ENS 2004 - Accessibilité dans les graphes Et/Ou — Automates alternants

Erratum

Le sujet original comporte une grosse erreur, qui rend la question 2 fausse (ce qui se répercute sur la suite de la première partie par le biais de la définition de $A_{\mathcal{G}}$ qui n'est plus possible).
Cependant, le sujet peut redevenir correct si l'on modifie la définition d'une application *compatible avec* $\mathcal{G}$ de la façon suivante :

Définition
Une application g de V dans 2^V est compatible avec $\mathcal{G}$ si, pour tout $s \in V$:
– si $f(s) = \vee$, alors :

$$\{s\} \cup \bigcup_{(s,s') \in E} g(s') \subseteq g(s)$$

– si $f(s) = \wedge$, alors :

$$\{s\} \cup \bigcap_{(s,s') \in E} g(s') \subseteq g(s)$$

De plus, au début de la deuxième partie **Automates alternants**, dans la définition du calcul d'un automate alternant, il faut bien sûr lire $\tau(q,a)$ et non simplement $\tau(q)$.

C'est cette version que nous utilisons dans ce corrigé.

I. Accessibilité dans les graphes Et/Ou

Question 1 Pour que ce résultat soit juste, il faut supposer que pour tout $s \in E$, si $f(s) = \vee$, alors $\left\{ s' \in E \,\middle|\, (s,s') \in E \right\} \neq \emptyset$ (la condition n'est pas nécessaire si $f(s) = \wedge$). Soit g l'application qui à tout sommet $s \in \mathcal{G}$ associe V. Pour tout $s \in V$, on a :

$$\bigcup_{(s,s') \in E} g(s') = \bigcap_{(s,s') \in E} g(s') = V$$

Ainsi, quelle que soit la valeur de $f(s)$, on a :

$$\{s\} \cup \bigcup_{(s,s') \in E} g(s') = \{s\} \cup \bigcap_{(s,s') \in E} g(s') = V \subseteq V = g(s)$$

L'application g est donc bien compatible avec $\mathcal{G}$.

Question 2 On a clairement, si $f(s) = \vee$:

$$\{s\} \cup \bigcup_{(s,s') \in E} g_1(s') \cap g_2(s') \quad \subseteq \quad \{s\} \cup \bigcup_{(s,s') \in E} g_1(s') \quad \subseteq \quad g_1(s)$$

On a de même l'inclusion dans $g_2(s)$, d'où :

$$\{s\} \cup \bigcup_{(s,s') \in E} g_1(s') \cap g_2(s') \quad \subseteq \quad g_1 \cap g_2(s)$$

Le cas où $f(s) = \wedge$, on en déduit que si g_1 et g_2 sont compatibles, alors $g_1 \cap g_2$ l'est aussi.

Question 3 On suppose que la question précédente est juste. On note $C_{\mathcal{G}}$ l'ensemble des applications compatibles avec $\mathcal{G}$. D'après la question **1**, cet ensemble est non vide. De plus, il est fini. On peut donc définir :

$$A_{\mathcal{G}} : s \mapsto \bigcap \left\{ g(s) \,\middle|\, g \in C_{\mathcal{G}} \right\}$$

D'après la question **2**, l'application $A_{\mathcal{G}}$ ainsi définie est bien compatible avec $\mathcal{G}$. Cela implique en particulier que :

$$\forall\, g \in C_{\mathcal{G}},\ \forall\, s \in V,\ A_{\mathcal{G}}(s) \subseteq g(s)$$

Question 4 On donne des programmes *Caml* pour chacun des algorithmes demandés. De plus, on choisit de représenter les ensembles par des listes, et on supposera pour des raisons d'efficacité que ces listes sont ordonnées par ordre croissant (on rappelle que l'on suppose que $V = \{1, \ldots, n\}$).

1. Test d'appartenance :

```
let rec appartient e l =
  match l with
    [] -> false
  | a :: l' ->
      if a = e then true
      else if a < e then appartient e l' else false ;;
```

   La complexité est en $O(|l|)$ où l désigne la liste représentant un ensemble.

