EXERCICES DE TD

Exercice 1 : Définitions de cours : syntaxe

Considérons les formules propositionnelles suivantes :

- $H_1 = r \vee (p \wedge ((\neg q) \rightarrow r))$
- $H_2 = p \wedge (r \wedge ((\neg q) \rightarrow (\neg p)))$
- $H_3 = ((q \vee \neg p) \rightarrow (\neg \neg q \vee \neg p)) \wedge ((\neg \neg q \vee \neg p) \rightarrow (\neg p \vee q))$

Pour chaque formule :

- Q. 1 dessiner son arbre syntaxique ;
 Q. 2 donner ses sous-formules ;
 Q. 3 donner ses variables propositionnelles

Exercice 2 : Conséquence sémantique

Q. 1 Soient A, B, C trois formules propositionnelles. Les conséquences sémantiques suivantes sont-elles vérifiées ? Justifier.

1. $\{A \vee B, A \rightarrow C, B \rightarrow C\} \models C$	6. $A \rightarrow B \models \neg A \rightarrow \neg B$
2. $A \rightarrow B \models \neg B \rightarrow \neg A$	7. $\{A \rightarrow B, \neg B\} \models \neg A$
3. $\{A \vee B, \neg A\} \models B$	8. $\{A \rightarrow B, \neg A\} \models \neg B$
4. $A \rightarrow B \models B \rightarrow A$	9. $\{\neg B \rightarrow \neg A, A\} \models B$
5. $\{\neg(A \vee B), C \rightarrow B\} \models \neg(A \vee C)$	10. $\{\neg A \rightarrow \neg B, A\} \models B$

Solution

On traite les 3 premières conséquences sémantiques au moyen de raisonnement sur les booléens. Les deux suivantes au moyen de tables de vérité. La dernière au moyen d'équivalence sémantique.

1. Oui. Soit μ tel que (1) $\llbracket A \vee B \rrbracket^\mu = V$, (2) $\llbracket A \rightarrow C \rrbracket^\mu = V$ et (3) $\llbracket B \rightarrow C \rrbracket^\mu = V$, montrons que $\llbracket C \rrbracket^\mu = V$. De (1) $\llbracket A \rrbracket^\mu + \llbracket B \rrbracket^\mu = V$ et donc $\llbracket A \rrbracket^\mu = V$ ou $\llbracket B \rrbracket^\mu = V$.
- Si $\llbracket A \rrbracket^\mu = V$, puisque de (2) $\underbrace{\llbracket A \rrbracket^\mu}_{=V} + \llbracket C \rrbracket^\mu = V$ il vient $\llbracket C \rrbracket^\mu = V$
 - Sinon si $\llbracket B \rrbracket^\mu = V$, puisque de (3) $\underbrace{\llbracket B \rrbracket^\mu}_{=V} + \llbracket C \rrbracket^\mu = V$ il vient $\llbracket C \rrbracket^\mu = V$

Finalement dans tous les cas $\llbracket C \rrbracket^\mu = V$ et on peut conclure.

2. Oui. Soit μ une valuation. $\llbracket A \rightarrow B \rrbracket^\mu = \overline{\llbracket A \rrbracket^\mu} + \llbracket B \rrbracket^\mu = \overline{\llbracket A \rrbracket^\mu} + \overline{\overline{\llbracket B \rrbracket^\mu}} = \overline{\overline{\llbracket B \rrbracket^\mu}} + \overline{\llbracket A \rrbracket^\mu} = \llbracket \neg B \rightarrow \neg A \rrbracket^\mu$. Finalement ceci étant vrai pour tout μ , il vient que $A \rightarrow B \equiv \neg B \rightarrow \neg A$ et a fortiori $A \rightarrow B \models \neg B \rightarrow \neg A$.
3. Oui. Soit μ tel que $\llbracket A \vee B \rrbracket^\mu = \llbracket A \rrbracket^\mu + \llbracket B \rrbracket^\mu = V$ et $\llbracket \neg A \rrbracket^\mu = V$. Alors $\llbracket A \rrbracket^\mu = F$, donc $\llbracket A \rrbracket^\mu + \llbracket B \rrbracket^\mu = \llbracket B \rrbracket^\mu = V$. Finalement, ceci étant vrai pour tout μ , $\{A \vee B, \neg A\} \models B$.
4. Non. Un contre-exemple : soit $\llbracket A \rrbracket^\mu = V$ et $\llbracket B \rrbracket^\mu = V$. Alors $\llbracket A \rightarrow B \rrbracket^\mu = V$ mais $\llbracket B \rightarrow A \rrbracket^\mu = F$. Donc $A \rightarrow B \not\models B \rightarrow A$.

5. Oui. Par table de vérité, on constate bien que si on considère les lignes pour lesquels $\neg(A \vee B)$ et $C \rightarrow B$ sont rendus V par l'environnement propositionnel alors $\neg(A \vee C)$ l'est aussi.

A	B	C	$A \vee B$	$\neg(A \vee B)$	$C \rightarrow B$	$\neg(A \vee C)$
F	F	F	F	V	V	V
F	F	V	F	V	F	F
F	V	F	V	F	V	V
F	V	V	V	F	V	F
V	F	F	V	F	V	F
V	F	V	V	F	F	F
V	V	F	V	F	V	F
V	V	V	V	F	V	F

6. Non. Un contre-exemple : soit $\llbracket A \rrbracket^\mu = F$ et $\llbracket B \rrbracket^\mu = V$. Alors $\llbracket A \rightarrow B \rrbracket^\mu = V$, mais $\llbracket \neg A \rightarrow \neg B \rrbracket^\mu = F$.
7. Oui. Par table de vérité, on constate bien que si on considère les lignes pour lesquels $A \rightarrow B$ et $\neg B$ sont rendus V par l'environnement propositionnel alors $\neg A$ l'est aussi.

A	B	$A \rightarrow B$	$\neg B$	$\neg A$
F	F	V	V	V
F	V	V	F	V
V	F	F	V	F
V	V	V	F	F

8. Non. Un contre-exemple : $\llbracket A \rrbracket^\mu = F$, $\llbracket B \rrbracket^\mu = V$. Alors $\llbracket A \rightarrow B \rrbracket^\mu = V$, $\llbracket \neg A \rrbracket^\mu = V$, mais $\llbracket \neg B \rrbracket^\mu = F$.
9. Oui. Du cours, $\{\neg B \rightarrow \neg A, A\} \models B$ si et seulement si $((\neg B \rightarrow \neg A) \wedge A) \rightarrow B$ est valide.
Or

$$\begin{aligned}
 ((\neg B \rightarrow \neg A) \wedge A) \rightarrow B &\equiv \neg((\neg \neg B \vee \neg A) \wedge A) \vee B \\
 &\equiv (\neg(B \vee \neg A) \vee \neg A) \vee B \\
 &\equiv ((\neg B \wedge A) \vee \neg A) \vee B \\
 &\equiv ((\neg B \vee \neg A) \wedge (A \vee \neg A)) \vee B \\
 &\equiv ((\neg B \vee \neg A) \wedge \top) \vee B \\
 &\equiv (\neg B \vee \neg A) \vee B \\
 &\equiv \top
 \end{aligned}$$

Finalemment $((\neg B \rightarrow \neg A) \wedge A) \rightarrow B$ est bien valide.

