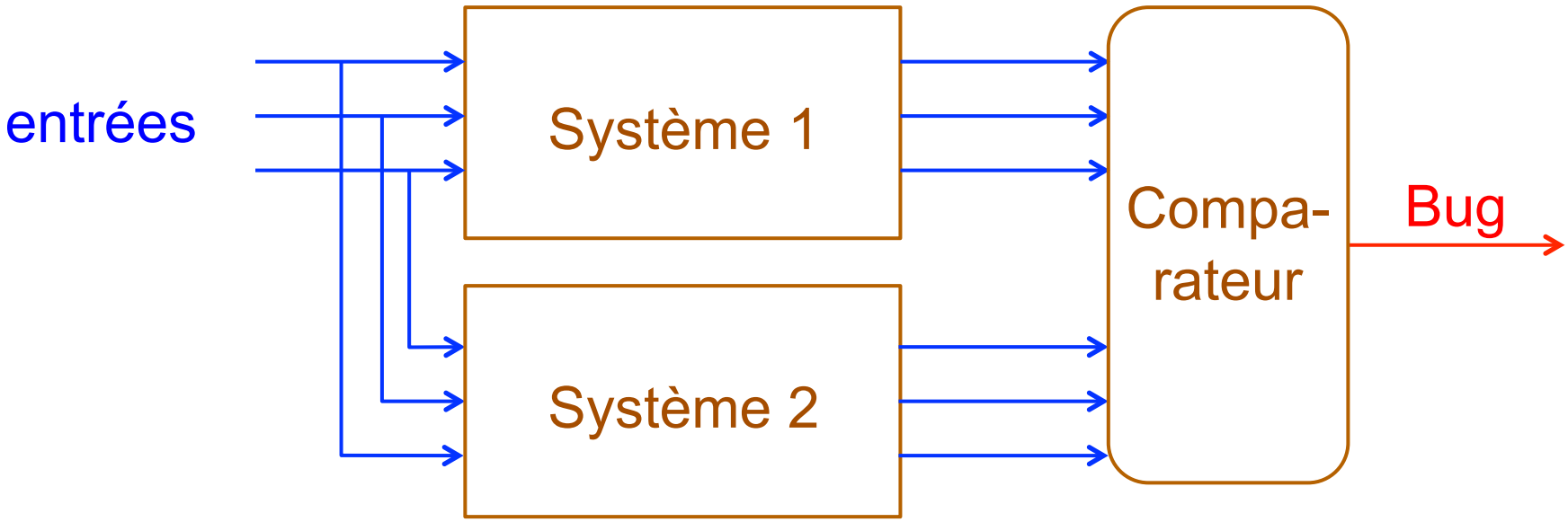
1. Présentation général du cours 2016
2. Rappels sur la complexité algorithmique
3. Le calcul Booléen
4. Les BDDs
5. Application des BDDs à la vérification
6. Les ZDDs
7. Conclusion

Vérification booléenne de sûreté



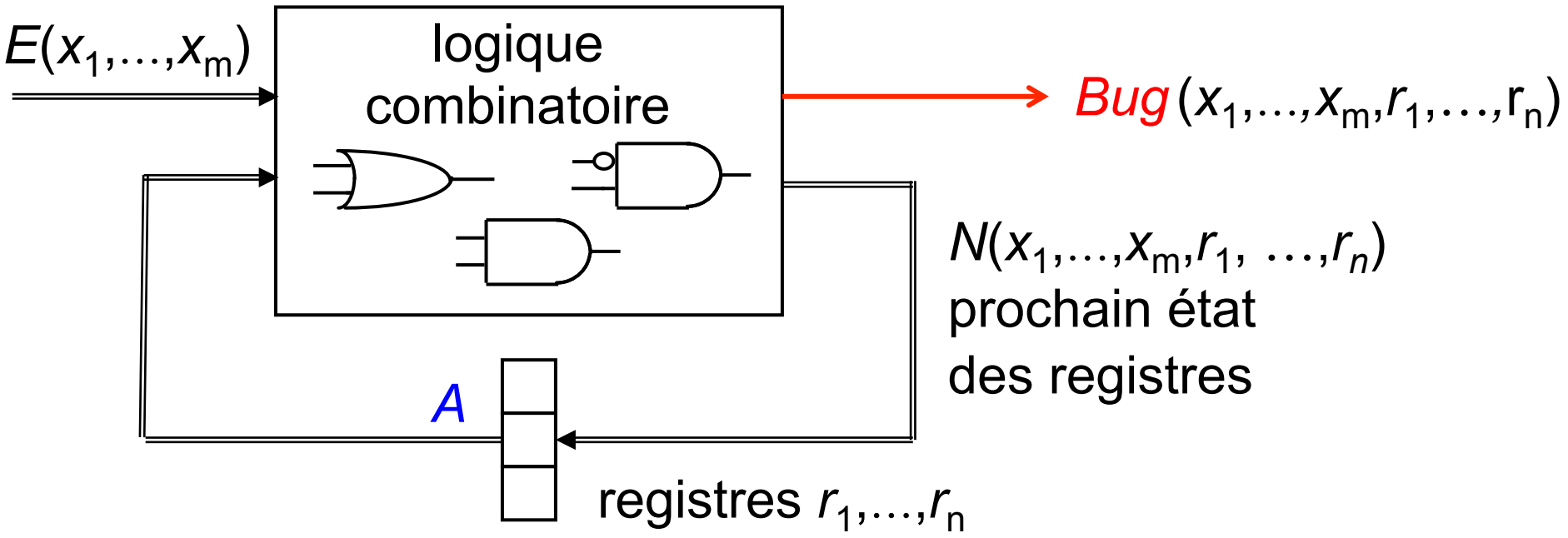
Formule booléenne exprimant que Bug n'est jamais émis :
pour tout état accessible R, O et toute entrée valide E
du couple système / observateur, le booléen Bug est faux.

