Binary Decision Diagrams

Attention, cet énoncé contient deux sujets en un.
- **Si vous êtes informaticien**, vous devez faire les questions 1, 2 et 3, et la programmation (Section 4). La section 4.6 est facultative.
- **Si vous êtes mathématicien**, vous devez faire les questions 1, 2 et 3, et les questions 4, 5, 6 (Section 3). Les questions 7 à 10 de la section 3.2 sont facultatives.

Exception : si vous êtes mathématicien et que vous êtes néanmoins à l'aise avec OCaml, vous pouvez choisir de faire le sujet informatique. Dans ce cas, dans le cadre de ce sujet, vous vous reclassez en informaticien.

1 Introduction

L'étude de la logique a non seulement un intérêt théorique certain (formaliser et comprendre le fonctionnement de la pensée humaine, donner des bases solides aux mathématiques), mais aussi de nombreuses applications pratiques (assistants de preuves, vérification de programmes...). Prenons par exemple le problème SAT, qui consiste à décider, étant donnée une formule de la logique propositionnelle, s'il existe une valuation de ses variables qui satisfait la formule. Ce problème est NP-Complet (Théorème de Cook), mais des algorithmes efficaces en pratique existent (on les appelle SAT solveurs). De nombreux problèmes en informatique, notamment en conception assistée par ordinateur de systèmes électroniques ou en vérification formelle, sont efficacement résolus en les exprimant en logique propositionnelle et en utilisant un SAT solveur. On évite ainsi d'avoir à concevoir des algorithmes ad-hoc pour résoudre chaque nouveau problème : il suffit d'implémenter la traduction vers SAT et d'utiliser un solveur existant.

Un des problèmes pour manipuler des formules est de choisir comment les représenter. Si les formules sont données sous forme normale disjonctive (DNF), le problème SAT devient polynomial, mais décider la validité d'une formule DNF est NP-difficile. A l'inverse, si la formule est sous forme normale conjonctive (CNF), la validité est polynomiale mais la satisfiabilité reste NP-difficile. Dans ce projet nous allons étudier une forme normale pour laquelle existent des algorithmes efficaces à la fois pour la satisfiabilité et la validité.

2 BDD et forme normale if-then-else

2.1 BDD

Une fonction booléenne $f : \mathbb{B}^n \to \mathbb{B}$ représente un ensemble de formules sémantiquement équivalentes sur les variables $\{x_1, \ldots, x_n\}$.

Une représentation possible d'une fonction booléenne est sous forme d'*arbre de décision*. La Figure 1 représente un arbre de décision pour la fonction booléenne définie par la formule $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$. Un arbre de décision pour un ensemble de variables $\{x_1, \ldots, x_n\}$ est un arbre binaire où chaque feuille est associée à une constante (0 ou 1), et chaque nœud interne v est associé à une variable $var(v)$. Prendre le fils *faible* (en pointillés) d'un nœud associé à la variable x dénote une valuation qui met x à 0. A l'inverse, prendre le *fort* (en plein), dénote la mise à 1 de la variable x.

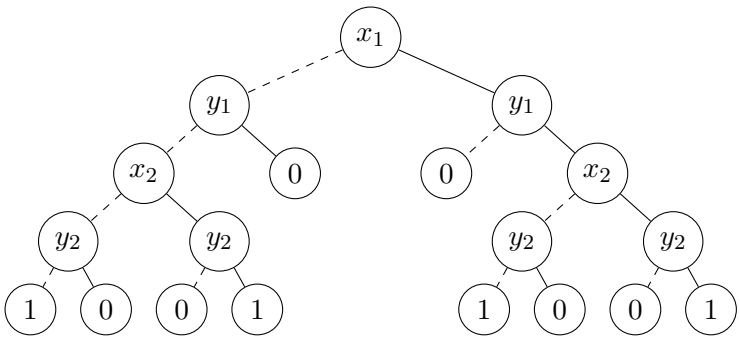


FIGURE 1 – Un arbre de décision pour $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$

La constante contenue par une feuille indique le résultat de la fonction booléenne pour les valuations définies par le chemin reliant la racine à cette feuille.

Cette représentation peut-être grandement condensée, en évitant de représenter plusieurs fois des sous-arbres égaux. Par exemple sur la Figure 1, les deux sous-arbres associés à x_2 sont identiques. On pourrait donc n'en représenter qu'un, et faire pointer les deux y_1 vers le même nœud. Si, de la même manière, on ne conserve qu'une feuille 0 et une feuille 1, on obtient un *diagramme de décision binaire (BDD)*. Ce n'est plus un arbre mais un graphe orienté acyclique (DAG), représenté en Figure 2. On voit qu'on économise 11 nœuds par rapport à l'arbre binaire de la Figure 1.

