Diagramme de décision binaire et TETRAVEX

Pierre-Yves STRUB
pierre-yves.strub@polytechnique.edu

INF441 — 2018

1 Logique Propositionnelle

Une proposition, ou formule propositionnelle, est une expression syntaxique et qui est censée parler de vérité. À la base de la syntaxe des propositions sont les variables propositionnelles, prises dans un ensemble infini dénombrable $\mathcal{V} = \{p, q, ...\}$. Les deuxièmes constituants de base du langage sont les connecteurs. Ces derniers sont des symboles permettant la construction de propositions plus élaborées. Les plus courants sont la *négation* ($\neg$), le *et logique* ($\wedge$), le *ou logique* ($\vee$), l'*implication* ($\implies$) et l'équivalence ($\iff$). On y adjoint deux symboles $\top$ et $\bot$ dénontant respectivement le vrai et le faux.

Définition 1 (Formule propositionnelle)**.** La syntaxe des *formules propositionelles* $\phi, \psi, ...$ est donnée par la syntaxe suivante :

$$\phi, \psi, ... ::= p \in \mathcal{V} \mid \top \mid \bot \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \implies \psi \mid \phi \iff \psi.$$

On se servira des parenthèses pour lever les ambiguïtés dans les formules complexes. Afin de ne pas trop alourdir les notations, on affecte des priorités et des associativités aux connecteurs. On liste ici les connecteurs du plus au moins prioritaire, avec leur associativité respective entre parenthèse :

$$\neg \text{ (gauche)} \mid \wedge \text{ (gauche)} \mid \vee \text{ (gauche)} \mid \implies \text{ (droite)} \mid \iff \text{ (non assoc.)}.$$

On se donne également une syntaxe d'entrée pour les formules où les variables propositionnelles sont représentées par des entiers :

$$\texttt{P}, \texttt{Q}, ... ::= n \in \mathbb{N} \mid \texttt{(P)} \mid \texttt{true} \mid \texttt{false} \mid \texttt{~P} \mid \texttt{P \&\& Q} \mid \texttt{P || Q} \mid \texttt{P -> Q} \mid \texttt{P <-> Q}.$$

où `true`, `false`, `~`, `&&`, `||`, `->` et `<->` représentent resp. les connecteurs $\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\implies$ et $\iff$ avec les mêmes priorités.

1.1 Interprétation

On note $\mathbb{B} = \{\top, \bot\}$ l'ensemble des booléens. Ces derniers permettent d'interpréter une formule propositionnelle comme une valeur de vérité, où $\top$ (resp. $\bot$) représente le *vrai* (resp. le *faux*). On équipe $\mathbb{B}$ d'une structure d'algèbre $(\mathbb{B}, \bot, \top, \ominus, \oplus, \odot)$, où $\ominus$, $\oplus$ et $\odot$ sont définis comme suit :

$$\begin{cases} \ominus : \quad \mathbb{B} \to \mathbb{B} \\ \qquad b \mapsto \top \text{ si } b = \bot & (\bot \text{ sinon}) \\ \\ \oplus : \quad \mathbb{B}^2 \to \mathbb{B} \\ (b_1, b_2) \mapsto \top \text{ si } \top \in \{b_1, b_2\} & (\bot \text{ sinon}) \\ \\ \odot : \quad \mathbb{B}^2 \to \mathbb{B} \\ (b_1, b_2) \mapsto \top \text{ si } b_1 = b_2 = \top & (\bot \text{ sinon}) \,. \end{cases}$$

Pour interpréter une formule propositionnelle comme une valeur de vérité, il faut au préalable savoir comment on interprète les variables propositionnelles. C'est le rôle des *valuations* :

Définition 2 (Valuation)**.** Une *valuation* ρ est une fonction de $\mathcal{V}$dans $\mathbb{B}$, c'est-à-dire une fonction qui associe à chaque variable propositionnelle une valeur de vérité. On écrit $\{p_1 \mapsto b_1, \dots, p_n \mapsto b_n\}$, ou $\{p_i \mapsto b_i\}_{1 \leq i \leq n}$, pour la valuation qui associe b_i à p_i (pour $i \in [1 \dots n]$), et qui associe $\bot$ à toutes les autres variables propositionnelles.