Vérification booléenne d'équivalence



Formule booléenne exprimant que Bug n'est jamais émis :
pour tout état accessible R1, R2 et toute entrée valide E
du couple de systèmes, le booléen Bug est faux.

Vérification par calcul d'accessibilité (01/04/2015)



Entrées légales E, état initial : A_0

Etats accessibles en 1 coup : $A_1 = A_0 \cup N(E,A_0)$

Etats accessibles en n+1 coups : $A_{n+1} = A_n \cup N(E,A_n)$

Test d'arrêt : $A_{n+1} = A_n$

Etats accessibles : $A = A_n$

Vision Booléenne des calculs

- Calcul des images directes :

- Soient $F : B^m \rightarrow B^n$, $X \subset B^m$. Image $F(X) \subset B^n$ de X par F :

$$I = F(X) = \{ \vec{y} \in B^n \mid \exists \vec{x} \in X . \vec{y} = f(\vec{x}) \}$$

- Si F et X sont les vecteurs de fonctions caractéristiques de F et X, la fonction caractéristique I de $F(X)$ est donnée par :

$$I(\vec{y}) = \exists(\vec{x}). X(\vec{x}) \wedge \Lambda_i (y_i \Leftrightarrow F_i(\vec{x}))$$

- Formule à vérifier pour que Bug ne passe jamais à 1 :

$$\neg(\exists \vec{x},\vec{r} . E(\vec{x}) \wedge A(\vec{r}) \wedge Bug(\vec{x},\vec{r}))$$

Calcul de contre-exemple

- Si *Bug* peut valoir 1, il faut trouver un contre-exemple en forme de suites d'entrées conduisant au bug.
- Ici il faut calculer des images réciproques d'ensembles

Soient des valeurs t .q. $E(\vec{x}) \wedge A(\vec{r}) \wedge Bug(\vec{x},\vec{r})$

Voyage en arrière pour retrouver l'état initial :

soit i minimum tel que $A_i(\vec{r})$. Alors, la formule suivante est satisfiable:

$$\exists \vec{x}^{i-1}, \vec{r}^{\,i-1}. E(\vec{x}^{i-1}) \wedge A_{i-1}(\vec{r}^{\,i-1}) \wedge (\vec{r} \Leftrightarrow N(\vec{x}^{i-1},\vec{r}^{\,i-1}))$$

ce qui détermine des valeurs $\vec{x}^{i-1}, \vec{r}^{\,i-1}$, et ainsi de suite jusqu'à $\vec{x}^{0}, \vec{r}^{\,i-1}$.

La suite des vecteurs d'entrée $(\vec{x}^{0},....,\vec{x}^{i-1},\vec{x})$ fournit le contre-exemple

Les quantificateurs (Coudert & Madre)

- Ils sont définissables

$$\exists x_0 . P(x_0) =_{def} P(0) \vee P(1)$$

$$\forall x_0 . P(x_0) =_{def} P(0) \wedge P(1)$$

- Mais potentiellement explosifs

$$\exists x_0, x_1 . P(x_0, x_1) =_{def} P(0,0) \vee P(0,1) \vee P(1,0) \vee P(1,1)$$

...

Implémentation naïve → exponentielle !