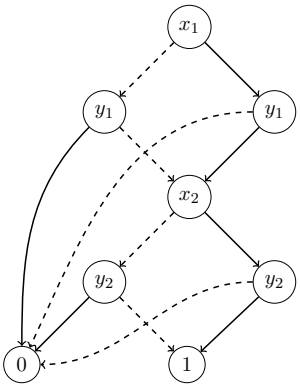


FIGURE 2 – Un BDD pour $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$

Définition 1 Un Diagramme de Décision Binaire (BDD) est un DAG (graphe orienté acyclique) avec une racine u et

- Un ou deux nœuds *terminaux*, de degré sortant nul, étiquetés 0 ou 1, et
- Un ensemble de nœuds internes de degré sortant deux, étiquetés par un nom de variable.

Pour un nœud interne v, le fils faible est donné par $low(v)$, le fils fort par $high(v)$, et la variable testée par $var(v)$.

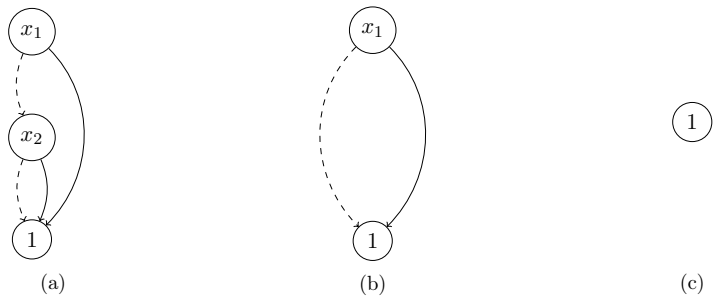


FIGURE 3 – Trois BDDs réalisant la fonction booléenne constante 1. (a) contient deux tests inutiles, dans (b) le test sur x_2 a été éliminé, et (c) ne contient plus de test inutile.

Un autre type d'optimisation peut-être effectué, comme illustré dans la Figure 3 : lorsqu'un nœud v a les mêmes fils faible et fort, cela signifie que le test sur $var(v)$ est inutile, et v peut être supprimé, en faisant pointer ses flèches entrantes vers son fils.

Définition 2 Un BDD $\mathcal{B}$ est *ordonné* (OBDD) s'il existe un ordre linéaire $x_1 < x_2 < \ldots < x_n$ sur les variables apparaissant dans $\mathcal{B}$ tel que, sur tous les chemins partant de la racine, les variables rencontrées respectent cet ordre.

Observer que les exemples des Figures 1 et 2 sont des OBDD. L'ordre dans lequel les tests sont effectués est le même dans tous les chemins : $x_1 < y_1 < x_2 < y_2$, ce n'est pas nécessairement le cas pour un BDD quelconque.

Noter qu'on ne rencontre pas forcément toutes les variables sur tous les chemins.

Définition 3 Un (O)BDD est *réduit* (R(O)BDD) s'il respecte les deux conditions suivantes :

Unicité : $var(u) = var(v)$, $low(u) = low(v)$ and $high(u) = high(v)$ implies $u = v$.

En d'autres mots, il n'y a pas deux nœuds qui ont la même étiquette, le même fils faible et le même fils fort.

Utilité : Aucun test n'est inutile : pour tout nœud interne u, $low(u) \neq high(u)$.

Dans la suite, on pourra assimiler un BDD avec sa racine.

Une propriété très intéressante (que vous allez prouver), est que pour un ordre linéaire des variables donné, à chaque fonction booléenne correspond un unique ROBDD.

2.2 Forme normale if-then-else

Pour établir cette propriété, on introduit maintenant une forme normale pour les formules de la logique propositionnelle qui correspond à la représentation sous forme de BDD.

Soit $\varphi \to \psi_1, \psi_0$ le connecteur ternaire *if-then-else* défini par

$$\varphi \to \psi_1, \psi_0 := (\varphi \wedge \psi_1) \vee (\neg\varphi \wedge \psi_0).$$

$\varphi \to \psi_1, \psi_0$ est vraie si φ et ψ_1 sont vraies, ou si φ est fausse et ψ_0 est vraie. φ est appelée la *formule test*.

