Durée : 4h. Aucun matériel électronique autorisé. Aucun document autorisé. Si une fonction ou un résultat mathématique est introduit dans l'énoncé, vous pouvez l'utiliser dans les questions suivantes, y compris si vous n'avez pas proposé de définition de cette fonction, de démonstration du résultat. Pour chaque définition de fonction en OCaml les annotations de types doivent indiquer le type de chacun des arguments ainsi que le type de retour. De plus chaque fonction auxiliaire (comprendre, non spécifiquement demandée dans l'énoncé) doit être accompagnée d'une description rapide de son comportement et du sens attaché à ses arguments. On trouvera en fin de sujet une annexe contenant des rappels de syntaxe OCaml, ainsi que des rappels sur l'utilisation du module `Hasthbl`.

Exercice 1 : Clauses de Horn

Dans cet exercice de logique $\mathscr{P}$ désigne l'ensemble des variables propositionnelles. On le suppose *fini* dans tout l'exercice.

1. Rappels, notations

Littéraux positifs, négatifs. Un littéral de la forme $p \in \mathscr{P}$ est dit *positif*, un littéral de la forme $\neg p$ avec $p \in \mathscr{P}$ est dit *négatif*.

Clauses disjonctives. Formes normales conjonctives. $\mathcal{M}(\Gamma)$ Dans tout cet exercice, les formules manipulées sont : ou bien des clauses disjonctives, ou bien des formules normales conjonctives. Les clauses disjonctives seront représentées par un ensemble de littéraux : $l_1 \vee l_2 \vee \ldots \vee l_r$ est représentée par $\{l_1, l_2, \ldots, l_r\}$. Par exemple la clause disjonctive $p \vee (\neg p) \vee (\neg q) \vee (\neg q)$ est représentée par l'ensemble $\{p, \neg p, \neg q\} = \{\neg p, p, \neg q\}$ Les formes normales conjonctives seront alors représentées par un ensemble d'ensemble de littéraux : $C_1 \wedge C_2 \wedge \ldots \wedge C_s$ est représentée par $\{C_1, C_2, \ldots, C_s\}$. Aussi, on étend naturellement la définition de $[\![\quad]\!]^\mu$ aux ensembles de littéraux (que sont les clauses disjonctives) et aux ensembles d'ensembles de littéraux (que sont les formes normales conjonctives) :
- Si $C = \{l_1, l_2, \ldots, l_r\}$ est un ensemble de $r \in \mathbb{N}$ littéraux et $\mu \in \mathbb{B}^{\mathscr{P}}$, $[\![C]\!]^\mu = [\![l_1]\!]^\mu + [\![l_2]\!]^\mu + \ldots + [\![l_r]\!]^\mu$, avec le cas particulier que $[\![\emptyset]\!]^\mu = \mathsf{F}$.
- Si $\Gamma = \{C_1, C_2, \ldots, C_s\}$ est un ensemble de $s \in \mathbb{N}$ clauses disjonctives (des ensembles de littéraux) et $\mu \in \mathbb{B}^{\mathscr{P}}$, $[\![C]\!]^\mu = [\![C_1]\!]^\mu \cdot [\![C_2]\!]^\mu \cdot \ldots \cdot [\![C_s]\!]^\mu$.

Dans la suite si Γ est une forme normale conjonctive, on note $\mathcal{M}(\Gamma)$ l'ensemble de ses modèles, à savoir les environnements propositionnels μ tels que $[\![\Gamma]\!]^\mu = \mathsf{V}$.

Notation $\vDash$. Dans tout cet exercice, on s'efforcera de n'utiliser la notation $\Gamma \vDash C$ que lorsque Γ est une forme normale conjonctive $\{C_1, C_2, \ldots, C_s\}$ et C une clause disjonctive. En pareil cas, l'énoncé $\{C_1, C_2, \ldots, C_s\} \vDash C$ signifie donc qu'un modèle de chaque C_i est un modèle de C, à savoir :

$$\{C_1, C_2, \ldots, C_s\} \vDash C \overset{\text{déf}}{\Leftrightarrow} \forall \mu \in \mathbb{B}^{\mathscr{P}}, (\forall i \in [\![1, s]\!], [\![C_i]\!]^\mu = \mathsf{V}) \Rightarrow [\![C]\!]^\mu = \mathsf{V}$$

2. Clauses de Horn

Clauses de Horn. On appelle *clause de Horn* une clause disjonctive dans laquelle au plus un littéral est positif, autrement dit : C, est une clause de Horn, si et seulement si $|C \cap \mathscr{P}| \leqslant 1$. Par exemple :
- $\{\neg p, \neg q\}$ est bien une clause de Horn : elle contient deux littéraux négatifs ;
- $\{\neg p, \neg q, r\}$ est bien une clause de Horn : elle contient deux littéraux négatifs et un littéral positif ;

- $\{\neg p, \neg q, r, s\}$ *n'*est *pas* une clause de Horn : elle contient deux littéraux négatifs et deux littéraux positifs.

Types de clauses de Horn. On distingue trois types de clauses de Horn :
- Les clauses de Horn qui sont dites des *faits* : ce sont les clauses contenant exactement un littéral positif et aucun littéral négatif, de telles clauses sont donc de la forme : $\{p\}$ pour $p \in \mathscr{P}$.
- Les clauses de Horn qui sont dites *strictes* : ce sont les clauses contenant exactement un littéral positif et au moins un littéral négatif, de telles clauses sont donc de la forme : $\{p, \neg q_1, \neg q_2, \ldots, \neg q_m\}$ avec $m \geqslant 1$, $p \in \mathscr{P}$ et $\forall i \in [\![1, m]\!], q_i \in \mathscr{P}$.
- Les clauses de Horn qui sont dites *négatives* : ce sont les clauses ne contenant aucun littéral positif, de telles clauses sont donc de la forme : $\{\neg q_1, \neg q_2, \ldots, \neg q_m\}$ avec $m \geqslant 0$ et $\forall i \in [\![1, m]\!], q_i \in \mathscr{P}$.

