DS 4 : CCINP 2014 option MP - corrigé

I. Logique et calcul des propositions

Question 1. Pour tout $i \in [\![1;n]\!]$, l'assertion D_i prononcée par le membre numéro i est vraie si et seulement si ce membre dit la vérité (ce qui est marqué par le variable propositionnelle M_i).

On a donc : $\forall i \in [\![1;n]\!] \quad D_i \leftrightarrow M_i$

On peut donc écrire : $\varphi = \bigwedge_{i=1}^{n} (D_i \leftrightarrow M_i)$

Question 2.

- $D_A = A \wedge \neg B \wedge C$
- $D_B = \neg C$
- $D_C = (B \vee \neg A) \wedge \neg(B \wedge \neg A)$ (il s'agit du « ou exclusif » entre A et $\neg B$)

Question 3.

A	B	C	D_A	$A \leftrightarrow D_A$	D_B	$B \leftrightarrow D_B$	D_C	$C \leftrightarrow D_C$	φ
0	0	0	0	1	1	0	1	0	0
0	0	1	0	1	0	1	1	1	1
0	1	0	0	1	1	1	0	1	1
0	1	1	0	1	0	0	0	0	0
1	0	0	0	1	1	0	0	1	0
1	0	1	1	1	0	1	0	0	0
1	1	0	0	1	1	1	1	0	0
1	1	1	0	1	0	0	1	1	0

On n'arrive pas à conclure complètement. Les informations dont on dispose permettent seulement de dire que A ment et que exactement un des deux autres dit la vérité. (Peut-être y avait-il une erreur dans l'énoncé.)

Question 4.

- $D_D = \neg F$
- $D_E = D \wedge F$
- $D_F = E$

Question 5. On remarquera que, pour toutes formules P et Q, on a : $(P \leftrightarrow Q) \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q)$

On a donc :

$$\begin{aligned}
\varphi' &\equiv (D \leftrightarrow D_D) \wedge (E \leftrightarrow D_E) \wedge (F \leftrightarrow D_F) \\
&\equiv [(D \wedge D_D) \vee (\neg D \wedge \neg D_D)] \wedge [(E \wedge D_E) \vee (\neg E \wedge \neg D_E)] \wedge [(F \wedge D_F) \vee (\neg F \wedge \neg D_F)] \\
&\equiv [(D \wedge \neg F) \vee (\neg D \wedge F)] \wedge [(E \wedge D \wedge F) \vee (\neg E \wedge \neg(D \wedge F))] \wedge [(F \wedge E) \vee (\neg F \wedge \neg E)] \\
&\equiv [(D \wedge \neg F) \vee (\neg D \wedge F)] \wedge [(E \wedge D \wedge F) \vee (\neg E \wedge (\neg D \vee \neg F))] \wedge [(F \wedge E) \vee (\neg F \wedge \neg E)] \qquad \text{(loi de De Morgan)} \\
&\equiv [(D \wedge \neg F) \vee (\neg D \wedge F)] \wedge [(E \wedge D \wedge F) \vee (\neg E \wedge \neg D) \vee (\neg E \wedge \neg F)] \wedge [(F \wedge E) \vee (\neg F \wedge \neg E)] \qquad \text{(distributivité)}
\end{aligned}$$

On développe ensuite la conjonction des deux premières parenthèses (en ne gardant que les termes qui ne s'annulent pas). On obtient :

$$\varphi' \equiv [(D \wedge \neg F \wedge \neg E) \vee (\neg D \wedge F \wedge \neg E)] \vee [(F \wedge E) \vee (\neg F \wedge \neg E)]$$

On fait de même pour les deux parenthèses restantes pour trouver :

$$\varphi' \equiv D \wedge \neg E \wedge \neg F$$

Conclusion : D dit la vérité alors que E et F mentent.

II. Automates et langages

Question 1. Attention : pour montrer que $\hat{h}$ est un homomorphisme de langage ε-libre, il faut bien montrer que $\hat{h}$ est un homomorphisme de langage. Ensuite, on montrera qu'il est ε-libre.

- $\hat{h}$ **est un homomorphisme de langage.** On doit montrer que $\hat{h}$ vérifie la définition II.2.
  - $\hat{h}(\varepsilon) = \varepsilon$ par définition.
  - On doit montrer que $\forall m_1, m_2 \in X^*, \hat{h}(m_1 \cdot m_2) = \hat{h}(m_1) \cdot h(m_2)$

    Soit m_1 et $m_2 \in X^*$. Comme $m_1 \in X^*$, il existe $n \in \mathbb{N}$ tel que $m_1 \in X^n$. Ainsi, si on montre que :

    $$\forall m_1 \in X^n, m_2 \in X^*, \hat{h}(m_1 \cdot m_2) = \hat{h}(m_1) \cdot \hat{h}(m_2)$$

    alors c'est gagné. On va montrer cette propriété par récurrence sur n.
    - Initialisation : $n = 0$. Alors $m_1 \in X^0 = \{\varepsilon\}$, donc $m_1 = \varepsilon$. Ainsi :

      $$\begin{aligned}\hat{h}(m_1 \cdot m_2) &= \hat{h}(\varepsilon \cdot m_2)\\ &= \hat{h}(m_2)\\ &= \hat{h}(\varepsilon) \cdot \hat{h}(m_2) \quad (\text{car } \hat{h}(\varepsilon) = \varepsilon)\\ &= \hat{h}(m_1) \cdot \hat{h}(m_2)\end{aligned}$$

    - Hérédité : soit $n > 0$, supposons la propriété vraie au rang n, montrons qu'elle est vraie au rang $n+1$.

      Soit $m_1 \in X^{n+1}$, alors il existe $x \in X$ et $m_1' \in X^n$ tels que $m_1 = x \cdot m_1'$. Ainsi :

      $$\begin{aligned}\hat{h}(m_1 \cdot m_2) &= \hat{h}(x \cdot m_1' \cdot m_2)\\ &= h(x) \cdot \hat{h}(m_1' \cdot m_2) && \text{par définition de } \hat{h}\\ &= h(x) \cdot \hat{h}(m_1') \cdot \hat{h}(m_2) && \text{par hypothèse de récurrence}\\ &= \hat{h}(x \cdot m_1') \cdot \hat{h}(m_2) && \text{par définition de } \hat{h}\\ &= \hat{h}(m_1) \cdot \hat{h}(m_2)\end{aligned}$$

      Ce qui achève la récurrence.

      (Remarque : c'est la première récurrence de ce devoir, je l'ai beaucoup détaillée, les suivantes seront plus succintes).
- $\hat{h}$ **est** ε**-libre.** Soit $m \in X^*, m \neq \varepsilon$, montrons que $\hat{h}(m) \neq \varepsilon$. Comme $m \neq \varepsilon$, alors il existe $n > 0$ tel que $m \in X^n$, et donc il existe $x \in X$ et $m' \in X^{n-1}$. On a alors $\hat{h}(m) = \hat{h}(x \cdot m') = h(x) \cdot \hat{h}(m')$. Or, $h(x) \neq \varepsilon$, car h est de co-domaine $Y^* \setminus \{\varepsilon\}$ et $h(x)$ est donc de longueur au moins 1. De même pour $\hat{h}(m)$, d'où finalement $\hat{h}(m) \neq \varepsilon$.

Question 2. Montrons que $\widehat{h_{|X}} = h$. Remarquons d'abord que $h_{|X}$ est une application qui vérifie l'hypothèse de la question précédente, et donc $\widehat{h_{|X}}$ est bien un homomorphisme de langage de domaine X^* et de co-domaine Y^* (tout comme h lui-même).

