CORRIGÉ DE L'ÉPREUVE D'INFORMATIQUE DES ENSI - 2002

Programmation en Caml

Partie I - Logique et calcul des propositions

La notation $\overline{P}$ signifie $\neg P$.

Question I.1 $I_R = J \wedge \overline{B}, \qquad I_J = (R \Rightarrow B) \equiv \overline{R} \vee B, \qquad I_B = \overline{B} \wedge (R \vee J).$

- Solutions avec une table de vérité

	R	J	B	I_R	I_J	I_B	$I_R \wedge I_J$	$I_J \wedge I_B$	$I_R \wedge I_B$
1	0	0	0		1				
2	0	0	1		1				
3	0	1	0	1	1	1	1	1	1
4	0	1	1		1				
5	1	0	0			1			
6	1	0	1		1				
7	1	1	0	1		1			1
8	1	1	1		1				

Question I.2 Les inscriptions sur les trois flacons sont toutes vraies si et seulement si on se trouve dans les conditions de la ligne 3, c'est à dire si le flacon jaune est le seul à contenir un poison.

Question I.3 Sachant que $(P \Rightarrow Q)$ est vraie si et seulement si pour toute valuation v des variables figurant dans les propositions P et Q, on a $v(P) \leq v(Q)$, on constate que :

$\star$ $(I_R \wedge I_J \Rightarrow I_B)$ et $(I_J \wedge I_B \Rightarrow I_R)$ sont vraies. $\star$ $(I_R \wedge I_B \Rightarrow I_J)$ est fausse (cf. ligne 7).

Donc il y a deux inscriptions dépendantes.

Question I.4 Si on suppose qu'aucun des flacons ne contient un poison, on se trouve alors dans la situation de la ligne 1 : les deux inscriptions I_R et I_B sont fausses (alors que I_J est vraie).

Question I.5 Les trois inscriptions sont vraies si et seulement si on se trouve dans la situation de la ligne 3 : seul le flacon jaune contient alors du poison.

Question I.6 Remarque : $\overline{P} \Leftrightarrow Q \equiv P \oplus Q$ où $\oplus$ désigne le ou exclusif (xor).

On veut que $(\overline{R} \Leftrightarrow I_R) \wedge (\overline{J} \Leftrightarrow I_J) \wedge (\overline{B} \Leftrightarrow I_B)$ soit vraie.

$*$ $(\overline{R} \Leftrightarrow I_R)$ est vraie aux lignes 3, 5, 6, 8.

$*$ $(\overline{J} \Leftrightarrow I_J)$ est vraie aux lignes 1, 2, 6, 7.

$*$ $(\overline{B} \Leftrightarrow I_B)$ est vraie aux lignes 2, 3, 4, 5, 6, 7, 8.

Elles sont donc toutes les trois vraies à la ligne 6 : alors seul le flacon jaune ne contient pas de poison.

• Solutions avec les lois de Morgan

On utilisera les formules (dites d'absorption) suivantes : $\boxed{P \wedge (\overline{P} \vee Q) \equiv P \wedge Q}$ et $\boxed{P \vee (\overline{P} \wedge Q) \equiv P \vee Q.}$

Question I.2 $I_R \wedge I_J \wedge I_B \equiv J \wedge \overline{B} \wedge (\overline{R} \vee B) \wedge \overline{B} \wedge (R \vee J) \equiv \overline{B} \wedge (B \vee \overline{R}) \wedge J \wedge (R \vee J) \equiv \overline{B} \wedge \overline{R} \wedge J.$

Les inscriptions sur les trois flacons sont toutes vraies si et seulement si le flacon jaune est le seul à contenir un poison.