3. Clauses de Horn comme formules de la logique propositionnelle

Q. 1 Soit une clause stricte $C = \{p, \neg q_1, \neg q_2, \ldots, \neg q_m\}$ avec $m \geqslant 1$, soit la formule $G = (q_1 \wedge q_2 \wedge \ldots \wedge q_m) \rightarrow p$ de la logique propositionnelle, montrer que $G \equiv C$, autrement dit, montrer que $\forall \mu \in \mathbb{B}^{\mathscr{P}}, [\![G]\!]^\mu = [\![C]\!]^\mu$.

Solution

Soit la formule $G = (q_1 \wedge q_2 \wedge \ldots \wedge q_m) \rightarrow p$. Montrons que $G \equiv C$.

$$\begin{aligned}
\forall \mu \in \mathbb{B}^{\mathscr{P}}, [\![G]\!]^\mu = \mathsf{V} &\Leftrightarrow [\![(q_1 \wedge q_2 \wedge \ldots \wedge q_m) \rightarrow p]\!]^\mu = \mathsf{V} \\
&\Leftrightarrow \overline{[\![(q_1 \wedge q_2 \wedge \ldots \wedge q_m)]\!]^\mu} + [\![p]\!]^\mu = \mathsf{V} \\
&\Leftrightarrow \overline{[\![q_1]\!]^\mu \cdot [\![q_2]\!]^\mu \ldots [\![q_m]\!]^\mu} + [\![p]\!]^\mu = \mathsf{V} \\
&\Leftrightarrow \overline{[\![q_1]\!]^\mu} + \overline{[\![q_2]\!]^\mu} + \ldots + \overline{[\![q_m]\!]^\mu} + [\![p]\!]^\mu = \mathsf{V} \\
&\Leftrightarrow [\![\neg q_1]\!]^\mu + [\![\neg q_2]\!]^\mu + \ldots + [\![\neg q_m]\!]^\mu + [\![p]\!]^\mu = \mathsf{V} \\
&\Leftrightarrow [\![\{\neg q_1, \neg q_2, \ldots, \neg q_m, p\}]\!]^\mu = \mathsf{V} \\
&\Leftrightarrow [\![C]\!]^\mu = \mathsf{V}
\end{aligned}$$

Q. 2 Pour chacun des cas suivants, on ne demande pas de preuve, seulement une formule (à la manière de celle proposée en **Q. 1**).

$a)$ Donner une formule de la logique propositionnelle, équivalente à la clause négative $\{\neg q_1, \neg q_2, \ldots, \neg q_m\}$ avec $m \geqslant 1$, n'utilisant que des variables propositionnelles, le connecteur $\wedge$, le connecteur $\rightarrow$, la constante $\bot$.

$b)$ Donner une formule de la logique propositionnelle, de plus petite taille possible, équivalente à la clause négative $C = \emptyset$.

$c)$ Donner une formule de la logique propositionnelle, de plus petite taille possible, équivalente au fait $C = \{p\}$ pour $p \in \mathscr{P}$.

Solution

a) $\{\neg q_1, \neg q_2, \ldots, \neg q_m\} \equiv (q_1 \wedge q_2 \wedge \ldots \wedge q_m) \rightarrow \bot$

b) $\emptyset \equiv \bot$

c) $\{p\} \equiv p$

4. Un peu de structure sur $\mathbb{B}^{\mathscr{P}}$

Une relation d'ordre $\sqsubseteq$ sur $\mathbb{B}^{\mathscr{P}}$. On munit $\mathbb{B}$ de la relation d'ordre totale $\preccurlyeq$ définie par l'inégalité suivante : $\mathsf{F} \preccurlyeq \mathsf{V}$. On munit $\mathbb{B}^{\mathscr{P}}$ de la relation d'ordre produit dérivant de $\preccurlyeq$, que l'on note $\sqsubseteq$, dont la définition est rappelée ci-dessous.

$$\mu \sqsubseteq \rho \overset{\text{déf}}{\Leftrightarrow} \forall p \in \mathscr{P}, \mu(p) \preccurlyeq \rho(p)$$

Q. 3 Dans le cas où $\mathscr{P} = \{q, r\}$, et $\rho = (q \mapsto \mathsf{V}, r \mapsto \mathsf{F})$, donner un environnement propositionnel μ tel que $\mu \sqsubseteq \rho$ et $\mu \neq \rho$.

Solution

$(q \mapsto \mathsf{F}, r \mapsto \mathsf{F})$

Q. 4 On admet que $\sqsubseteq$ est bien une relation d'ordre. Est-elle totale ? Est-elle bien fondée ?

Solution

Question de cours.

Une loi interne $\sqcap$ sur $\mathbb{B}^{\mathscr{P}}$. On munit $\mathbb{B}^{\mathscr{P}}$ d'une loi interne $\sqcap : \mathbb{B}^{\mathscr{P}} \times \mathbb{B}^{\mathscr{P}} \to \mathbb{B}^{\mathscr{P}}$.

$$\forall (\mu, \rho) \in (\mathbb{B}^{\mathscr{P}})^2, \mu \sqcap \rho \overset{\text{déf}}{=} \begin{cases} \mathscr{P} & \to \mathbb{B} \\ p & \mapsto \mu(p) \cdot \rho(p) \end{cases}$$

Ainsi $\mu \sqcap \rho$ est l'environnement propositionnel associant $\mu(p) \cdot \rho(p)$ à toute variable propositionnelle p. On admet que $\sqcap$ est une loi associative et commutative de $\mathbb{B}^{\mathscr{P}}$.

Q. 5 Dans le cas où $\mathscr{P} = \{p, q, r\}$, $\mu = (p \mapsto \mathsf{V}, q \mapsto \mathsf{V}, r \mapsto \mathsf{F})$ et $\rho = (p \mapsto \mathsf{F}, q \mapsto \mathsf{V}, r \mapsto \mathsf{V})$, donner $\mu \sqcap \rho$.

Solution

$(p \mapsto \mathsf{F}, q \mapsto \mathsf{V}, r \mapsto \mathsf{F})$.

Q. 6 Montrer que la loi $\sqcap$ admet un élément neutre. Préciser lequel.

Solution

La fonction constante, égale à V, notée $\boldsymbol{V} \overset{\text{déf}}{=} \begin{cases} \mathscr{P} & \to \mathbb{B} \\ p & \mapsto \mathsf{V} \end{cases}$, est un élément neutre. En effet, pour tout $p \in \mathscr{P}$, $(\boldsymbol{V} \sqcap \rho)(p) = \boldsymbol{V}(p) \cdot \rho(p) = \mathsf{V} \cdot \rho(p) = \rho(p)$.

