Exercice 1. Théorème de complétude

Le but de cet exercice est de montrer le théorème de complétude pour la logique propositionnelle dans le cas où l'ensemble des variables est dénombrable. Le théorème reste vrai dans le cas non dénombrable, mais la preuve est plus difficile.

Théorème 1 (Theorème de compacité)**.** *Soit T une théorie, alors :*

$$T \text{ admet un modèle} \Leftrightarrow \text{Pour tout } \Gamma \subset T \text{ fini, } \Gamma \text{ admet un modèle.}$$

Le preuve du théorème de compacité est l'objet de l'exercice 2.

Théorème 2 (Complétude de la logique propositionnelle)**.** *Soit T une théorie et F une formule logique.*

$$\text{Si } F \text{ est valide dans } T \text{ alors } T \vdash F.$$

Soit $V = \{v_1, v_2, v_3, \ldots\}$ l'ensemble des variables propositionnelles et D l'ensemble des distributions de vérité. Pour tout $n \in \mathbb{N}$, on note $V_n = \{v_1, v_2, \ldots, v_n\}$ et D_n l'ensemble des distributions de vérité de V_n dans $\{0,1\}$. Ainsi $D = \{0,1\}^V$ et $D_n = \{0,1\}^{V_n}$.

Pour tout $n \in \mathbb{N}$ et tout $\mu \in D_n$, on note Γ_μ l'ensemble de formules logiques :

$$\Gamma_\mu = \Big\{\neg v : v \in V_n, \mu(v) = 0\Big\} \cup \Big\{v : v \in V_n, \mu(v) = 1\Big\}$$

Par exemple, si $n = 3$, $\mu(v_1) = \mu(v_3) = 0$ et $\mu(v_2) = 1$, alors $\Gamma_\mu = \{\neg v_1, v_2, \neg v_3\}$.

1. Montrer que pour toute formule A et pour tout entier $n \in \mathbb{N}$:

   $$\text{si } \Gamma_\mu \vdash A \text{ est prouvable pour tout } \mu \in D_n, \text{ alors } \vdash A \text{ est prouvable.}$$

2. Soit $n \in \mathbb{N}$ un entier et F une formule logique dont toutes les variables propositionnelles sont dans V_n. Par récurrence sur $|F|$ (le nombre de noeuds dans l'arbre représentant F), montrer que pour tout $\mu \in D_n$:

   $$\begin{cases} \text{Si } E_\mu(F) = 0 \text{ alors } \Gamma_\mu \vdash \neg F \\ \text{Si } E_\mu(F) = 1 \text{ alors } \Gamma_\mu \vdash F. \end{cases}$$

3. En déduire que si F est une tautologie alors $\vdash F$.

Soient $(A_i)_{i\in\mathbb{N}}$ une famille de formules logiques. On note :

$$\begin{cases} Disj(A_1) = A_1 \\ \forall k \geqslant 1 : Disj(A_1, \ldots, A_{k+1}) = Disj(A_1, \ldots, A_k) \vee A_{k+1} \end{cases}$$

Attention au parenthésage : $Disj(A_1, \ldots, A_{k+1}) \neq A_1 \vee Disj(A_2, \ldots, A_{k+1})$.

4. Soit $k \in \mathbb{N}^*$. Montrer que la règle $\vee_{e,k}$ est admissible en logique classique :

   $$\frac{\Gamma \vdash Disj(A_1, \ldots, A_k) \qquad \Gamma, A_1 \vdash A \qquad \ldots \qquad \Gamma, A_k \vdash A}{\Gamma \vdash A} \vee_{e,k}$$

5. En admettant le théorème de compacité, montrer le théorème de complétude.
6. Soit T une théorie. À l'aide du théorème de complétude, montrer que $T \vdash \bot$ si et seulement si T n'admet pas de modèle.
7. Construire une théorie T telle que $T \nvdash \bot$ et pour toute formule F : $T \vdash F$ ou $T \vdash \neg F$.

Remarque. Le résultat de la dernière question est à mettre en parallèle avec le théorème d'incomplétude de Gödel en logique de premier ordre.

Exercice 2. Théorème de compacité

On reprend les notations de l'exercice 1. En particulier, $V = \{v_1, v_2, v_3, \ldots\}$ est l'ensemble des variables propositionnelles supposé dénombrable.

Dans le théorème de compacité (voir l'exercice 1), l'implication directe est immédiate : si T admet un modèle μ, alors μ est un modèle de Γ pour tout $\Gamma \subset T$ fini. Le but de l'exercice est de montrer l'implication réciproque.

Définition 3. Soit T une théorie. On dit que T est ***finiement satisfiable*** (FS) si pour tout $\Gamma \subset T$ fini, il existe un modèle de Γ.

Soit T une théorie FS, montrons que T admet un modèle.