Nous pouvons maintenant définir l'interprétation d'une formule propositionnelle :

Définition 3 (Interprétation d'une formule propositionnelle)**.** Soit ρ une valuation et ϕ une formulle propositionnelle. L'interprétation de ϕ sous ρ, notée $[\![\phi]\!]_\rho$, est définie par induction sur la structure de la formule ϕ :

$$\begin{array}{ccc} [\![p]\!]_\rho = \rho(p) & [\![\top]\!]_\rho = \top & [\![\bot]\!]_\rho = \bot \\ [\![\phi \vee \psi]\!]_\rho = [\![\phi]\!]_\rho \oplus [\![\psi]\!]_\rho & [\![\phi \wedge \psi]\!]_\rho = [\![\phi]\!]_\rho \odot [\![\psi]\!]_\rho & [\![\neg\phi]\!]_\rho = \ominus[\![\phi]\!]_\rho \\ [\![\phi \implies \psi]\!]_\rho = [\![\neg\phi \vee \psi]\!]_\rho & [\![\phi \iff \psi]\!]_\rho = [\![(\phi \implies \psi) \wedge (\psi \implies \phi)]\!]_\rho\,. & \end{array}$$

La notation d'interprétation nous permet de définition les notions de satisfiabilité et de validité d'une formule propositionnelle :

Définition 4 (Satisfiabilité / Validité)**.** Une formule propositionnelle ϕ est *satisfiable* ssi il existe une valuation ρ telle que $[\![\phi]\!]_\rho = \top$ — ce qu'on note $\rho \vDash \phi$. Elle est *valide* si $[\![\phi]\!]_\rho = \top$ pour toute valuation ρ — ce qu'on note $\vDash \phi$.

En règle générale, il est difficile de déterminer si une formule propositionnelle est satisfiable ou non. [1] On s'intéresse dans la suite à des représentations, plus ou moins compacte, de ces formules.

2 Arbres et diagrammes de décision

De manière general une formule ϕ qui dépend de n variables propositionnelles $p_1, \dots, p_n$ peut se voir comme une fonction booléenne à n arguments $\overline{\phi} : \mathbb{B}^n \to \mathbb{B}$ définie par $\phi(b_1, \dots, b_n) \triangleq [\![\phi]\!]_{\{p_i \mapsto b_i\}_{1 \leq i \leq n}}$. Les notions de validité, satisfiabilité ou d'insatisfiabilité d'une formule peuvent être ramenées à des propriétés de la fonction booléenne associée. E.g., une formule ϕ est valide si et seulement si sa fonction booléenne $\overline{\phi}$ est la fonction constante égale à $\top$.

Un *arbre de décision* est une structure arborescente permettant la représentation du graphe d'une fonction booléenne $\overline{\phi}$. La figure 1 présente un arbre de décision pour la fonction booléenne associée à la formule $(p \implies q) \vee (r \wedge s)$.

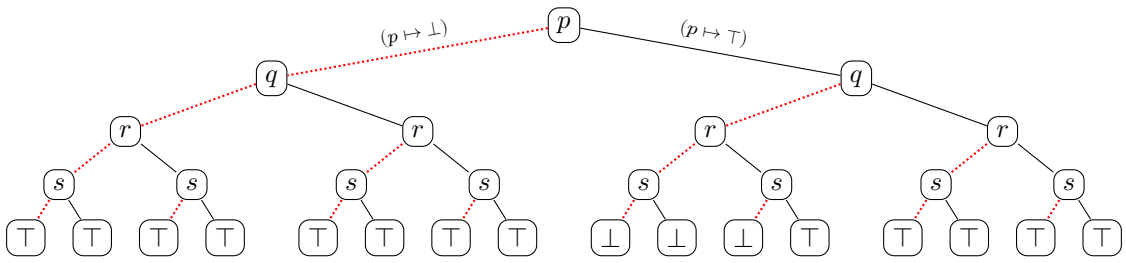

FIGURE 1 – Arbre de décision pour $(p \implies q) \vee (r \wedge s)$

Informellement, chaque noeud interne de l'arbre est étiqueté par une variable propositionnelle et représente une décision quant à l'interprétation de cette dernière, suivant que l'on suive la branche gauche ou droite. Ainsi, la racine de l'arbre de la figure 1 est étiquetée par la variable propositionnelle p, et la

1. Il s'agit d'un problème NP-complet.

branche gauche (resp. droite) représente la décision d'interpréter p par $\bot$ (resp. $\top$). Un chemin dans un arbre de décision représente alors une valuation (partielle). [2] Les feuilles de l'arbre, quant à elles, sont étiquetées par une valeur de vérité de $\mathbb{B}$. Ces valeurs donnent l'interprétation de la formule, telle que représentée par l'arbre, pour la valuation (partielle) obtenue en suivant le chemin de la racine jusqu'à la feuille — étant supposé que l'interprétation de la formule ne dépend pas des variables qui ne font pas parti de ce chemin. Par exemple, toujours dans la figure 1, le chemin :

$$\texttt{droite} \cdot \texttt{gauche} \cdot \texttt{droite} \cdot \texttt{gauche}$$

donne la valuation $\rho \triangleq \{p \mapsto \top, q \mapsto \bot, r \mapsto \top, s \mapsto \bot\}$. Ce chemin amène a une feuille étiquetée par $\bot$ qui correspond bien à l'interprétation de $(p \implies q) \vee (r \wedge s)$ sous ρ.