Dans la suite, on note $\boldsymbol{V}$ l'élément neutre de $\sqcap$.

Q. 7 Montrer que $\forall (\rho, \mu) \in (\mathbb{B}^{\mathscr{P}})^2$, $(\mu \sqcap \rho) \sqsubseteq \mu$.

Solution

Soit $p \in \mathscr{P}$ par définitions : $(\mu \sqcap \rho)(p) = \mu(p) \cdot \rho(p) \preccurlyeq \mu(p)$ ceci étant vrai pour tout $p \in \mathscr{P}$, $\mu \sqcap \rho \sqsubseteq \mu$.

Dans la suite, lorsque $\mathscr{B} \in \wp(\mathbb{B}^{\mathscr{P}})$ est un ensemble fini $\{\mu_1, \mu_2, \ldots, \mu_m\}$ avec $m \geqslant 0$ d'environnements propositionnels, on note $\bigsqcap_{\mu \in \mathscr{B}} \mu \overset{\text{déf}}{=} \mu_1 \sqcap \mu_2 \ldots \sqcap \mu_m$ si $m \geqslant 1$ et $\bigsqcap_{\mu \in \mathscr{B}} \mu \overset{\text{déf}}{=} \boldsymbol{V}$ sinon.

5. Espace des modèles des clauses de Horn

FNCH. Notation $[\![\Gamma]\!]^\mu$. Notation $\mathcal{M}(\Gamma)$. On appelle *Forme normale conjonctive de Horn*, abrégé en FNCH dans la suite, une forme normale conjonctive dont toutes les clauses sont des clauses de Horn.

Q. 8 Lorsque $\mathcal{P} = \{p, q, r\}$, dresser une table de vérité commune aux trois clauses disjonctives $\{p\}$, $\{r\}$, $\{\neg p, q, \neg r\}$. En déduire l'ensemble $\mathcal{M}(\Gamma)$ des environnements propositionnels rendant simultanément vraies ces trois clauses.

Solution

p	q	r	$\{p\}$	$\{r\}$	$\{\neg p, q, \neg r\}$	
F	F	F	F	F	V	
F	F	V	F	V	V	
F	V	F	F	F	V	
F	V	V	F	V	V	
V	F	F	V	F	V	
V	F	V	V	V	F	
V	V	F	V	F	V	
V	V	V	V	V	V	★

Finalement $\mathcal{M}(\Gamma) = (p \mapsto \mathsf{V}, q \mapsto \mathsf{V}, r \mapsto \mathsf{V})$.

Q. 9 Montrer que si $\Gamma = \{\{p\}, \{\neg p, r\}, \{\neg r, \neg q, s\}, \{q\}\}$ alors $\Gamma \vDash \{s\}$.

Solution

Soit μ tel que $[\![\Gamma]\!]^\mu = \mathsf{V}$, soit $[\![p]\!]^\mu = \mu(p) = \mathsf{V}$, $\overline{[\![p]\!]^\mu} + [\![r]\!]^\mu = \mathsf{V} = [\![r]\!]^\mu = \mu(r)$, $[\![q]\!]^\mu = \mu(q) = \mathsf{V}$ et finalement $\overline{[\![r]\!]^\mu} + \overline{[\![q]\!]^\mu} + [\![s]\!]^\mu = \mu(s) = \mathsf{V}$, alors $[\![s]\!]^\mu = \mathsf{V}$.

Q. 10 Montrer que si $C = \{p, \neg q_1, \neg q_2, \ldots, \neg q_m\}$ est telle que $C \in \Gamma$ et $\forall i \in [\![1, m]\!], \Gamma \vDash \{q_i\}$ alors $\Gamma \vDash \{p\}$.

Solution

Soit μ tel que $[\![\Gamma]\!]^\mu$, de l'énoncé : $\forall i \in [\![1, m]\!], [\![q_i]\!]^\mu = \mathsf{V}$. De plus $[\![C]\!]^\mu = \mathsf{V}$ donc $[\![p]\!]^\mu + \underbrace{\overline{[\![q_1]\!]^\mu}}_{\mathsf{F}} + \underbrace{\overline{[\![q_2]\!]^\mu}}_{\mathsf{F}} + \ldots + \underbrace{\overline{[\![q_m]\!]^\mu}}_{\mathsf{F}} = \mathsf{V}$ soit finalement $[\![p]\!]^\mu = \mathsf{V}$.

Q. 11 Soit Γ une FNCH, ne contenant pas de clause négative (autrement dit : constituée uniquement de faits et de clauses strictes), montrer que Γ est satisfiable.

Solution

Soit C une clause de Γ, elle est de la forme $\{p, \neg q_1, \neg q_2, \ldots, \neg q_m\}$ avec $m \geqslant 0$. Ainsi $[\![C]\!]^{\boldsymbol{V}} = \mathsf{V}$ et donc $[\![\Gamma]\!]^{\boldsymbol{V}} = \mathsf{V}$. Finalement Γ est satisfiable et a $\boldsymbol{V}$ comme modèle.

Q. 12 Montrer que si $C \overset{\text{déf}}{=} \{\neg q_1, \neg q_2, \ldots, \neg q_m\}$ est telle que $C \in \Gamma$ et $\forall i \in [\![1, m]\!], \Gamma \vDash \{q_i\}$ alors Γ est insatisfiable.

Solution

Supposons par l'absurde Γ satisfiable, soit μ tel que $[\![\Gamma]\!]^\mu$, de l'énoncé : $\forall i \in [\![1, m]\!], [\![q_i]\!]^\mu = \mathsf{V}$.

De plus $[\![C]\!]^\mu = \mathsf{V}$ donc $\underbrace{\overline{[\![q_1]\!]^\mu}}_{\mathsf{F}} + \underbrace{\overline{[\![q_2]\!]^\mu}}_{\mathsf{F}} + \ldots + \underbrace{\overline{[\![q_m]\!]^\mu}}_{\mathsf{F}} = \mathsf{V}$ soit finalement $\mathsf{F} = \mathsf{V}$ ce qui est absurde.