1. Soit $v \in V$ une variable propositionnelle. Montrer que $T \cup \{v\}$ ou $T \cup \{\neg v\}$ est FS.
2. Construire une théorie T' vérifiant $T \subset T'$ qui soit FS et telle que pour toute variable $v \in V$, on ait $v \in T'$ ou $\neg v \in T'$.
3. Montrer que T admet un modèle.

Exercice 3. Admissibilité de la règle d'affaiblissement

Soit R l'ensemble des règles de la logique classique privé de la règle d'affaiblissement (aff). Montrer que (aff) est admissible dans R.

Exercice 4. Arbres de décision (concours X/ENS 1999)

Préliminaires

Soit $\mathcal{B} = \{0, 1\}$ l'ensemble des booléens. Dans cet exercice, on considère que les opérateurs $\neg$, $\wedge$, $\vee$ et $\Rightarrow$ sont des fonctions. Ainsi :

$$\neg : \mathcal{B} \to \mathcal{B} \qquad \wedge : \mathcal{B}^2 \to \mathcal{B} \qquad \vee : \mathcal{B}^2 \to \mathcal{B} \qquad \Rightarrow : \mathcal{B}^2 \to \mathcal{B}$$

On utilisera la notation infixe pour les opérateurs binaires $\wedge$, $\vee$ et $\Rightarrow$. Par exemple, on notera $b_0 \wedge b_1$ à la place de $\wedge(b_0, b_1)$. On notera également $\neg b_0$ à la place de $\neg(b_0)$. Pour rappel, voici les valeurs prises par ces fonctions :

b	$\neg b$
0	1
1	0

b_0	b_1	$b_0 \vee b_1$	$b_0 \wedge b_1$	$b_0 \Rightarrow b_1$
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	1

À partir de maintenant, on fixe un entier n supérieur ou égal à 1. On note π_i la projection qui à tout n-uplet de booléens $(b_0, b_1, \ldots, b_{n-1})$ associe le booléen b_i. Étant données deux fonctions booléennes $g_1, g_2 : \mathcal{B}^n \to \mathcal{B}$, on note $(g_1 \wedge g_2)$ la fonction booléenne de $\mathcal{B}^n \to \mathcal{B}$ définie pour tout $b \in \mathcal{B}^n$ par $(g_1 \wedge g_2)(b) = g_1(b) \wedge g_2(b)$. On utilisera également les notations $\neg g_1$, $g_1 \vee g_2$ et $g_1 \Rightarrow g_2$ qui désignent des fonctions de $\mathcal{B}^n \to \mathcal{B}$.

Étant donné un ensemble ordonné de n variables notées $b_0, b_1, \ldots, b_{n-1}$, on définit les ***arbres de décision*** sur ces variables de la façon suivante :

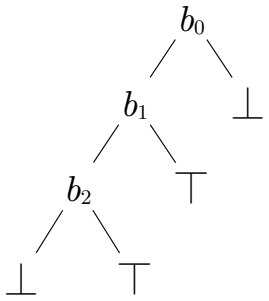
(C 1) Les symboles $\bot$ et $\top$ sont des arbres de décision.

(C 2) $test_{b_i}(v, f)$ est un arbre de décision, si v et f sont des arbres de décision tels que :

(C 2.1) Les arbres v et f sont différents.

(C 2.2) Toutes les variables de v et f sont des b_j avec $j > i$.

Dans un nœud interne $test_{b_i}(v, f)$, b_i est la ***variable testée***, v est le ***fils positif*** et f le ***fils négatif***.
Un arbre de décision est donc d'abord un arbre dont les feuilles sont $\bot$ ou $\top$ et dont les nœuds internes sont binaires et étiquetés par une variable. En outre, les deux fils d'un nœud interne sont distincts et tous les chemins de la racine aux feuilles passent par des nœuds d'étiquettes strictement croissantes. Voici par exemple, un arbre de décision :



qui peut également s'écrire $test_{b_0}(test_{b_1}(test_{b_2}(\bot, \top), \top), \bot)$. On remarquera que dans un nœud $test_{b_i}(v, f)$, le fils positif v est systématiquement le fils gauche, et le fils négatif f le fils droit.
En revanche, les deux arbres ci-dessous ne sont pas des arbres de décision. Le premier arbre viole la contrainte (C 2.2) sur l'ordre des variables, tandis que le second viole la contrainte (C 2.1) sur les fils distincts.

Étant données trois applications g, g' et g'' de $\mathcal{B}^n \to \mathcal{B}$, $test(g, g', g'')$ est une application de $B^n \to B$ définie de la manière suivante :

- Si $g(b_0, b_1, \ldots, b_{n-1})$ vaut 1, alors $test(g, g', g'')(b_0, b_1, \ldots, b_{n-1})$ vaut $g'(b_0, b_1, \ldots, b_{n-1})$.
- Sinon, $g(b_0, b_1, \ldots, b_{n-1})$ vaut 0 et $test(g, g', g'')(b_0, b_1, \ldots, b_{n-1})$ vaut $g''(b_0, b_1, \ldots, b_{n-1})$.

