Binary Decision Diagrams

Rémi Hutin & Joshua Peignier

6 Mai 2016

1 Introduction

Ce document traite la version informatique du sujet proposé, et présente donc des réponses détaillées aux question 1 à 3, ainsi que les points intéressants et résultats de la partie qui concerne le code.
Pour compiler et exécuter notre code, extrayez le contenu de l'archive fournie dans un répertoire de votre choix. Rendez-vous dans ce répertoire à l'aide d'un terminal, puis compilez avec `make`, et exécutez avec `./main`. Notre code, en plus des fonctions demandées dans le sujet, présente un exemple de la résolution du problèmes des n dames pour $n = 12$.
Remarque préliminaire : Dans tout ce document, on admettra l'existence d'une bijection entre l'ensemble des formules INF et l'ensemble des BDD.

2 BDD et forme normale if-then-else

Question 1 Soient φ une formule, x une variable et V une valuation.
On veut montrer que $\varphi \equiv \varphi \uparrow^x$.
Remarquons que, par définition, $\varphi \uparrow^x \equiv (x \wedge \varphi[1/x]) \vee (\neg x \wedge \varphi[0/x])$.

Voici donc la table de vérité de $\varphi \uparrow^x$ en fonction des valeurs de x et de φ pour chaque valuation (on ne remplit que les cases nécessaires).

x	φ	$\varphi[1/x]$	$x \wedge \varphi[1/x]$	$\varphi[0/x]$	$\neg x \wedge \varphi[0/x]$	$\varphi \uparrow^x$
0	0		0	0	0	0
0	1		0	1	1	1
1	0	0	0		0	0
1	1	1	1		0	1

Il suffit alors de remarquer que les colonnes de φ et $\varphi \uparrow^x$ sont égales.

Question 2 On veut montrer que toute formule est équivalente à une formule INF. Procédons par induction. (Suivant la remarque préliminaire établie en introduction, on confondra les formules INF et leurs BDD associés.)

Remarque : Dans cette démonstration, lorsqu'on traitera le cas des connecteurs binaires, on sera amené à remplacer des feuilles d'une formule INF ${f'}_1$ par une deuxième formule INF ${f'}_2$

(le tout pour former une formule INF φ'). Notons que nous devons toutefois procéder à des simplifications : si ${f'}_2$ contient une formule INF $F = x \to F_1, F_0$, et que x est également présente dans ${f'}_1$, alors dans le diagramme de φ', il faut s'assurer qu'on ne teste pas x deux fois sur le même chemin. Donc dans une branche qui se trouve sur le fils faible de x dans ${f'}_1$, si un noeud x réapparaît plus loin, on supprime ce deuxième noeud x ainsi que son fils fort F_1, et on le remplace par son fils faible F_0 (et symétriquement si on est sur une branche sur le fils fort de x dans ${f'}_1$). On sous-entendra ces suppressions de noeuds par la suite, mais elles sont nécessaires pour la construction de la formule INF.

- Si notre formule φ est réduite à une variable x, sans utilisation d'opérateurs, alors φ est équivalente à la formule INF $x \to 1, 0$.
- Si la formule φ s'écrit $\neg f$ et qu'on suppose que f est équivalente à une formule INF f', alors φ est équivalente à la formule INF φ' obtenue en transformant, dans le BDD correspondant à f', les feuilles 0 en 1 et les feuilles 1 en 0. Ainsi, φ est vraie si et seulement si f est fausse, si et seulement si f' est fausse, si et seulement si φ' est vraie.
- Si φ s'écrit $f_1 \wedge f_2$ et qu'on suppose que f_1 et f_2 sont toutes les deux équivalentes à des formules INF ${f'}_1$ et ${f'}_2$, alors on peut construire une formule INF φ' équivalente à φ à partir de ${f'}_1$; il suffit, dans le BDD correspondant à ${f'}_1$, de remplacer les feuilles 1 par le BDD complet de ${f'}_2$.

  Ainsi, si on parcourt φ' avec une valuation telle que ${f'}_1$ soit fausse, on tombe sur 0, donc φ' est fausse. Et si on la parcourt avec une valuation qui rende ${f'}_1$ vraie, alors on arrive sur la racine de ${f'}_2$, et on continue les tests, donc φ' a la même valeur que ${f'}_2$. Donc φ' est vraie si et seulement si ${f'}_1$ et ${f'}_2$ sont vraies. Donc φ' est bien équivalente à φ.
- Si φ s'écrit $f_1 \vee f_2$ et qu'on suppose que f_1 et f_2 sont toutes les deux équivalentes à des formules INF ${f'}_1$ et ${f'}_2$, on procède de la même manière qu'avant en construisant une formule φ' à partir de ${f'}_1$, mais cette fois, on remplace ses feuilles 0 par la formule INF de ${f'}_2$.