10. Non. Un contre-exemple : $\llbracket A \rrbracket^\mu = V$, $\llbracket B \rrbracket^\mu = F$. Alors $\llbracket \neg A \rightarrow \neg B \rrbracket^\mu = V$, $\llbracket A \rrbracket^\mu = V$, mais $\llbracket B \rrbracket^\mu = F$.

Exercice 3 : Fraternité [CCP2010]

Les universités anglophones nord-américaines sont célèbres pour leurs associations estudiantines, les fraternités, ou sororités. Celles-ci développent souvent une culture du secret.

Vous venez de rejoindre la fraternité $\Gamma\alpha\chi$. Lors de votre initiation, vous avez appris les règles gou-

vernant les discussions pour que seuls les membres puissent en comprendre le contenu. Lorsque plusieurs membres prennent la parole sur un sujet donné, soit ils disent tous la vérité, soit ils mentent tous. Si quelqu'un intervient sur plusieurs sujets, il peut avoir un rôle différent pour chaque sujet. Pour être définitivement admis, vous devez démontrer votre maîtrise de ces règles. Vous participez à une discussion avec trois membres de la fraternité que nous appellerons A , B et C . Ceux-ci vous indiquent comment rejoindre la salle d'intronisation. Si vous n'arrivez pas à rejoindre directement cette salle, vous ne serez pas admis.

Un premier sujet est abordé concernant la pièce dans laquelle se tiendra la cérémonie :

1. A : « La cérémonie se tiendra dans le gymnase »
2. C : « Non, elle ne se tiendra pas dans cette pièce »
3. A : « Ou alors dans le réfectoire »

Nous noterons G et R les variables propositionnelles associées à la pièce où se tiendra la cérémonie. Nous noterons A_1 et C_1 les formules propositionnelles correspondant aux affirmations de A et C sur le premier sujet.

Puis, un second sujet est traité concernant les escaliers qui permettent de rejoindre la pièce où se tiendra la cérémonie.

1. A : « Le premier escalier conduit à l'intronisation »
2. C : « Tu as raison »
3. A : « Le troisième escalier y conduit aussi »
4. B : « Si le deuxième escalier y conduit, alors le troisième n'y conduit pas »
5. C : « Le deuxième n'y conduit pas »

Nous noterons E_1 , E_2 , E_3 les variables propositionnelles correspondant au fait que le premier, le deuxième, le troisième escalier conduisent à la salle de cérémonie.

Nous noterons A_2 , B_2 et C_2 les formules propositionnelles correspondant aux affirmations de A , B et C sur le second sujet.

Q. 1 Représenter le premier sujet abordé sous la forme d'une formule propositionnelle dépendant des formules A_1 , et C_1 .

Solution

$$(A_1 \wedge C_1) \vee (\neg A_1 \wedge \neg C_1)$$

Q. 2 Représenter les informations données par les participants sous la forme de deux formules propositionnelles A_1 et C_1 dépendant des variables G et R .

Solution

$$A_1 = G \vee R, C_1 = \neg G$$

Q. 3 Déterminer dans quelle pièce vous devez vous rendre pour rejoindre la cérémonie.

Solution

$$\begin{aligned} (A_1 \wedge C_1) \vee (\neg A_1 \wedge \neg C_1) &\equiv ((G \vee R) \wedge (\neg G)) \vee (\neg(G \vee R) \wedge \neg(\neg G)) \\ &\equiv (G \wedge \neg G) \vee (R \wedge \neg G) \vee (((\neg G) \wedge (\neg R)) \wedge G) \\ &\equiv (R \wedge \neg G) \end{aligned}$$

Aussi si ρ est un modèle de la formule, on a que $\rho(R) = V$ et $\rho(G) = F$, la cérémonie se tiendra donc dans le réfectoire et non dans le gymnase.

Q. 4 Représenter le second sujet abordé sous la forme d'une formule propositionnelle dépendant des formules A_2 , B_2 et C_2 .

Solution

$$(A_1 \wedge B_1 \wedge C_1) \vee (\neg A_1 \wedge \neg B_1 \wedge \neg C_1)$$

Q. 5 Représenter les informations données par les participants sous la forme de trois formules propositionnelles A_2 , B_2 et C_2 dépendant des variables E_1 , E_2 et E_3 .

Solution

$$A_2 = E_1 \wedge E_3, B_2 = E_2 \rightarrow \neg E_3, C_2 = A_2 \wedge \neg E_2$$

Q. 6 En utilisant le calcul des propositions (résolution avec les tables de vérité), déterminer l'escalier que vous devez suivre pour rejoindre la cérémonie.

Solution

E_1	E_2	E_3	A_2	B_2	C_2	$(A_1 \wedge B_1 \wedge C_1) \vee (\neg A_1 \wedge \neg B_1 \wedge \neg C_1)$
F	F	F	F	V	F	F
F	F	V	F	V	F	F
F	V	F	F	V	F	F
F	V	V	F	F	F	V
V	F	F	F	V	V	F
V	F	V	V	V	V	V
V	V	F	F	V	F	F
V	V	V	V	F	F	F

Il faut donc prendre l'escalier 3.

Q. 7 En admettant que les trois participants aient menti, pouviez vous prendre d'autres escaliers ? Si oui, le ou lesquels ?

Solution

Dans le cas où les 3 ont menti, il est aussi possible de prendre l'escalier 2.