On doit donc montrer que $\forall m \in X^*, \widehat{h_{|X}}(m) = h(m)$. Comme pour la question précédente, on va montrer par récurrence sur n que $\forall n \in \mathbb{N}, \forall m \in X^n, \widehat{h_{|X}}(m) = h(m)$.

- Initialisation : $n = 0$. Soit $m \in X^0$, donc $m = \varepsilon$. On a bien : $\widehat{h_{|X}}(\varepsilon) = \varepsilon = h(\varepsilon)$.
- Hérédité : soit $n > 0$, supposons la propriété vraie au rang n, montrons qu'elle est vraie au rang $n+1$.

  Soit $m \in X^{n+1}$, alors $m = x \cdot m'$ avec $x \in X$ et $m' \in X^n$. On a alors :

  $$\begin{aligned}\widehat{h_{|X}}(m) &= \widehat{h_{|X}}(x \cdot m')\\ &= h(x) \cdot \widehat{h_{|X}}(m') && \text{par définition de } \widehat{h_{|X}}\\ &= h(x) \cdot h(m') && \text{par hypothèse de récurrence}\\ &= h(x \cdot m') && \text{car } h \text{ est un homomorphisme}\\ &= h(m)\end{aligned}$$

  Ce qui achève la récurrence.

Question 3. On a $e = a(ba \mid aba)^*$. Ainsi :

$$\begin{aligned}
\tilde{h}(e) &= \tilde{h}(a(ba \mid aba)^*) \\
&= \tilde{h}(a)\tilde{h}((ba \mid aba)^*) \\
&= \tilde{h}(a)\big(\tilde{h}(ba \mid aba)\big)^* \\
&= \tilde{h}(a)\big(\tilde{h}(ba) \mid \tilde{h}(aba)\big)^* \\
&= \tilde{h}(a)\big(\tilde{h}(b)\tilde{h}(a) \mid \tilde{h}(a)\tilde{h}(b)\tilde{h}(a)\big)^* \\
&= h(a)(h(b)h(a) \mid h(a)h(b)h(a))^* \\
&= 10(010 \mid 10010)^*
\end{aligned}$$

Question 4. Montrons que pour toute expression régulière e sur l'alphabet X, alors $\hat{h}(e)$ est une expression régulière sur Y. On montre la propriété **par induction** sur e.

- Si $e = \emptyset$, alors $\hat{h}(\emptyset) = \emptyset$ est bien une expression régulière sur Y.
- Si $e = \varepsilon$, alors $\hat{h}(\varepsilon) = \varepsilon$ est bien une expression régulière sur Y.
- Si $e = a$ avec $a \in X$, alors $\hat{h}(a) = h(a) \in Y^*$. Il s'agit donc d'un mot fini sur l'alphabet Y qu'on peut interpréter comme une expression régulière sur Y.
- Si $e = e_1 \mid e_2$, avec e_1 et e_2 deux expressions régulières dont on suppose qu'elle vérifient l'hypothèse d'induction : $\hat{h}(e_1)$ et $\hat{h}(e_2)$ sont des expressions régulière sur Y. Alors $\hat{h}(e) = \hat{h}(e_1 \mid e_2) = \hat{h}(e_1) \mid \hat{h}(e_2)$ est donc une expression régulière sur Y comme union de deux expressions régulières sur Y.
- Le cas $e = e_1 \cdot e_2$ est similaire.
- Le cas $e = e_1^*$ est similaire.

Question 5. Comme pour la question précédente, on raisonne par induction sur e.

- Si $e = \emptyset$, alors :
  - $\mathcal{L}\big(\tilde{h}(\emptyset)\big) = \mathcal{L}(\emptyset) = \{\}$ (l'ensemble vide est noté $\{\}$ pour le distinguer de l'expression régulière vide $\emptyset$),
  - $\hat{h}(\mathcal{L}(\emptyset)) = \hat{h}(\{\}) = \big\{\hat{h}(m) \mid m \in \{\}\big\} = \{\}$

  d'où l'égalité.
- Si $e = \varepsilon$, alors :
  - $\mathcal{L}\big(\tilde{h}(\varepsilon)\big) = \mathcal{L}(\varepsilon) = \{\varepsilon\}$
  - $\hat{h}(\mathcal{L}(\varepsilon)) = \hat{h}(\{\varepsilon\}) = \big\{\hat{h}(m) \mid m \in \{\varepsilon\}\big\} = \big\{\hat{h}(\varepsilon)\big\} = \{\varepsilon\}$

  d'où l'égalité.
- Si $e = a$ avec $a \in X$, alors :
  - $\mathcal{L}\big(\tilde{h}(a)\big) = \mathcal{L}(h(a)) = \{h(a)\}$
  - $\hat{h}(\mathcal{L}(a)) = \hat{h}(\{a\}) = \big\{\hat{h}(m) \mid m \in \{a\}\big\} = \big\{\hat{h}(a)\big\} = \big\{\hat{h}(a \cdot \varepsilon)\big\} = \big\{h(a) \cdot \hat{h}(\varepsilon)\big\} = \{h(a)\}$

  d'où l'égalité.
- Si $e = e_1 \mid e_2$, avec e_1 et e_2 deux expressions régulières dont on suppose qu'elle vérifient l'hypothèse d'induction : $\mathcal{L}\big(\tilde{h}(e_1)\big) = \hat{h}(\mathcal{L}(e_1))$ et $\mathcal{L}\big(\tilde{h}(e_2)\big) = \hat{h}(\mathcal{L}(e_2))$. On a alors :

$$\begin{aligned}
\mathcal{L}\big(\tilde{h}(e)\big) &= \mathcal{L}\big(\tilde{h}(e_1 \mid e_2)\big) \\
&= \mathcal{L}\big(\tilde{h}(e_1) \mid \tilde{h}(e_2)\big) && \text{définition de } \tilde{h} \\
&= \mathcal{L}\big(\tilde{h}(e_1)\big) \cup \mathcal{L}\big(\tilde{h}(e_2)\big) && \text{par définition de } \mathcal{L} \\
&= \hat{h}(\mathcal{L}(e_1)) \cup \hat{h}(\mathcal{L}(e_2)) && \text{par hypothèse d'induction}
\end{aligned}$$

  D'autre part, on a :

$$\begin{aligned}
\hat{h}(\mathcal{L}(e)) &= \hat{h}(\mathcal{L}(e_1 \mid e_2)) \\
&= \hat{h}(\mathcal{L}(e_1) \cup \mathcal{L}(e_2)) && \text{par définition de } \mathcal{L} \\
&= \hat{h}(\mathcal{L}(e_1)) \cup \hat{h}(\mathcal{L}(e_2)) && \text{image d'union par une fonction}
\end{aligned}$$

  d'où l'égalité.
- Si $e = e_1 \cdot e_2$, avec e_1 et e_2 deux expressions régulières dont on suppose qu'elle vérifient l'hypothèse d'induction : $\mathcal{L}\big(\tilde{h}(e_1)\big) = \hat{h}(\mathcal{L}(e_1))$ et $\mathcal{L}\big(\tilde{h}(e_2)\big) = \hat{h}(\mathcal{L}(e_2))$. Pour conduire la preuve, on a besoin d'un résultat intermédiaire :

Lemme 1. Soit h un homomorphisme de langages de domaine X et de co-domaine Y. Soient L_1 et L_2 deux langages sur X (L_1 et $L_2 \subset X^*$). On a alors $h(L_1 \cdot L_2) = h(L_1) \cdot h(L_2)$

Preuve : du lemme ci-dessus. On procède par double inclusion.