Définition 4 Soit φ une formule et x une variable. L'expansion de Shannon de φ sur x est la formule

$$\varphi\uparrow^x = x \to \varphi[1/x], \varphi[0/x]$$

où $\varphi[b/x]$ est la formule φ où toutes les occurrences de x sont remplacées par b.

Question 1 Montrer que pour toute formule φ et toute variable x, $\varphi \equiv \varphi\uparrow^x$.

Définition 5 Une formule propositionnelle est sous *Forme Normale If-then-else (INF)* si elle est formée à partir de l'opérateur if-then-else, des constantes 0 et 1, et que chaque formule test est une variable (non niée).

Question 2 Démontrer que toute formule est équivalente à une formule INF.

Par exemple, la formule $\varphi = (x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$ est équivalente à la formule INF φ' :

$$\begin{array}{lll} \varphi' & = & x_1 \to \psi_1, \psi_0 \\ \psi_0 & = & y_1 \to 0, \psi_{00} \\ \psi_1 & = & y_1 \to \psi_{00}, 0 \end{array} \qquad \begin{array}{lll} \psi_{00} & = & x_2 \to \psi_{001}, \psi_{000} \\ \psi_{000} & = & y_2 \to 0, 1 \\ \psi_{001} & = & y_2 \to 1, 0 \end{array}$$

Il est aisé de voir la correspondance entre φ' et le ROBDD de la Figure 2.

Pour définir la fonction booléenne associée à un BDD, nous commençons par définir la formule associée : un nœud terminal devient une constante booléenne, et un nœud interne étiqueté x devient une expression if-then-else où la condition est x et les deux sous formules sont obtenues à partir des fils fort et faible :

Définition 6 Soit un BDD u. La formule associée φ^u est définie par :
- si $var(u) = 0$, $\varphi^u = 0$
- si $var(u) = 1$, $\varphi^u = 1$
- si $var(u) = x$, $\varphi^u = x \to \varphi^{high(u)}, \varphi^{low(u)}$

Définition 7 Soit un ROBDD u et $x_1 < x_2 < \ldots < x_n$ son ordre sur les variables. La fonction booléenne f^u associée à u est définie par :

$$\begin{array}{rccl} f^u : & \mathbb{B}^n & \to \mathbb{B} & \\ & b_1, \ldots, b_n & \mapsto \nu(\varphi^u) & \text{où } \nu(x_i) = b_i \end{array}$$

Proposition 1 Pour un ordre $x_1 < \ldots < x_n$ sur les variables donné, pour toute fonction $f : \mathbb{B}^n \to \mathbb{B}$, il existe un unique ROBDD u tel que $f^u = f$.

Question 3 Prouver la Proposition 1.

3 Influence de l'ordre des variables sur la taille d'un ROBDD

Cette section est à faire uniquement par les étudiants du cursus mathématiques.

La Proposition 1 montre que pour un ordre des variables fixé, la représentation sous forme de ROBDD d'une fonction booléenne est unique. Nous allons maintenant nous intéresser à l'influence de l'ordre des variables sur cette représentation, et en particulier sur sa taille.

3.1 Un exemple extrémal

Définition 8 La *taille* d'un BDD est son nombre de nœuds internes (feuilles non comprises).

Pour commencer, on va préciser un peu les propriétés de l'unique ROBDD correspondant à une fonction booléenne f (cf. Proposition 1), en termes de son nombre de nœuds internes.

Définition 9 Soit $f : \mathbb{B}^n \to \mathbb{B}$ une fonction booléenne. Pour $k < n$ et $\vec{x} = (x_0, \ldots, x_{k-1}) \in \mathbb{B}^k$ on définit la restriction $f_{\vec{x}}$ de f sur $\vec{x}$ comme :

$$f_{\vec{x}} : \mathbb{B}^{n-k} \to \mathbb{B}$$
$$(x_k, \ldots, x_{n-1}) \mapsto f(x_0, \ldots, x_{k-1}, x_k, \ldots, x_{n-1})$$

Définition 10 Soit $f : \mathbb{B}^n \to \mathbb{B}$. On dit que f dépend essentiellement de son premier paramètre, qu'on note $\mathrm{dep}(f)$, ssi $f_{(0)}$ et $f_{(1)}$ sont distinctes.