Cet arbre peut-être également être lu *à l'envers*. Par exemple, la présence d'une feuille étiquetée par $\top$ nous permet de dire que la formule associée est satisfiable. De plus, il nous suffit de considérer le chemin, à partir de la racine, menant à la feuille pour construire une valuation satisfaisant la formule — il est même possible d'énumérer toutes les valuations qui satisfont la formule. De même, la présence ou non d'une feuille étiquetée par $\bot$ nous permet de conclure quant à la validité de la formule.

L'arbre de la figure 1 est la forme d'arbre de décision la plus simple possible : il considère, dans l'ordre $p \preceq q \preceq r \preceq s$, toutes les possibibilités pour l'interprétation des variables propositionnelles. Un tel arbre aura toujours un nombre exponentiel (fonction du nombre de variables propositionnelle) de noeuds.

Il existe différentes heuristiques pour compresser les arbres de décision. Une première est du supprimer tous les noeuds de décision qui n'influent pas sur l'interprétation. Par exemple, dans l'arbre de la figure 1, le sous-arbre gauche du noeud étiqueté par p ne possède que des feuilles elles-même étiquetées par $\top$ — en effet, toute valuation ρ telle que $\rho(p) = \top$ rend la formule $(p \implies q) \vee (r \wedge s)$ satisfiable. En un certain sens, ce sous-arbre ne sert à rien et peut-être avantageusement remplacé par une feuille étiquetée par $\top$, comme démontré figure 2. À noter que l'ordre des variables considérées lors des décisions aura une influence sur la taille de l'arbre compressé — la figure 3 donne un autre arbre de décision compressé pour la même fonction booléenne, mais avec un ordre différent pour les variables. Quand l'ordre des décisions est décidé a priori (comme c'est le cas dans les figures 2 et 3), on dit que l'arbre de décision est ordonné.

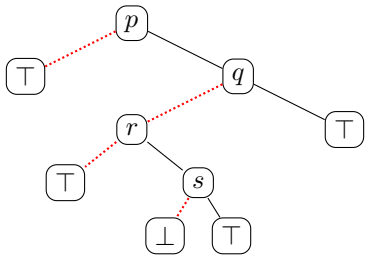

FIGURE 2 – Arbre de décision compressé pour $(p \implies q) \vee (r \wedge s)$ — $(p \preceq q \preceq r \preceq s)$

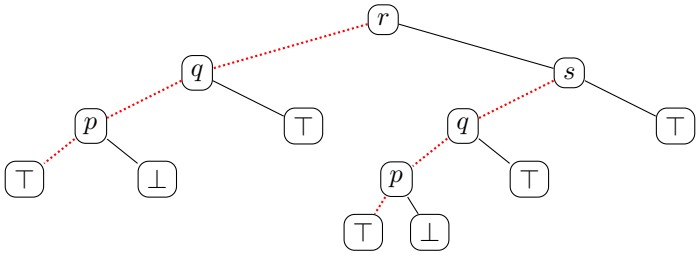

FIGURE 3 – Arbre de décision compressé pour $(p \implies q) \vee (r \wedge s)$ — $(r \preceq s \preceq q \preceq p)$

Une autre optimisation possible est l'abandon de la structure d'arbre pour celle de graphe acyclique. En effet, un arbre de décision binaire peut comporter des sous arbres identiques qu'il est alors possible de

2. On peut restreindre l'arbre à ne pas contenir de chemins avec deux fois la même étiquette ou décider que si une variable apparaît plusieurs fois dans un chemin, seule la dernière décision fait foi.

partager. La figure 4 présente l'arbre de décision de figure 3 avec du partage. Quand le partage est maximal, on obtient ce qu'on appelle un *diagramme de décision binaire* (BDD) (*ordonné* si l'ordre des variables est décidé a priori).

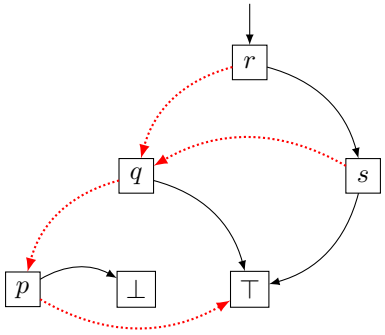

FIGURE 4 – Diagramme de décision binaire ordonné pour $(p \implies q) \vee (r \wedge s)$ — $(r \preceq s \preceq q \preceq p)$

On s'intéresse dans ce projet à l'implémentation de tel diagramme en OCAML et à leur application à la résolution d'un puzzle.