Q. 13 Donner un exemple de FNCH qui est insatisfiable.

Solution

L'ensemble $\Gamma = \{\{r\}, \{\neg r\}\}$ constitué du fait r et de la clause de Horn négative $\{\neg r\}$ n'est pas satisfiable.

Q. 14 Soient $(\mu, \rho) \in (\mathbb{B}^{\mathscr{P}})^2$ deux environnements propositionnels et Γ une FNCH, montrer que si $[\![\Gamma]\!]^\mu = \mathsf{V}$ et $[\![\Gamma]\!]^\rho = \mathsf{V}$ alors $[\![\Gamma]\!]^{\mu \sqcap \rho} = \mathsf{V}$.

Solution

Soient donc $(\mu, \rho) \in (\mathbb{B}^{\mathscr{P}})^2$ et Γ une FNCH, telle que $[\![\Gamma]\!]^\mu = \mathsf{V}$ et $[\![\Gamma]\!]^\rho = \mathsf{V}$. Soit $C \in \Gamma$, montrons que $[\![C]\!]^{\rho \sqcap \mu} = \mathsf{V}$. Soit $\{\neg q_1, \neg q_2, \ldots, \neg q_m\}$, l'ensemble des littéraux négatifs de la clause disjonctive C.

- S'il existe $i \in [\![1, m]\!]$ tel que $\rho(q_i) = \mathsf{F}$ ou $\mu(q_i) = \mathsf{F}$ (supposons sans perdre en généralité que ce soit $\rho(q_i) = \mathsf{F}$), alors $(\rho \sqcap \mu)(q_i) = \mathsf{F}$ donc $[\![\neg q_i]\!]^{\rho \sqcap \mu} = \mathsf{V}$ donc $[\![C]\!]^{\rho \sqcap \mu} = \mathsf{V}$
- Sinon si pour tout $i \in [\![1, m]\!]$ $\rho(q_i) = \mathsf{V}$ et $\mu(q_i) = \mathsf{V}$ alors C est une clause stricte (sinon $[\![C]\!]^\rho = \mathsf{F}$) de la forme $\{p, \neg q_1, \neg q_2, \ldots, \neg q_m\}$ et $\rho(p) = \mu(p) = \mathsf{V}$ (sinon, si $\rho(p) = \mathsf{F}$, $[\![C]\!]^\rho = \mathsf{F}$, de même pour μ), ainsi $(\rho \sqcap \mu)(p) = \mathsf{V}$ et $[\![C]\!]^{\rho \sqcap \mu} = \mathsf{V}$.

Q. 15 Soit Γ une FNCH satisfiable. Montrer que $\mathcal{M}(\Gamma)$ admet un plus petit élément ♣ pour la relation d'ordre $\sqsubseteq$.

Solution

Γ est satisfiable, ainsi $\mathcal{M} \neq \emptyset$, considérons alors $\mu^\star \stackrel{\text{déf}}{=} \bigsqcap_{\mu \in \mathcal{M}} \mu$. Par récurrence depuis **Q. 14**, on déduit que $\mu^\star$ est un modèle de Γ. Par **Q. 7**, on déduit que $\forall \mu \in \mathcal{M}, \mu^\star \sqsubseteq \mu$. $\mu^\star$ est donc le plus petit élément de $\mathcal{M}$.

Q. 16 Proposer une FNCH Γ, et trois environnements propositionnels μ_1, μ_2 et μ_3 tels que $\mu_1 \sqsubseteq \mu_2 \sqsubseteq \mu_3$, $[\![\Gamma]\!]^{\mu_1} = \mathsf{V}$, $[\![\Gamma]\!]^{\mu_2} = \mathsf{F}$ et $[\![\Gamma]\!]^{\mu_3} = \mathsf{V}$.

Q. 17 Soit Γ^- un ensemble de clauses de Horn négatives. Montrer que si μ est un modèle de Γ^- alors tout environnement propositionnel $\mu' \sqsubseteq \mu$ est aussi un modèle de Γ^-.

Solution

Soient de tels Γ^-, μ et μ'. Soit $C \in \Gamma^-$, C est de la forme $\{\neg q_1, \neg q_2, \ldots, \neg q_n\}$, ainsi il existe $i \in [\![1, n]\!]$ tel que $\mu(q_i) = \mathsf{F}$ (sinon $[\![C]\!]^\mu = \mathsf{F}$). Par comparaison, $\mu'(q_i) = \mathsf{F}$ et donc $[\![C]\!]^{\mu'} = \mathsf{F}$. Ceci étant vrai pour tout $C \in \Gamma^-$ on en déduit que μ' est un modèle de Γ^-.

Q. 18 Soit une formule G de la logique propositionnelle (sur le même ensemble de variables $\mathscr{P}$), existe-t-il nécessairement une FNCH Γ telle que $G \equiv \Gamma$?

♣. On rappelle, à toutes fins utiles, qu'un élément x est dit être *le plus petit élément* d'un ensemble X, muni d'une relation d'ordre $\preccurlyeq$ dès lors que $x \in X$ et x est un minorant de X, à savoir $\forall y \in X, x \preccurlyeq y$. Si un ensemble X admet un plus petit élément, celui-ci est unique.

Solution

Non, la propriété 15 est trop forte. Considérons par exemple la formule $p \vee q$ l'ensemble de ses modèles est $\{(\mathsf{F},\mathsf{V}),(\mathsf{V},\mathsf{V}),(\mathsf{V},\mathsf{F})\}$ qui n'admet pas de plus petit élément.

6. Résolution du problème SAT pour les FNCH

On a établi en question **Q. 11** que toute FNCH sans clause négative est satisfiable. De plus la question **Q. 10** nous suggère l'Algorithme 1 permettant le calcul du plus petit modèle d'une FNCH sans clause négative.

Notations C^-, C^+. Dans cet algorithme, si C est une clause de Horn, on note C^- l'ensemble des variables propositionnelles apparaissant dans un littéral négatif de C, autrement dit $C^- = \{p \in \mathscr{P} \mid \neg p \in C\}$. Par exemple si $C = \{p, \neg q, \neg r\}$, C^- désigne $\{q, r\}$, si $C = \{p\}$ alors $C^- = \emptyset$. De même, l'algorithme suivant manipule des clauses de Horn, *qui ne sont pas des clauses négatives*, ainsi toute clause de Horn manipulée par l'algorithme contient un et un seul littéral positif, on note C^+ cette unique variable propositionnelle apparaissant positivement dans C. Par exemple si $C = \{p, \neg q, \neg r\}$, C^+ désigne p.