Définition 11 Soit $f : \mathbb{B}^n \to \mathbb{B}$. Pour $k < n$, on définit l'ensemble $S_k(f)$ des k-restrictions de f comme :

$$S_k(f) = \{f_{\vec{x}} : \vec{x} \in \mathbb{B}^k \text{ et } \mathrm{dep}(f_{\vec{x}})\}$$

Dans la définition ci-dessus, la condition $\mathrm{dep}(f_{\vec{x}})$ peut paraître peu naturelle : on ne considère que les restrictions qui dépendent essentiellement de leur premier paramètre. La raison est que les k-restrictions de f vont correspondre exactement aux nœuds de la ROBDD de f indexés par la k-ième variable. Dans ce contexte la condition $\mathrm{dep}(f_{\vec{x}})$ élimine les nœuds inutiles (dans le sens de la notion d'utilité de la section précédente).

Question 4 Soit $f : \mathbb{B}^n \to \mathbb{B}$, et u comme dans la Proposition 1. Démontrer que le nombre de nœuds de u indexés par la k-ième variable est exactement $|S_k(f)|$.

On va maintenant exhiber un exemple extrême de dépendance de la taille d'un ROBDD en l'ordre des variables. Pour ce faire on définit la famille de formules $G(n)$ sur $2n$ variables $(x_0, \ldots, x_{2n-1})$ comme suit :

$$G(n) = (x_0 \wedge x_1) \vee (x_2 \wedge x_3) \vee \cdots \vee (x_{2n-2} \wedge x_{2n-1})$$

Soit $g_1(n)$ le ROBDD de $G(n)$ pour l'ordre des variables $(x_0, \ldots, x_{2n-1})$.
Soit $g_2(n)$ le ROBDD de $G(n)$ pour l'ordre des variables $(x_0, x_2, \ldots, x_{2n-2}, x_1, x_3, \ldots, x_{2n-1})$.

Question 5 Montrer que $g_1(n)$ contient exactement $2n$ nœuds internes.

Question 6 Montrer que $g_2(n)$ contient exactement 2^{n-1} nœuds indexés par x_{2n-2}.
(Facultatif) montrer que $g_2(n)$ contient 2^{n+1} nœuds au total.

3.2 Comportement asymptotique

Cette section est facultative. La faire peut rapporter des points bonus. Vous pouvez admettre des questions.

Ainsi, pour les formules $G(n)$, la taille des ROBDD correspondants peut croître linéairement ou exponentiellement en le nombre de variables, suivant le choix de l'ordre des variables. Cependant, pour $G(n)$, il existe tout de même un ordre de variables favorable, tel que la taille du ROBDD est faible (linéaire). On peut se demander si c'est toujours le cas. Nous allons voir que non.

Dans la suite, on s'intéresse à la taille du plus petit ROBDD qui représente une fonction donnée, en jouant sur l'ordre des variables. Soit $A(n)$ le nombre de fonctions $\mathbb{B}^n \to \mathbb{B}$ telles que le plus petit ROBDD représentant la fonction soit de taille au plus $2^{n-1}/n$. Soit $B(n) = 2^{2^n}$ le nombre total de fonctions $\mathbb{B}^n \to \mathbb{B}$. On va montrer, étape par étape, la proposition suivante :

Proposition 2

$$\lim_{n\to\infty} \left(\frac{A(n)}{B(n)}\right) = 0$$

Cette proposition implique en particulier que le nombre de fonctions qui admettent un ROBDD de taille polynomiale tend vers zéro quand le nombre de variables augmente.

La proposition découle d'un argument de comptage : même en faisant varier l'ordre des variables, il y a peu de ROBDD de taille $< 2^{n-1}/n$ par rapport au nombre de fonctions $\mathbb{B}^n \to \mathbb{B}$. Pour le montrer, il faut donc majorer le nombre de ROBDD de taille $< 2^{n-1}/n$. On note $K = \lfloor 2^{n-1}/n \rfloor$.

On considère donc un ROBDD r de taille K. Tout d'abord, on fixe un ordre des variables (parmi $n!$ possibilités). On énumère les nœuds de r par $u_0, \ldots, u_{K-1}$, en choisissant un ordre cohérent avec celui des variables, i.e. pour $i < j < K$, les variables formant les labels des nœuds u_i et u_j sont croissantes (au sens large) pour l'ordre des variables. Les feuilles portent les numéros les plus élevés.

On note $\binom{a}{b}$ le coefficient binomial correspondant à choisir b éléments parmi a (aussi noté C_a^b).