Question I.3

$* \ \big(I_R \wedge I_J \Rightarrow I_B\big) \equiv \big(J \wedge \overline{B} \wedge (B \vee \overline{R}) \Rightarrow I_B\big) \equiv \big(J \wedge \overline{B} \wedge \overline{R} \Rightarrow \overline{B} \wedge (R \vee J)\big)$
$\equiv (\overline{J} \vee B \vee R) \vee \big(\overline{B} \wedge (R \vee J)\big) \equiv (\overline{J} \vee B \vee R \vee \overline{B}) \wedge (\overline{J} \vee B \vee R \vee J) \equiv \mathbf{1} \wedge \mathbf{1} \equiv \mathbf{1}.$

$* \ \big(I_J \wedge I_B \Rightarrow I_R\big) \equiv \big((\overline{R} \vee B) \wedge \overline{B} \wedge (R \vee J) \Rightarrow I_R\big) \equiv \big(\overline{R} \wedge \overline{B} \wedge (R \vee J) \Rightarrow I_R\big)$
$\equiv \big(\overline{B} \wedge \overline{R} \wedge J \Rightarrow I_R\big) \equiv \big(\overline{B \vee R \vee \overline{J}} \Rightarrow J \wedge \overline{B}\big) \equiv (B \vee R \vee \overline{J}) \vee (J \wedge \overline{B})$
$\equiv (B \vee R \vee \overline{J} \vee J) \wedge (B \vee R \vee \overline{J} \vee \overline{B}) \equiv \mathbf{1} \wedge \mathbf{1} \equiv \mathbf{1}.$

$* \ \big(I_R \wedge I_B \Rightarrow I_J\big) \equiv \big((J \wedge \overline{B} \wedge (R \vee J) \Rightarrow I_J\big) \equiv \big((J \wedge \overline{B} \Rightarrow I_J\big) \equiv \overline{J} \vee B \vee \overline{R} \vee B \equiv \overline{R} \vee \overline{J} \vee B.$
Donc cette implication est fausse lorsque $R = J = 1$ et $B = 0$ (cf. ligne 7).

En résumé, il y a donc deux relations de dépendance.

Question I.4 Si on suppose que $R = J = B = 0$, alors $I_R = J \vee \overline{B} = 0$, $I_J = \overline{R} \vee B = 1$ et $I_B = \overline{B} \wedge (J \vee R) = 0$.
Donc I_R et I_B sont fausses.

Question I.5 $I_R \wedge I_J \wedge I_B = \overline{B} \wedge \overline{R} \wedge J$ d'après bf I.2
$I_R = I_J = I_B = 1 \Longleftrightarrow I_R \wedge I_J \wedge I_B = 1 \equiv (J = 1 \wedge B = 0 \wedge R = 0).$
Donc seul le flacon jaune contient du poison.

Question I.6 Mettons $E = (\overline{R} \Leftrightarrow I_R) \wedge (\overline{J} \Leftrightarrow I_J) \wedge (\overline{B} \Leftrightarrow I_B)$ sous forme conjonctive.

$* \ (\overline{R} \Leftrightarrow I_R) \equiv (\overline{R} \Longrightarrow I_R) \wedge (I_R \Rightarrow \overline{R}) \equiv (R \vee I_R) \wedge (\overline{I_R} \vee \overline{R}) \equiv \big(R \vee (J \wedge \overline{B})\big) \wedge \big((\overline{J} \vee B) \vee \overline{R}\big) \equiv (R \vee J) \wedge (R \vee \overline{B}) \wedge (B \vee \overline{J} \vee \overline{R}).$

$* \ (\overline{J} \Leftrightarrow I_J) \equiv (\overline{J} \Longrightarrow I_J) \wedge (I_J \Rightarrow \overline{J}) \equiv (J \vee I_J) \wedge (\overline{I_J} \vee \overline{J}) \equiv (J \vee \overline{R} \vee B) \wedge \big((R \wedge \overline{B}) \vee \overline{J}\big) \equiv (\overline{R} \vee J \vee B) \wedge (R \vee \overline{J}) \wedge (\overline{B} \vee \overline{J}).$

$* \ (\overline{B} \Leftrightarrow I_B) \equiv (B \vee I_B) \wedge (\overline{I_B} \vee \overline{B}) \equiv \big(B \vee (\overline{B} \wedge (J \vee R))\big) \wedge \big(B \vee (\overline{J \vee R}) \vee \overline{B}\big) \equiv B \vee (J \vee R) \wedge \mathbf{1} \equiv B \vee J \vee R.$