Exercice 4 : Alice aux pays des merveilles [CCP 2002]

Lors de ses aventures au pays des merveilles, Alice est souvent accompagnée par le chat de Cheshire. Ce félin énigmatique s'exprime sous la forme d'affirmations logiques qui sont toujours vraies. Alice se trouve dans un couloir dont toutes les portes à sa taille sont fermées. La seule porte ouverte est nettement trop petite pour qu'elle puisse l'emprunter. Une étagère est fixée au-dessus de cette porte. Le chat dit alors à Alice : "L'un des flacons posés sur cette étagère contient un liquide qui te permettra de prendre une taille plus adéquate. Mais attention, les autres flacons peuvent contenir un poison fatal." Trois flacons sont effectivement posés sur l'étagère. Le premier est rouge, le second jaune, le troisième bleu. Une étiquette est collée sur chaque flacon. Alice lit l'inscription figurant sur chaque étiquette :

- Flacon rouge : le flacon jaune contient un poison. Le flacon bleu ne contient pas un poison ;
- Flacon jaune : si le flacon rouge contient un poison, alors le flacon bleu aussi ;

- Flacon bleu : je ne contiens pas un poison, mais au moins l'un des deux autres flacons contient un poison.
- Q. 1 Axiomatiser les énoncés ci-dessus au moyen de trois formules I_R, I_J, I_B , représentant les énoncés des flacons rouge, jaune et bleu.
- Q. 2 La conjonction de ces trois formules est elle satisfiable ?
- Q. 3 Une des inscriptions est-elle une conséquence sémantique des deux autres ?
- Q. 4 Dans le cas où aucun des trois flacons ne contient pas de poison, est-ce qu'une ou plusieurs(s) inscription(s) est(sont) fausse(s) ? Si oui, laquelle ou lesquelles ?
- Q. 5 Si les trois inscriptions sont vraies, est-ce qu'un ou plusieurs flacons contient(ent) un poison ? Si oui le(s)quel(s) ?
- Q. 6 Si seuls les flacons ne contenant pas de poison ont une inscription vraie, est-ce qu'un ou plusieurs flacon(s) ne contient(nent) pas de poison ? Si oui, lequel ou lesquels ?

Solution

Non

Exercice 5 : Barre de Scheffer

Nous enrichissons la logique d'un nouveau symbole binaire : nand , dont l'interprétation est définie de la manière suivante : $\llbracket H_1 \text{ nand } H_2 \rrbracket^\rho = \overline{\llbracket H_1 \rrbracket^\rho \cdot \llbracket H_2 \rrbracket^\rho}$.

Q. 1 Pour chaque formule suivante, donner une formule équivalente utilisant uniquement nand comme connecteur de formules.

- $H_1 \wedge H_2$
- $H_1 \vee H_2$
- $H_1 \leftrightarrow H_2$
- $\neg H_1$
- $H_1 \rightarrow H_2$

Solution

- $\neg H_1 \equiv H_1 \text{ nand } H_1$.
- $H_1 \wedge H_2 \equiv \neg(H_1 \text{ nand } H_2) \equiv (H_1 \text{ nand } H_2) \text{ nand } (H_1 \text{ nand } H_2)$
- $H_1 \vee H_2 \equiv \neg(\neg H_1 \wedge \neg H_2) \equiv \neg H_1 \text{ nand } \neg H_2 \equiv (H_1 \text{ nand } H_1) \text{ nand } (H_2 \text{ nand } H_2)$
- $H_1 \rightarrow H_2 \equiv \neg(H_1 \wedge \neg H_2) \equiv H_1 \text{ nand } \neg H_2 \equiv H_1 \text{ nand } (H_2 \text{ nand } H_2)$
- $H_1 \leftrightarrow H_2 \equiv (H_1 \rightarrow H_2) \wedge (H_2 \rightarrow H_1) \equiv H_1 \text{ nand } (H_2 \text{ nand } H_2) \equiv \dots$

Q. 2 Faire de même avec l'opérateur nor dont l'interprétation est $\llbracket H_1 \text{ nor } H_2 \rrbracket^\rho = \overline{\llbracket H_1 \rrbracket^\rho + \llbracket H_2 \rrbracket^\rho}$

Solution

De même.

Exercice 6 : Formules duales

Dans cet exercice on se limite aux formules n'utilisant pas $\rightarrow$ et $\leftrightarrow$. On définit le *dual* d'une formule H , dénoté H^* comme étant la formule obtenue en inversant $\top$ et $\perp$ et en inversant $\wedge$ et $\vee$.

Q. 1 Donner une définition inductive du dual d'une formule.

Solution

On définit la fonction $\square^*$ par induction de la manière suivante :

$$\begin{aligned}
 \top^* &= \perp \\
 \perp^* &= \top \\
 p^* &= p \\
 (G \wedge H)^* &= G^* \vee H^* \\
 (G \vee H)^* &= G^* \wedge H^* \\
 (\neg G)^* &= \neg(G^*)
 \end{aligned}
 \quad \text{pour } p \in \mathcal{P}$$

Q. 2 Montrer que si deux formules sont équivalentes, alors leurs duales sont aussi équivalentes.

Solution

Étant donné un environnement propositionnel $\rho \in \mathbb{B}^\mathcal{P}$, on définit l'environnement propositionnel $\rho^\square$ comme étant $p \mapsto \rho(p)$. Montrons par induction la propriété suivante sur $\mathcal{F}$, $P(G) : \forall \rho \in \mathbb{B}^\mathcal{P}, \llbracket G^* \rrbracket^\rho = \llbracket \neg G \rrbracket^{\rho^\square}$.

- Montrons $P(\perp)$: Soit $\rho \in \mathbb{B}^\mathcal{P}$, $\llbracket \perp^* \rrbracket^\rho = \llbracket \top \rrbracket^\rho = \llbracket \neg \perp \rrbracket^\rho = \llbracket \neg \perp \rrbracket^{\rho^\square}$
- Montrons $P(\top)$: Soit $\rho \in \mathbb{B}^\mathcal{P}$, $\llbracket \top^* \rrbracket^\rho = \llbracket \perp \rrbracket^\rho = \llbracket \neg \top \rrbracket^\rho = \llbracket \neg \top \rrbracket^{\rho^\square}$
- Soit $p \in \mathcal{P}$, montrons $P(p)$: Soit $\rho \in \mathbb{B}^\mathcal{P}$, $\llbracket p^* \rrbracket^\rho = \llbracket p \rrbracket^\rho = \rho(p) = \overline{\rho^\square(p)} = \llbracket \neg p \rrbracket^{\rho^\square}$
- Soit G telle que $P(G)$, montrons $P(\neg G)$: Soit $\rho \in \mathbb{B}^\mathcal{P}$, $\llbracket (\neg G)^* \rrbracket^\rho = \llbracket \neg G^* \rrbracket^\rho = \overline{\llbracket G^* \rrbracket^\rho} = \overline{\llbracket \neg G \rrbracket^{\rho^\square}} = \llbracket \neg(\neg G) \rrbracket^{\rho^\square}$
- Soit G et H telles que $P(G)$ et $P(H)$, montrons $P(G \wedge H)$: Soit $\rho \in \mathbb{B}^\mathcal{P}$, $\llbracket (G \wedge H)^* \rrbracket^\rho = \llbracket G^* \vee H^* \rrbracket^\rho = \llbracket G^* \rrbracket^\rho + \llbracket H^* \rrbracket^\rho = \llbracket \neg G \rrbracket^{\rho^\square} + \llbracket \neg H \rrbracket^{\rho^\square} = \overline{\llbracket \neg G \rrbracket^{\rho^\square} \cdot \llbracket \neg H \rrbracket^{\rho^\square}} = \overline{\llbracket G \rrbracket^{\rho^\square} \cdot \llbracket H \rrbracket^{\rho^\square}} = \overline{\llbracket G \wedge H \rrbracket^{\rho^\square}} = \llbracket \neg(G \wedge H) \rrbracket^{\rho^\square}$
- Soit G et H telles que $P(G)$ et $P(H)$, montrons $P(G \vee H)$: Soit $\rho \in \mathbb{B}^\mathcal{P}$, $\llbracket (G \vee H)^* \rrbracket^\rho = \llbracket G^* \wedge H^* \rrbracket^\rho = \llbracket G^* \rrbracket^\rho \cdot \llbracket H^* \rrbracket^\rho = \llbracket \neg G \rrbracket^{\rho^\square} \cdot \llbracket \neg H \rrbracket^{\rho^\square} = \overline{\llbracket \neg G \rrbracket^{\rho^\square} + \llbracket \neg H \rrbracket^{\rho^\square}} = \overline{\llbracket G \rrbracket^{\rho^\square} + \llbracket H \rrbracket^{\rho^\square}} = \overline{\llbracket G \vee H \rrbracket^{\rho^\square}} = \llbracket \neg(G \vee H) \rrbracket^{\rho^\square}$