Question 7 Montrer que le nombre de possibilités pour les labels de $u_0, \ldots, u_{K-1}$ est plus petit que $\binom{n+K}{n}$. *Indice : comme les labels (= variables) apparaissent dans l'ordre croissant, fixer les labels revient à choisir les numéros des nœuds où chaque label apparaît pour la première fois (s'il apparaît).*

Question 8 Montrer que le nombre de possibilités pour les arêtes de r est plus petit que $((K+1)!)^2$.

Question 9 Déduire des questions précédentes que le nombre de fonctions $\mathbb{B}^n \to \mathbb{B}$ dont le ROBDD de plus petite taille a moins de K nœuds est borné supérieurement par :

$$n!\binom{n+K}{n}((K+1)!)^2 = (K+1)(K+1)!(K+n)!$$

Question 10 Finalement, prouver la Proposition 2.

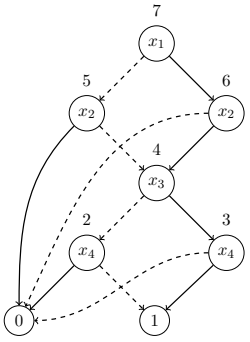
4 Implémentation de ROBDDs

Cette section est à faire uniquement par les étudiants du cursus informatique.

On s'intéresse maintenant à la construction d'un ROBDD à partir d'une formule de la logique propositionnelle. La preuve par induction de la Question 2 donne une procédure pour construire un OBDD à partir d'une formule. Une solution serait donc de construire d'abord un OBDD, puis de le réduire. Une solution plus intéressante est de réduire l'OBDD au cours de sa construction. C'est ce que nous allons faire.

4.1 Implémenter les BDDs

Afin de décrire le principe des fonctions nécessaires à cette opération, nous choisissons une représentation explicite des ROBDDs. Chaque nœud est représenté par un entier, 0 et 1 étant réservés pour les nœuds terminaux. Les variables $x_1 < x_2 < \ldots < x_n$ sont aussi représentées par des entiers : chaque variable x_i est représentée par son indice i dans l'ordre choisi. Le ROBDD est stocké dans une table $T : u \mapsto (i, l, h)$ qui associe à (l'indice d') un nœud u ses trois attributs $var(u) = i, low(u) = l$ et $high(u) = h$. La Figure 4 montre la représentation du ROBDD de la Figure 2, avec les variables renommées en $x_1 < x_2 < x_3 < x_4$.



$T : u \mapsto (i, l, h)$

u	var	low	$high$
0	$+\infty$		
1	$+\infty$		
2	4	1	0
3	4	0	1
4	3	2	3
5	2	4	0
6	2	0	4
7	1	5	6

FIGURE 4 – Représentation d'un ROBDD avec l'ordre $x_1 < x_2 < x_3 < x_4$. Les nombres à côté des nœuds sont leurs noms, ou id, utilisés dans la représentation. 0 et 1 sont réservés pour les nœuds terminaux. Les nombres dans la colonne *var* sont les indices des variables dans l'ordonnancement. Aux noeuds terminaux 0 et 1, qui ne correspondent à aucune variable, on associe MAXINT (cela se révèle pratique pour certaines fonctions que vous aurez à implémenter).

La librairie fournie, dont la signature se trouve dans le fichier `doc.mli`, propose une implémentation de ces tableaux, utilisant des tables de hachage.

4.2 Make

La fonction `make` prend un indice de variable i, un fils faible l et un fils fort h, et renvoie un nœud pour (i, l, h).

Dans le but d'éviter les doublons lors de la création d'un BDD, il est nécessaire de pouvoir dire, étant donné un triplet (i, l, h), s'il existe déjà un noeud u tel que $var(u) = i$, $low(u) = l$ et

$high(u) = h$. Cela peut se faire à l'aide d'un tableau H représentant l'"inverse" de T : $H(i, l, h) = u$ ssi $T(u) = (i, l, h)$. C'est ce qui est fait dans la librairie fournie.

Vous écrirez une fonction `make : tableT -> tableH -> variable -> id -> id -> id` qui permet de créer de nouveaux noeuds tout en assurant les conditions d'unicité et d'utilité introduites à la Définition 3.

4.3 Apply

Pour toutes les opérations booléennes sur les BDDs, le principe est le même. Il est basé sur l'expansion de Shannon et les trois résultats suivants, qu'on admettra.