Algorithme 1 : Calcul du plus petit modèle

Entrée : Une FNCH Γ, *sans clause négative*
Sortie : Le plus petit modèle de Γ

$Q \leftarrow \emptyset$;
Done $\leftarrow \emptyset$;
tant que $\exists C \in \Gamma \setminus \mathsf{Done}$ tel que $C^- \subseteq Q$ **faire**
    Soit une telle clause C ;
    Soit p l'unique variable propositionnelle telle que $p \in C$ ($p = C^+$) ;
    $Q \leftarrow Q \cup \{p\}$;
    Done $\leftarrow$ Done $\cup \{C\}$;
Soit l'environnement $\mu : \begin{cases} \mathscr{P} & \to \mathbb{B} \\ p & \mapsto \mathsf{V} \text{ si } p \in Q, \mathsf{F} \text{ sinon} \end{cases}$;
retourner μ

Dans les questions suivantes on démontre la terminaison et la correction de l'Algorithme 1. Aussi, dans le reste de cette section, on fixe une FNCH, Γ, sans clause négative, on note $\mu^\star$ le plus petit modèle de Γ (pour la relation $\sqsubseteq$), et finalement $Q^\star \overset{\text{déf}}{=} \{p \in \mathscr{P} \mid \mu^\star(p) = \mathsf{V}\}$ l'ensemble des variables propositionnelles p telles que $\mu^\star(p) = \mathsf{V}$. On cherche à démontrer que sur l'entrée Γ, l'algorithme 1 renvoie bien $\mu^\star$.

Q. 19 Établir que les deux propriétés suivantes sont des invariants de la boucle **tant que** de l'algorithme 1.

$$\mathscr{I}_1 : Q \subseteq Q^\star$$
$$\mathscr{I}_2 : \forall C \in \mathsf{Done}, C^+ \in Q$$

Solution

Démontrons que les propriétés $\mathscr{I}_1$ et $\mathscr{I}_2$ sont bien des invariants.

Initialisation Initialement $Q = \emptyset$, et Done $= \emptyset$ assurant donc trivialement les deux propriétés.

Propagation Notons $\underline{Q}$, $\underline{\mathsf{Done}}$ les valeurs de Q et Done avant une itération de boucle et notons

$\overline{Q}$ et $\overline{\mathsf{Done}}$ les valeurs de Q et Done après cette même itération. Supposons que $\underline{Q}$ et $\underline{\mathsf{Done}}$ vérifient les propriétés $\mathcal{I}_1$ et $\mathcal{I}_2$, autrement dit $\underline{Q} \subseteq Q^\star$ et $\forall D \in \underline{\mathsf{Done}}, C^+ \in \underline{Q}$. Soit alors la clause C sélectionnée à cette itération de boucle : $C \in \Gamma$ telle que $C^- \subseteq \underline{Q}$. C est de la forme $\{p, \neg q_1, \neg q_2, \ldots, \neg q_m\}$ avec $\forall i \in [\![1, m]\!], q_i \in \underline{Q} \subseteq Q^\star$. Ainsi $\forall i \in [\![1, m]\!], \mu^\star(q_i) = \mathsf{V}$. Finalement de **Q. 10** on déduit que $\mu^\star(p) = \mathsf{V}$. Ainsi $p \in Q^\star$, donc $\overline{Q} = \underline{Q} \cup \{p\} \subseteq Q^\star$. Soit finalement $D \in \overline{\mathsf{Done}}$ si $D \in \underline{\mathsf{Done}}$ alors $D^+ \in \underline{Q} \subseteq \overline{Q}$, sinon si $D = C$ alors $D^+ = p$ et $D^+ \in \overline{Q}$.

Q. 20 Démontrer la terminaison de l'algorithme. On explicitera *très précisément* les hypothèses des propriétés utilisées pour justifier de la terminaison de l'algorithme.

Solution

Dans l'espace ordonné $(\mathbb{N}, \leqslant)$, bien fondé, $|\Gamma \setminus \mathsf{Done}|$ est un variant de la boucle **tant que** de l'algorithme. En effet :

- $|\Gamma \setminus \mathsf{Done}| \in \mathbb{N}$
- Notons $\underline{\mathsf{Done}}$ la valeur de Done avant une itération de boucle et notons $\overline{\mathsf{Done}}$ la valeur de Done après cette même itération. $\overline{\mathsf{Done}} = \underline{\mathsf{Done}} \cup \{C\}$ avec $C \in \Gamma \setminus \underline{\mathsf{Done}}$, ainsi $|\Gamma \setminus \overline{\mathsf{Done}}| = |\Gamma \setminus \underline{\mathsf{Done}}| - 1 < |\Gamma \setminus \underline{\mathsf{Done}}|$.

Puisque la boucle **tant que** admet un variant, elle termine.

Q. 21 Démontrer la correction de l'algorithme 1.

Solution

En fin (puisqu'on a établi la terminaison en **Q. 20**) de boucle la variable Q est telle que :

- $Q \subseteq Q^\star$ (par invariant **Q. 19**)
- $\forall C \in \mathsf{Done}, C^+ \in Q$ (par invariant **Q. 19**)
- $\forall C \in \Gamma \setminus \mathsf{Done}, \exists p \in C^-, p \notin Q$ (par négation de la condition de boucle)

Finalement, montrons que μ est un modèle de Γ. Soit $C \in \Gamma$.

- Si $C \in \mathsf{Done}$, alors $C^+ \in Q$, donc $\mu(C^+) = \mathsf{V}$, ainsi $[\![C]\!]^\mu = [\![C^+]\!]^\mu + \ldots = \mathsf{V}$
- Si $C \in \Gamma \setminus \mathsf{Done}$, soit $q \in C^-$ tel que $q \notin Q$, donc $\mu(q) = \mathsf{F}$, mais alors $[\![C]\!]^\mu = \overline{[\![q]\!]^\mu} + \ldots = \mathsf{V}$

Ceci étant vrai pour tout $C \in \Gamma$, on conclut que $[\![\Gamma]\!]^\mu = \mathsf{V}$. Par ailleurs montrons que $\mu \sqsubseteq \mu^\star$. Soit $p \in \mathscr{P}$, si $\mu(p) = \mathsf{F}$ alors $\mu(p) \preccurlyeq \mu^\star(p)$, si $\mu(p) = \mathsf{V}$ alors $p \in Q$, donc $p \in Q^\star$ donc $\mu^\star(p) = \mathsf{V}$, donc $\mu(p) \preccurlyeq \mu^\star(p)$. Finalement $\mu \sqsubseteq \mu^\star$ donc $\mu = \mu^\star$.