2. Test d'égalité :

```
let rec egalite l1 l2 =
  match (l1, l2) with
    [], [] -> true
  | a1 :: l1', a2 :: l2' ->
      if a1 = a2 then egalite l1' l2' else false
  | _ -> false ;;
```

Puisque l'on parcourt les deux listes élément par élément, la complexité est en $O(\min(|l_1|,|l_2|))$. L'intérêt d'avoir des listes ordonnées est ici flagrant car sinon, la complexité aurait été en $O(|l_1| \times |l_2|)$.

3. Union de deux ensembles :

```
let rec union l1 l2 =
  match (l1, l2) with
    [], _ -> l2
  | _, [] -> l1
  | a1 :: l1', a2 :: l2' ->
      if a1 = a2 then a1 :: (union l1' l2')
      else if a1 < a2 then a1 :: (union l1' l2)
      else a2 :: (union l1 l2') ;;
```

À nouveau, à chaque étape de récursion, on supprime un élément au pire un seul élément de l'une des listes. Ainsi, la complexité est en $O(|l_1|+|l_2|)$.

Remarque On peut en donner une version récursive terminale. Pour cela, une petite fonction utilitaire s'impose :

```
let rec rev_append l1 l2 =
  match l1 with
    [] -> l2
  | a :: l1' -> rev_append l1' (a :: l2) ;;

let union l1 l2 =
  let rec union_aux l1 l2 accu =
    match (l1, l2) with
      [], _ -> rev_append accu l2
    | _, [] -> rev_append accu l1
    | a1 :: l1', a2 :: l2' ->
        if a1 = a2 then union_aux l1' l2' (a1 :: accu)
        else if a1 < a2 then union_aux l1' l2 (a1 :: accu)
        else union_aux l1 l2' (a2 :: accu)
  in
  union_aux l1 l2 [] ;;
```

4. Union d'ensembles indexés par un ensemble :

```
let union_g_de_s g v =
  let rec aux v accu =
    match v with
      [] -> accu
    | e :: v' -> aux v' (union accu g.(e))
  in
  aux v [] ;;
```

La complexité de chaque union est en "O" de la somme des cardinaux des ensembles. Chacune des unions est donc en $O(|V|)$. Pour chaque élément de S, on effectue un accés en temps constant, et une union d'ensembles. La complexité de la fonction est donc $O(|V| \times |S|)$.

5. Test de compatibilité :

On utilise pour le tableau `f`, le type `sorte = Et | Ou` pour indiquer la sorte de sommet rencontré. On fait apparaître explicitement les paramètres `sommets` et `f` qui codent respectivement les tableaux G et f.

```
let compatible sommets f g s v =
  let rec aux v_a_faire =
    match v_a_faire with
      [] -> true
    | s :: v_a_faire' ->
        match f.(s) with
          Ou ->
            let union_ensemble = union_g_de_s g sommets.(s) in
            egalite g.(s) (union [s] union_ensemble)
        | Et ->
            let inter_ensemble = inter_g_de_s g sommets.(s) in
            egalite g.(s) (union [s] inter_ensemble)
  in aux v ;;
```

Cette fonction utilise une fonction `inter_g_de_s` similaire à `union_g_de_s` qui calcule l'intersection des ensembles correspondants. Il est clair que ces deux fonctions ont la même complexité.

Le déroulement de l'algorithme est simple : pour chaque sommet, on vérifie la relation de g pour ce sommet. Ainsi, pour chaque sommet s on effectue une union ou une intersection, en $O(|V|^2)$, plus la réunion avec $\{s\}$ (qui correspond à un surcoût négligeable de $O(|V|)$). Ainsi, la complexité totale est en $O(|V|^3)$.

Question 5

1. Les ensembles $A_n(s)$ étant croissants pour l'inclusion, on ne fait figurer dans le

tableau que les nouveaux éléments. On a les étapes suivantes :

	1	2	3	4	5	6	7	8	9
A_0	1	2	3	4	5	6	7	8	9
A_1	3	1	4,6		8		8	3	6,8
A_2	4,6	3			3	8	3	4,6	3
A_3		4,6	8	3	4,6	3	4,6		4
A_4	8			6		4			
A_5		8							
A_6				8					

2. Pour s fixé, les $(A_n(s))$ constituent une suite croissante pour l'inclusion de parties d'un ensemble fini. Chacune de ces suites est donc stationnaire. De plus, comme il y a un ensemble fini d'états, la suite (A_n) est elle-même stationnaire.