On remarque de plus que $(\rho^\square)^\square = \rho$.

Ainsi soit deux formules G et H telles que $G \equiv H$, soit $\rho \in \mathbb{B}^\mathcal{P}$, on a $\llbracket G^* \rrbracket^\rho = \llbracket \neg G \rrbracket^{\rho^\square} = \overline{\llbracket G \rrbracket^{\rho^\square}} = \overline{\llbracket H \rrbracket^{\rho^\square}} = \llbracket \neg H \rrbracket^{\rho^\square} = \llbracket H^* \rrbracket^\rho$ et finalement on a bien que $G^* \equiv H^*$.

Q. 3 Montrer si une formule est valide, sa formule duale est non satisfiable.

Solution

Soit G une formule valide alors $G \equiv \top$, on a donc $G^* \equiv \top^* = \perp$ et donc G^* est insatisfiable.

Exercice 7 : Logique avec If

Logique propositionnelle alternative $\mathcal{F}_{if}$ Étant donné un ensemble de variables propositionnelles $\mathcal{Q}$, l'ensemble des formules de la logique propositionnelle *alternative* $\mathcal{F}_{if}$ sur $\mathcal{Q}$, est défini inductivement comme suit.

- $\top$ et $\perp$ sont des formules de $\mathcal{F}_{if}$.
- Toutes les variables propositionnelles de $\mathcal{Q}$ sont dans $\mathcal{F}_{if}$.
- Si C, G et H sont trois formules de $\mathcal{F}_{if}$, alors $\text{if } C \text{ then } G \text{ else } H$ est une formule de $\mathcal{F}_{if}$.

Interprétation des formules de $\mathcal{F}_{\text{if}}$ Étant donné un ensemble de variables propositionnelles $\mathcal{Q}$ et un environnement propositionnel ρ sur $\mathcal{Q}$, l'interprétation des formules de $\mathcal{F}_{\text{if}}$ est définie comme suit.

$$\begin{aligned} \llbracket \top \rrbracket^\rho &= \text{V} & \llbracket p \rrbracket^\rho &= \rho(p) \text{ si } p \in \mathcal{Q} \\ \llbracket \perp \rrbracket^\rho &= \text{F} & \llbracket \text{if } C \text{ then } G \text{ else } H \rrbracket^\rho &= \llbracket C \rrbracket^\rho \cdot \llbracket G \rrbracket^\rho + \overline{\llbracket C \rrbracket^\rho} \cdot \llbracket H \rrbracket^\rho \end{aligned}$$

1. Représentabilité des fonctions booléennes par formules de $\mathcal{F}_{\text{if}}$

Q. 1 Donner une formule de $\mathcal{F}_{\text{if}}$ dont la sémantique est la fonction booléenne $(p, q, r) \mapsto p + q + r$.

Solution

if p then $\top$ else (if q then $\top$ else r)

Q. 2 Exprimer la sémantique de if C then G else H comme une disjonction, selon la valeur de $\llbracket C \rrbracket^\rho$.

Solution

Si $\llbracket C \rrbracket^\rho = \text{V}$ alors $\llbracket \text{if } C \text{ then } G \text{ else } H \rrbracket^\mu = \llbracket G \rrbracket^\mu$, sinon $\llbracket \text{if } C \text{ then } G \text{ else } H \rrbracket^\mu = \llbracket H \rrbracket^\mu$.

Q. 3 Démontrer que toute fonction booléenne peut être représentée par une formule de la logique avec $\mathcal{F}_{\text{if}}$.

Solution

On montre le résultat par récurrence sur le cardinal de l'ensemble $\mathcal{Q}$ des variables propositionnelles.

- Si $\mathcal{Q} = \emptyset$ alors il y a deux fonctions booléennes : la fonction qui a la valuation vide ($()$) associe V et celle qui a l'environnement vide ($()$) associe F. Cette première fonction est représentée par la formule $\top$, la seconde par la formule $\perp$.
- Sinon, soit $\mathcal{Q} = \mathcal{Q}' \cup \{p\}$ où $p \notin \mathcal{Q}'$. Soit f une fonction booléenne sur $\mathcal{Q}$. Soit g_V la fonction booléenne définie sur $\mathcal{Q}'$ par $g_V(\mu) = f(\mu \uplus (p \mapsto \text{V}))$ où $\mu \uplus (p \mapsto \text{V})$ est la valuation définie sur $\mathcal{Q}$, qui coïncide avec μ sur $\mathcal{Q}'$ et qui V sur p . Soit de même g_F la fonction booléenne définie sur $\mathcal{Q}'$ par $g_F(\mu) = f(\mu \uplus (p \mapsto \text{F}))$. g_V et g_F sont des fonctions booléennes définies sur les variables propositionnelles $\mathcal{Q}'$. Par hypothèse de récurrence, soient donc deux formules de $\mathcal{F}_{\text{if}}$ G_V et G_F dont g_V et g_F sont les sémantiques. Considérons alors la formule $G \stackrel{\text{def}}{=} \text{if } p \text{ then } G_V \text{ else } G_F$. Soit μ un environnement propositionnel sur $\mathcal{Q}$.