On se donne aussi une syntaxe pour représenter les BDD de manière textuelle, présenté en figure 5. Le format est orienté ligne. Chaque ligne `id v t f` représente un noeud du BDD où `id` est un identifiant unique, `v` est la variable propositionnelle qui étiquette le noeud, `t` (resp. `f`) est l'identifiant du sous-diagramme dans le cas de la décision $\texttt{v} \mapsto \top$ (resp. $\texttt{v} \mapsto \bot$). Il existe deux identifiants primitifs `@t` et `@f` pour les deux feuilles resp. étiquetées par $\top$ et $\bot$.

```
id1 v1 t1 f1
id2 v2 t2 f2
...
idn vn tn fn
```
(a) Format général

```
0 r 1  2
1 s @t 2
2 q @t 3
3 p @f @t
```
(b) Codage du BDD de la figure 4

FIGURE 5 – Format de sortie pour les BDD

3 Tetravex

Tetravex est un jeu de réflexion de type puzzle. Le plateau est une grille de taille $n \times p$, et il est fourni (au moins) $n \cdot p$ carrés dont chaque bord est annoté par un nombre. Le but est de placer $n \cdot p$ de ces carrés sur la grille de sorte deux carrés adjacents est le même nombre sur leur côté en commun. La figure 6 présente une grille de Tetravex 3×3 résolue.

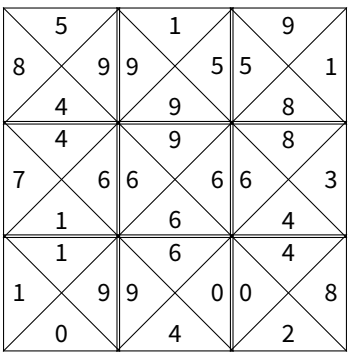

FIGURE 6 – Grille de Tetravex 3×3 résolue

Pour la suite, on va se donner un format d'entrée pour une partie de Tetravex. Ce dernier est un format texte orienté ligne tel que présenté en figure 7. La première ligne est composée de deux nombres qui sont respectivement le nombre de lignes et de colonnes de la grille. Les lignes suivantes décrivent les carrés à disposition, chaque ligne représentant un carré en enumérant les nombres sur ses bords dans l'ordre *haut, bas, gauche, droite*.

```
n p
a1 b1 c1 d1
a2 b2 c2 d2
...
an bn cn dn
```
(a) Format général

```
2 2
1 9 9 5
9 6 6 6
4 2 0 8
5 4 8 9
4 1 7 6
```
(b) Exemple

FIGURE 7 – Format d'entrée pour Tetravex

4 Travail demandé

Dans ce projet, on vous demande la réalisation d'une bibliothèque OCAML pour la création et manipulation de diagrammes de décision binaire. Cette bibliothèque contiendra :

- un type pour les formules propositionnelles,
- un type pour les diagrammes de décision binaire (BDD),
- une API de manipulation de ces BDD (opérations logiques, évaluation pour une interprétation, test de satisfiabilité, de validité),
- une fonction de traduction des formules vers les BDD.

Vos BDD devront présenter toutes les optimisations présentées dans ce document. Toutes la bibliothèque sera paramétrée par un type abstrait & ordonné pour les variables propositionnelles.

Enfin, en se reposant sur votre bibliothèque sur les BDD, vous écrirez une procédure dédiée à la résolution d'une partie de Tetravex.

On rappelle enfin l'importance d'écrire un code modulaire et de fournir des programmes de test.

5 Rendu

Il est important de suivre scrupuleusement les conditions de rendu suivantes. Elles seront utilisées par les tests qui permettent d'évaluer votre projet.

Vous devez fournir une archive qui contient l'ensemble de votre code source. Cette archive contiendra un `Makefile` à sa racine dont l'exécution compilera votre programme qui devra s'appeler `bdd`. Ce dernier devra avoir le comportement suivant :

- `./bdd dump` : prend une formule sur l'entrée standard et écrit son BDD correspondant sur la sortie standard.
- `./bdd valid` : prend une formule sur l'entrée standard et sort avec un statut égal à 0 si et seulement si la formule en entrée est valide.
- `./bdd satisfiable` : prend une formule sur l'entrée standard et écrit sur la sortie standard une valuation qui satisfait la formule. Le format de sortie est orienté ligne, chaque ligne contenant une variable propositionnelle et sa valuation (en utilisant `@t` pour $\top$ et `@f` pour $\bot$).
- `./bdd tetravex` : prend une description de Tetravex sur l'entrée standard et écrit, si elle existe, une solution sur la sortie standard. Le format de sortie est la suite de carrés ligne par ligne puis colonne par colonne — en utilisant le même format en entrée pour la description d'un carré. Si le Tetravex n'est pas résoluble, votre programme n'écrit rien et sort avec un statut différent de 0.