- Soit $m \in h(L_1 \cdot L_2)$, il existe donc $w \in L_1 \cdot L_2$ tel que $m = h(w)$, et donc il existe $w_1 \in L_1$ et $w_2 \in L_2$ tel que $m = h(w_1 \cdot w_2)$. Par définition d'un homomorphisme de langage, on a donc $m = h(w_1) \cdot h(w_2)$. Or, par définition, $h(w_1) \in h(L_1)$ et $h(w_2) \in h(L_2)$, d'où $m \in h(L_1) \cdot h(L_2)$, ce qu'il fallait démontrer.
- Réciproquement, soit $m \in h(L_1) \cdot h(L_2)$, alors il existe $m_1 \in h(L_1)$ et $m_2 \in h(L_2)$ tels que $m = m_1 \cdot m_2$. Mais alors, il existe $w_1 \in L_1$ tel que $m_1 = h(w_1)$ et il existe $w_2 \in L_2$ tel que $m_2 = h(w_2)$. On a donc : $m = h(w_1) \cdot h(w_2)$, et donc $m = h(w_1 \cdot w_2)$ par définition d'un homomorphisme de langage. Or, $w_1 \cdot w_2 \in L_1 \cdot L_2$, d'où $m \in h(L_1 \cdot L_2)$.

□

On revient à la preuve ; on a alors :

$$\begin{aligned}
\mathcal{L}(\tilde{h}(e)) &= \mathcal{L}(\tilde{h}(e_1 \cdot e_2)) \\
&= \mathcal{L}(\tilde{h}(e_1) \cdot \tilde{h}(e_2)) && \text{définition de } \tilde{h} \\
&= \mathcal{L}(\tilde{h}(e_1)) \cdot \mathcal{L}(\tilde{h}(e_2)) && \text{par définition de } \mathcal{L} \\
&= \hat{h}(\mathcal{L}(e_1)) \cdot \hat{h}(\mathcal{L}(e_2)) && \text{par hypothèse d'induction} \\
&= \hat{h}(\mathcal{L}(e_1) \cdot \mathcal{L}(e_2)) && \text{par le lemme ci-dessus} \\
&= \hat{h}(\mathcal{L}(e_1 \cdot e_2)) && \text{par définition de } \mathcal{L} \\
&= \hat{h}(\mathcal{L}(e))
\end{aligned}$$

d'où l'égalité.

- Si $e = e_1^*$, avec e_1 une expression régulière dont on suppose qu'elle vérifie l'hypothèse d'induction : $\mathcal{L}(\tilde{h}(e_1)) = \hat{h}(\mathcal{L}(e_1))$. On a comme pour le cas précédent besoin d'un lemme intermédiaire similaire :

Lemme 2. Soit h un homomorphisme de langages de domaine X et de co-domaine Y. Soient L un langage sur X ($L \subset X^*$). On a alors $h(L^*) = h(L)^*$

Preuve : du lemme ci-dessus. On procède par double inclusion.

- Soit $m \in h(L^*)$, il existe donc $w \in L^*$ tel que $m = h(w)$, et donc il existe $w_1, w_2, ..., w_n \in L$ (avec $n \in \mathbb{N}$) tels que $m = h(w_1 \cdot w_2 \cdot ... \cdot w_n)$. Par définition d'un homomorphisme de langage, on a donc $m = h(w_1) \cdot h(w_2) \cdot ... \cdot h(w_n)$. Or, par définition, les $h(w_i) \in h(L)$, d'où $m \in h(L) \cdot h(L) \cdot ... \cdot h(L) = h(L)^n \subset h(L)^*$.
- Réciproquement, soit $m \in h(L)^*$, alors il existe $m_1, m_2, ..., m_n \in h(L)$ (avec $n \in \mathbb{N}$) tels que $m = m_1 \cdot m_2 \cdot ... \cdot m_n$. Mais alors, il existe $w_1, w_2, ..., w_n \in L$ tel que $m_1 = h(w_1)$, $m_2 = h(w_2)$, ..., $m_n = h(w_n)$. On a donc : $m = h(w_1) \cdot h(w_2) \cdot ... \cdot h(w_n)$, et donc $m = h(w_1 \cdot w_2 \cdot ... \cdot w_n)$ par définition d'un homomorphisme de langage. Or, $w_1 \cdot w_2 \cdot ... \cdot w_n \in L^n \subset L^*$, d'où $m \in h(L^*)$.

□

On revient à la preuve ; on a alors :

$$\begin{aligned}
\mathcal{L}(\tilde{h}(e)) &= \mathcal{L}(\tilde{h}(e_1^*)) \\
&= \mathcal{L}(\tilde{h}(e_1)^*) && \text{définition de } \tilde{h} \\
&= \mathcal{L}(\tilde{h}(e_1))^* && \text{par définition de } \mathcal{L} \\
&= \hat{h}(\mathcal{L}(e_1))^* && \text{par hypothèse d'induction} \\
&= \hat{h}(\mathcal{L}(e_1)^*) && \text{par le lemme ci-dessus} \\
&= \hat{h}(\mathcal{L}(e_1^*)) && \text{par définition de } \mathcal{L} \\
&= \hat{h}(\mathcal{L}(e))
\end{aligned}$$

Ce qui achève (enfin) la preuve par induction.

*Remarque : cette preuve est **très** longue. J'imagine qu'on aurait pu balayer la preuve des deux lemmes intermédaires comme une conséquence assez directe de la définition d'un homomorphisme de langage.*

Question 6. Avec les notations de l'énoncé, soit L_X un langage régulier sur X. Il existe donc une expression régulière e sur l'alphabet X telle que $L_X = \mathcal{L}(e)$.

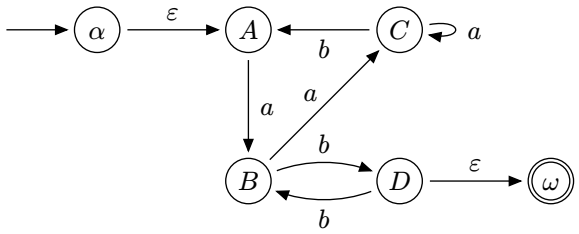
Remarquons par ailleurs que d'après la question 2, on a $h = \widehat{h_{|X}}$.

Ainsi, on a : $h(L_X) = h(\mathcal{L}(e)) = \widehat{h_{|X}}(\mathcal{L}(e)) = \mathcal{L}\left(\widetilde{h_{|X}}(e)\right)$ d'après la question 5.

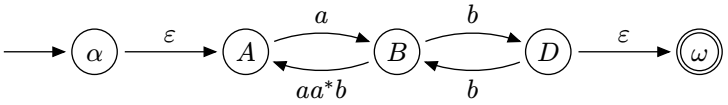
La question 4 permet de conclure, car on en déduit que $\widetilde{h_{|X}}(e)$ est une expression régulière sur Y. Cette expression dénote un langage régulier sur Y. On a alors : $h(L_X) = L_Y$.

Question 7. Cette question ne demande pas de justification, mais ce corrigé détaille le calcul de l'expression régulière par la méthode de l'élimination des états.

