DS1 (éléments de réponses)

Exercice 1

Question 1. Notons φ_1, φ_2 et φ_3 les trois formules étudiées.

- φ_1 est satisfiable, en prenant par exemple la valuation qui associe *vrai* aux trois variables p_i.
- φ_2 n'est pas satisfiable car pour que la formule soit vraie, il faudrait que p_2 et $\neg p_2$ soient simultanément vérifiées.
- φ_3 n'est également pas satisfiable. En effet, $(\neg p_1 \vee \neg p_2) \wedge (p_1 \vee \neg p_2)$ est logiquement équivalente à $\neg p_2$, et donc φ_3 est logiquement équivalente à $p_2 \wedge (\neg p_2) \wedge (p_1 \vee \neg p_2 \vee \neg p_3)$ qui n'est clairement pas satisfiable.

Question 2. Soit H une formule de Horn et considérons la valuation v associant la valeur *faux* à toutes les variables p_i. Nous pouvons écrire $H = C_1 \wedge \cdots \wedge C_m$ où $m \geq 0$ et où les C_i sont des clauses comportant au moins un littéral négatif. Pour tout i, nous avons donc $v(C_i) = \textit{vrai}$, puis $v(H) = \textit{vrai}$ (en remarquant que cela marche, par convention, même si $m = 0$). H est donc satisfiable.

Question 3. Soit H une formule de Horn de la forme $H = p_k \wedge C_2 \cdots \wedge C_m$, avec $C_i \neq \neg p_k$ pour tout i compris entre 2 et m. Pour chacun de ces i, soit C'_i définie de la façon suivante :

- si p_k est un littéral de C_i, alors $C'_i = \top$;
- sinon, C'_i est la clause obtenue en supprimant de C_i les éventuels littéraux de la forme $\neg p_k$.

Soit enfin H' la formule de Horn conjonction des C'_i distincts de T : $' = \bigwedge_{i,\ C'_i \neq \top} C'_i$. Par construction, nous avons bien $\mathcal{V}_{H'} \subset \mathcal{V}_H$. Montrons que H et H' sont simultanément satisfiables.

- si H est satisfiable, soit v une valuation telle que $v(H) = \textit{vrai}$. Nous avons alors $v(p_k) = \textit{vrai}$, et $v(C_i) = \textit{vrai}$ pour tout i. Si i est tel que $C'_i \neq \top$, C_i est égal (à l'ordre près des littéraux) ou à $C'_i \wedge (\neg p_k)$, ou à C'_i. Comme $v(\neg p_k) = faux$, nous avons dans les deux cas $v(C'_i) = v(C_i) = \textit{vrai}$, puis $v(H') = \textit{vrai}$: H' est satisfiable.
- Si H' est satisfiable, soit v une valuation telle que $v(H') = \textit{vrai}$. Comme p_k n'est pas une variable de H', nous pouvons (quitte à changer $v(p_k)$) supposer que $v(p_k) = \textit{vrai}$. Pour i compris entre 2 et m, deux cas se présentent :
  - ou bien C_i contient le littéral p_k, et alors $v(C_i) = \textit{vrai}$;
  - ou bien C_i ne contient pas le littéral p_k, et alors C_i est de la forme $C'_i \vee (\neg p_k)$ ou C'_i, puis $v(C_i) = v(C'_i) = \textit{vrai}$.

  Nous obtenons donc $v(H) = \textit{vrai}$ et H est satisfiable.

Question 4. Soit $H = C_1 \wedge \cdots \wedge C_m$ une formule de Horn. Pour déterminer la satisfiabilité de H, on utilise l'algorithme récursif suivant :

- si $H = \top$ alors H est satisfiable ;
- sinon, on cherche i tel que la clause C_i ne contienne pas de littéraux négatifs. Si un tel i n'existe pas, H est satifiable d'après la question (2). Sinon, la clause C_i est de la forme p_k où $1 \leqslant k \leqslant n$, puisque les clauses d'une formule de Horn contiennent au plus un littéral positif. Si l'une des autres clauses de H est réduite à $(\neg p_k)$, la formule H n'est pas satisfiable. Sinon, on applique récursivement l'algorithme à la formule de Horn H' construite en appliquant la question (3) (on sait que H est satisfiable si et seulement si H' l'est).

Cet algorithme se termine car, en cas d'appel récursif, l'ensemble $\mathcal{V}_{H'}$ est strictement contenu dans $\mathcal{V}_H$. Dans le pire des cas, au bout de n appel récursif, on appliquera l'algorithme à la formule T.