⇒ contrôler de près la portée des quantificateurs
en partitionnant astucieusement la fonction :

$\exists x_0, x_1, x_2 . P \wedge Q \Leftrightarrow \exists x_0 . (\exists x_1 . P \wedge \exists x_2 . Q)$ si $x_2 \notin P$ et $x_1 \notin Q$

Images efficaces (Coudert & Madre)

- Images directes

$$F(A)(\vec{y}) = \exists(\vec{x}).\ A(\vec{x}) \wedge (\vec{y} \Leftrightarrow F(\vec{x}))$$

Interpolation : connaissant F et A, chercher une fonction F' telle $F'(A) = F(A)$ mais ayant un BDD plus petit

Opérateur restrict, cf. séminaire de J-C. Madre

- Images réciproques : pas détaillées ici

Tout cela est bien sûr automatique
et joyeusement ignoré de l'utilisateur

Calcul de la constructivité d'un circuit (26 mars 2014)

- Faire des calculs BDDs de point fixe analogues mais en logique à trois valeurs $\perp$, 0, 1, ce qui est équivalent à la propagation des valeurs en logique constructive
- Pour cela, travailler avec des paires de booléens (d,v), avec $\perp = (0,0)$, $0 = (1,0)$ et , $1 = (1,1)$

Implémenté dans le compilateur Esterel v5, option –causal.

Pour un circuit cyclique, permet la vérification formelle, et permet de construire un circuit acyclique équivalent

Usage majeur chez Dassault Aviation,
cf. séminaire de E. Ledinot du 16 avril 2013

Agenda

1. Présentation général du cours 2016
2. Rappels sur la complexité algorithmique
3. Le calcul Booléen
4. Les BDDs
5. Exemple d'applications des BDDs
6. Les ZDDs
7. Conclusion

Les ZDDs : Zero-Suppressed Decision Diagrams

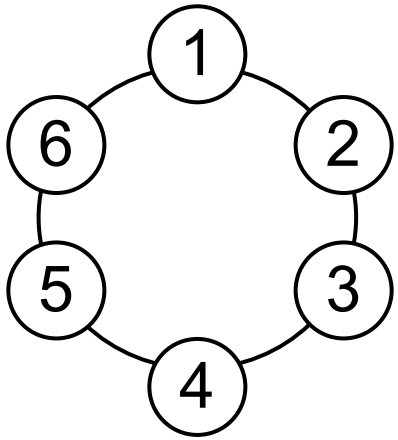
- Shin-ichi Minato, 1993 : les mêmes diagrammes, mais avec un autre sémantique.
- Représentation efficace d'ensembles de parties d'ensembles, supprimant beaucoup de flèches vers 0

  Pour *mux*(*x*,*A*,*B*) :
  - BDDs : ne pas créer un noeud si $A = B$
  - ZDDs : ne pas créer un noeud si $A = 0$

- Très efficace quand les 0 sont nombreux, et évitent l'apparition des variables inutiles

Les mêmes graphes, les mêmes algorithmes, mais des applications différentes !

Codage d'ensembles de parties (D. Knuth)

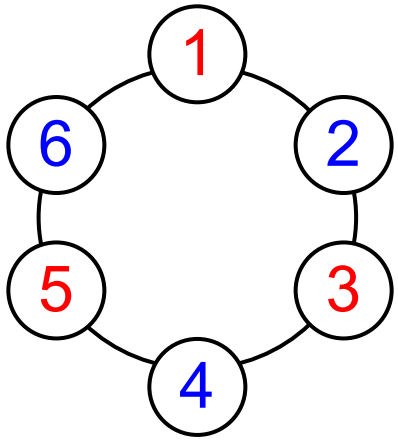


$x_1x_2x_3x_4x_5x_6 = 101010 \rightarrow \{1,3,5\}$

- Ensemble indépendant : pas d'arc entre deux éléments
- Noyau : ensemble indépendant maximal

Cf. Knuth vol.4, 7.1.4, page ?

Codage d'ensembles de parties (D. Knuth)



- Ensemble indépendant : pas d'arc entre deux éléments
- Noyau : ensemble indépendant maximal

Noyau :