3. Tout d'abord, la limite A définie à la question précédente est, d'après sa définition, compatible avec $\mathscr{G}$, donc on a : $\forall\, s \in V,\ A_{\mathscr{G}}(s) \subseteq A(s)$.

   Mais l'inclusion inverse est elle aussi vérifiée. En effet, $\forall\, s \in V,\ A_0(s) \subseteq A_{\mathscr{G}}(s)$, et si pour $n \in \mathbf{N}$, on a $\forall\, s \in V,\ A_n(s) \subseteq A_{\mathscr{G}}(s)$ alors la croissance de l'union et de l'intersection en chacuns de ses arguments implique que l'on a alors :

   $$\forall\, s \in V,\ A_{n+1}(s) \subseteq A_{\mathscr{G}}(s)$$

   On en déduit par récurrence que ces inclusions sont vraies pour tout n. En particulier, elles sont vraies au rang M défini à la question précédente. Autrement dit, $\forall\, s \in V,\ A(s) \subseteq A_{\mathscr{G}}(s)$.

   Ainsi, la limite A obtenue est la relation d'accessibilité $A_{\mathscr{G}}$ de $\mathscr{G}$.

4. En calculant les relations (A_n) successives, on obtient donc la relation d'accessibilité $A_{\mathscr{G}}$. La complexité des étapes se décompose ainsi :
   – L'initialisation de A_0 a un coût en $|V|$ (qui sera négligé par rapport aux étapes suivantes).
   – Si l'on a A_n, d'après la question **4** 4., le coût pour obtenir A_{n+1} est $O(|V|^3)$. Maintenant, au pire des cas, on ajoute un unique élément dans un seul des $A_n(s)$. Aussi, dans le pire des cas, il faut $|V|^2$ étapes.

   Ainsi, en conclusion, la complexité dans le pire des cas est en $O(|V|^5)$.

Question 6 Un algorithme simple permettant de calculer $A^{-1}_{\mathscr{G}}(S)$ est le suivant :

1. On calcule $A_{\mathscr{G}}$ grâce à l'algorithme précédent.
2. Pour chacun des sommets $s \in V$, on calcule l'intersection de $A_{\mathscr{G}}(s)$ avec S. On retourne l'ensemble des sommets pour lesquels l'intersection précédente est nulle.

En Caml, cela peut se programmer ainsi :

```
let a_g_moins_1 g s =
  let a_g = accessibilite g in
  let rec aux v_restants accu =
    match v_restants with
      [] -> rev accu
    | v :: v_restants' ->
        aux v_restants' (
          match (inter a_g.(v) s) with
           [] -> accu
          | _ -> v :: accu
        )
    in aux v [] ;;
```

La complexité est nettement dominée par le calcul de $A_{\mathscr{G}}$ en $O(|V|^5)$, puisque pour l'énumération des sommets, chaque intersection est en $O(|V|)$ et le test pour savoir si l'ensemble obtenu est vide ou non est en temps constant, ainsi que l'ajout éventuel du sommet à l'ensemble retourné. Ainsi, l'étape 1. précédente est en $O(|V|^5)$ et l'étape 2. est en $O(|V|^2)$.

Question 7

1. On peut tout d'abord remarquer que chaque sommet $s \in V$ n'est traité au plus qu'une fois dans la boucle extérieure, puisque il faut que ce sommet vérifie en début de boucle $n(s) = 0$ et que la choix suivant où il est susceptible d'être ajouté à S, cela s'effectue après une décrémentation de $n(s)$.

   Ainsi, la boucle extérieure est itérée au plus $|V|$ fois. À chaque itération, les affectations de s, S, L et T se font en temps constant. Ensuite, la boucle parcourant les éléments de $\mathrm{prec}(s)$ effectue pour chaque arête aboutissant à s un traitement de temps constant. La boucle intérieure est donc au plus exécutée une fois pour chaque arête du graphe.

   On a donc au total une complexité en $O(|\mathscr{G}|)$.

2. Montrons qu'à chaque test pour éventuellement entrer dans la boucle extérieure, les points suivants sont toujours vrais :

   (a) $\forall\, s \in V \setminus \{s_0\},\ f(s) = \wedge \Rightarrow n(s) = |G(s)| - |G(s) \cap T|$;

   (b) $\forall\, s \in V \setminus \{s_0\},\ f(s) = \vee \Rightarrow n(s) = 1 - |G(s) \cap T|$.