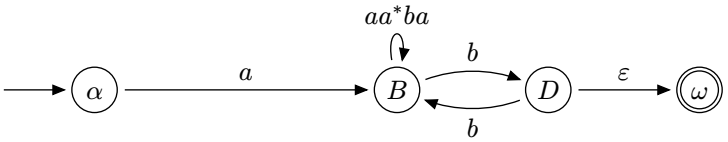
On normalise d'abord l'automate, en insérant un unique état initial α n'ayant aucun transition entrante, et un unique état acceptant n'ayant aucune transition sortante. On obtient :



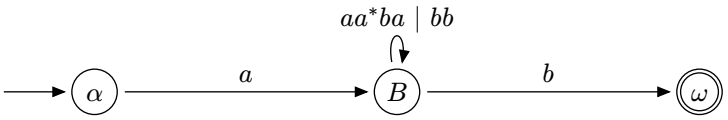
On élimine ensuite les états un par un. On commence par l'état C :



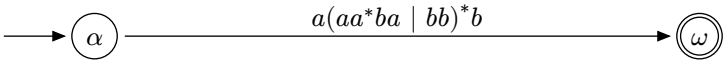
Ensuite on élimine l'état A :



Ensuite on élimine l'état D :



Finalement, on obtient en éliminant B :



D'où l'expression $e = a(aa^*ba \mid bb)^*b$ qui dénote le langage reconnu par l'automate $\mathcal{E}$.

Remarque : choisir un ordre d'élimination différent peut mener à une expression régulière différente (mais équivalente !)

Question 8. Soit $o, d \in Q$, soit $m_1, m_2 \in X^*$. On a donc $m_1 \in X^n$ avec $n \in \mathbb{N}$. On raisonne par récurrence sur n.

- Initialisation : $n = 0$. $m_1 \in X^0$, donc $m_1 = \varepsilon$. On montre les deux implications :
  - Si $d \in \delta^*(o, m_1 \cdot m_2) = \delta^*(o, m_2)$. Par ailleurs on a $\delta^*(o, \varepsilon) = \{o\}$ par définition.
    Ainsi, en posant $q = o$, on a bien $q \in \delta^*(o, \varepsilon)$ et $d \in \delta^*(q, m_2)$, d'où l'existance demandée.
  - Si il existe $q \in Q$ tel que $q \in \delta^*(o, m_1)$ et $d \in \delta^*(q, m_2)$, comme $m_1 = \varepsilon$, on a $\delta^*(o, m_1) = \{o\}$ et donc nécessairement $q = o$. On a donc bien $d \in \delta^*(o, m_1 \cdot m_2)$.
- Hérédité : soit $n > 0$, supposons la propriété vraie au rang n, montrons qu'elle est vraie au rang $n + 1$. On a donc l'hypothèse de récurrence suivante :

$$\forall o, d \in Q, \forall m_1 \in X^n, \forall m_2 \in X^*, d \in \delta^*(o, m_1 \cdot m_2) \Leftrightarrow \exists q \in Q, (q \in \delta^*(o, m_1) \wedge d \in \delta^*(q, m_2))$$

  Soit $m_1 \in X^{n+1}$, $m_2 \in X^*$. Ainsi $m_1 = x \cdot m_1'$ avec $x \in X$ et $m_1' \in X^n$. On raisonne alors par équivalence :

$$\begin{aligned} d \in \delta^*(o, m_1 \cdot m_2) &\Leftrightarrow d \in \delta^*(o, x \cdot m_1' \cdot m_2) \\ &\Leftrightarrow \exists q \in Q, (q \in \delta^*(o,x) \wedge d \in \delta^*(q, m_1' \cdot m_2)) && \text{par définition de } \delta^* \\ &\Leftrightarrow \exists q, q' \in Q, (q \in \delta^*(o,x) \wedge q' \in \delta^*(q, m_1') \wedge d \in \delta^*(q', m_2)) && \text{par hypothèse de récurrence} \\ &\Leftrightarrow \exists q' \in Q, (q' \in \delta^*(o, x \cdot m_1') \wedge d \in \delta^*(q', m_2)) && \text{par définition de } \delta^* \\ &\Leftrightarrow \exists q' \in Q, (q' \in \delta^*(o, m_1) \wedge d \in \delta^*(q', m_2)) \end{aligned}$$

Ce qui est précisemment ce qu'on voulait démontrer, et achève la récurrence.

Question 9. Il faut bien faire attention à appliquer la définition de façon méticuleuse. Soyons particulièrement attentif au fait que l'énoncé a la mauvais idée d'intervertir les rôles de X et Y entre la définition de l'image réciproque d'un automate et la question...

On a donc $X = \{a, b\}$, $Y = \{0, 1\}$, et $h : Y \to X^*$, vérifiant $h(0) = ab$ et $h(1) = b$.

L'automate $\mathcal{E}$ est défini par $\mathcal{E} = (Q, X, I, T, \delta)$ par l'énoncé.

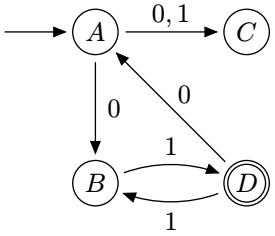
On a alors $\mathcal{B} = \hat{h}^{-1}(\mathcal{E})$ défini par $\mathcal{B} = (Q, Y, I, T, \delta_{\hat{h}^{-1}})$, on garde donc les même états, états initiaux et acceptant. Les transitions sont étiquetées par $Y = \{0, 1\}$; reste à déterminer la fonction de transition $\delta_{\hat{h}^{-1}}$.

Par définition, on ajoute à $\mathcal{B}$ une transition de la forme $o \xrightarrow{x} d$ si il existe un **chemin** $d \xrightarrow{\hat{h}(x)}{}^* o$ dans $\mathcal{E}$. Ainsi, pour $d, o \in Q$:
- on ajoute la transition $o \xrightarrow{0} d$ à la condition que $d \xrightarrow{ab}{}^* o$ dans $\mathcal{E}$.
- on ajoute la transition $o \xrightarrow{1} d$ à la condition que $d \xrightarrow{b}{}^* o$ dans $\mathcal{E}$.

Par lecture de l'automate $\mathcal{E}$, on observe que :
- $B \xrightarrow{b}{}^* D$, $D \xrightarrow{b}{}^* B$ et $C \xrightarrow{b}{}^* A$
- $A \xrightarrow{ab}{}^* D$, $B \xrightarrow{ab}{}^* A$ et $C \xrightarrow{ab}{}^* A$

Ce qui donne dans $\mathcal{B}$ les transitions suivantes :
- $D \xrightarrow{1} B$, $B \xrightarrow{1} D$ et $A \xrightarrow{1} C$
- $D \xrightarrow{0} A$, $A \xrightarrow{0} B$ et $A \xrightarrow{0} C$

Finalement, on obtient l'automate $\mathcal{B}$:

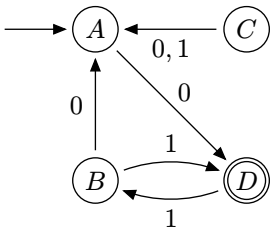


En répondant aux questions suivantes, on peut constater que l'énoncé contient une erreur : la définition de l'image réciproque d'automate devrait être :

$$\forall x \in X, \forall o \in Q, \forall d \in Q, d \in \delta_{\hat{h}^{-1}}(o, x) \Leftrightarrow d \in \delta^*\left(o, \hat{h}(x)\right)$$

(inversion entre les états o et d à droite de l'équivalence). Sans cette correction, impossible de répondre aux questions suivantes...