Ainsi $E \equiv (R \vee J \vee B) \wedge (\overline{R} \vee J \vee B) \wedge (R \vee J) \wedge (R \vee \overline{J}) \wedge (R \vee \overline{B}) \wedge (\overline{J} \vee \overline{B}) \wedge (\overline{R} \vee \overline{J} \vee B).$
Donc $E \equiv (J \vee B) \wedge R \wedge (R \vee \overline{B}) \wedge (\overline{J} \vee \overline{B}) \wedge (\overline{R} \vee \overline{J} \vee B)$,
d'où $E \equiv (J \vee B) \wedge (\overline{J} \vee \overline{B}) \wedge R \wedge (\overline{R} \vee \overline{J} \vee B) \equiv (J \vee B) \wedge (\overline{J} \vee \overline{B}) \wedge (\overline{J} \vee B) \wedge R \equiv (J \vee B) \wedge \overline{J} \wedge R.$
Ainsi $\boxed{E \equiv R \wedge \overline{J} \wedge B.}$
En résumé, $E = 1$ si et seulement si alors seul le flacon jaune ne contient pas de poison.

Partie II - Automates et langages

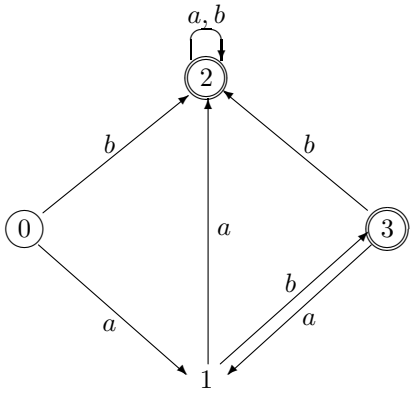
Pour la troisième fois en quatre ans, le sujet porte sur un produit d'automates déterministes complets avec presque les mêmes questions qu'en 1999 et 2000.

Question II.1 C étant un rebut, $\boxed{E_1 = a(ba)^*.}$

Question II.2 $\boxed{E_2 = (a+b)(a+b)^*.}$ $L(\mathcal{E}_2)$ est l'ensemble des mots non vides sur l'alphabet $X = \{a, b\}$.

Question III.3 Formons la table des transitions d'un automate déterministe complet et accessible équivalent à $\mathcal{E}_1 \oplus \mathcal{E}_2$.

		a	b
0 : initial	(A,D)	(B,E)	(C,E)
1 :	(B,E)	(C,E)	(A,E)
2 : final	(C,E)	(C,E)	(C,E)
3 : final	(A,E)	(B,E)	(C,E)

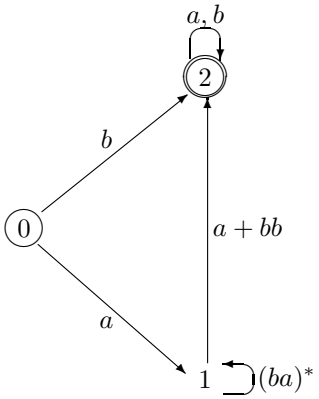


Question II.4 Le langage reconnu par cet automate est la réunion de deux langages L' et L''.

. L' est le langage reconnu lorsque l'état 3 est le seul état final. L'état 2 est alors un rebut. Donc L' admet pour expression régulière $E' = ab(ab)^* \equiv a(ba)^*b$.

. L'' est le langage reconnu lorsque l'état 2 est le seul état final.

La suppression de l'état 3 conduit à l'automate symbolique suivant :



Donc $E'' = \big(b + a(ba)^*(a + bb)\big)(a + b)^*$.