$101010 \rightarrow \{1,3,5\}$

$100100 \rightarrow \{1,4\}$

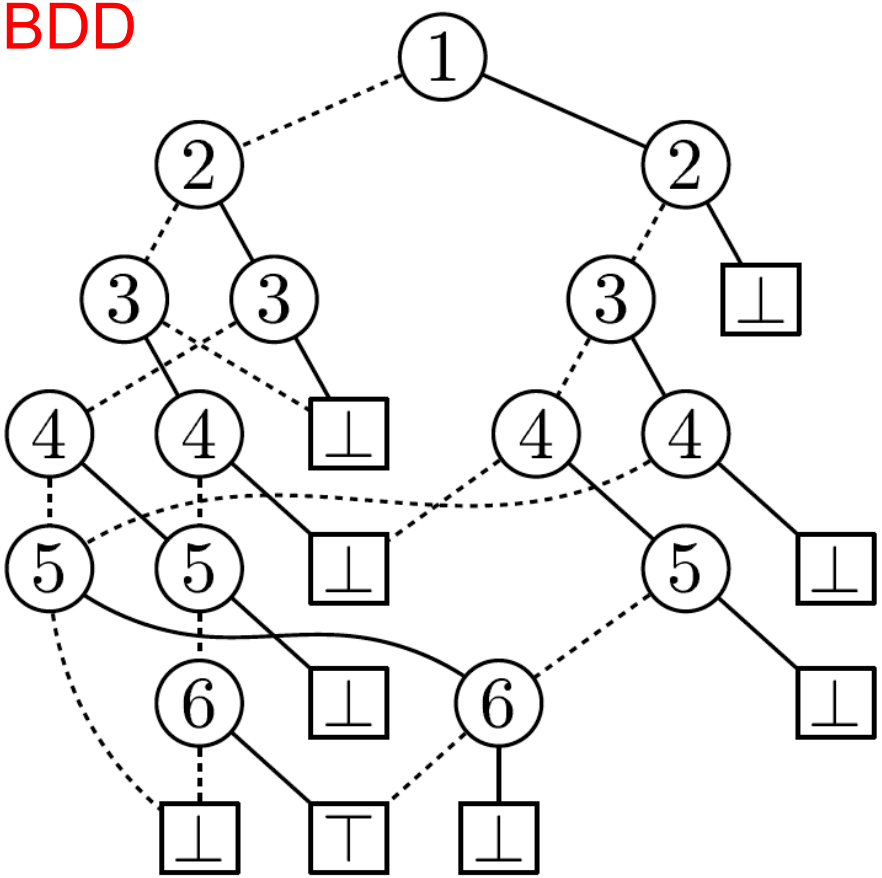
$010101 \rightarrow \{2,4,6\}$

$010010 \rightarrow \{2,5\}$

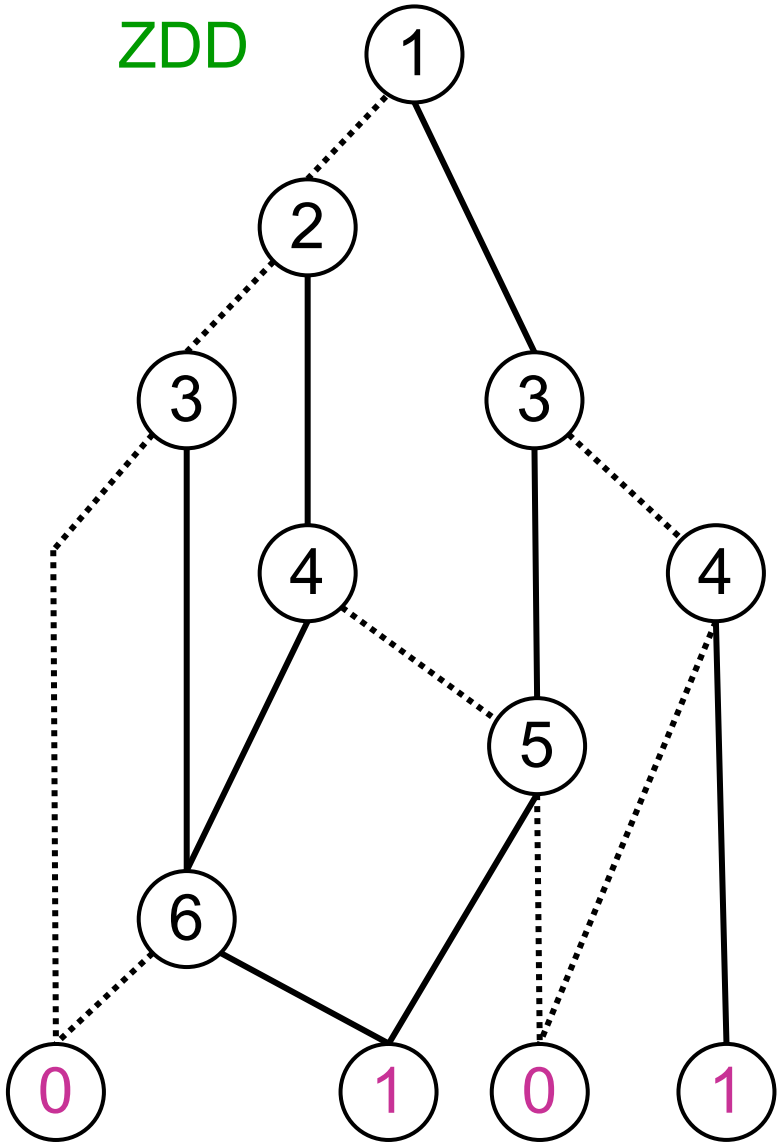
$001001 \rightarrow \{3,6\}$

Noyau : { {1,3,5}, {1,4}, {2,4,6}, {2,5}, {3,6} }

BDD



ZDD

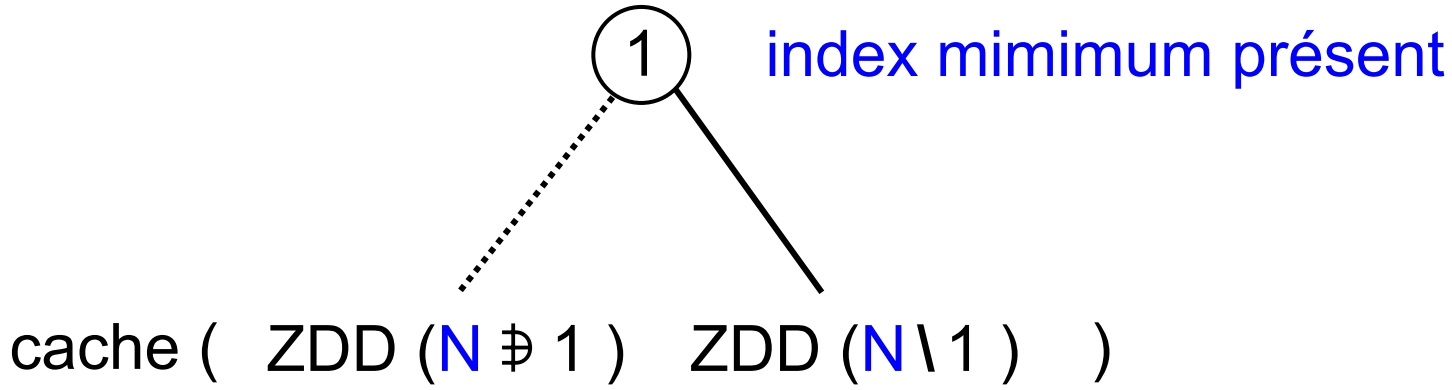