Cela revient simplement à inverser les transitions que nous avons établies précedemment, on considère donc pour la suite que l'automate $\hat{h}^{-1}(\mathcal{E})$ obtenu est :



Question 10. La méthode d'élimination des états donne l'expression $e = 0(11 \mid 100)^*$.

On a alors l'expression $\tilde{h}(e) = ab(bb \mid babab)^*$

Comparons alors le langage reconnu par $\tilde{h}(e)$: $\mathcal{L}\left(\tilde{h}(e)\right)$, et celui reconnu par l'automate $\mathcal{E}$.

On remarque facilement que tout mot dénoté par $\tilde{h}(e)$ est accepté par l'automate : informellement, en lisant le préfixe ab dans $\mathcal{E}$, on arrive sur l'état D de façon déterministe. Puis, lire le bb ou le mot $babab$ ramène de nouveau sur ce même état D de façon déterministe. Quel que soit le nombre de répétition, le mot est donc accepté. On a donc :

$$\mathcal{L}(\tilde{h}(e)) = \hat{h}(\mathcal{L}(e)) = \hat{h}\Big(\mathcal{L}\big(\hat{h}^{-1}(\mathcal{E})\big)\Big) \subset \mathcal{L}(\mathcal{E})$$

(notons un léger abus de notation de l'énoncé ici, c'est bien $\hat{h}\Big(\mathcal{L}\big(\hat{h}^{-1}(\mathcal{E})\big)\Big)$ qui nous intéresse.)

On n'a pas l'inclusion inverse : en effet le mot $aabab$ est accepté par l'automate $\mathcal{E}$, mais pas par l'expression (le mot ne commence pas par ab).

Remarque : ce n'est pas demandé, mais le langage reconnu par $\hat{h}\big(\hat{h}^{-1}(\mathcal{E})\big)$ est en fait le langage reconnu par $\mathcal{E}$, restreint aux mots m qui peuvent s'écrire de la forme $h(m')$ avec $m' \in Y$.

Question 11. L'équivalence demandée est la même que celle de la définition de l'image réciproque d'un automate, généralisée à un mot $m \in X^*$ plutôt qu'un symbole $x \in X$. (*Remarque : c'est en voyant cette équivalence qu'on peut se douter de l'erreur d'énoncé évoquée plus tôt.*)

Soit $m \in X^*$, donc $m \in X^n$ avec $n \in \mathbb{N}$. On va encore une fois raisonner par récurrence sur n.

- Initialisation : $n = 0$. On a donc $m = \varepsilon$. On a alors $\hat{h}(\varepsilon) = \varepsilon$ par définition de $\hat{h}$.

  Ainsi : $\delta^*_{\hat{h}^{-1}}(o, \varepsilon) = \{o\}$ et $\delta^*\big(o, \hat{h}(\varepsilon)\big) = \delta^*(o, \varepsilon) = \{o\}$, par définition de la fermeture réflexive et transitive (déf 11) dans les deux cas. D'où l'équivalence.
- Hérédité : soit $n > 0$, supposons la propriété vraie au rang n, montrons qu'elle est vraie au rang $n+1$.

  On a donc $m \in X^{n+1}$, ainsi $m = x \cdot m'$ avec $x \in X$ et $m' \in X^n$. On raisonne par équivalence.

$$\begin{aligned}
d \in \delta^*_{\hat{h}^{-1}}(o, m) &\Leftrightarrow d \in \delta^*_{\hat{h}^{-1}}(o, x \cdot m') \\
&\Leftrightarrow \exists q \in Q, \big(q \in \delta_{\hat{h}^{-1}}(o, x) \wedge d \in \delta^*_{\hat{h}^{-1}}(q, m')\big) && \text{par définition de l'étoile } * \text{ de } \delta_{\hat{h}^{-1}} \\
&\Leftrightarrow \exists q \in Q, \Big(q \in \delta^*\big(o, \hat{h}(x)\big) \wedge d \in \delta^*_{\hat{h}^{-1}}(q, m')\Big) && \text{par définition (corrigée) de } \delta_{\hat{h}^{-1}} \\
&\Leftrightarrow \exists q \in Q, \Big(q \in \delta^*\big(o, \hat{h}(x)\big) \wedge d \in \delta^*\big(q, \hat{h}(m')\big)\Big) && \text{par hypothèse de récurrence} \\
&\Leftrightarrow d \in \delta^*\big(o, \hat{h}(x) \cdot \hat{h}(m')\big) && \text{par la question 8} \\
&\Leftrightarrow d \in \delta^*\big(o, \hat{h}(x \cdot m')\big) && \text{par définition de } \hat{h} \\
&\Leftrightarrow d \in \delta^*\big(o, \hat{h}(m)\big)
\end{aligned}$$

Ce qui achève la récurrence.

Question 12. Reprenons les définitions.

Soit un automate $\mathcal{A} = (Q, Y, I, T, \delta)$. On a par définition l'automate $\hat{h}^{-1}(\mathcal{A}) = (Q, X, I, T, \delta_{\hat{h}^{-1}})$.

Ainsi, $\mathcal{L}\big(\hat{h}^{-1}(\mathcal{A})\big)$ est le langage reconnu par ce dernier automate.

- On s'intéresse à $\hat{h}\Big(\mathcal{L}\big(\hat{h}^{-1}(\mathcal{A})\big)\Big)$: l'image par $\hat{h}$ de ce langage. Comme suggéré par la question 9, on peut montrer l'inclusion :

  $$\hat{h}\Big(\mathcal{L}\big(\hat{h}^{-1}(\mathcal{A})\big)\Big) \subset \mathcal{L}(\mathcal{A})$$

  Soit $m \in \hat{h}\Big(\mathcal{L}\big(\hat{h}^{-1}(\mathcal{A})\big)\Big)$.
  - Par la définition 4, il existe $w \in \mathcal{L}\big(\hat{h}^{-1}(\mathcal{A})\big)$ tel que $m = \hat{h}(w)$.
  - Par la définition 12, il existe $i \in I$ et $t \in T$ tels que $t \in \delta^*_{\hat{h}^{-1}}(i, w)$.
  - Par la question 11, on a donc $t \in \delta^*\big(i, \hat{h}(w)\big)$, et donc $t \in \delta^*(i, m)$.
  - Et par la définition 12, c'est que $m \in \mathcal{L}(A)$.

  D'où l'inclusion.
- On aurait aussi pu montrer l'égalité suivante : *(les deux propositions auraient été acceptées)*

  $$\mathcal{L}\big(\hat{h}^{-1}(\mathcal{A})\big) = \hat{h}^{-1}(\mathcal{L}(\mathcal{A}))$$

  En effet, par équivalences, on a :

$$\begin{aligned}
w \in \mathcal{L}\big(\hat{h}^{-1}(\mathcal{A})\big) &\Leftrightarrow \exists i \in I, \exists t \in T, t \in \delta^*_{\hat{h}^{-1}}(i, w) && \text{par définition de } \mathcal{L} \\
&\Leftrightarrow \exists i \in I, \exists t \in T, t \in \delta^*\big(i, \hat{h}(w)\big) && \text{par la question 11} \\
&\Leftrightarrow \hat{h}(w) \in \mathcal{L}(A) && \text{par définition de } \mathcal{L} \\
&\Leftrightarrow w \in \hat{h}^{-1}(\mathcal{L}(A)) && \text{par la définition 5}
\end{aligned}$$

  D'où l'égalité.

Question 13. Soit L_Y un langage régulier sur Y. Par le théorème de Kleene, il existe un automate $\mathcal{A}$ sur l'alphabet Y tel que $L_Y = \mathcal{L}(\mathcal{A})$.