◦ Si $\mu(p) = \text{V}$ alors $\llbracket G \rrbracket^\mu = \llbracket G_V \rrbracket^\mu = \llbracket G_V \rrbracket^{\mu|_{\mathcal{Q}'}} = g_V(\mu|_{\mathcal{Q}'}) = g(\mu|_{\mathcal{Q}'} \uplus (p \mapsto \text{V})) = g(\mu)$

◦ Si $\mu(p) = \text{F}$ alors $\llbracket G \rrbracket^\mu = \llbracket G_F \rrbracket^\mu = \llbracket G_F \rrbracket^{\mu|_{\mathcal{Q}'}} = g_F(\mu|_{\mathcal{Q}'}) = g(\mu|_{\mathcal{Q}'} \uplus (p \mapsto \text{F})) = g(\mu)$

Finalement la sémantique de G est bien g .

Q. 4 Donner une fonction $\varphi : \mathcal{F}_{\text{if}} \rightarrow \mathcal{F}$ associant une formule de $\mathcal{F}$ à chaque formule de $\mathcal{F}_{\text{if}}$, φ doit être telle que $\forall H \in \mathcal{F}_{\text{if}}, \llbracket \varphi(H) \rrbracket = \llbracket H \rrbracket$.

Solution

On définit inductivement φ .

- $\varphi(\top) = \top$
- $\varphi(\perp) = \perp$
- $\varphi(p) = p$
- $\varphi(\text{if } C \text{ then } G \text{ else } H) = (\varphi(C) \wedge \varphi(G)) \vee (\neg\varphi(C) \wedge \varphi(H))$

L'équivalence demandée découle de la définition de la sémantique de $\text{if } C \text{ then } G \text{ else } H$.

Q. 5 Conclure quant à la représentabilité des fonctions booléennes dans la logique des propositions.

Solution

Soit g une fonction booléenne, il existe G une formule de $\mathcal{F}_{\text{if}}$ telle que $\llbracket G \rrbracket = g$ et donc $\varphi(G)$ est une formule de la logique propositionnelle telle que $\llbracket \varphi(G) \rrbracket = \llbracket G \rrbracket = g$. Finalement on retrouve le résultat de cours sur l'exhaustivité de la logique propositionnelle.

2. Représentabilité des formules de la logique propositionnelle par la logique alternative

Q. 6 Démontrer, en utilisant les résultats précédents, que pour toute formule H de la logique propositionnelle, il existe une formule équivalente à H en logique alternative.

Solution

Soit H une formule de la logique propositionnelle. $\llbracket H \rrbracket$ est une fonction booléenne, il existe donc une formule G de $\mathcal{F}_{\text{if}}$ telle que $\llbracket G \rrbracket = \llbracket H \rrbracket$. Autrement dit G est une formule de $\mathcal{F}_{\text{if}}$ équivalente à H .

On s'intéresse maintenant à la construction d'une telle formule.

Q. 7 Donner une fonction $\psi : \mathcal{F} \rightarrow \mathcal{F}_{\text{if}}$ associant une formule de $\mathcal{F}_{\text{if}}$ à chaque formule de $\mathcal{F}$, ψ doit être telle que $\forall H \in \mathcal{F}, \llbracket \psi(H) \rrbracket = \llbracket H \rrbracket$.

Solution

On définit ψ par induction structurelle.

- $\psi(\top) = \top$
- $\psi(\perp) = \perp$
- $\psi(p) = p$
- $\psi(\neg G) = \text{if } G \text{ then } \perp \text{ else } \top$
- $\psi(G \vee H) = \text{if } G \text{ then } \top \text{ else } H$
- $\psi(G \wedge H) = \text{if } G \text{ then } H \text{ else } \perp$
- $\psi(G \rightarrow H) = \text{if } G \text{ then } H \text{ else } \top$
- $\psi(G \leftrightarrow H) = \text{if } G \text{ then } H \text{ else } (\text{if } H \text{ then } \perp \text{ else } \top)$

On restreint la logique alternative en n'autorisant que des formules $\text{if } C \text{ then } G \text{ else } H$ dans lesquelles C est une variable propositionnelle, on dénote $\mathcal{F}_{\text{if}}^-$ cette logique.

Règles de réécriture. On appelle règle de réécriture (c'est une définition par l'exemple) un énoncé de la forme $H_1 \wedge H_2 \rightsquigarrow \neg((\neg H_1) \vee (\neg H_2))$. Une telle règle décrit comment des sous-formules d'une formule doivent être successivement transformées. Un ensemble de règles de réécriture $(g_1 \rightsquigarrow d_1, \dots, g_n \rightsquigarrow d_n)$ décrit donc un algorithme :

Algorithme 1 : Application d'un système de réécriture $(g_1 \rightsquigarrow d_1, \dots, g_n \rightsquigarrow d_n)$ à une formule H

Entrée : Une formule H

- 1 **tant que** il existe une sous-formule G de H de la forme g_i **faire**
 - 2 $\lfloor$ remplacer G dans H par d_i ;
-

Par exemple la règle de réécriture $H_1 \wedge H_2 \rightsquigarrow \neg((\neg H_1) \vee (\neg H_2))$ permet la transformation d'une formule en une formule équivalente ne contenant pas de connecteur $\wedge$.

Q. 8 Proposer un ensemble de règles de réécriture dont l'application successive permet la transformation d'une formule de $\mathcal{F}_{if}$ en une formule de $\mathcal{F}_{if}^-$ équivalente. Démontrer la correction de votre ensemble de règles de réécriture.

Solution

- $(\text{if } \top \text{ then } G \text{ else } H) \rightsquigarrow G$
- $(\text{if } \perp \text{ then } G \text{ else } H) \rightsquigarrow H$
- $(\text{if}(\text{if}C\text{then}I\text{else}J)\text{then}G\text{else}H) \rightsquigarrow \text{if}C\text{then}(\text{if}I\text{then}G\text{else}H)\text{else}(\text{if}J\text{then}G\text{else}H)$

Pour démontrer la correction, il suffit de remarquer que :