Ainsi $L(\mathcal{E}_1 \oplus \mathcal{E}_2)$ est représenté par $\boxed{E = E' + E'' = a(ba)^*b + \big(b + a(ba)^*a + a(ba)^*bb\big)(a + b)^*.}$

Question II.5 Si δ_1 et δ_2 sont définies sur $X \times Q_1$ et $X \times Q_2$ respectivement, alors $\delta_{1\oplus 2}$ est définie sur $X \times (Q_1 \times Q_2)$, donc $A_1 \oplus A_2$ est un automate déterministe complet.

Question II.6 Montrons par induction structurelle que $\forall\, m \in X^*,\ \delta^*_{1\oplus 2}\big(m, (q_1, q_2)\big) = \big(\delta^*_1(m, q_1), \delta^*_2(m, q_2)\big)$ (**1**).

. C'est vrai si $m = \Lambda$ car $\delta^*_{1\oplus 2}\big(\Lambda, (q_1, q_2)\big) = (q,q_2) = \big(\delta^*_1(\Lambda, q_1), \delta^*_2(\Lambda, q_2)\big)$.

Supposons (**1**) vraie pour un mot m et montrons qu'elle est aussi vraie pour le mot $m.x$ où x est quelconque dans X.

$$\begin{aligned}
\delta^*_{1\oplus 2}\big(m.x, (q_1, q_2)\big) &= \delta_{1\oplus 2}\Big(x, \delta^*_{1\oplus 2}\big(m, (q_1, q_2)\big)\Big) \quad \text{par définition de } \delta^*_{1\oplus 2} \\
&= \delta_{1\oplus 2}\Big(x, \big(\delta^*_1(m, q_1), \delta_2(m, q_2)\big)\Big) \quad \text{d'après l'hypothèse d'induction} \\
&= \Big(\delta_1\big(x, \delta^*_1(m, q_1)\big), \delta_2\big(x, \delta^*_2(m, q_2)\big)\Big) \quad \text{par définition de } \delta_{1\oplus 2} \\
&= \big(\delta^*_1(m.x, q_1), \delta^*_2(m.x, q_2)\big) \quad \text{par définition de } \delta^*_1 \text{ et } \delta^*_2.
\end{aligned}$$

Question II.7 Si L_1 et L_2 sont deux langages sur l'alphabet X, notons $L_1 \oplus L_2$ leur différence symétrique définie par :

$$\boxed{L_1 \oplus L_2 = (L_1 \cap L_1^C) \cup (L_1^C \cap L_2) = (L_1 \cup L_2) \cap (L_1 \cap L_2)^C.}$$

$$\begin{aligned}
m \in L(A_1 \oplus A_2) &\iff \delta_{1\oplus 2}\big(m, (i_1, i_2)\big) \in \big(T_1 \times (Q_2 \setminus T_2)\big) \cup \big((Q_1 \setminus T_1) \times T_2\big) \\
&\iff \big(\delta^*_1(m, i_1) \in T_1 \wedge \delta^*_2(m, i_2) \notin T_2\big) \vee \big(\delta^*_1(m, i_1) \notin T_1 \wedge \delta^*_2(m, i_2) \in T_2\big) \\
&\iff \big(m \in L(A_1) \ \wedge\ m \notin L(A_2)\big) \vee \big(m \notin L(A_1) \ \wedge\ m \in L(A_2)\big) \\
&\iff \big(m \in L(A_1) \ \vee\ m \in L(A_2)\big) \wedge \big(m \notin L(A_1) \cap L(A_2)\big)
\end{aligned}$$

Question II.8 On a donc $\boxed{L(A_1 \oplus A_2) = L(A_1) \oplus L(A_2).}$

Remarque : vérifions la réponse obtenue au **II.4**.

Comme $L_2^C = \{\Lambda\}$, alors $L_1 \cap L_2^C = \emptyset$, donc $L_1 \oplus L_2 = L_1^C \cap L_2$ est formé des mots non vides qui ne sont pas dans L_1. Ce sont tous les mots :

. qui commencent par b associés à $b(a+b)^*$

. qui ont pour préfixe un mot de $L\big((a(ba)^*\big)$, suivi d'un a, puis de lettres quelconques correspondant à l'expression rationnelle $a(ba)^*a(a+b)^*$

. qui ont pour préfixe un mot de $L\big((a(ba)^*\big)$, suivi d'un bb, puis de lettres quelconques correspondant à l'expression rationnelle $a(ba)^*bb(a+b)^*$.