   (c) $\forall\, s \in S \cup T,\ s_0 \in A_{\mathscr{G}}(s)$;

   (d) $\forall\, s \in V,\ s \in S \cup T \ \Leftrightarrow\ n(s) \leq 0$;

   C'est clairement vrai au début de l'exécution, puisque $T = \emptyset$ et $s_0 \in A_{\mathscr{G}}(s_0)$.

   Supposons maintenant que les deux propriétés sont vraies avant d'entrer dans la boucle, et supposons de plus que $S \neq \emptyset$. Soit s l'élément de S qui va être transféré

à T. Posons $S' = S \setminus \{s\}$ et $T' = T \cup \{s\}$. Maintenant, pour chaque $t \in V$, définissons :

$$n'(t) = n(t) - 1 \text{ si } t \in \text{prec}(s) \quad \text{et} \quad n'(t) = n(t) \text{ sinon}$$

Définissons de plus $S'' = S' \cup \{\, t \in \text{prec}(s) \mid n'(t) = 0 \,\}$. Il est clair que S'' et T' représentent les valeurs des listes S et T en sortie de boucle, et n' représente le tableau n au même point. Montrons maintenant que les propriétés (a), (b) et (c) sont vérifiées par S'', T' et n'.

Pour (a) et (b), soit $t \in V \setminus \{s_0\}$. Si $n'(t) = n(t)$, alors $t \notin \text{prec}(s)$, et on a l'égalité $G(t) \cap T' = G(t) \cap T$. Si par contre $n'(t) = n(t) - 1$, alors $t \in \text{prec}(s)$. Dans ce cas, $G(t) \cap T' = (G(t) \cap T) \cup \{s\}$. Les propriétés voulues en découlent directement.

Pour (c) et (d), il suffit que la propriété voulue est vérifiée par les éléments qui viennent d'être ajoutés à $S \cup T$, autrement aux éléments de $(S'' \cup T') \setminus (S \cup T)$. Soit t un tel élément. On a en particulier $t \neq s_0$. Si $f(t) = \wedge$, cela signifie que $n'(t) = 0$ et donc que $G(t) \subseteq T'$. En particulier, on a $G(t) \subseteq S \cup T$ et donc :

$$\forall\, t' \in G(t),\ s_0 \in A_{\mathscr{G}}(t')$$

On en déduit, par compatibilité de $A_{\mathscr{G}}$, que $s_0 \in A_{\mathscr{G}}(t)$. On procède de façon similaire si $f(t) = \wedge$, ce qui achève de montrer que (c) est vérifiée en sortie de boucle. Concernant (d), il est clair que $t \in S \cup T \ \Rightarrow\ n(t) \leq 0$. Réciproquement, si $n'(t) \leq 0$, deux solutions sont possibles pour $n(t)$: si $n(t) \leq 0$ alors $t \in S \cup T \subseteq S'' \cup T'$. Sinon, si $n(t) = 1$, alors cela signifie que $t \in \text{prec}(s)$ et que t a été ajouté dans la boucle intérieure. On a donc bien $t \in S'' \subseteq S'' \cup T'$.

On déduit que, lorsque l'algorithme s'achève, on a $T \subseteq A^{-1}_{\mathscr{G}}(\{s_0\})$, puisque dans ce cas $S = \emptyset$.

Montrons maintenant l'inclusion inverse. Pour tout $s \in A^{-1}_{\mathscr{G}}(\{s_0\})$, on définit, en reprenant la suite (A_n) de la question **5** :

$$r(s) = \min \{\, n \in \mathbf{N} \mid s_0 \in A_n(s) \,\}$$

Nous allons montrer que par récurrence sur $r(s)$ que pour tout $s \in A^{-1}_{\mathscr{G}}(\{s_0\})$, on a $s \in T$. L'initialisation est triviale, puisque $s_0 \in A_0(s) \ \Leftrightarrow\ s = s_0$ et que $s_0 \in T$. Maintenant, soit n un entier naturel quelconque, et supposons que :

$$\forall\, s \in A^{-1}_{\mathscr{G}}(\{s_0\}),\ r(s) \leq n \ \Rightarrow\ s \in T$$

Soit maintenant un élément $s \in A^{-1}_{\mathscr{G}}(\{s_0\})$ tel que $r(s) = n+1$. Par définition de $r(s)$ et de A_{n+1}, cela signifie que : $\forall\, t \in G(s)$, $s_0 \in A_n(t)$ et donc que : $\forall\, t \in G(s)$, $t \in T$.