- $\text{if } \top \text{ then } G \text{ else } H \equiv (\top \wedge G) \vee (\neg \top \wedge H) \equiv G$
- $\text{if } \perp \text{ then } G \text{ else } H \equiv (\perp \wedge G) \vee (\neg \perp \wedge H) \equiv H$
-

$$\begin{aligned}
& (\text{if}(\text{if}C\text{then}I\text{else}J)\text{then}G\text{else}H) \\
& \equiv ((\text{if}C\text{then}I\text{else}J) \wedge G) \vee (\neg(\text{if}C\text{then}I\text{else}J) \wedge H) \\
& \equiv (((C \wedge I) \vee (\neg C \wedge J)) \wedge G) \vee (\neg((C \wedge I) \vee (\neg C \wedge J)) \wedge H) \\
& \equiv (((C \wedge I \wedge G) \vee (\neg C \wedge J \wedge G)) \vee (((\neg C \vee \neg I) \wedge (C \vee \neg J)) \wedge H) \\
& \equiv (((C \wedge I \wedge G) \vee (\neg C \wedge J \wedge G)) \\
& \quad \vee (\neg C \wedge C \wedge H) \vee (\neg C \wedge J \wedge H) \vee (\neg I \wedge C \wedge H) \vee (\neg I \wedge \neg J \wedge H) \\
& \equiv (((C \wedge I \wedge G) \vee (\neg C \wedge J \wedge G)) \\
& \quad \vee (\neg C \wedge J \wedge H) \vee (C \wedge \neg I \wedge H) \vee (\neg I \wedge \neg J \wedge H) \\
& \equiv (((C \wedge I \wedge G) \vee (\neg C \wedge J \wedge G)) \\
& \quad \vee (\neg C \wedge J \wedge H) \vee (C \wedge \neg I \wedge H) \vee ((C \vee \neg C) \wedge \neg I \wedge \neg J \wedge H) \\
& \equiv (((C \wedge I \wedge G) \vee (\neg C \wedge J \wedge G)) \\
& \quad \vee (\neg C \wedge J \wedge H) \vee (C \wedge \neg I \wedge H) \vee (C \wedge \neg I \wedge \neg J \wedge H) \\
& \equiv ((C \wedge I \wedge G) \vee (\neg C \wedge J \wedge G)) \vee (\neg C \wedge J \wedge H) \vee (C \wedge \neg I \wedge H) \\
& \equiv (C \wedge (I \wedge G \vee \neg I \wedge H)) \vee (\neg C \wedge (J \wedge G \vee J \wedge H)) \\
& \equiv \text{if}C\text{then}(\text{if}I\text{then}G\text{else}H)\text{else}(\text{if}J\text{then}G\text{else}H)
\end{aligned}$$

Q. 9 Est-il nécessaire d'appliquer cet algorithme aux formules produites à la **Q. 3** ?

Solution

Non, nous avons "fabriqué" des formules déjà sous la forme voulue.

Exercice 8 : Syntaxe et sémantique de la logique propositionnelle

Dans tout cet exercice on suppose que les formules de la logique propositionnelle sont définies sur un ensemble de variables propositionnelles $\mathcal{P} = \{x_0, x_1, x_2, \dots, x_{n-1}\}$ admettant une indexation par des entiers naturels d'un intervalle $\llbracket 0, n-1 \rrbracket$.

Valuations. Une valuation est une fonction de $\mathcal{P}$ dans l'ensemble des booléens $\mathbb{B}$. Une telle fonction représente une affectation des variables propositionnelles. Étant donné que les variables propositionnelles admettent une indexation par les des entiers naturels d'un intervalle $\llbracket 0, n-1 \rrbracket$, on peut représenter une valuation au moyen d'un tableau de booléens. La valuation $(x_0 \mapsto F, x_1 \mapsto V, x_2 \mapsto F)$ sera représenté par le tableau `[|false; true; false|]`. Ainsi on se munit de la définition de type ci-dessous.

```
1 | type valu = bool array
```

Syntaxe de la logique propositionnelle. On rappelle, qu'étant donné un ensemble de variables propositionnelles $\mathcal{Q}$, on définit les formules de la logique propositionnelle $\mathcal{F}$ comme étant le plus petit ensemble obtenu inductivement de la manière suivante :

- $\perp, \top \in \mathcal{F}^2$
- Si $p \in \mathcal{Q}, p \in \mathcal{F}$
- Si $G \in \mathcal{F}$ et $H \in \mathcal{F}$ alors $G \vee H \in \mathcal{F}, G \wedge H \in \mathcal{F}, G \rightarrow H \in \mathcal{F}, G \leftrightarrow H \in \mathcal{F}$
- Si $G \in \mathcal{F}$ alors $\neg G \in \mathcal{F}$

Une telle définition nous conduit naturellement à définir le type *formule* ci-dessous.

```
1 | type formule =
2 |   | Top
3 |   | Bot
4 |   | Var   of int
5 |   | And   of formule * formule
6 |   | Or    of formule * formule
7 |   | Imply of formule * formule
8 |   | Equiv of formule * formule
9 |   | Not   of formule
```

compagnon_logique.ml. Dans le fichier compagnon, on trouvera des définitions des types introduits ci-dessus, ainsi qu'une fonction `affiche : formule -> unit` permettant l'affichage des formules de la logique propositionnelle.

1. Syntaxe

- Q. 1** Définir une fonction `empty_vars : formule -> bool` prenant en argument une formule G de la logique propositionnelle et retournant si cette formule est sans variable propositionnelle ou non. Par exemple pour la formule $(x_1 \wedge x_2)$ la fonction devra s'évaluer à `false`, mais pour la formule $\top \rightarrow (\neg \perp)$ la fonction devra s'évaluer à `true`.

Solution

```
1  (** Teste si la formule [g] est sans variable. *)
2  let rec empty_vars (g: formule): bool =
3      match g with
4      | Top
5      | Bot          -> true
6      | Var _        -> false
7      | Not(g)       -> (empty_vars g)
8      | And(g, d)
9      | Or(g, d)
10     | Imply(g, d)
11     | Equiv(g, d) -> (empty_vars g) && (empty_vars d)
```

Q. 2 Définir une fonction `taille : formule -> int` prenant en argument une formule G de la logique propositionnelle et retournant la taille de cette formule.

Solution

```
1  (** Calcule la taille de la formule [g]. *)
2  let rec taille (g: formule): int =
3      match g with
4      | Top
5      | Bot
6      | Var _        -> 1
7      | Not(g)       -> 1 + (taille g)
8      | And(g, d)
9      | Or(g, d)
10     | Imply(g, d)
11     | Equiv(g, d) -> 1 + (taille g) + (taille d)
```

Q. 3 Définir une fonction `vars : formule -> int` prenant en argument une formule G de la logique propositionnelle et retournant le plus petit entier $n \in \mathbb{N}$ tels que les variables propositionnelles de G sont toutes indicées par des entiers de $\llbracket 0, n - 1 \rrbracket$.

Solution

```
1  (** Retourne le plus petit entier n tel que les variables propositionnelles
2      de G sont toutes indicées par des entiers de [|0, n - 1|] *)
3  let rec vars (g: formule): int =
4      match g with
5      | Top
6      | Bot          -> 0
7      | Var i        -> i + 1
8      | Not(g)       -> (vars g)
9      | And(g, d)
10     | Or(g, d)
11     | Imply(g, d)
12     | Equiv(g, d) -> max (vars g) (vars d)
```

2. Sémantique.

Q. 4 Définir une fonction `interprete : formule -> valu -> bool` permettant l'interprétation d'une formule dans une valuation. On lèvera une exception si la valuation ne contient pas toutes les variables apparaissant dans la formule.