. qui ont pour préfixe un mot de $L\big((a(ba)^*\big)$, suivi d'un b, correspondant à l'expression rationnelle $a(ba)^*b$.

On retrouve $E = a(ba)^*b + \big(b + a(ba)^*a + a(ba)^*bb\big)\big)(a+b)^*$.

Partie III - Algorithmique et programmation en CaML

Question III.1 Fonction `coder : suite -> cle -> boolean list`

```
type boolean = O | I ;;              (* Pour simuler 0 et 1 *)

type suite == char list ;;
type code  == boolean list ;;
type cle   == (char * code) list ;;

let coder  (s:suite) (c:cle)=
  let rec code_of_char t cl =  match cl with
         |      []   -> failwith "Erreur dans coder"
         | (x,l)::r  -> if x=t then l
                                else code_of_char t r
  and coder_aux ss = match ss with
         |     []  ->  []
         | t::q -> (code_of_char t c)@(coder_aux q)

  in coder_aux s ;;
```

Question III.2 La fonction auxiliaire `code_of_char t cl` renvoie le code du caractère `t` selon la clé de codage `cl` en parcourant la liste `cl` des couples `(x,l)` jusqu'à ce que `x=t`.
La fonction auxiliaire `coder_aux` parcourt une suite `ss` de caractères et renvoie la liste concaténée des codes de chacun d'eux.

Question III.3 La complexité de la fonction `code_of_char` est en $O(|c|)$ et la complexité totale de la fonction `coder` est en $\underline{O(|s|\,|c|)}$.

Question III.4 Si l'arbre associé à la clé de codage est complet et si $b_1 \dots b_r \in \mathcal{I}$, alors il existe dans l'arbre un chemin étiqueté par $b_1 \dots b_r$ partant de la racine de l'arbre et aboutissant à une feuille v_i.
Pour chaque $i \in [\![1;r]\!]$ le nœud situé à la hauteur $i-1$ situé sur ce chemin est un nœud interne n : il a donc deux fils. L'un d'eux contient la feuille v_i et l'autre contient au moins une feuille v_j. Comme l'étiquette du chemin conduisant au nœud n est $b_1 \dots b_{i-1}$, alors celle de la feuille v_j est de la forme $b_1 \dots b_{i-1} b'_i \dots b'_s \in \mathcal{I}$ avec $b'_i = \neg b_i$.
On démontre ainsi que la clé de codage associée à un arbre complet est optimale.

Question III.5 Caractérisation des arbres complets associés à des clés de codage séparables.
Soit c une clé de codage associée à un arbre complet a.
c est séparable si et seulement si aucun préfixe d'un élément $b \in \mathcal{I}$ n'est dans I, c'est à dire si sur tout chemin allant de la racine à une feuille v de code $b \in \mathcal{I}$, aucun nœud intermédiaire ne contient de caractère.

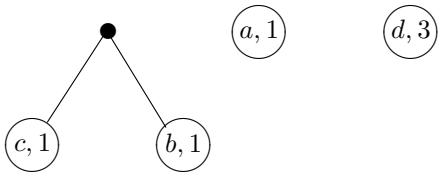
Question III.6 Fonction `nombre : arbre -> int`

```
let rec nombre a = match a with
       |     Vide     -> 0
       | Feuille(_,n) -> n
       | Noeud(ag,ad) -> (nombre ag) + (nombre ad) ;;
```

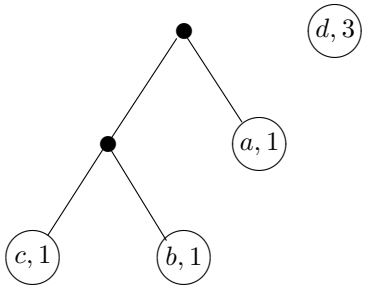
Question III.7 L'arbre obtenu dépend de l'ordre dans lequel se présente les quatre feuilles au départ.