  Ainsi, si on choisit une valuation qui rende ${f'}_1$ est vraie, le parcours dans φ' avec cette valuation nous amène à un noeud 1, donc est φ' est vraie. Sinon, on arrive sur la racine de ${f'}_2$ et on teste cette dernière ; alors φ' a la valeur de ${f'}_2$. Donc φ' est vraie si et seulement si au moins l'une des deux formules entre ${f'}_1$ et ${f'}_2$ est vraie. Donc φ' est équivalente à φ.
- Si φ s'écrit $f_1 \Rightarrow f_2$ et qu'on suppose que f_1 et f_2 sont toutes les deux équivalentes à des formules INF ${f'}_1$ et ${f'}_2$, on procède de la même manière qu'avant en construisant une formule φ' à partir de ${f'}_1$; mais cette fois, on change les feuilles 0 de ${f'}_1$ en 1, et on remplace ses feuilles 1 par ${f'}_2$.

  Ainsi, une valuation qui rend ${f'}_1$ fausse nous amène sur la valeur 1 dans φ', donc φ' est vraie. Et une valuation qui rend ${f'}_1$ vraie nous amène à tester la valeur de ${f'}_2$, qui est celle que prendra φ'. Donc φ' est fausse si et seulement si ${f'}_1$ est vraie et ${f'}_2$ fausse. D'où l'équivalence de φ' et φ.
- Si φ s'écrit $f_1 \Leftrightarrow f_2$ et qu'on suppose que f_1 et f_2 sont toutes les deux équivalentes à des formules INF ${f'}_1$ et ${f'}_2$, on procède de la même manière qu'avant en construisant une formule φ' à partir de ${f'}_1$: on remplace les feuilles 1 de ${f'}_1$ par la formule ${f'}_2$, et on remplace les feuilles 0 de ${f'}_1$ par la formule ${f'}_2$ en prenant soin d'échanger les valeurs 1 et 0 sur les feuilles de cette dernière. On peut ainsi vérifier que φ' est équivalente à φ.

On en conclut que toute formule est équivalente à une formule INF.

Question 3 On peut procéder par récurrence sur n. Soit donc la propriété $P(n)$: "pour un ordre sur les variables $x_1 < ... < x_n$ donné, pour toute fonction booléenne $f : \mathbb{B}^n \to \mathbb{B}$, il existe un unique ROBDD u tel que $f^u = f$."

- Montrons $P(1)$:
  Si f ne prend qu'une variable x, alors deux cas se présentent :

  – si $f(1) = f(0)$ (donc si f est constante), alors le BDD réduit au noeud $f(0)$ convient pour décrire f. Ce BDD est ordonné (notons que tout BDD où on ne fait des tests que sur au plus une variable est ordonné) et réduit, car il respecte de manière évidente les principes d'unicité et d'utilité. C'est donc un ROBDD, donc il en existe au moins un. Et celui-ci est l'unique ROBDD qui permet de représenter f pour cet ordre. En effet, tout autre BDD convenant doit soit être composé d'un unique noeud $f(0)$ (on retrouve donc le ROBDD précédemment mentionné), soit avoir une autre racine, qui est forcément x, puisque c'est la seule variable à tester. Mais comme tous les tests aboutissent à une unique valeur, le principe d'unicité permet de réduire ce diagramme au ROBDD précédemment mentionné. On a donc l'unicité du ROBDD correspondant.

  – $si f(1) \neq f(0)$ (donc si f peut prendre deux valeurs différentes), il suffit de construire le BDD correspondant à la formule INF $x \to f(1), f(0)$. Il est ordonné, et il est réduit, car il respecte les principe d'unicité et d'utilité. L'unicité de ce ROBDD pour la propriété $P(1)$ découle du fait qu'un seul test est nécessaire. Un ROBDD qui convient doit avoir x pour racine ; mais aucun autre noeud ne peut être étiqueté par x, car un tel noeud serait accessible depuis la racine x, et x serait donc testé plusieurs fois sur certains chemins. On doit donc obligatoirement mettre des valeurs constantes comme fils fort et faible de la racine, et il n'y a qu'une possibilité pour la fonction f donnée. On a donc l'existence et l'unicité du ROBDD correspondant.