Solution

```
1  (** Retourne l'interprétation de la formule [g] dans la valuation [mu]. *)
2  let rec interprete (g: formule) (mu: valu): bool =
3      match g with
4      | Top          -> true
5      | Bot          -> false
6      | Var i        ->
7          if i > Array.length mu then
8              failwith "pas une variable de l'environnement propositionnel"
9          else mu.(i)
10     | Not(g)        -> not (interprete g mu)
11     | And(g, h)     -> (interprete g mu) && (interprete h mu)
```

Q. 5 Définir une fonction `next : valu -> unit` prenant en argument une valuation et le modifiant pour qu'elle contienne la valuation suivante dans la numérotation "canonique" des tableaux de booléens.

Indication. Cette numérotation canonique est celle de la représentation en base 2 d'un entier naturel. Ainsi pour $n = 3$, la numérotation des tableaux est :

0) [false ; false ; false]	3) [false ; true ; true]
1) [false ; false ; true]	4) [true ; false ; false]
2) [false ; true ; false]	5) ...

Solution

```
1  (** Modifie le tableau de booléen passé en paramètre de sorte qu'il
2      contienne le prochain tableau de booléen pour la numérotation
3      "canonique" des tableaux de booléens (celle qui coïncide avec
4      l'incrémement des entiers). *)
5  let next (mu: valu): unit =
6      let i = ref (Array.length mu - 1) in
7      let retenue = ref true in
8      while (!i >= 0 && !retenue) do
9          retenue := mu.(!i);
10         mu.(!i) <- not mu.(!i);
11         decr i;
12     done
```

Q. 6 Définir une fonction `sat : formule -> valu option♣` permettant de résoudre le problème de la satisfiabilité d'une formule. On retournera donc `None` si la formule est insatisfiable et `Some( $\mu$ )` sinon, où μ est un modèle de la formule.

♣. Voir topo en fin d'énoncé pour type 'a option

Solution

```
1  (** Résout le problème sat sur la formule g. *)
2  let sat (g: formule): valu option =
3      let n = vars g in
4      let mu = Array.make n false in
5      let found = ref None in
6      let i = ref 0 in
7      (* 1 lsl n = 2^n *)
8      while (!i < (1 lsl n) && !found = None) do
9          if interprete g mu then found := Some mu
10         else next mu;
11         incr i
12     done;
13     !found
```

Interlude. Considérons l'énigme ci-dessous.

Vous êtes perdus dans le désert et vous suivez une piste depuis de longues heures quand vous débouchez soudain sur une bifurcation. Vous savez que les deux pistes qui s'ouvrent à vous peuvent éventuellement conduire à une oasis, mais aussi vous perdre à tout jamais. Chacune d'elles est gardée par un sphinx qui s'anime à votre arrivée et commence à parler : Le premier vous dit : « une au moins des deux pistes conduit à une oasis. » Le second ajoute : « la piste de droite se perd dans le désert. » Sachant que les deux sphinx disent tous deux la vérité, ou bien mentent tous deux, que faites vous ?

Q. 7 Résoudre l'énigme ci-dessus au moyen de la fonction `sat`.

Q. 8 Définir une fonction `est_valide : formule -> bool` permettant de tester si une formule est valide.

Solution

```
1  (** Résout le problème valide sur la formule g. *)
2  let valide (g: formule): bool =
3      let n = vars g in
4      let mu = Array.make n false in
5      let ok = ref true in
6      let i = ref 0 in
7      (* 1 lsl n = 2^n *)
8      while (!i < (1 lsl n) && !ok) do
9          ok := interprete g mu;
10         incr i
11     done;
12     !ok
```

Q. 9 Définir une fonction `est_cons_semantique : formule -> formule -> bool` permettant de tester si une formule est conséquence sémantique d'une autre.

Solution

```
1 (** Teste si [h] est conséquence sémantique de [g]. *)
2 let est_cons_semantique (g: formule) (h: formule): bool =
3   valide (ImPLY(g, h))
```

Q. 10 Définir une fonction `equiv : formule -> formule -> bool` permettant de tester si deux formules sont équivalentes.

Solution

```
1 (** Teste si [h] est équivalente à [g]. *)
2 let equiv (g: formule) (h: formule): bool =
3   (est_cons_semantique g h) && (est_cons_semantique h g)
```

Q. 11 Définir une fonction `models : formule -> valu list` donnant l'ensemble des modèles d'une formule.

Solution

```
1 (** Donne l'ensemble des modèles de [g]. *)
2 let models (g: formule): valu list =
3   let n = vars g in
4   let mu = Array.make n false in
5   let res = ref [] in
6   (* 1 lsl n = 2^n *)
7   for i = 0 to ((1 lsl n) - 1) do
8     if interprete g mu then res := (Array.copy mu) :: !res;
9     next mu
10  done;
11  !res
```

Exercice 9 : Construction d'une formule à partir d'une fonction

Cet exercice nécessite d'avoir fait l'exercice précédent. On définit le type ci-dessous.

```
1 | type fct_bool = valu -> bool
```

Q. 1 Donner une fonction `formule_of_fct_bool : string list -> fct_bool -> formule` calculant une formule dont la sémantique est la fonction booléenne passée en argument. On utilisera dans cette question l'algorithme vu en classe et produisant une disjonction de conjonction de littéraux. On fournit en argument l'ensemble des variables propositionnelles sur lequel est définie la fonction booléenne. On pourra donc travailler en plusieurs temps :

- Écriture d'une fonction prenant en argument un couple (p, b) et retournant le littéral p si $b = \text{true}$ et $\neg p$ sinon.
- Écriture d'une fonction prenant en argument un environnement propositionnel μ et retournant une clause conjonctive dont μ est le seul modèle (notons H_μ une telle formule).
- Écriture de la fonction `formule_of_fct_bool` construisant la disjonction de toutes les clauses H_μ pour μ dans l'image réciproque par la fonction booléenne de V (autrement dit : les environnements pour lesquels la fonction booléenne en question vaut vrai).

Q. 2 Donner une fonction `formule_of_fct_bool2 : string list -> fct_bool -> formule` calculant une formule dont la sémantique est la fonction booléenne passée en argument. On utilisera dans cette question l’algorithme vu en classe et produisant une conjonction de disjonction de littéraux. On fournit en argument l’ensemble des variables propositionnelles sur lequel est définie la fonction booléenne.

Exercice 10 : Application de règles de réécriture

On s’intéresse à l’utilisation de *règles de réécritures* pour la transformation automatique de formules.