Il est difficile de majorer le temps de calcul sans préciser le type de donnée utilisé pour stocker les formules. Nous supposerons dans cette question qu'un littéral est un couple (i, b) où i est un entier et b un booléen, $(i, vrai)$ et $(i, faux)$ représentant respectivement p_i et $\neg p_i$. Une clause sera alors une liste de littéraux, et une formule sous forme normale conjonctive sera une liste de clauses. Pour traiter le problème, on commence à tester si l'une des clauses est de la forme p_k, ce qui demande un nombre d'opérations de l'orde de m (on teste si une des m listes est de taille 1, et associée à un littéral de la forme $(k, vrai)$). Si la recherche échoue, le calcul est terminé. Sinon, il faut encore de l'ordre de m opérations pour vérifier si l'une des autres clauses est de la forme $\neg p_k$. Si c'est le cas, le calcul est terminé en un temps $\Theta(m)$. Sinon, il faut définir la formule H', ce qui demande au plus de l'ordre de nm opérations (on parcourt les m clauses, qui sont des listes de longueurs au plus $2n$). Comme $n(H') \leqslant n(H) - 1$, nous obtenons en notant $T(n, m)$ le temps de calcul dans les pire des cas pour traiter une formule de Horn contenant n variables et m clauses :

$$T(n, m) \leqslant Knm + T(n-1, m)$$

On pourrait aussi remplacer $T(n-1, m)$ par $T(n-1, m-1)$ puisque H' contient au moins une clause de moins que H, mais cela complique le calcul. Ainsi, nous avons :

$$T(n,m) \leqslant Knm + T(n-1,m) \leqslant Km(n+n-1) + T(n-2,m) \leqslant \cdots \leqslant Km(n+n-1+n-2+\cdots+1) + T(0,m)$$

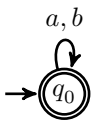
puis $T(n,m) = O(mn^2)$ puisque $T(0,m) =$ cste (c'est le temps de vérifier que la formule est associée à la liste vide).

Question 5. On remarque que la clause C_1 est réduite à p_2 et que ni C_2, ni C_3 ne sont réduites à $(\neg p_2)$. On pose donc $H' = (\neg p_1) \wedge (p_1) \wedge (p_1 \vee \neg p_3) = C'_1 \wedge C'_2 \wedge C'_3$. Comme $C'_2 = p_1$ et $C'_1 = (\neg p_1)$, H' n'est pas satisfiable et H ne l'est également pas.

Exercice 2

Question 1.

□ **1.1.** Soit $\Sigma = \{a, b\}$ un alphabet. Considérons l'automate suivant.



Soit X_0 le langage reconnu par cet automate depuis l'état q_0. On peut écrire :

$$X_0 = \varepsilon + (a+b)X_0$$

Cette équation permet l'application du lemme d'Arden tel que formulé dans l'énoncé. Un premier résultat est obtenu par le calcul suivant.

$$X_0 = (a+b)^*\varepsilon = (a+b)^*$$

Un deuxième calcul décompose le calcul sous la forme suivante.

$$X_0 = aX_0 + bX_0 + \varepsilon = a^*(bX_0 + \varepsilon) = a^*bX_0 + a^* = (a^*b)^*a^*$$

Enfin, un troisième calcul est le suivant.

$$X_0 = bX_0 + aX_0 + \varepsilon = b^*(aX_0 + \varepsilon) = b^*aX_0 + b^* = (b^*a)^*b^*$$

Ce qui établit le résultat de l'énoncé.

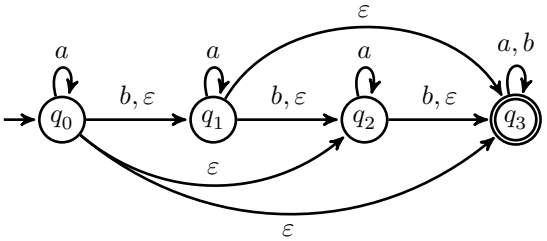
□ **1.2.** Pour cette question, une approche algébrique suffit. On utilise notamment le résultat $a^*a^* = a^*$. Ainsi :