Proposition 3 Pour toute variable x et formules ψ_0, ψ_1 :

$$\neg(x \to \psi_1, \psi_0) \equiv x \to \neg\psi_1, \neg\psi_0$$

Soit l'ensemble de connecteurs *binaires* $OP = \{\wedge, \vee, \to, \leftrightarrow\}$ (ne pas confondre le connecteur ternaire if-then-else avec l'implication).

Proposition 4 Pour tout $\circ \in OP$, pour toute variable x et formules ψ_0, ψ_1, ψ' :

$$(x \to \psi_1, \psi_0) \circ \psi' \equiv x \to (\psi_1 \circ \psi'), (\psi_0 \circ \psi')$$

Proposition 5 Pour tout $\circ \in OP$, pour toute variable x et formules $\psi_0, \psi'_0, \psi_1, \psi'_1$:

$$(x \to \psi_1, \psi_0) \circ (x \to \psi'_1, \psi'_0) \equiv x \to (\psi_1 \circ \psi'_1), (\psi_0 \circ \psi'_0)$$

Vous écrirez une fonction `apply_neg : tableT -> tableH -> id -> id` qui calcule la négation d'un bdd.

Vous écrirez aussi une fonction `apply : tableT -> tableH -> op -> id -> id -> id` qui prend un opérateur booléen binaire, deux bdd et construit le bdd résultat. Pour ces deux fonctions, il est recommandé d'utiliser la programmation dynamique pour éviter de faire plusieurs fois les mêmes calculs.

4.4 Build

Vous écrirez une fonction `build : tableT -> tableH -> prop formula -> id`, qui prend une table T, une table inverse H, une formule propositionnelle sur les variables $x_1 < \ldots < x_n$, construit dans T et H le ROBDD correspondant et renvoie l'id de la racine. Note : utilisez directement les fonctions précédentes, plutôt que l'expansion de Shannon.

4.5 Sat, Valid, Anysat

La fonction `sat` (resp. `valid`) prend un ROBDD et répond vrai si la fonction booléenne qu'il représente est satisfiable (resp. valide), et faux sinon. Noter que, par la Propritété 1, si un ROBDD représente la fonction constante 0 ou 1, alors il est nécessairement réduit à une feuille.

La fonction `anysat` prend un BDD et renvoie, s'il en existe une, une valuation qui satisfait la fonction booléenne représentée.

Vous écrirez la fonction `sat : id -> bool`, la fonction `valid : id -> bool`, et la fonction `anysat : tableT -> id -> (variable * bool) list`.

4.6 Application : les n dames

Cette section est facultative. La faire peut rapporter des points bonus.

Le problème classique des n dames est de trouver comment placer n dames sur un échiquer de taille $n \times n$, sans qu'aucune dame n'en menace une autre directement.

Le but est de résoudre ce problème en utilisant les ROBDDs. Vous écrirez une fonction `dames : int -> int` qui prend un entier n, et qui renvoie une solution du problème.

4.7 Sources

Téléchargez l'archive contenant les sources et librairies `OCaml` sur `http://people.irisa.fr/Brice.Minaud/tp_bdd/Sources.zip`.

Un `Makefile` vous est fourni. Les deux cibles sont :

- un exécutable `main` construit à partir du fichier `main.ml`, qui fournit un exemple d'utilisation. Vous pouvez écrire les fonctions demandées directement dans `main.ml`.
- un fichier `.ocamlinit`, chargé automatiquement par la commande `ocaml` lançant le toplevel. Toutes les librairies sont chargées et ouvertes, de manière à ce que vous n'ayiez pas à vous soucier des modules avec lesquels préfixer les fonctions des librairies. Cela peut vous faciliter les tests.

5 Consignes générales

Placez vos fichiers dans un dossier compressé qui porte vos noms (il est fortement conseillé de se mettre en binôme).

Vous enverrez par mail le **6 mai 2016, avant 23h59** (1pt de pénalité par heure de retard), à `brice.minaud@irisa.fr` un rapport structuré dans lequel vous répondrez aux questions. Une attention particulière sera portée sur la clarté de vos explications et la rigueur de vos raisonnements.

Si vous êtes informaticien, vous enverrez en même temps un dossier compressé contenant votre implémentation. Veillez à commenter votre code de manière lisible, courte et précise. Vous donnerez aussi dans le rapport un bref commentaire de votre exécutable : mode d'emploi pour compiler/exécuter, choix effectués, où vous vous êtes arrêtés le cas échéant (en particulier, si certaines fonctions sont incomplètes, signalez-le). Une attention particulière sera portée sur la clarté de vos commentaires, et sur la lisibilité de votre code.