Un arbre de décision représente une application de $\mathcal{B}^n \to \mathcal{B}$ de la manière suivante :

- L'arbre $\bot$ représente l'application constante qui à tout n-uplet de booléens $(b_0, b_1, \ldots, b_{n-1})$ associe 0, l'arbre $\top$ représente l'application constante qui à tout n-uplet de booléens associe 1.
- L'arbre $test_{b_i}(v, f)$ représente l'application $test(\pi_i, g', g'')$, où les arbres v et f représentent respectivement les applications g' et g''.

Étude théorique

1. Donner des arbres de décision qui représentent les cinq fonctions π_0, $\neg\pi_1$, $\pi_0 \wedge \neg\pi_1$, $\neg\pi_1 \wedge \pi_0$ et $\neg(\pi_0 \Rightarrow \pi_1)$.
2. Montrer que si l'arbre a représente la fonction constante égale à 1 alors a est réduit à la feuille $\top$.

Les applications partielles $g_{i\leftarrow 0}$ et $g_{i\leftarrow 1}$ de $\mathcal{B}^n \to \mathcal{B}$ sont définies pour toute application g de $\mathcal{B}^n \to \mathcal{B}$, comme suit :

$$g_{i\leftarrow 0}(b_0, \ldots, b_i, \ldots, b_{n-1}) = g(b_0, \ldots, 0, \ldots, b_{n-1}),$$
$$g_{i\leftarrow 1}(b_0, \ldots, b_i, \ldots, b_{n-1}) = g(b_0, \ldots, 1, \ldots, b_{n-1}).$$

3. Montrer que toute fonction booléenne est représentable par un arbre de décision. Indication : on pourra utiliser les applications partielles $g_{i\leftarrow 0}$ et $g_{i\leftarrow 1}$.
4. Soit $g : \mathcal{B}^n \to \mathcal{B}$ une application.
   (a) Exprimer g et $\neg g$ à l'aide de la projection π_i, des applications partielles $g_{i\leftarrow 0}$ et $g_{i\leftarrow 1}$, et des connecteurs $\neg$, $\wedge$ et $\vee$. Justifier votre réponse.
   (b) Soient en outre g' et g'' dans $\mathcal{B}^n \to \mathcal{B}$, exprimer $test(g, g', g'')$ à l'aide de la projection π_i, des applications partielles $g_{i\leftarrow 0}$, $g_{i\leftarrow 1}$, $g'_{i\leftarrow 0}$, $g'_{i\leftarrow 1}$, $g''_{i\leftarrow 0}$ et $g''_{i\leftarrow 1}$, et de la fonction $test$. Justifier votre réponse.

Implémentation

Les arbres de décision seront représentés en machine par des objets de type `abd` (pour ***arbre binaire de décision***), dont voici les définitions :

```
type variable = int;;

type abd =
  | Bool of bool
  | Test of variable * abd * abd;;
```

5. Programmer la fonction `(abd_proj: variable -> abd)` qui prend en argument un indice de variable et renvoie un arbre de décision représentant π_i.
6. (a) Programmer la fonction `(abd_neg: abd -> abd)` qui prend en argument un arbre de décision représentant une fonction booléenne g et renvoie un arbre de décision représentant la fonction $\neg g$.
   (b) Justifier que votre fonction `abd_neg` rend un arbre de décision bien formé si son argument est un arbre de décision bien formé.
   (c) Montrer que votre fonction est correcte. On demande ici une preuve détaillée.
7. Programmer la fonction `(abd_egal: abd -> abd -> bool)` qui prend en argument deux arbres de décision et renvoie `true` si ces arbres sont structurellement égaux, et `false` autrement.
   Note. Vous ne devez pas définir la fonction `abd_egal` comme suit :

   ```
   let abd_egal a0 a1 = a0 = a1;;
   ```

   Toutes les définitions d'esprit similaire sont bien évidemment également interdites.
8. Programmer la fonction `(abd_partiel: variable -> bool -> abd -> abd)` qui prend en arguments, un indice de variable i, un booléen b, ainsi qu'une fonction booléenne g représentée par un arbre de décision ; et renvoie un arbre de décision qui représente $g_{i\leftarrow b}$.
9. Programmer la fonction `(abd_test: abd -> abd -> abd -> abd)` qui prend en arguments trois fonctions booléennes c, v et f représentées par des arbres de décision, et qui renvoie l'arbre de décision qui représente $test(c, v, f)$.
10. Donner les fonctions

    ```
    abd_et: abd -> abd -> abd
    abd_ou: abd -> abd -> abd
    abd_implique: abd -> abd -> abd
    ```

    qui réalisent les opérations $f \wedge g$, $f \vee g$ et $f \Rightarrow g$, où f et g sont des fonctions booléennes représentées en machine par des arbres de décision.