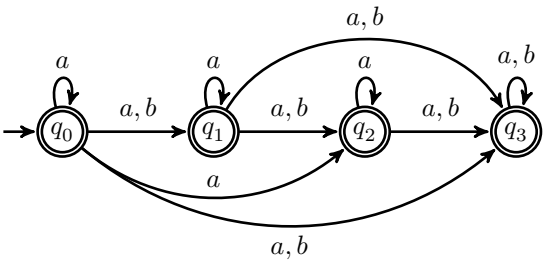
$$\begin{aligned}
a^*(a+b)^* &= a^*(a^*b)^*a^* \\
&= a^*\left(\varepsilon + (a^*b)^+\right)a^* \\
&= a^*\left(\varepsilon + (a^*b)(a^*b)^*\right)a^* \\
&= a^*a^* + a^*a^*b(a^*b)^*a^* \\
&= a^* + a^*b(a^*b)^*a^* \\
&= a^* + (a^*b)^+a^* \\
&= \left(\varepsilon + (a^*b)^+\right)a^* \\
&= (a^*b)^*a^* \\
&= (a+b)^*
\end{aligned}$$

Le second résultat s'établit en échangeant les rôles de a et b dans le premier résultat.

Question 2.

□ **2.1.** Plusieurs méthodes sont envisageables pour répondre à cette question. On propose ci-dessous d'abord de calculer la fermeture transitive des ε-transitions de l'automate, ensuite d'éliminer les ε-transitions par fermeture arrière.





□ **2.2.** En notant X_i le langage reconnu par l'automate à partir de l'état q_i, on peut écrire le système suivant.

$$\begin{aligned}
X_0 &= \varepsilon + aX_0 + (a|b)X_1|aX_2|(a|b)X_3\\
X_1 &= \varepsilon|aX_1|(a|b)X_2|(a|b)X_3\\
X_2 &= \varepsilon|aX_2|(a|b)X_3\\
X_3 &= \varepsilon|(a|b)X_3
\end{aligned}$$

En partant de la dernière équation et en remontant vers la première, le lemme d'Arden permet le calcul des X_i. Tout d'abord :

$$X_3 = (a|b)^*$$

Puis, en utilisant le résultat de la question 1.2 :

$$X_2 = a^*(\varepsilon|(a|b)^+) = a^*(a|b)^* = (a|b)^*$$

Ensuite :

$$X_1 = a^*\left(\varepsilon \mid (a|b)(a|b)^* \mid (a|b)(a|b)^*\right) = a^*\left(\varepsilon \mid (a|b)^+\right) = a^*(a|b)^*$$

Exercice 3

Question 1.

```
let rec longueur = function
  | [] -> 0
  | _::t -> 1 + longueur t
```

Question 2.

```
let rec sous_liste l k long =
  if k > 0 then sous_liste (List.tl l) (k-1) long
  else if long = 0 then []
  else (List.hd l) :: sous_liste (List.tl l) 0 (long-1)
```

Un algorithme naïf

Question 3. 1 et 3 contiennent une répétition, pas les autres.

Question 4. On peut supposer que w commence par a, et que l'autre lettre est b. Pour être sans répétition, après a on ne peut poursuivre que par b, puis à nouveau par un a. Si on essaye d'ajouter une quatrième lettre, cela ne peut être a (sinon on aurait un facteur aa), mais pas b non plus car alors w commence par $abab$ et a donc la répétition ab. D'où le résultat par contraposée.

Question 5.

□ **5.1.** On coupe la liste en son milieu et on vérifie que les deux parties sont égales. J'exploite l'égalité de listes de OCaml ; c'est une comparaison terme à terme.

```
let estCarre w =
  let l = longueur w in
  if l mod 2 = 1 then
    false
  else let ls2 = l/2 in
    (sous_liste w 0 ls2) = (sous_liste w ls2 ls2)
```

□ **5.2.** Dans le modèle de complexité proposé, le calcul de longueur et les extractions de sous-listes ne coutent rien et la comparaison est en $O(|w|/2)$. D'où une complexité linéaire.

Question 6. Il faut faire attention à ne pas se retrouver avec une liste de taille $< 2m$. Pour éviter de recalculer la longueur à chaque fois on utilise une fonction auxiliaire.

```
let contientRepetitionAux w m =
  let l = longueur w in
  let rec tester k w' =
    if l < k+2*m then
      false
    else match w' with
      | [] -> false
      | _::t -> estCarre (sous_liste w' 0 (2*m)) || tester (k+1) t
  in
  tester 0 w
```

Question 7. Soit c un facteur carré de w : il existe x, y, z tels que $w = yxxz$ et $c = xx$. On a $|w| = |y| + |z| + 2|x|$ donc $|x| \leq \frac{|w|}{2}$.

Question 8.