Comme pour la question 6, on remarque que d'après la question 2, on a $h = \widehat{h_{|X}}$.

Soit $\mathcal{B} = \widehat{h_{|X}}^{-1}(\mathcal{A})$. D'après la question 12 (le second point), on a $\mathcal{L}(\mathcal{B}) = \widehat{h_{|X}}^{-1}(\mathcal{L}(\mathcal{A})) = \widehat{h_{|X}}^{-1}(L_Y) = h^{-1}(L_Y)$.

Ainsi, le langage $h^{-1}(L_Y)$ est reconnu par l'automate $\mathcal{B}$, qui est un automate sur l'alphabet X. De nouveau par le théorème de Kleene, $h^{-1}(L_Y)$ est donc un langage régulier sur X.

III. Algorithmique et programmation en OCaml

III. A. Exercice : le tri par sélection

Question 1. On commence par appeler `trier [3; 1; 4; 2]`

↳ Appel **`trier`** avec = `[3; 1; 4; 2]`
  ↳ Appel **`selection`** avec = `[3; 1; 4; 2]`
    ↳ Appel **`aux`** avec = `3` et = `[1; 4; 2]`
      ↳ Appel **`aux`** avec = `1` et = `[4; 2]`
        ↳ Appel **`aux`** avec = `1` et = `[2]`
          ↳ Appel **`aux`** avec = `1` et = `[]`
            Retour **`aux`** : `(1, [])`

          Retour **`aux`** : `(1, [2])`

        Retour **`aux`** : `(1, [4; 2])`

      Retour **`aux`** : `(1, [3; 4; 2])`

    Retour **`selection`** : `(1, [3; 4; 2])`
  ↳ Appel **`trier`** avec = `[3; 4; 2]`
    ↳ Appel **`selection`** avec = `[3; 4; 2]`
      ↳ Appel **`aux`** avec = `3` et = `[4; 2]`
        ↳ Appel **`aux`** avec = `3` et = `[2]`
          ↳ Appel **`aux`** avec = `2` et = `[]`
            Retour **`aux`** : `(2, [])`

          Retour **`aux`** : `(2, [3])`

        Retour **`aux`** : `(2, [4; 3])`

      Retour **`selection`** : `(2, [4; 3])`
    ↳ Appel **`trier`** avec = `[4; 3]`
      ↳ Appel **`selection`** avec = `[4; 3]`
        ↳ Appel **`aux`** avec = `4` et = `[3]`
          ↳ Appel **`aux`** avec = `3` et = `[]`
            Retour **`aux`** : `(3, [])`

          Retour **`aux`** : `(3, [4])`

        Retour **`selection`** : `(3, [4])`
      ↳ Appel **`trier`** avec = `[4]`
        ↳ Appel **`selection`** avec = `[4]`
          ↳ Appel **`aux`** avec = `4` et = `[]`
            Retour **`aux`** : `(4, [])`

          Retour **`selection`** : `(4, [])`
        ↳ Appel **`trier`** avec = `[]`
          Retour **`trier`** : `[]`

        Retour **`trier`** : `[4]`

      Retour **`trier`** : `[3; 4]`

    Retour **`trier`** : `[2; 3; 4]`

  Retour **`trier`** : `[1; 2; 3; 4]`

Question 2. **Attention**, pour cette question, il n'est pas question d'utiliser le fait que **`selection`** extrait et renvoie le minimum de la liste donnée en paramètre ; c'est précisemment ce qu'on essaye de prouver, autrement dit on veut montrer que la fonction **`selection`** est correcte.

Comme suggéré par l'énoncé, on va pour cela donner une spécification de la fonction **`aux`**, et la prouver.

Soit **`c`** un élément, **`a`** une liste, posons **`(m, r)`** = **`aux c a`** les valeurs renvoyées par l'appel à **`aux`**, avec **`m`** un élément et **`r`** une liste.

On va montrer les faits suivants :

(1) Les éléments de **`(m :: r)`** sont exactement les éléments de **`(c :: a)`** (possiblement réordonnés).
(2) **`m`** est le minimum des éléments de **`(c :: a)`**.
(3) **`m`** est $\leq$ à tous les éléments de **`r`**.

On montre ces propriétés en raisonnant par induction sur la liste **`a`** (ou de façon équivalente, par récurrence sur la longueur de **`a`**).

- Cas de base : pour `a` $=$ `[]`. Alors `aux c a` renvoie `(c, a)`, d'où `m = c` et `r = a = []`. Vérifions les 2 propriétés :
  (1) les listes `(m :: r)` et `(c :: a)` contiennent toutes les deux un unique élément : `c`.
  (2) `m` est bien le minimum des éléments de `(c :: a) = [m]`
  (3) `m` est bien $\leq$ à tous les éléments de `r = []` par propriété de l'ensemble vide.
- Cas inductif : la liste `a` est de la forme `a` $=$ `(t :: q)`, et on suppose les 3 propriétés vérifiées pour la liste `q`.

  Par lecture du code de `aux`, il y a deux cas à considérer :
  - Si `c < t` : On pose `(m', r') = aux c q`, et on a alors `m = m'` et `r = (t :: r')`. Ainsi :
    (1) On doit vérifier que `(m :: r)` et `(c :: a)` contiennent les mêmes éléments. Or `(m :: r) = (m' :: t :: r')` et `(c :: a) = (c :: t :: q)`.

    De plus, l'hypothèse d'induction donne que `(m' :: r')` et `(c :: q)` contiennent les mêmes éléments. En ajoutant l'élément `t` de part et d'autre, on obtient le résultat voulu.
    (2) On doit vérifier que `m` est le minimum des éléments de `(c :: a) = (c :: t :: q)`. Or l'hypothèse d'induction nous donne que `m' = m` est le minimum des élements de `(c :: q)`. Comme `c < t`, on conclut directement.
    (3) On doit vérifier que `m` est $\leq$ à tous les éléments de `r = (t :: r')`. Or, l'hypothèse d'induction donne que `m' = m` est $\leq$ à tous les éléments de `r'`. L'hypothèse d'induction donne aussi (par le point (2)) que `m` est plus petit que `c` et donc que `t`, ce qui permet de conclure.
  - Deuxième cas, si `c` $\geqslant$ `t` : On pose cette fois `(m', r') = aux t q`, et on a alors `m = m'` et `r = (c :: r')`. Ainsi :
    (1) On doit vérifier que `(m :: r)` et `(c :: a)` contiennent les mêmes éléments. Or `(m :: r) = (m' :: c :: r')` et `(c :: a) = (c :: t :: q)`.

    De plus, l'hypothèse d'induction donne que `(m' :: r')` et `(t :: q)` contiennent les mêmes éléments. En ajoutant l'élément `c` de part et d'autre, on obtient le résultat voulu.
    (2) La preuve est similaire au cas précédent.
    (3) Idem.

Ce qui achève la preuve par induction.

Prouvons pour finir les trois propriétés (a), (b) et (c). Soit l'entier `m`, les listes `s` et `r` de tailles respectives `n` et `p` comme définie dans l'énoncé, de façon à avoir `(m, r) = (selection s)`. Notons que `s` est nécessairement non vide (les appels `hd s` et `tl s` provoqueraient une erreur sinon). Supposons donc `s` non vide, telle que `s = c :: a'`. L'appel `selection s` renvoie immédiatement la valeur `(aux c a)`. D'où `(m, r) = aux c a`. On a ainsi :

(a) D'après la propriété (1), les listes `(m, r)` et `s = (c, a)` contiennent les mêmes éléments. Cela prouve (a).
(b) Toujours d'après la propriété (1), les listes `(m, r)` et `s = (c, a)` sont de même longueur, donc $1 + |\mathtt{r}| = |\mathtt{s}|$, d'où $n = p + 1$.
(c) La propriété (3) donne directement le résultat (c).