Q. 22 En utilisant l'algorithme précédent et la question **Q. 17**, proposer un algorithme permettant de résoudre le problème de la satisfiabilité d'une FNCH. On justifiera de sa correction.

Solution

L'algorithme suivant convient :

- Soit Γ une FNCH, notons Γ^+ l'ensemble de ses clauses non négatives et Γ^- l'ensemble de ses clauses négatives ;
- On calcule $\mu^\star$ le plus petit modèle de Γ^+ (par l'algorithme ci-dessus) ;
- Si $\forall C \in \Gamma^-, [\![C]\!]^{\mu^\star} = \mathsf{V}$, renvoyer V, sinon renvoyer F.

En effet montrons que Γ est satisfiable si et seulement si $\mu^\star$ est un modèle de Γ^-. Si Γ est satisfiable, elle admet un modèle μ qui est donc un modèle de Γ^+ et de Γ^-, $\mu^\star \sqsubseteq \mu$, c'est donc

un modèle de Γ^-, or c'est un modèle de Γ^+, c'est donc un modèle de Γ. Si $\mu^\star$ est un modèle de Γ^-, puisque c'est un modèle de Γ^+, c'est un modèle de Γ.

7. Implémentation en OCaml

On se propose dans cette section de réaliser une implémentation en OCaml d'une variante de l'algorithme présenté en section précédente.
On représente les variables par des chaînes de caractères.

```
type var = string
```

Une clause de Horn est alors représentée par une structure contenant deux champs :

- `conclusion` contenant un élément de type `var option`. `None` est utilisé pour indiquer que la clause ne contient pas de littéral positif, `Some(p)` pour indiquer que la clause contient le littéral positif `p`.
- `premisses` contenant la liste des variables négatives de la clause (celles notées q_1, q_2, ...q_n depuis le début de l'exercice).

```
type horn_clause = {
  conclusion : var option ;      (* p ou rien *)
  premisses  : var list          (* q_1, q_2, ... q_n *)
}
```

La clause $\{s, \neg p, \neg q\}$ peut♣ être représentée par l'objet OCaml `c_ex_1` ci-dessous. La clause $\{\neg p, \neg s\}$ peut♡ être représentée par l'objet OCaml `c_ex_2` ci-dessous.

```
let c_ex_1 = {conclusion = Some "s"; premisses = ["p"; "q"]}
let c_ex_2 = {conclusion = None    ; premisses = ["p"; "s"]}
```

Q. 23 Définir trois fonctions `est_fait`, `est_stricte`, `est_negative`, toutes de signature `horn_clause -> bool` permettant de tester respectivement si la clause passée en argument est un fait, est stricte, est négative.

Solution

```
let est_fait (hc: horn_clause) : bool     = hc.premisses  = []
let est_stricte (hc: horn_clause) : bool  = hc.premisses <> [] &&
 ↪  hc.conclusion <> None
let est_negative (hc: horn_clause) : bool = hc.conclusion = None
```

Une Fnch est un ensemble de clauses de Horn, ce que l'on représente en OCaml au moyen du type `horn_set` ci-dessous.

```
type horn_set = horn_clause array
```

Aussi, la Fnch $\{\{p\}, \{r, \neg p\}, \{s, \neg p, \neg q\}, \{q\}, \{\neg p, \neg s\}\}$ peut être représentée au moyen de la valeur OCaml suivante.

♣. On utilise *peut* et non pas *doit* car la représentation n'est pas unique : l'ordre des prémisses dans la liste n'est pas imposé.
♡. De même.

```
let hs_ex_1 =
  [|
    {conclusion = Some "p"; premisses = []}         ; (* {p} *)
    {conclusion = Some "r"; premisses = ["p"]}      ; (* {r, ¬p} *)
    {conclusion = Some "s"; premisses = ["p"; "q"]}; (* {s, ¬p, ¬q} *)
    {conclusion = Some "q"; premisses = []}         ; (* {q} *)
    {conclusion = None    ; premisses = ["p"; "s"]}; (* {¬p, ¬s} *)
  |]
```

L'algorithme que nous implémentons dans cette section suit, dans les grandes lignes, celui de la section précédente. Cet algorithme reçoit en argument une FNCH `hs`. On maintient une liste `q` des variables propositionnelles sélectionnées pour être mises à Vdans l'environnement réponse. On tient à jour un tableau `nb_premisses_restantes` de même taille que `hs`, dont la case d'indice i indique : le nombre de variables propositionnelles de `hs.(i).premisses` qui ne sont pas encore sélectionnées par l'algorithme ou `-1` si la clause a été traitée. On tient finalement à jour, une table de hachage `table`, indiquant pour chaque variable propositionnelle, la liste des indices (dans `hs`) des clauses de Horn dans lesquelles la variable apparaît comme prémisse. L'algorithme est alors le suivant.
Tant qu'il existe une clause `hs.(i)` telle que `nb_premisses_restantes.(i)` est zéro ;

- Si `hs.(i)` est sans conclusion : l'ensemble est non satisfiable.
- Si `hs.(i)` a pour conclusion une variable déjà sélectionnée on passe `nb_premisses_restantes.(i)` à `-1`.
- Si `hs.(i)` a pour conclusion une variable p non déjà sélectionnée, on ajoute p aux sélectionnées (la liste `q`), on décrémente de 1 le nombre de prémisses de toutes les clauses de Horn dans lesquels p apparaît. On passe, ici aussi, `nb_premisses_restantes.(i)` à `-1`.

Q. 24 Définir une fonction `trouve_zero (nb_premisses_restantes: int array): int option` prenant en argument un tableau d'entier et renvoyant :

- `None` si aucune case du tableau ne contient `0` ;
- `Some(i)` si `i` est le plus petit indice du tableau contenant `0`.