Règles de réécriture. On appelle *règle de réécriture* ♣ un énoncé de la forme $H_1 \wedge H_2 \rightsquigarrow \neg((\neg H_1) \vee (\neg H_2))$. Une telle règle décrit comment des sous-formules d’une formule doivent être successivement transformées. Un ensemble de règles de réécriture $(g_1 \rightsquigarrow d_1, \dots, g_n \rightsquigarrow d_n)$ décrit alors un algorithme.

Algorithme 2 : Application d’un système de réécriture $(g_1 \rightsquigarrow d_1, \dots, g_n \rightsquigarrow d_n)$ à une formule H

Entrée : Une formule H

- 1 **tant que** il existe une sous-formule G de H “de la forme” g_i **faire**
 - 2 |_ remplacer G dans H par d_i ;
-

Par exemple la règle de réécriture $H_1 \wedge H_2 \rightsquigarrow \neg((\neg H_1) \vee (\neg H_2))$ permet la transformation d’une formule en une formule équivalente ne contenant pas de connecteur $\wedge$. En effet en utilisant cette règle la formule $(p \wedge (q \wedge r))$ est transformée en $\neg(\neg p \vee \neg(q \wedge r))$, elle-même transformée en $\neg(\neg p \vee \neg(\neg q \vee \neg r))$.

On représente un ensemble de règles de réécriture en OCaml par une fonction de type `formule -> formule option`[♥]. On définit donc le type : `type rewrite = formule -> formule option`. Par exemple la règle de réécriture $H_1 \wedge H_2 \rightsquigarrow \neg((\neg H_1) \vee (\neg H_2))$ peut être encodé au moyen de la fonction :

```

1 let ex =
2   fun (f: formule) -> match f with
3     | And(h1, h2) -> Some(Not(Or(Not(h1), Not(h2))))
4     | _ -> None

```

Ainsi lorsque la fonction retourne `None`, c’est qu’aucune règle de réécriture n’a pu être appliquée sur la “tête” de la formule, sinon c’est qu’une réécriture peut avoir lieu, auquel cas la fonction nous fournit la formule transformée.

L’ensemble de règles de réécriture $(H_1 \wedge H_2 \rightsquigarrow \neg((\neg H_1) \vee (\neg H_2)), H_1 \vee H_2 \rightsquigarrow \neg((\neg H_1) \wedge (\neg H_2)))$ peut être encodé au moyen de la fonction :

```

1 let ex =
2   fun (f: formule) -> match f with
3     | And(h1, h2) -> Some(Not(Or(Not(h1), Not(h2))))
4     | Or(h1, h2) -> Some(Not(And(Not(h1), Not(h2))))
5     | _ -> None

```

On remarque au passage que cet ensemble de règles de réécriture a le mauvais goût de fournir un algorithme ne terminant pas.

♣. C’est une définition par l’exemple, la notion de règle de réécriture n’est pas à votre programme

♥. Voir topo sur le type ‘a option en fin d’énoncé

Q. 1 Donner une fonction **val** `rewrite_one : rewrite -> formule -> formule option`, permettant l'application d'une règle de réécriture sur la formule H passée en argument. S'il est possible d'appliquer des réécritures à plusieurs sous-formules de H on choisira d'appliquer la réécriture sur la première sous-formule de H rencontrée dans l'ordre de parcours préfixe des sous-formules. Si aucune sous-formule de H ne conduit à une réécriture, votre fonction devra alors s'évaluer à `None`.

Exemples :

Avec le système de réécriture exemple ci-dessus, et la formule $H = (p \wedge q) \rightarrow (q \wedge r)$, la fonction doit calculer la formule $\neg(\neg p \vee \neg q) \rightarrow (q \wedge r)$

Q. 2 En déduire une fonction **val** `rewrite : rewrite -> formule -> formule` calculant le résultat de l'algorithme d'application des règles de réécriture décrit ci-avant. Votre fonction pourra ne pas terminer, en fonction du système de réécriture.

Application. On cherche à construire une formule équivalente qui ne contient que des variables propositionnelles et les connecteurs $\neg$ et $\wedge$.

Q. 3 Proposer un système de réécriture permettant cette transformation.

Q. 4 Tester votre système de réécriture avec votre fonction `rewrite` ci-avant définie.

Q. 5 Donner les arguments justifiant de la terminaison et de la correction de votre système de réécriture.

Le type 'a option en OCAML

Il n'est pas rare que l'on souhaite étendre un type OCAML `t` pour y ajouter une valeur. Considérons par exemple l'ensemble des entiers `int` et la fonction de division :

```
1 let ma_division (x: int) (y: int) : int =
2   print_string "je fais une division";
3   (x / y)
```

Cette fonction est bien définie mais lève une erreur quand on l'appelle avec une valeur nulle pour l'entier `y`. Aussi on souhaite ajouter une valeur spéciale, qui ne soit pas un entier, aux entiers, on pourra alors retourner cette valeur dans le cas d'une division par zéro.

OCAML fournit pour cela le type prédéfini `'a option`, défini de la manière suivante :

```
1 type 'a option =
2   | None
3   | Some of 'a
```

Aussi un élément de type `'a option` est : ou bien la valeur `None`, ou bien la valeur `Some(x)` où `x` est de type `'a`. On peut alors redéfinir notre division de la manière suivante :

```
1 let ma_division (x: int) (y: int) : int option =
2   print_string "je fais une division";
3   if y = 0 then None
4   else Some(x / y)
```

Noter le type de retour de la fonction.

```
1 # ma_division 4 2 ;;
2 je fais une division- : int option = Some 2
3 # ma_division 4 0 ;;
4 je fais une division- : int option = None
```

Bien sûr, en tant que type défini par énumération, on peut raisonner par disjonction sur une valeur de type 'a option au moyen de la construction `match ... with ...`. Ainsi on peut définir la fonction ci-dessous effectuant une division par 2 au moyen de la fonction `ma_division`.

```
1 | let division_par_2 (x: int) : int =
2 |   match ma_division x 2 with
3 |   | None -> failwith "ma_division : division par zéro"
4 |   | Some(r) -> r
```

Noter le type de retour de la fonction.

```
1 | # division_par_2 3 ;;
2 | je fais une division- : int = 1
```

Une utilisation classique du type 'a option est en substitution au type `bool`. Considérons par exemple une fonction de recherche : *existe-t-il un élément dans le tableau vérifiant telle propriété?* Une telle fonction aurait un type de retour booléen : oui ou non. On pourrait toutefois enrichir une telle fonction en : *existe-t-il un élément dans le tableau vérifiant telle propriété? Si oui le retourner.* Une telle fonction aurait un type de retour optionnel : non (`None`) ou oui et voici l'élément (`Some(élément)`).