Mais ainsi, en reprenant les propriétés (a) et (b) précédentes de T, cela signifie que $n(s) \leq 0$ et donc d'après (d) que $s \in T$

En conclusion, on a montré par récurrence que $A^{-1}_{\mathscr{G}}(\{s_0\}) \subseteq T$, d'où l'égalité voulue.

3. On peut procéder ainsi :

   Pour $s \in V$, faire
   prec$(s) := \emptyset$;
   si $f(s) = \vee$ alors $n(s) := 1$ sinon $n(s) := 0$;
   Pour $(s, s') \in E$, faire
   prec$(s') :=$ cons$(s,$ prec$(s'))$;
   si $f(s) = \wedge$ alors $n(s) := n(s) + 1$

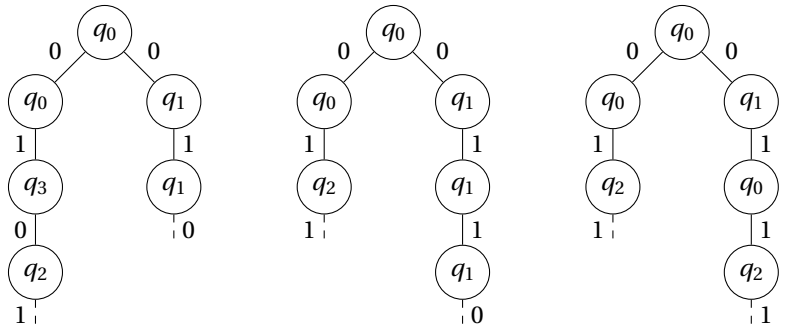
   Cet algorithme est clairement en $O(|\mathscr{G}|)$, puisque l'on parcourt V puis E et pour chacune de ces boucles, le traitement pour chaque élément se fait en temps constant.

4. On peut donc résoudre le problème d'accessibilité en temps linéaire : on commence par calculer les tableaux prec et n en temps constant avec l'algorithme précédent. Ensuite, on applique l'algorithme de la figure 2 de l'énoncé avec au départ $S = \{t\}$. Enfin, le résultat est le test de l'appartenance de s à l'ensemble T rendu.

   Les deux premières étapes sont en $O(|\mathscr{G}|)$ et la troisième en $O(|V|)$, la complexité totale est donc en $O(|\mathscr{G}|)$.

II. Automates alternants

Question 8 Puisqu'il n'y a pas d'états finaux, toutes les feuilles doivent être de hauteur strictement inférieure à la longueur du mot du chemin.



Question 9 Étant donné un automate fini non-déterministe $\mathscr{A}_1$, on définit l'automate alternant $\mathscr{A}_2$ de la façon suivante :
– son ensemble d'états Q, d'états initiaux et d'états finals est le même que pour $\mathscr{A}_1$;

– ses transitions sont définies, en notant T l'ensemble des transitions de $\mathscr{A}_1$, par :

$$\forall\,(p,a)\in Q\times\Sigma,\ \tau(p,a)=\vee\ \textbf{et}\ \delta(p,a)=\left\{\,q\in Q\ \middle|\ (p,a,q)\in T\,\right\}$$

La taille de l'automate alternant ainsi obtenu est égale à $|Q|+|T|=|\mathscr{A}_1|$.

Les deux automates reconnaissent le même langage, puisqu'un calcul réussi pour $\mathscr{A}_2$ est un arbre dont la racine est un état initial, chaque nœud interne a exactement 1 fils (accepté l'unique feuille de l'arbre qui est un état final), et chaque passage d'un nœud à un nœud fils correspond à une transition de $\mathscr{A}_1$.

Question 10 Étant donné un automate alternant $\mathscr{A}$, définissons un automate alternant $\mathscr{A}'$ de même états et états initiaux que $\mathscr{A}$, et d'états finaux $F'=Q\setminus F$. De plus, concernant les transitions, on note $\delta'=\delta$ et τ est défini par $\tau'(p,a)=\wedge\Leftrightarrow\tau(p,a)=\vee$.