Knuth vol.4, 7.1.4., fig. 12 p. 208
fig. 122 p. 249
⊥=0, ⊤= 1, implicitement partagés

Construction récursive du ZDD

Noyau N : { {1,3,5}, {1,4}, {2,4,6}, {2,5}, {3,6} }

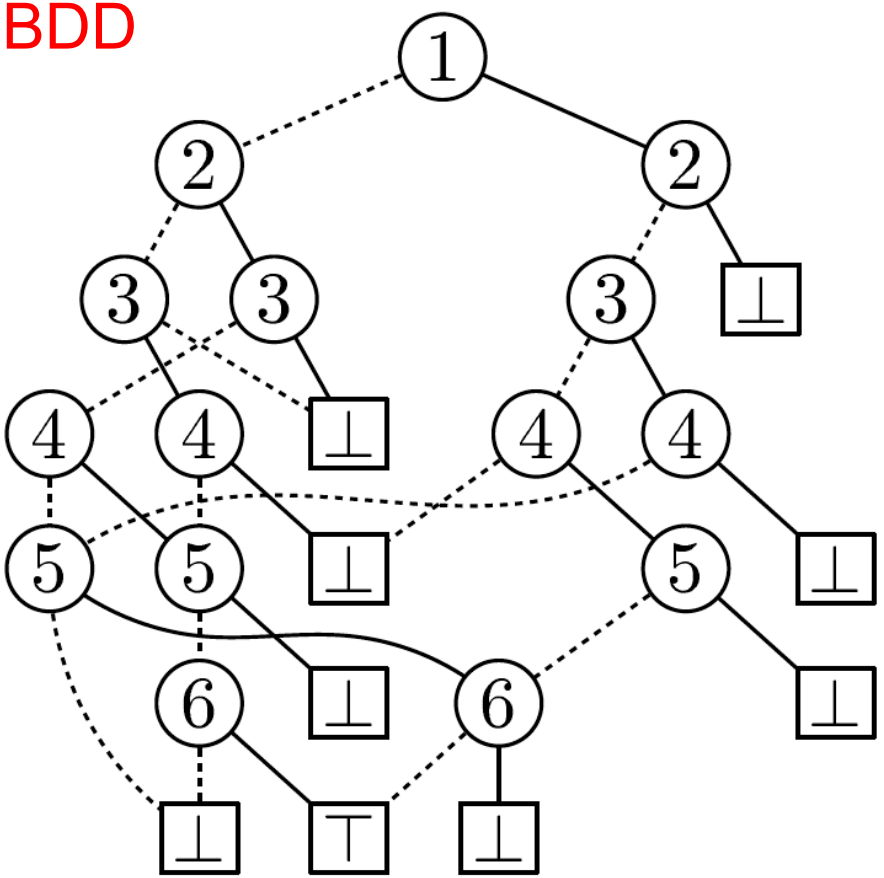
enlever les 1 : $N \setminus 1$ = { {3,5}, {4} }
les parties sans 1 : $N \not\ni 1$ = { {2,4,6}, {2,5}, {3,6} }