□ **8.1.** On procède naïvement en testant tous les m possibles.

```
let contientRepetition w =
  let l = longueur w in
  let rec tester m =
    if m > l/2 then
      false
    else
      contientRepetitionAux w m || tester (m+1)
  in
  tester 1
```

□ **8.2.** `contientRepetitionAux` est en $O(m|w|)$. D'où la complexité totale en $O(|w|^2)$.

Algorithme de Main-Lorentz

Question 9. Il s'agit de *ababa*.

Question 10. Supposons que uv contient un carré centré sur u. En reprenant les notations de l'énoncé, on a $w = w'w''$, $u = u'ww' = u'w'w''w'$ et $v = w''v''$. Notons i la position du début de w'' dans u, de sorte que $u'w' = u[0, i-1]$ et $w''w' = u[i, |u|-1]$. Par construction, w' est un suffixe commun à $u[0, i-1]$ et u, donc $|\operatorname{lcs}(u[0, i-1], u)| \geq |w'|$, et w'' est un préfixe commun à $u[i, |u|-1]$ et v, donc $|\operatorname{lcp}(u[i, |u|-1], v)| \geq |w''|$. Ainsi, la quantité étudiée est plus grande que $|w'| + |w''|$, qui vaut $|w|$, soit encore $|u[i, |u|-1]| = |u| - 1 - i + 1 = |u| - i$. D'où le sens direct.

Montrons la réciproque. Travaillons dans $u[i, |u|-1]$, mot de longueur $|u| - i$. Ce mot commence par les lettres de $\operatorname{lcp}(u[i, |u|-1], v)$ et s'achève par des lettres de $\operatorname{lcs}(u[0, i-1], u)$ (car c'est un suffixe de u). La relation sur les longueurs assure que ces deux mots comprennent toutes les lettres de $u[i, |u|-1]$: si l'égalité est réalisée, alors $u[i, |u|-1] = \operatorname{lcp}(u[i, |u|-1], v) \cdot \operatorname{lcs}(u[0, i-1], u)$; sinon, il y a chevauchement mais on peut l'éliminer en prenant un préfixe w'' et un suffixe w' un peu plus courts de sorte que $u[i, |u|-1] = w''w'$, $v = w''v''$ et $u[0, i-1] = u'w'$, ce qui établit le résultat.

Question 11. $\operatorname{pref}_u = [6, 1, 0, 0, 0, 1]$ et $\operatorname{pref}_{u,v} = [1, 3, 0, 0, 0, 1]$.

Question 12.

i	f	g	$\operatorname{pref}[i]$
0	–	0	12
1	1	3	2
2	1	3	1
3	3	3	0
4	4	12	8
5	4	12	2
6	4	12	1
7	4	12	0
8	4	12	4
9	4	12	2
10	4	12	1
11	4	12	0

Question 13. Notons v le miroir de u. Soit i un indice. $v[0, i]$ est le miroir de $u[|u| - 1 - i, |u| - 1]$, donc, puisque le passage au miroir change les préfixes en suffixes, on a $\operatorname{suff}_u[i] = \operatorname{pref}_v[|u| - 1 - i]$. On peut donc calculer suff_u en calculant v puis pref_v puis en renversant le tableau obtenu : les opérations nouvelles sont linéaires, donc on reste dans la même gamme de complexité que pour le calcul de pref_u.

Question 14.

□ **14.1.** On calcule les tableaux suff_u et $\text{pref}_{u,v}$ à l'aide des algorithmes précédents puis on vérifie par essais exhaustifs (avec une boucle) l'existence d'un i tel que $\text{suff}_u[i-1] + \text{pref}_{u,v}[i] \geq |u| - i$. Le cas $i = 0$ doit être géré à part car $\text{suff}_u[-1]$ n'existe pas (conceptuellement c'est zéro).

□ **14.2.** Le calcul des deux tableaux coute $O(|u|)$ (admis par l'énoncé). Le reste de l'algorithme ne fait aucune comparaison de caractère, d'où la complexité globale $O(|u|)$.

Question 15.

□ **15.1.** Soit w le mot à étudier. Si $|w| \leq 1$, on renvoie Faux. Sinon on découpe $w = uv$ avec u et v de taille moitié (à un près). On étudie récursivement u, puis si nécessaire v. S'il n'y a aucun carré, on applique la procédure définie ci-dessus pour chercher un carré centré sur u, puis si besoin une procédure semblable pour un carré centré sur v.

□ **15.2.** On a la relation de récurrence $C_n \approx 2\,C_{\frac{n}{2}} + O(n)$. C'est la même relation que celle du tri fusion, donc on a de même $C_n = O(n\,\log(n))$.