Au départ : $(c,1)$ $(b,1)$ $(a,1)$ $(d,3)$

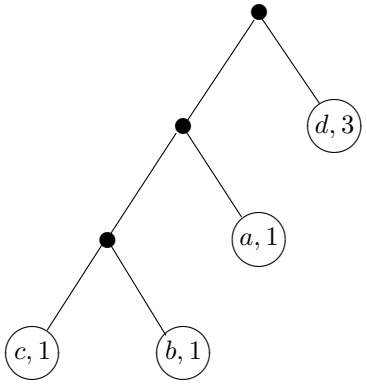
Étape 1 :



Étape 2 :



Étape 3 :



Question III.8 La contribution dans H du coût de v_i et de v_j est égale à $\mathcal{C}(v_i)\,\mathcal{O}(v_i) + \mathcal{C}(v_j)\,\mathcal{O}(v_j)$.
Dans H', elle est égale à $\mathcal{C}(v_j)\,\mathcal{O}(v_i) + \mathcal{C}(v_i)\,\mathcal{O}(v_j)$.
Donc $\Delta = \mathcal{C}(H) - \mathcal{C}(H') = \mathcal{C}(v_i)\big(\mathcal{O}(v_i) - \mathcal{O}(v_j)\big) + \mathcal{C}(v_j)\big(\mathcal{O}(v_j) - \mathcal{O}(v_i)\big) = \underline{\big(\mathcal{C}(v_i) - \mathcal{C}(v_j)\big)\big(\mathcal{O}(v_i) - \mathcal{O}(v_j)\big)}$.

Question III.9 Considérons toutes les feuilles de H de profondeur maximale h et notons m leur nombre (m est pair car l'arbre est complet). Notons $I = \big\{i,\ i \in [\![1;p]\!] \ /\ \mathcal{C}(v_i) = h\big\}$.
Raisonnons par l'absurde en supposant qu'il existe une feuille v_j située à une hauteur $< h$ et dont l'occurence $\mathcal{O}(v_j)$ est strictement inférieure à celle d'un v_i avec $i \in I$. En échangeant dans H les feuilles v_i et v_j, on obtient un arbre H' tel que $\mathcal{C}(H) - \mathcal{C}(H') = \big(h - \mathcal{C}(v_j)\big)\big(\mathcal{O}(v_i) - \mathcal{O}(v_j)\big) > 0$, donc $\mathcal{C}(H') < \mathcal{C}(H)$, ce qui contredit la minimalité de H.

Question III.10 Supposons H minimal et H' non minimal.

Notons h la hauteur de H et K la contribution du coût relatif aux feuilles autres que v_i et v_j dans H (et donc aussi dans H').

On a $\mathcal{C}(H) = K + h.\big(\mathcal{O}(v_i) + \mathcal{O}(v_j)\big)$ et $\mathcal{C}(H') = K + (h-1).\big(\mathcal{O}(v_i) + \mathcal{O}(v_j)\big)$, donc $\mathcal{C}(H) = \mathcal{C}(H') + \mathcal{O}(v_i) + \mathcal{O}(v_j)$.

Posons $\mathcal{V} = \{v_1, \ldots, v_p\}$ et $\mathcal{V}' = \mathcal{V} \setminus \{v_i, v_j\}$.

Puisque H' n'est pas minimal pour $\mathcal{V}' \cup \{n\}$, il existe un arbre de Huffman H'' pour $\mathcal{V}' \cup \{n\}$ tel que $\mathcal{C}(H'') < \mathcal{C}(H')$.

L'arbre H_1 obtenu à partir de H'' en remplaçant la feuille n par l'arbre binaire de hauteur 1 dont les feuilles sont v_i et v_j est un arbre de Huffman pour $\mathcal{V}$ et vérifie : $\mathcal{C}(H_1) = \mathcal{C}(H'') + \mathcal{O}(v_i) + \mathcal{O}(v_j)$.