Notons $L(\mathscr{A},p)$ l'ensemble des mots correspondant à un chemin réussi pour $\mathscr{A}$ de racine p. Montrons par récurrence sur la longueur des mots que l'on a :

$$\forall\,(p,u)\in Q\times\Sigma^\star,\ u\in L(\mathscr{A},p)\ \Leftrightarrow\ u\notin L(\mathscr{A},p)$$

Pour le mot vide, on peut écrire :

$$\varepsilon\in L(\mathscr{A},p)\ \Leftrightarrow\ p\in F\ \Leftrightarrow\ p\notin Q\setminus F\ \Leftrightarrow\varepsilon\notin L(\mathscr{A}',p)$$

Si maintenant le résultat est vrai pour tout mot de longueur n, soit u un mot de longueur $n+1$. On écrit $u=a\cdot v$ avec $a\in\Sigma$ et $|v|=n$. Soit $p\in Q$. Si $\tau(p,a)=\wedge$, on a :

$$\begin{aligned}u\in L(\mathscr{A},p)&\Leftrightarrow\forall\,q\in\delta(p,a),\ v\in L(\mathscr{A},q)\\&\Leftrightarrow\forall\,q\in\delta(p,a),\ v\notin L(\mathscr{A}',q)\\&\Leftrightarrow\textbf{non}\big(\exists\,q\in\delta(p,a)\colon v\in L(\mathscr{A}',q)\big)\\&\Leftrightarrow u\notin L(\mathscr{A}',p)\end{aligned}$$

On procède de façon similaire si $\tau(p,a)=\vee$.

Le temps de calcul pour obtenir $\mathscr{A}'$ en fonction de $\mathscr{A}$ est en $O(|\mathscr{A}|)$ puisqu'on se contente de calculer le complément de F dans Q (de complexité en $O(|Q|)$) et, pour chacun des états p et des lettres a, on change la valeur de $\tau(p,a)$.

Question 11 On construit récursivement l'ensemble des états qui, pour un mot donné, sont le départ d'un calcul réussi sur ce mot. On définit pour cela $Q(\varepsilon)=F$ et :

$$\begin{aligned}\forall\,(a,u)\in\Sigma\times\Sigma^\star,\ Q(a\cdot u)=&\left\{\,q\in Q\ \middle|\ \tau(q,a)=\wedge\ \textbf{et}\ \delta(q,a)\subseteq Q(u)\,\right\}\\&\cup\left\{\,q\in Q\ \middle|\ \tau(q,a)=\vee\ \textbf{et}\ \delta(q,a)\cap Q(u)\neq\emptyset\,\right\}\end{aligned}$$

On vérifie aisément que le calcul de $Q(a\cdot u)$ à partir de $Q(u)$ se fait en $O(|\mathscr{A}|)$ (pour chaque état $q\in Q$, on parcourt $\delta(q,a)$) et qu'un mot w est accepté par $\mathscr{A}$ si, et seulement si, $q_0\in Q(w)$. Ainsi, on teste si w est accepté par $\mathscr{A}$ en $O(|w|\times|\mathscr{A}|)$.

Question 12 On construit l'automate non-déterministe $\mathscr{A}_2$ ainsi : son ensemble d'états est l'ensemble des parties de Q_1 : $Q_2=\wp(Q_1)$. Les états finaux sont les parties non vides de F_1, l'état initial est $\{q_0\}$ est les transitions sont définies par :

$$(P,a,P')\in T_2\ \Leftrightarrow\ \begin{cases}\forall\,p\in P,\ \tau(p,a)=\wedge\Rightarrow\delta(p,a)\subseteq P'\\ \forall\,p\in P,\ \tau(p,a)=\vee\Rightarrow\delta(p,a)\cap P'\neq\emptyset\end{cases}$$

Par récurrence sur la longueur des mots, on montre que pour tout mot u et tout état $P\in Q_2$, il existe un chemin étiqueté par w et allant de P à une partie de F_1 si, et seulement si, $P\subseteq Q(u)$ où $Q(u)$ est l'ensemble défini à la question précédente. En effet, si $u=\varepsilon$, comme $Q(\varepsilon)=F$, cela revient à $P\subseteq F$. Sinon, si $u=a\cdot v$, il faut :

$$P\subseteq Q(a\cdot v)\ \Leftrightarrow\ \exists P'\colon(P,a,P')\in T_2\ \textbf{et}\ P'\subseteq Q(v)$$