- Supposons $P(n)$ pour n donné, et montrons $P(n+1)$: On se donne $n+1$ variables x_1 à x_{n+1} et l'ordre $x_1 < ... < x_{n+1}$. Soit f une fonction booléenne à $n+1$ variables, notées b_1 à b_{n+1}. On considère les fonctions booléennes à n variables f_0 et f_1 telles que $f_0(b_2, ..., b_{n+1}) = f(0, b_2, ..., b_{n+1})$ et $f_1(b_2, ..., b_{n+1}) = f(1, b_2, ..., b_{n+1})$.

  f_0 et f_1 sont des fonctions booléennes à n variables, et les variables x_2 à x_{n+1} suivent l'ordre $x_2 < ... <_x n+1$. Donc d'après $P(n)$, il existe un unique ROBDD u_0 tel que $f^{u_0} = f_0$, et de même il existe un unique ROBDD u_1 tel que $f^{u_1} = f$. On peut donc construire un BDD pour f en prenant x_1 pour racine, et en lui donnant pour fils faible la racine de u_0 et pour fils fort la racine de u_1. On peut vérifier avec les définitions de f_0 et f_1 que le BDD u ainsi construit est tel que $f^u = f$. De plus, l'ordre donné sur les variables x_1 à $_{n+1}$ est respecté dans u, car il a pour racine x_1 et que chacun de ses fils est ordonné suivant $x_2 < ... < x_{n+1}$. On a donc un *OBDD*, qu'on peut réduire en fusionnant tous les noeuds qui auraient la même étiquette, le même fils fort et le même fils faible, et en supprimant tous les test inutiles (ceux-ci sont déjà a priori supprimés dans les ROBDD u_0 et u_1 ; il faudrait donc seulement supprimer la racine x_1 si son test est inutile). On a donc construit un ROBDD.

  L'unicité de celui-ci est imposée par l'ordre qu'on s'est donné. Si le test de x_1 est inutile, on est ramené au cas de $P(n)$, qui est notre hypothèse, et qui assure l'existence et l'unicité du ROBDD construit. Sinon, un tel *ROBDD* doit forcément avoir x_1 pour racine ; et comme fils fort et fils faible, il doit avoir deux autres ROBDD associés respectivement à f_1 et f_0. Mais ceux-ci sont uniques, d'après la propriété $P(n)$. Il ne peut donc pas en exister d'autre non plus pour f. On a donc démontré $P(n+1)$.

On en conclut que pour tout n, pour un ordre sur les variables $x_1 < ... < x_n$ donné, pour toute fonction booléenne $f : \mathbb{B}^n \rightarrow \mathbb{B}$, il existe un unique ROBDD u tel que $f^u = f$.

3 Implémentation de ROBDD

Nous avons réussi à implémenter l'ensemble des fonctions demandées, en particulier l'extension des n dames ; les résultats que nous obtenons semblent cohérents.

Pour le problème des n dames, nous avons créé la fonction `dames : int -> prop formula`, qui génère la formule modélisant le problème pour une taille donnée. L'idée intuitive pour modéliser le problème est qu'il doit y avoir exactement 1 dame présente dans chaque ligne et chaque colonne de l'échiquier, et qu'il doit y avoir au plus 1 dame présente dans chaque diagonale.

Notre fonction traduit donc cette idée en formule logique. On commence donc par associer à chaque case une variable logique. Par exemple, nous $n = 4$:

x_0	x_1	x_2	x_3
x_4	x_5	x_6	x_7
x_8	x_9	x_{10}	x_{11}
x_{12}	x_{13}	x_{14}	x_{15}

Puis, notre fonction va générer les formules correspondantes. Par exemple, pour la première ligne, la formule générée est :

$$(x_0 \wedge \neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_0 \wedge x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_0 \wedge \neg x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_0 \wedge \neg x_1 \wedge \neg x_2 \wedge x_3)$$

On réitère ce procédé pour toutes les lignes, colonnes et diagonales, et on construit une formule globale obtenue par la conjonction de toutes les formules ainsi obtenues. Cette formule est ensuite traduite en ROBDD par la fonction `build`, et une solution est trouvée par la fonction `anysat`.

Dans un premier temps, nous avons implémenté les fonctions sans utiliser la programmation dynamique. Les résultats étaients corrects, mais les performances étaient assez faibles (nous n'avons pu résoudre le problème des n dames que pour $n \leq 6$). Ensuite, nous avons utilisé la programmation dynamique avec des tables de hachages, dans le but d'éviter de faire plusieurs fois les mêmes calculs. Nous avons pu observer une très nette amélioration des performances, et avons pu résoudre le problème pour $n = 12$ en un temps correct, comme le montre la Figure 1.

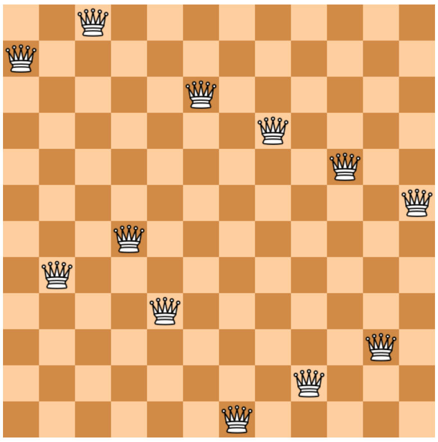

Figure 1: Une solution au problème des 12 dames