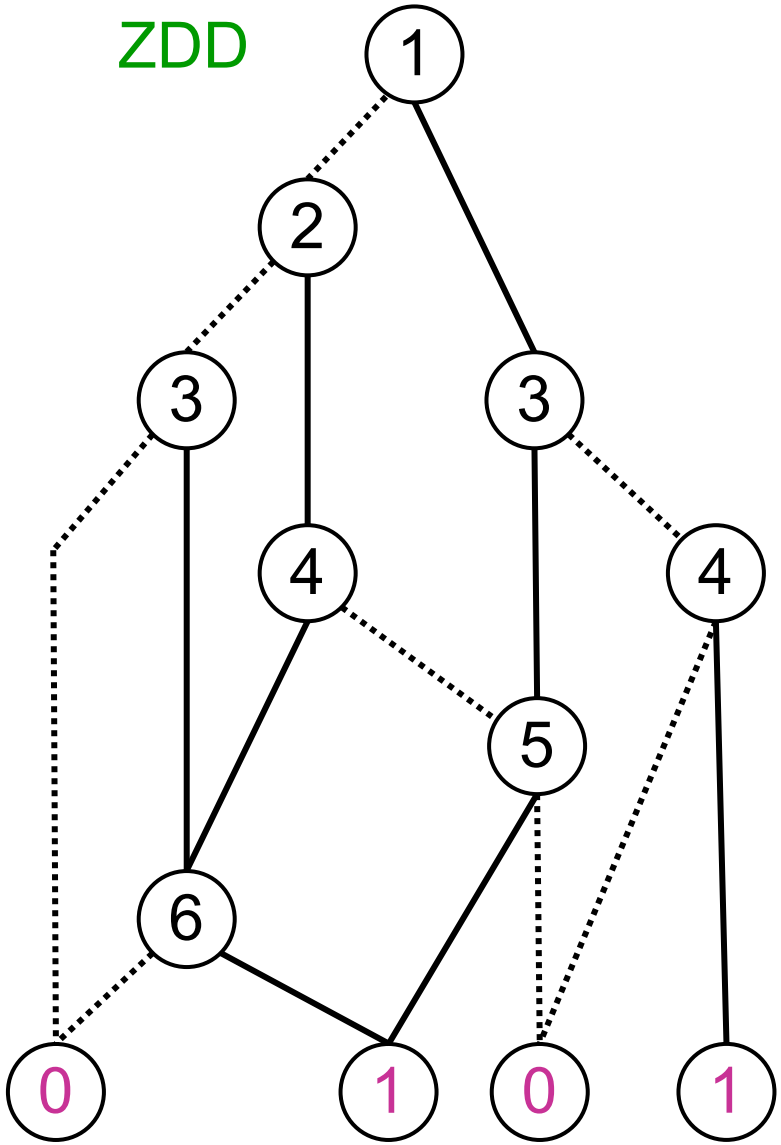
D. Knuth :
ZDD = the most beautiful structure in Computer Science

Noyau : { {1,3,5}, {1,4}, {2,4,6}, {2,5}, {3,6} }

BDD



ZDD



Knuth vol.4, 7.1.4., fig. 12 p. 208
fig. 122 p. 249
⊥=0, ⊤= 1, implicitement partagés