Question 3. On montre les propriétés par récurrence sur la longueur de `s`.

- Initialisation : pour `s = []`. Alors `r = (trier []) = []`. Ainsi :
  (a) $m = |\mathtt{s}| = 0$ et $n = |\mathtt{r}| = 0$, d'où l'égalité.
  (b) Acquis car le « pour tout » est vide.
  (c) Idem.
- Hérédité : la liste `s` est non vide. On pose `(m', r') = selection s`, puis `q = (tri r')`, et on a alors `r = (m' :: q)`.

  D'après le point (b) de la question 2, on $|\mathtt{s}| = |\mathtt{r'}| + 1$, donc $|\mathtt{r'}| = |\mathtt{s}| - 1$. On peut donc appliquer l'hypothèse de récurrence à la liste `r'` car elle est de taille strictement inférieure à `s`. Montrons les points (a), (b) et (c).
  (a) Le point (a) de l'hypothèse de récurrence donne $|\mathtt{r'}| = |\mathtt{q}|$, d'où $|\mathtt{r}| = |\mathtt{r'}| + 1 = |\mathtt{s}|$; ce qui donne bien la propriété (a).
  (b) Le point (a) de la question 2 donne (sans les notations de l'énoncé pour simplifier...) :

  « Les listes `(m' :: r')` et `s` contiennent les mêmes éléments. »

  Le point (b) de l'hypothèse de récurrence (pour l'appel à `tri r'`) donne :

  « Les listes `q` et `r'` contiennent les mêmes éléments. »

  On en déduit directement que les listes `r = (m' :: q)` et `s` contiennent les même éléments, ce qu'on voulait montrer.
  (c) D'après le point (c) de la question 2, on sais que `m'` est plus petit que tous les éléments de `r'`.

  D'après le point (c) de l'hypothèse de récurrence (pour l'appel à `tri r'`), on sait que les éléments de `q` sont triés.

Les éléments de `q` et de `r'` étant les même (point (b) de l'hypothèse de récurrence), alors `m'` est plus petite que tous les éléments de `q`.

On en déduit que les éléments de `r = (m' :: q)` sont triés, ce qu'on voulait montrer.

Question 4.

- `aux c a` termine car chaque appel à `aux` effectue un appel récursif sur une liste de longueur strictement plus petite. Autrement dit, la longueur de la liste `a` est un variant, ce qui assure la terminaison de `aux`.
- `selection s` se contente d'appeler les fonctions `hd`, `tl`, qui terminent bien (à condition que la liste `s` soit non vide, on s'en assurera à l'étape suivante), puis d'appeler `aux`. L'appel `selection s` termine donc.
- `tri s` :
  - termine directement si `s = []`
  - sinon si `s` $\neq$ `[]`, on appelle `selection s`, qui termine, puis `tri` sur la liste renvoyée par `selection s`. Cette dernière liste est de taille strictement plus petite (d'après la question 2, propriété (b)), cette taille est donc un variant qui assure la terminaison de `tri s`.

Question 5. Je ne comprends pas cette question, il n'y a ni meilleur cas, ni pire cas pour le tri par sélection. On est toujours en $O(n^2)$. Ou alors n'importe quelle liste s est donc à la fois un meilleur et un pire cas...

En effet :

- La complexité de `aux` vérifie l'équation :

$$\begin{cases} C(0) = \mathcal{O}(1) \\ C(n) = C(n-1) + \mathcal{O}(1) \end{cases}$$

  En effet le cas de base s'effectue en temps constant, et pour une liste d'entrée de taille $n > 0$ on effectue un appel récursif pour une liste de taille $n-1$, le reste des opérations (comparaison, construction de liste, de couple) se fait en temps constant. Cette équation se résout directement en $C(n) = \mathcal{O}(n)$.
- La complexité de `selection` est donc directement en $\mathcal{O}(n)$ avec n la taille de la liste donnée en paramètre.
- La complexité de `tri` vérifie l'équation :

$$\begin{cases} C(0) = \mathcal{O}(1) \\ C(n) = C(n-1) + \mathcal{O}(n) \end{cases}$$

  En effet le cas de base s'effectue en temps constant, et pour une liste d'entrée de taille $n > 0$ on effectue un appel à `aux` en $\mathcal{O}(n)$ et un appel récursif pour une liste de taille $n-1$.

  Cette équation se résout directement en $C(n) = \mathcal{O}(n^2)$.

Question 6. Il s'agit de reformuler la définition 5 sous la forme d'une propriété, que l'on nomme $\mathrm{VAQ}(a)$ (probablement pour « Validité d'un Arbre Quaternaire »). Point par point, on a :

- « les tailles, abscisses et ordonnées de chaque noeud $n \in \mathcal{N}(a)$ sont strictement positives ». On pose :

$$P_1(a) \iff \forall n \in \mathcal{N}(a), \quad T(n) > 0 \wedge X(n) > 0 \wedge Y(n) > 0$$

- « les tailles des quatre fils f_{SO}, f_{SE}, f_{NO}, f_{NE} de chaque division $d \in \mathcal{D}(a)$ sont identiques et égales à la moitié de la taille de d ». L'énoncé n'introduisant pas la notation associée, si $d \in \mathcal{D}(a)$, on pose $f_{\mathrm{SO}}(d)$ son fils sud-ouest, et de même pour les 3 autres fils. On pose alors :

$$P_2(a) \iff \forall d \in \mathcal{D}(a), \quad T(f_{\mathrm{SO}}(d)) = T(f_{\mathrm{SE}}(d)) = T(f_{\mathrm{NO}}(d)) = T(f_{\mathrm{NE}}(d)) = \frac{T(d)}{2}$$

- « les abscisses et ordonnées des fils de chaque division $d \in \mathcal{D}(a)$ sont cohérentes avec la position géographique de chaque fils f_{SO}, f_{SE}, f_{NO}, f_{NE} et avec l'abscisse, l'ordonnée et la taille de la division d ». On pose :

$$\begin{aligned} P_3(a) \iff \forall d \in \mathcal{D}(a), \quad & X(f_{\mathrm{SO}}(d)) = X(d) && \wedge\, Y(f_{\mathrm{SO}}(d)) = Y(d) \\ & \wedge\, X(f_{\mathrm{SE}}(d)) = X(d) + T(d)/2 && \wedge\, Y(f_{\mathrm{SE}}(d)) = Y(d) \\ & \wedge\, X(f_{\mathrm{NO}}(d)) = X(d) && \wedge\, Y(f_{\mathrm{NO}}(d)) = Y(d) + T(d)/2 \\ & \wedge\, X(f_{\mathrm{NE}}(d)) = X(d) + T(d)/2 && \wedge\, Y(f_{\mathrm{NE}}(d)) = Y(d) + T(d)/2 \end{aligned}$$

- « au moins deux des quatre fils f_{SO}, f_{SE}, f_{NO}, f_{NE} de chaque division $d \in \mathcal{D}(a)$ contiennent des blocs de couleurs différentes ». Cela revient à dire que pour chaque division b, il n'existe pas de couleur telle que tous les blocs de ses 4 enfants soient tous de cette couleurs. Ainsi, en posant $\mathrm{Couleurs}$ l'ensemble des couleurs disponibles, on propose :

$$P_4(a) \iff \forall d \in \mathcal{D}(a), \quad \neg(\exists c \in \mathrm{Couleurs}, \quad \forall b \in \mathcal{B}(d), \quad C(b) = c)$$