Mais cette équivalence est évidente d'après les définitions de $Q(a\cdot v)$ et de T_2. En effet, si $P\subseteq Q(a\cdot v)$, en posant :

$$P'=\bigcup\left\{\,\delta(p,a)\ \middle|\ p\in P\ \textbf{et}\ \tau(p,a)=\wedge\,\right\}\cup\bigcup\left\{\,\delta(p,a)\cap Q(v)\ \middle|\ p\in P\ \textbf{et}\ \tau(p,a)=\vee\,\right\}$$

on a alors bien $(P,a,P')\in T_2$ et $P'\subseteq Q(v)$. La réciproque est triviale.

La complexité de l'algorithme est exponentielle en la taille de $\mathscr{A}_1$.

Question 13

1. Si un automate d'au plus 2^n états accepte le mot 1^{2^n}, tout chemin réussi ayant mot pour étiquette passe deux fois par le même état. On peut donc, d'après le lemme de l'étoile, écrire 1^{2^n} sous la forme $x\cdot y\cdot z$ avec $y\neq\varepsilon$ et tel que pour tout n, $x\cdot y^n\cdot z$ est accepté par l'automate. Son langage ne peut donc pas être réduit à $\{1^{2^n}\}$.
2. L'honnêteté m'oblige à laisser la parole à l'auteur du sujet pour cette dernière question.

On supposera dans la suite sans perte de généralité que $n\geq 2$. On définit :

$$\begin{aligned}Q=&\left\{\,q_i\ \middle|\ i\in[\![1,n]\!]\,\right\}\cup\left\{\,q_{i,k}\ \middle|\ 1\leq i\leq k\leq n\,\right\}\\&\cup\left\{\,\overline{q_i}\ \middle|\ i\in[\![1,n]\!]\,\right\}\cup\left\{\,\overline{q_{i,k}}\ \middle|\ 1\leq i\leq k\leq n\,\right\}\cup\{q_0\}\end{aligned}$$

L'état initial est q_0, les états finaux sont donnés par :

$$F=\left\{\,\overline{q_{i,i}}\ \middle|\ i<n\,\right\}\cup\{q_{n,n}\}$$

Les transitions sont définies par :

$$\begin{aligned}
& q_0 \to \overline{q_1} \wedge \ldots \wedge \overline{q_n} \\
\forall\, k < n, \quad & q_k \to q_{1,k} \vee \ldots \vee q_{k-1,k} \vee \overline{q_{k,k}} \\
\forall\, i < k < n, \quad & q_{i,k} \to q_i \wedge q_k \\
\forall\, k < n, \quad & q_{k,k} \to \overline{q_1} \wedge \ldots \wedge \overline{q_{k-1}} \wedge q_k \\
\forall\, k \leq n, \quad & \overline{q_k} \to \overline{q_{1,k}} \vee \ldots \vee \overline{q_{k-1,k}} \vee q_{k,k} \\
\forall\, i < k \leq n, \quad & \overline{q_{i,k}} \to q_i \wedge \overline{q_k} \\
\forall\, k < n, \quad & \overline{q_{k,k}} \to \overline{q_1} \wedge \ldots \wedge \overline{q_k}
\end{aligned}$$

On montre, par récurrence que $1 \leq i \leq 2^{n-1}$ qu'il existe un calcul de l'automate sur 1^{2i-1} tel que l'ensemble des états étiquetant des nœuds de profondeur $2i-1$ est :

$$\{\, q_j \mid \text{le } j\text{-ème bit de } i-1 \text{ est } 1 \,\} \cup \{\, \overline{q_j} \mid \text{le } j\text{-ème bit de } i-1 \text{ est } 0 \,\}$$

Si $i-1$, alors $2i-1=1$ et on a un calcul de l'automate sur le mot 1 dont les nœuds de profondeur 1 sont étiquettés $\overline{q_1}, \ldots, \overline{q_n}$. Or, tous les bits de $i-1$ sont nuls.