Solution

```
let trouve_zero (nb_premisses: int array) : int option =
  let n = Array.length nb_premisses in
  let i = ref 0 in
  while !i < n && nb_premisses.(!i) <> 0 do
    i := !i + 1
  done;
  if !i = n then None
  else Some(!i)
```

Q. 25 Définir `decremente_premisses (l: int list) (nb_premisses_restantes: int array) : unit` prenant en arguments une liste d'indices du tableau `nb_premisses_restantes` et décrémentant toutes les valeurs contenues dans les cases du tableau se trouvant à des indices dans `l`. Par exemple si `nb_premisses_restantes` est le tableau `[|1; 3; 5; 1|]` et `l` est liste `[1; 3|]`, le tableau `nb_premisses_restantes` devra être *modifié* en `[|1; 2; 5; 0|]`.

Solution

```
let rec decremente_premisses (l: int list) (nb_premisses: int array): unit
↪  =
  match l with
```

```
  | [] -> ()
  | p :: q ->
    nb_premisses.(p) <- nb_premisses.(p) - 1 ;
    decremente_premisses q nb_premisses
```

Q. 26 Définir une fonction `pre_traitement_nb_premisses (hs: horn_set) : int array`, prenant en argument une FNCH et renvoyant un tableau de même taille que le tableau `hs` en argument et contenant dans sa case d'indice `i` la longueur de la liste des prémisses de la clause `hs.(i)`.

Solution

```
let pre_traitement_nb_premisses (hs: horn_set): int array =
  (* calcule un tableau dont la i-ème case indique le nombre de prémisses
   ↪  de
     l'ensemble de Horn hs.(i) *)
  let n = Array.length hs in
  let rep = Array.make n 0 in
  for i = 0 to (n - 1) do
    rep.(i) <- List.length hs.(i).premisses
  done;
  rep
```

Q. 27 Définir `ajout_liaison (l: var list) (i: int) (tbl : (var, int list) Hashtbl.t) : unit`, prenant en arguments une liste de variables, un entier et une table de hachage associant des listes d'entiers à des variables. Cette fonction devra modifier la table de hachage pour que chaque liste `il` d'entiers associée (par la table `tbl`) à une variable `p` (de la liste `l`) soit modifiée par un ajout en tête de l'entier `i`. Ainsi si la table de hachage `tbl` représente les liaisons $(\texttt{"p"} \mapsto \texttt{[2; 3]}, \texttt{"q"} \mapsto \texttt{[5]}, \texttt{"r"} \mapsto \texttt{[6]})$, et que la fonction `ajout_liaison` est appelée avec les arguments `["p"; "q"; "s"]`, `4` et `tbl`, alors la table `tbl` devra être modifiée pour représenter les liaisons $(\texttt{"p"} \mapsto \texttt{[4; 2; 3]}, \texttt{"q"} \mapsto \texttt{[4; 5]}, \texttt{"r"} \mapsto \texttt{[6]}, \texttt{"s"} \mapsto \texttt{[4]})$.

Solution

```
let rec ajout_liaison (l: var list) (i: int) (tbl : (var, int list)
 ↪  Hashtbl.t): unit =
  (* ajoute, pour chaque variable v de l, la liaison (v, i) à la table tbl.
     Ainsi si tbl ne fait aucune liaison depuis v, alors on ajoutera une
      ↪  liaison (v, [i]),
     si tbl a déjà une liaison (v, occur), on remplace cette liaison par la
      ↪  liaison (v, i :: occur)
  *)
  match l with
  | [] -> ()
  | x :: q ->
    if Hashtbl.mem tbl x then
        let occur = Hashtbl.find tbl x in
        Hashtbl.replace tbl x (i :: occur)
    else Hashtbl.add tbl x [i];
    ajout_liaison q i tbl
```

Q. 28 Définir une fonction `pre_traitement_table (hs: horn_set): (var, int list) Hashtbl.t`

prenant en argument une FNCH et renvoyant une table de hachage associant à chaque variable propositionnelle p la liste des indices des clauses de `hs` dans lesquelles p apparaît comme prémisse. Si une variable n'apparaît pas elle peut ne pas être liée par la table de hachage. À titre d'exemple, sur l'ensemble `hs_ex_1` ci-dessus, la fonction devra renvoyer la table de hachage représentant les liaisons : (`"p"` $\mapsto$ `[1; 2; 4]`, `"q"` $\mapsto$ `[2]`, `"s"` $\mapsto$ `[4]`). L'ordre dans les listes n'importe pas. On utilisera la **Q. 27**

Solution

```
let pre_traitement_table (hs: horn_set): (var, int list) Hashtbl.t =
  (* calcule, pour chaque variable, la liste des numéros des clauses de
  ↪  Horn
     dans laquelle elle apparaît *)
  let res = Hashtbl.create 12 in
  for i = 0 to ((Array.length hs) - 1) do
    ajout_liaison hs.(i).premisses i res
  done ;
  res
```

Q. 29 Définir une fonction `horn_sat (hs: horn_set): var list option` prenant en argument une FNCH Γ et telle que :

- Si Γ est insatisfiable la fonction s'évalue en `None`
- Si Γ est satisfiable, de plus petit modèle $\mu^\star$, alors la fonction s'évalue à `Some(q)` où `q` est l'ensemble ♣ des variables propositionnelles p telles que $\mu^\star(p) = \mathsf{V}$.

Sur l'exemple `hs_ex_1` la fonction devra renvoyer `None`. Sur l'exemple `hs_ex_2` ci-dessous, la fonction devra renvoyer `Some(["s"; "q"; "r"; "p"])`.

```
let hs_ex_2 =
  [|
     {conclusion = Some "p"; premisses = []};
     {conclusion = Some "r"; premisses = ["p"]};
     {conclusion = Some "s"; premisses = ["p"; "q"]};
     {conclusion = Some "q"; premisses = []};
   |]
```

Solution

```
let horn_sat hs =
  let nb_premisses_restantes = pre_traitement_nb_premisses hs in
  let table = pre_traitement_table hs in
  let todo = ref (trouve_zero nb_premisses_restantes) in
  let is_sat = ref true in
  let q = ref [] in
  while !todo <> None && !is_sat do
    let i = Option.get !todo in
    begin
      match hs.(i).conclusion with
      | None -> is_sat := false
      | Some(v) when not (List.mem v !q) ->
        begin
```

♣. représenté par une liste sans doublons, l'ordre des éléments n'importe pas.

```
          nb_premisses_restantes.(i) <- -1;
          q := v :: !q ;
          decremente_premisses (if Hashtbl.mem table v then Hashtbl.find
           ↪  table v else []) nb_premisses_restantes
        end
      | Some(v) ->
        nb_premisses_restantes.(i) <- -1;
    end;
    todo := trouve_zero nb_premisses_restantes
  done;
  if !is_sat then Some (!q) else None
```

Annexe, quelques éléments de syntaxe OCaml

On rappelle ci-dessous quelques éléments de syntaxe OCaml.