Finalement, on pose :

$$\mathrm{VAQ}(a) \iff P_1(a) \wedge P_2(a) \wedge P_3(a) \wedge P_4(a)$$

Question 7.

```
let scinder a =
  match a with
  | Division(_, _, _, _, _, _, _) → a
  | Bloc(x, y, t, c) →
      let demi = t / 2 in
      let so = Bloc(x, y, demi, c) in
      let se = Bloc(x + demi, y, demi, c) in
      let no = Bloc(x, y + demi, demi, c) in
      let ne = Bloc(x + demi, y + demi, demi, c) in
      Division(x, y, t, so, se, no, ne)
```

Question 8. On passe par des fonctions intermédiaires (non demandées) pour simplifier l'écriture :

```
let taille q =
  match q with
  | Bloc(_, _, t, _) → t
  | Division(_, _, t, _, _, _, _) → t

let abscisse q =
  match q with
  | Bloc(x, _, _, _) → x
  | Division(x, _, _, _, _, _, _) → x

let ordonnee q =
  match q with
  | Bloc(_, y, _, _) → y
  | Division(_, y, _, _, _, _, _) → y
```

La fonction demandée s'écrit alors :

```
let fusionner so se no ne =
  let t = taille so in
  let x = abscisse so in
  let y = ordonnee so in
  Division(x, y, 2 * t, so, se, no, ne)
```

Question 9.

```
let rec profondeur a =
  match a with
  | Bloc _ → 0
  | Division(_, _, _, so, se, no, ne) →
      let pso = profondeur so in
      let pse = profondeur se in
      let pno = profondeur no in
      let pne = profondeur ne in
      1 + max pso (max pse (max pno pne))
```

Question 10.

```
let rec consulter x y a =
  match a with
  | Bloc(_, _, _, c) → c
  | Division(x0, y0, t, so, se, no, ne) →
      let m = t / 2 in
      if x < x0 + m then
        if y < y0 + m then
          consulter x y so
        else
          consulter x y no
      else
        if y < y0 + m then
          consulter x y se
        else
          consulter x y ne
```

Question 11. La principale difficulté de cette fonction est qu'on doit gérer le cas où après avoir modifié une case, les quatre fils d'une division deviennent de la même couleur, on doit alors regrouper cette division en un unique bloc.

```
let rec peindre x y c a =
  match a with
  | Bloc(x0, y0, 1, _) →
      (* Si c'est un bloc de taille 1, on le peint directement *)
      Bloc(x0, y0, 1, c)
  | Bloc(x0, y0, t, _) →
      (* Si c'est un bloc de taille > 1, on le scinde, et on peint la division *)
      peindre x y c (scinder a)
  | Division(x0, y0, t, so, se, no, ne) →
      (* Dans le cas d'une division, on modifie le sous-arbre correspondant au point (x, y) *)
      let m = t / 2 in
      let so', se', no', ne' =
        if x < x0 + m then
          if y < y0 + m then
            (peindre x y c so, se, no, ne)
          else
            (so, se, peindre x y c no, ne)
        else
          if y < y0 + m then
            (so, peindre x y c se, no, ne)
          else
            (so, se, no, peindre x y c ne)
      in
      (* On verifie si les so', se', no', ne' sont des blocs de la même couleur,
         dans ce cas on les regroupe, sinon on garde la division  *)
      match so', se', no', ne' with
      | Bloc(_, _, _, c1), Bloc(_, _, _, c2), Bloc(_, _, _, c3), Bloc(_, _, _, c4)
          when c1 = c2 && c1 = c3 && c1 = c4 →
          Bloc(x0, y0, t, c1)
      | _ →
          Division(x0, y0, t, so', se', no', ne')
```

Question 12.

```
let rec valider a =
  match a with
  | Bloc(x, y, t, c) →
      let p1 = x > 0 && y > 0 && t > 0 in
      p1
  | Division(x, y, t, so, se, no, ne) →
      let m = t / 2 in
      let p1 = x > 0 && y > 0 && t > 0 in
      let p2 =
        taille so = m && taille se = m && taille no = m && taille ne = m
      in
      let p3 =
        abscisse so = x && ordonnee so = y &&
        abscisse se = x + m && ordonnee se = y &&
        abscisse no = x && ordonnee no = y + m &&
        abscisse ne = x + m && ordonnee ne = y + m
      in
      let vso = valider so in
      let vse = valider se in
      let vno = valider no in
      let vne = valider ne in
      let p4 =
        match so, se, no, ne with
        | Bloc(_, _, _, c1), Bloc(_, _, _, c2), Bloc(_, _, _, c3), Bloc(_, _, _, c4)
            when c1 = c2 && c1 = c3 && c1 = c4 → false
        | _ → true
      in
        p1 && p2 && p3 && p4 && vso && vse && vno && vne
```

Question 13.

```
let sauvegarder arbre =
  let t = taille arbre in
  let pos = ref (t*t) in
  let rec aux a acc =
    match a with
    | Bloc(x, y, t, couleur) →
        if t = 1 then begin
          let s = (!pos, couleur) :: acc in
          decr pos; (* pos := !pos - 1 *)
          s
        end else
          let bloc = Bloc(x, y, t / 2, couleur) in
          let s1 = aux bloc acc in
          let s2 = aux bloc s1 in
          let s3 = aux bloc s2 in
          let s4 = aux bloc s3 in
          s4
    | Division(_, _, _, so, se, no, ne) →
        let s1 = aux ne acc in
        let s2 = aux no s1 in
        let s3 = aux se s2 in
        let s4 = aux so s3 in
        s4
  in
  aux arbre []
```

Question 14.

```
let restaurer sequence =
(* On est obligé de calculer la racine carrée de la longueur de la liste pour
   connaître le coté d'un bloc de l'arbre. Je ne vois pas comment faire sans ... *)
let n = int_of_float (sqrt (float_of_int (List.length sequence))) in

let rec aux sequence x y t =
  if t = 1 then
    (* Créer un bloc à la position (x, y) avec la couleur donnée *)
    match sequence with
    | (_, couleur) :: q →
      (Bloc(x, y, 1, couleur), q)
    | _ → failwith "Erreur : séquence incomplète"
  else
    (* Diviser l'arbre en 4 sous-arbres et appliquer la reconstruction récursive *)
    let t2 = t / 2 in
    let (so, reste1) = aux sequence x y t2 in
    let (se, reste2) = aux reste1 (x + t2) y t2 in
    let (no, reste3) = aux reste2 x (y + t2) t2 in
    let (ne, reste4) = aux reste3 (x + t2) (y + t2) t2 in

    (* Si les quatres blocs générés ont la même couleur, on les regroupe en un unique bloc *)
    let b = match so, se, no, ne with
      | Bloc(_, _, _, c1), Bloc(_, _, _, c2), Bloc(_, _, _, c3), Bloc(_, _, _, c4)
          when c1 = c2 && c1 = c3 && c1 = c4 → Bloc(x, y, t, c1)
      | _ → fusionner so se no ne
    in

    (* On renvoie le bloc et ce qu'il reste de la séquence *)
    (b, reste4)
in
let (arbre, _) = aux sequence 1 1 n in
arbre
```