On a alors $\mathcal{C}(H_1) < \mathcal{C}(H)$, ce qui contredit la minimalité de H.

Question III.11 On appelle taille d'un arbre de Huffman le nombre de ses feuilles. Ainsi tout arbre de Huffman pour $\{v_1, \ldots, v_p\}$ est de taille p.

Considérons l'énoncé $\mathcal{E}_p$ suivant :

l'arbre de Huffman A_p pour $\mathcal{V} = \{v_1, \ldots, v_p\}$ construit par l'algorithme de Huffman $\mathcal{H}$ est minimal.

. $\mathcal{E}_1$ est vrai car il n'y a qu'un seul arbre de Huffman pour $\mathcal{V} = \{v_1\}$ (réduit à une feuille), donc minimal puisqu'unique.

. Supposons $\mathcal{E}_{p-1}$ vrai, c'est à dire A_{p-1} minimal.

A_p désignant l'arbre de Huffman obtenu par $\mathcal{H}$ pour $\mathcal{V} = \{v_1, \ldots, v_p\}$, il existe un arbre de Huffman H_p de coût minimal pour $\mathcal{V}$. Dans H_p, deux feuilles situées à la profondeur maximale h contiennent des caractères v_i et v_j d'occurence minimale d'après **III.9**.

Si dans H_p, on échange des feuilles situées à la hauteur maximale h, on ne modifie pas le coût de H_p. On peut donc supposer que dans H_p, les feuilles v_i et v_j ont le même père. L'arbre H'_p de taille $p-1$ obtenu en enlevant les deux feuilles v_i et v_j et en les remplaçant par leur père n d'occurence $\mathcal{O}(n) = \mathcal{O}(v_i) + \mathcal{O}(v_j)$ est d'après **III-10** minimal pour $\mathcal{V}' \cup \{n\}$.

Donc H'_p a d'après l'hypothèse de récurrence le même coût que l'arbre de Huffman A_{p-1} produit à partir de A_p en remplaçant dans A_p les deux feuilles v_i et v_j par leur père n.

Il en résulte que $\mathcal{C}(A_p) = C(A_{p-1}) + \mathcal{O}(v_i) + \mathcal{O}(v_j) = C(H'_p) + \mathcal{O}(v_i) + \mathcal{O}(v_j) = C(H_p)$, donc H_p est minimal.

Question III.12 Fonction `comparer : arbre -> arbre -> bool`

```
let comparer a1 a2 = (nombre a1) <= (nombre a2) ;;
```

Question III.13 Fonction `inserer : arbre -> arbre list -> arbre list`

```
let rec inserer a (l:liste) = match l with
       |  []  -> [a]
       | b::r -> if comparer a b then a::l else b::(inserer a r) ;;
```

Question III.14 Fonction `trier : arbre list -> arbre list`

```
let rec trier (l:liste) = match l with
       |  []  -> []
       | b::r -> inserer b (trier r) ;;
```

Question III.15 Fonction `ajouter : char -> arbre list -> arbre list`

```
let rec ajouter v (l:liste) = match l with
       |             []  -> [Feuille(v,1)]
       | Feuille(u,n)::q -> if u=v then Feuille(u,n+1)::q
                                   else Feuille(u,n)::(ajouter v q)
       |      _          -> failwith "Erreur dans ajouter" ;;
```

Question III.16 Fonction `compter : char list -> arbre list`

```
let rec compter (s:suite) = match s with
       |  []   -> []
       | x::q  -> ajouter x (compter q) ;;
```

Question III.17 Fonction `fusionner : liste -> arbre`

```
let rec fusionner (l:liste)  =  match l with
      |    []    -> Vide
      |   [a]    -> a
      | a::b::q  -> fusionner (inserer Noeud(a,b) q) ;;
```

Question III.18 La fonction auxiliaire `aux li` reçoit une liste `li` d'arbres de Huffman triée par occurrences croissantes et remplace les deux premiers `a` et `b` par l'arbre de Huffman `n` ayant pour fils `a` et `b`, puis insère ce nouvel arbre dans la queue de la liste.
Les appels récursifs s'arrêtent lorsqu'il n'y a plus qu'un arbre dans la liste : c'est alors l'arbre de Huffman minimal souhaité.