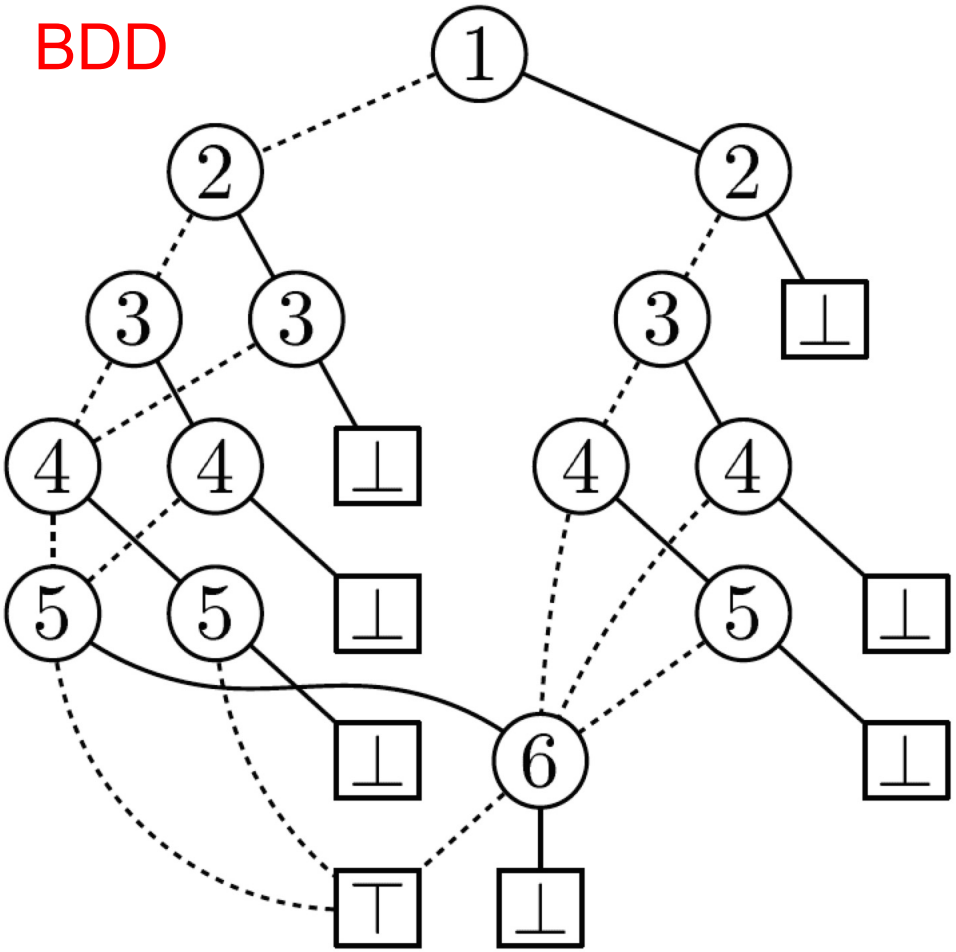
Ensembles indépendants

- pas d'arc entre deux éléments

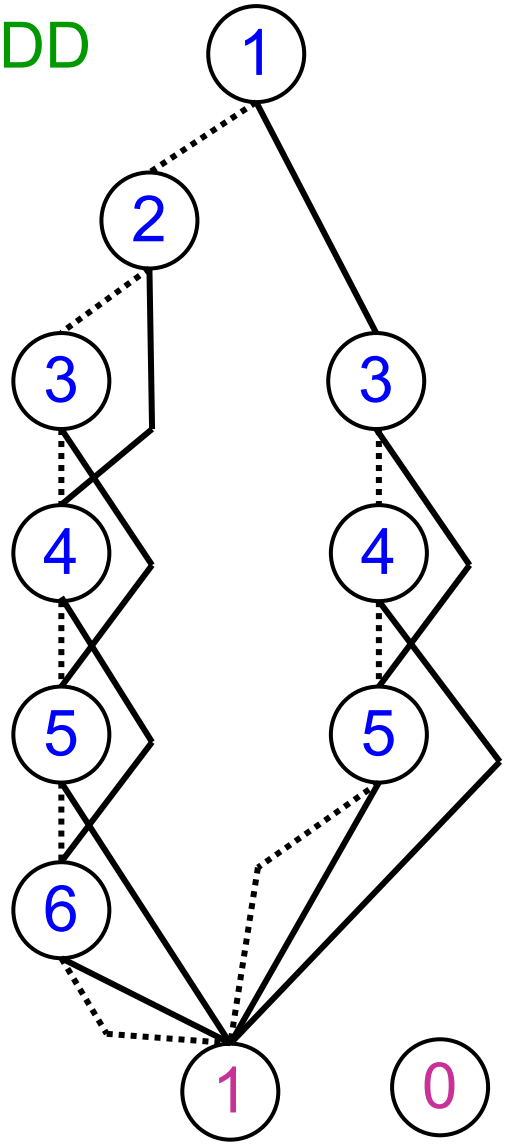
000000 → { }	010010 → {2,5}
100000 → {1}	010010 → {2,6}
101000 → {1,3}	001000 → {3}
101010 → {1,3,5}	001010 → {3,5}
100100 → {1,4}	001001 → {3,6}
100010 → {1,5}	000100 → {4}
100001 → {1,6}	000101 → {4,6}
010000 → {2}	000010 → {5}
010100 → {2,4}	000001 → {6}