Itération impérative non bornée. Le langage OCaml fournit l'expression suivante.

while e_c **do** e_b **done**

où
- e_c est une expression de type **bool** ;
- e_b est une expression de type **unit** ;

L'évaluation de cette expression conduit alors à :
- L'évaluation de e_c, si celle-ci conduit à `false` on s'arrête, sinon
- L'évaluation de e_b ;
- L'évaluation de e_c, si celle-ci conduit à `false` on s'arrête, sinon
- L'évaluation de e_b ;
- …
- À une évaluation en la valeur `()`

```
# let i = ref 0 in
  while !i < 5 do
    print_int !i ; print_string " " ;
    i := !i + 2
  done ;;
0 2 4 - : unit = ()
```

Quelques fonctions du module `List`. Dans cette annexe on rappelle quelques fonctions du module `List` de OCaml. Ces fonctions peuvent être utilisées dans tout le problème.
- `List.length : 'a list -> int` permet le calcul de la longueur d'une liste.
- `List.mem : 'a -> 'a list -> bool` permet de tester l'appartenance d'un élément à une liste. Aussi l'appel `(List.mem 3 [1; 2])` s'évalue à `false`, et l'appel `(List.mem 3 [3; 4])` s'évalue à `true`.
- `List.iter : ('a -> unit) -> 'a list -> unit` est une fonctionnelle permettant le calcul d'un schéma d'itération sur une liste.

  `(List.iter f [`x_1`; `x_2`; ...; `x_n`])` s'évalue de même que `(f `x_1`; f `x_2`; ...; f `x_n`)`

  Aussi l'appel `(List.iter (fun x -> print_string (x ^ " ")) ["toto"; "titi"])` produit l'affichage de la chaîne de caractères `"toto titi "`.

- `List.map : ('a -> 'b) -> 'a list -> 'b list` est une fonctionnelle permettant le calcul d'un schéma d'application sur une liste.

  `(List.map f [`x_1`; `x_2`; ...; `x_n`])` vaut `[(f `x_1`); (f `x_2`); ...; (f `x_n`)]`

  Aussi l'appel `(List.map (fun x -> 2 * x) [1; 2; 3])` s'évalue en `[2; 4; 6]`.
- `List.filter : ('a -> 'b) -> 'a list -> 'b list` est une fonctionnelle permettant le calcul d'un schéma de filtrage sur une liste.

  `(List.filter f [`x_1`; `x_2`; ...; `x_n`])` vaut `[`$\begin{cases} x_1 & \text{si (f } x_1\text{)} \\ & \text{sinon} \end{cases}$`; ...; `$\begin{cases} x_n & \text{si (f } x_n\text{)} \\ & \text{sinon} \end{cases}$`]`

  Aussi `(List.filter (fun x -> x mod 2 = 0) [1; 2; 2; 3; 4])` s'évalue en `[2; 2; 4]`.
- `List.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a` est une fonctionnelle permettant le calcul d'un schéma d'accumulation sur une liste.

  `(List.fold_left f a [`x_1`; `x_2`; ...; `x_n`])` vaut `(f (... (f (f a `x_1`) `x_2`) ...) `x_n`)`

  Aussi l'appel `(List.fold_left (fun acc x -> x + acc) 0 [1; 2; 3; 4])` s'évalue en `10`.

Utilisation du module `Hashtbl`. Le module `Hashtbl` d'OCaml implémente le type abstrait Dictionnaire qui permet la gestion d'ensembles dynamiques d'associations clés-valeurs, dont les clés sont 2 à 2 distinctes.
Les `Hashtbl` permettent ainsi de représenter :

→ des ensembles dynamiques (les éléments sont les clés, on peut leur associer une valeur booléenne indiquant la présence ou non, on peut aussi imaginer associer une valeur (quelconque) aux clés qui sont dans l'ensemble et ne pas en associer à celles qui ne sont pas dans l'ensemble) ;
→ des multi-ensembles (en remplaçant la présence booléenne par un nombre d'occurrences entier) ;
→ ou plus généralement une fonction sur un ensemble fini.

Une table dont les clés sont de type `'a` et dont les valeurs sont de type `'b` est de type `('a, 'b) Hashtbl.t`.
On crée une table de hachage comme suit, en précisant `n` une estimation du nombre de clés que contiendra la table. Cette valeur est donnée seulement en vue d'améliorer les performances de la table, elle ne limite pas le nombre de clés.

```
1  let tbl = Hashtbl.create n
```

Pour manipuler la table, on dispose des fonctions élémentaires suivantes.

- `Hashtbl.mem : ('a, 'b) Hashtbl.t -> 'a -> bool` qui teste la présence d'une clé dans la table.
- `Hashtbl.find : ('a, 'b) Hashtbl.t -> 'a -> 'b` qui donne la valeur associée à une clé présente dans la table. Si la clé n'est pas présente dans la table, l'exception `Not_found` est levée.
- `Hashtbl.add : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit` qui permet d'ajouter une association clé-valeur à la table. Si la clé était déjà présente, la valeur associée est masquée par la nouvelle valeur.
- `Hashtbl.replace : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit` agit comme `add` sauf que si la clé était déjà présente, la valeur qui lui était associée est écrasée.

De plus on dispose aussi d'une fonctionnelle `fold` pour itérer sur les associations de la table.

```
Hashtbl.fold : ('a -> 'b -> 'c -> 'c) -> ('a, 'b) Hashtbl.t -> 'c -> 'c
```

On remarque que l'ordre des arguments dans la fonction d'accumulation diffère de celui pour la fonctionelle `List.fold_left`, en effet ici l'accumulateur est passé en troisième argument.