Question III.19 Fonction `huffman : suite -> arbre`

```
let huffman (s:suite) = fusionner(trier(compter s)) ;;
```

Question III.20 Fonction `construire : arbre -> (char * boolean list) list`

```
let  construire  a =
  let rec aux aa li accu = match aa with
       |      Vide    -> failwith "Erreur dans construire"
       | Feuille(v,_) -> (v,rev(li))::accu
       | Noeud(ag,ad) -> (aux ag (O::li) accu)@(aux ad (I::li) accu)
in aux a [] [] ;;
```

Question III.21 Fonction `reconstruire : cle -> arbre`

```
let reconstruire (c:cle) =
  let rec inclure (x,li) a = match (li,a) with
         |   ([],Vide)         -> Feuille(x,0)
         | (t::q,Noeud(ag,ad)) -> if t=0 then Noeud(inclure (x,q) ag, ad)
                                         else Noeud(ag, inclure (x,q) ad)
         | (t::q,aa)           -> if t=0 then Noeud(inclure (x,q) Vide,aa)
                                         else Noeud(aa,inclure (x,q) Vide)
         |      _              -> failwith "Erreur dans reconstruire"

  and  parcourir cc = match cc with
         |       []  ->    Vide
         | (v,li)::r -> inclure(v,li) (parcourir r)
  in parcourir c ;;
```

Question III.22 Fonction `decoder : code -> arbre -> char list`

```
let decoder (s:code) a =
   let rec parcourir ss aa = match (ss,aa) with
          |     ([],Feuille(v,n)) -> [v]     (* x est le dernier caractere *)
          | ((t::q),Feuille(v,n)) -> v::(parcourir ss a)
          | ((t::q),Noeud(ag,ad)) -> if t=0 then parcourir q ag
                                            else parcourir q ad
          |             _         -> failwith "Erreur dans decoder"
   in parcourir s a ;;
```

Question III.23 Il s'agit de la fonction `nombre` qui est appelée deux fois par la fonction `compare` elle-même appelée par la fonction `inserer`.

Cette dernière fonction est appelée par $\frac{p\,(p-1)}{2}$ fois dans le pire des cas par la fonction de tri par insertion et aussi par la fonction `fusionner`.

Question III.24

Pour réduire le coût des appels de la fonction `nombre`, on pourrait modifier le type arbre de la façon suivante :

```
type arbre = Vide | Feuille of (char * int) | Noeud of (int * arbre * arbre) ;;
```

Ainsi on stocke en plus dans chaque nœud interne le nombre de feuilles du sous-arbre dont il est la racine.
Ceci compliquerait assez peu la fonction `huffman`, mais par contre la fonction `nombre` devient simple car on lit immédiatement à la racine l'occurence de l'arbre sans le parcourir entièrement, ce qui constitue un gain important pour les appels de la fonction `inserer` par la fonction `fusionner`.

D'autre part, on peut limiter le nombre d'appels de la fonction `comparer` en effectuant un tri dichomique en $O(p \ln p)$ plutôt qu'un tri par insertion. Cependant, comme on manipule des listes et non pas des vecteurs, ce n'est pas pratique lorsqu'on veut couper une liste en deux moitiés de longueur presque égales.

Question III.25 Fonction `ajouter2 : char -> liste -> arbre list`

```
let ajouter2 v (l:liste) =
     let rec aux ll = match ll with
        |             []  -> (false,[])
        | Feuille(u,n)::q -> if u=v then (true,inserer (Feuille(u,n+1)) q)
                                    else let(ok,li) = aux q
                                        in (ok,Feuille(u,n)::li)
        |      _          -> failwith "Erreur dans ajouter2"
    in let (ok,li)= aux l
       in if ok then li else Feuille(v,1)::l ;;
```

Fin du corrigé