Ensembles indépendants

BDD



ZDD



Knuth vol.4, 7.1.4., fig. 12 p. 208
fig. 122 p. 249

Impliquants premiers

- Définition

  DNF : $A = (\overline{x} \wedge \overline{y} \wedge z) \vee (z \wedge \overline{t}) = \overline{x}\overline{y}z + z\overline{t}$

  Les monômes sont des impliquants : $\overline{x}\overline{y}z \Rightarrow A$ et $z\overline{t} \Rightarrow A$

  Mais pas forcément miminaux : si $B = x + x\overline{y}$, alors $x\overline{y}$ est inutile

  Impliquant premier = impliquant minimal en taille

  fonction = somme des impliquants premiers

→ Minimisation des expressions logiques dans les circuits

→Analyse de défaillances : si A désigne une panne,

- les impliquants premiers sont les causes élémentaires de la panne
- leur calcul peut être étendu aux probabilités de pannes

→ Recherche d'associations fréquentes (*patterns*) en analyse de données (*Bigdata*)

Calcul des impliquants premiers (Reusch 1975)

1. On a $I(x) = \{x\}$ et $I(\bar{x}) = \{\bar{x}\}$
2. Soit $A = mux(x, A_1, A_0)$. Posons $I_1 = I(A_1)$, $I_0 = I(A_0)$ et $I_\cap = I_1 \cap I_0$.
   Alors :
   $I(A) = I_\cap \cup \{xm \mid m \in I_1 - I_\cap\} \cup \{\bar{x}m \mid m \in I_0 - I_\cap\}$

- Les impliquants étant des sous–ensembles de littéraux, les ZDDs sont idéaux pour ces opérations ensemblistes (+ recherche de tous les impliquants premiers contenant y, etc.)

BDDs et ZDDs ont la même machine de calcul, mais réalisent des opérations bien différentes !

Conclusion

- *DDs : une structure de données novatrice adaptée au calcul Booléen et à de nombreux problèmes combinatoires
  - BDDs : fonctions Booléennes
  - ZDDs : parties d'ensembles
  - BMDs (Bryant) : multiplieurs
  - IDDs (Vuillemin) : très grands nombres creux
  - QuadTrees en géométrie
  - ..
- D'excellentes implémentations en accès libre (CUDD, Boulder)
- Mais maintenant supplantée par les solveurs Booléens pour le la satisfaction Booléenne (problème SAT)...

→ Soyez-là au prochain cours / séminaire !

Références historiques

C. Y. Lee. *Representation of Switching Circuits by Binary-Decision Programs*.
Bell Systems Technical Journal, 38:985–999, 1959.

S. B. Akers. *Binary Decision Diagrams*.
IEEE Transactions on Computers, C-27(6):509–516, June 1978.

R. T. Boute. *The Binary Decision Machine as a programmable controller*.
EUROMICRO Newsletter, Vol. 1(2):16–22, January 1976.

R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation.
IEEE Transactions on Computers, C-35(8):677–691, 1986.

J-P. Billon. *Perfect Normal Forms for Discrete Functions.*
Bull Research Report 87019, 1987.

J-P. Billon, J-C. Madre. *Original Concepts of PRIAM, an Industrial Tool for Efficient Formal Verification of Combinational Circuits*
In *The Fusion of Hardware Design and Verification*, G,. Milne Editor, North-Holland 1988

K. S. Brace, R. L. Rudell and R. E. Bryant. "Efficient Implementation of a BDD Package".
In Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC 1990), pages 40–45. IEEE Computer Society Press, 1990.

R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, ACM Computing Surveys, Vol. 24, No. 3 (September, 1992), pp. 293–318.

Références modernes

M. Fujita, Y. Matsunaga, and T. Kakuda. *On Variable Ordering for Binary Decision Diagrams for the application of Multi-Level Logic Synthesis.*
Proc. *EDAC'91,* European Design Automation Conference 1991.

R. Rudell. *Dynamic Variable Ordering for Ordered Binary Decision Diagrams.*
Proc. *ICCAD'93*, International Conference of on Computer-Aided Design 1993.

Shin-ichi Minato. *Zero-Suppressed Binary Decision Diagrams*
Proc DAC'1993, Design Automation Conference, 1993, pp. 272-277

La bible

D. Knuth. *The Art of Computer Programming, Vol. 4 : Combinatorial Algorithms*
Section 7.1.4 : Binary Decision Diagrams
Addison Wesley, 2014