Supposons maintenant la propriété vraie pour i. Soit j l'indice du bit nul de $i-1$ de poids le plus faible. Comme $i-1 \leq 2^{n-1}-1$ on a $j \leq n-1$. Pour chaque nœud étiqueté q_k avec $k<j$, on construit un successeur $\overline{q_{k,k}}$ qui a lui-même pour successeurs $\overline{q_1}, \ldots, \overline{q_k}$. Le nœud étiqueté $\overline{q_j}$ a pour sucesseur $q_{j,j}$ qui a lui-même pour successeurs $\overline{q_1}, \ldots, \overline{q_{j-1}}, q_j$. Enfin, pour $k>j$, les nœuds étiquetés q_k ont pour successeurs $q_{j,k}$ qui ont eux-même pour successeurs q_j et q_k et les nœuds étiquetés $\overline{q_k}$ ont pour successeurs $\overline{q_{j,k}}$ qui ont pour successeurs q_j et $\overline{q_k}$.

En appliquant ce résultat pour $i = 2^{n-1}$, le calcul de l'automate sur 1^{2^n-1} a pour feuilles les états $q_1, \ldots, q_{n-1}, \overline{q_n}$. À l'étape suivante, on obtient pour feuilles $\overline{q_{1,1}}, \ldots, \overline{q_{n-1,n-1}}, q_{n,n}$ d'où un calcul réussi pour 1^{2^n}.

Réciproquement, montrons par récurrence sur la profondeur d'un calcul réussi que les mots acceptés à partir d'un état q_i (pour $i \geq 1$) sont de la forme 1^{2m+1} où le i-ème bit de m est 0 et ceux qui sont acceptés à partir d'un état $\overline{q_i}$ (pour $i<n$) sont de la forme 1^{2m+1} où le i-ème bit de m est 0. Enfin, ceux qui sont acceptés à partir de $\overline{q_n}$ sont de la forme 1^{2m+1} où $m < 2^{n-1}$.

Pour un calcul de profondeur 1, dont la racine est étiquetée q_i, il suffit que $i<n$ et en effet le i-ème bit de $m=0$ est 0. À partir de $\overline{q_i}$, le seul cas réussi correspond à $i=n$.

Si maintenant $k<n$ et ρ est un calcul réussi à partir de q_k, deux cas se présentent. Soit q_k a pour seul fils $q_{j,k}$ avec $j<k$, lequel a lui-même deux fils q_j et q_k. Dans ce cas, par hypothèse de récurrence, le mot accepté est de la forme $1^2 \cdot 1^{2m+1}$ où $m < 2^{n-1}$ et le j-ème et le k-ème bit de m valent 0. Dans ce cas, $m+1$ a encore 0 pour k-ème bit et la récurrence est montrée. Sinon, q_k a pour fils $\overline{q_{k,k}}$ qui a lui-même pour fils $\overline{q_1}, \ldots, \overline{q_k}$. Par hypothèse de récurrence, le mot accepté est de la forme $1^2 \cdot 1^{2m+1}$ tel que les k bits de poids faible de m valent 1. Dans ce cas, le k-ème bit de $m+1$ vaut à nouveau 0.

Si cette fois, pour $k<n$, on considère ρ un calcul réussi à partir de $\overline{q_k}$, deux cas se présentent. Si $\overline{q_k}$ a pour seul fils $q_{k,k}$ qui a lui-même pour descendants $\overline{q_1}, \ldots, \overline{q_{k-1}}, q_k$. Dans ce cas, le mot accepté est de la forme 1^{2m+3} où les $k-1$ bits de poids faible de m valent 1 et le k-ème bit vaut 0. Dans ce cas, le k-ème bit de $m+1$ est bien 1. Sinon, $\overline{q_k}$ a pour successeur $\overline{q_{j,k}}$ avec $j<k$, qui a lui-même pour successeurs q_j et $\overline{q_k}$. Par hypothèse de récurrence, le mot accepté est de la forme 1^{2m+3} et le j-ème bit de m est 0, le k-ème bit de m est 1. Il en résulte que le k-ème bit de $m+1$ est 1. Cela achève la récurrence.

Finalement, si ρ est un calcul réussi à partir de $\overline{q_n}$, alors ou bien $\overline{q_n}$ a pour fils $q_{n,n}$ qui est une feuille du calcul et le mot accepté est 1, ou bien $\overline{q_n}$ a pour fils $\overline{q_{j,n}}$ (avec $j<n$) qui a lui-même pour fils q_j et $\overline{q_n}$. Dans ce cas, par hypothèse de récurrence, le mot accepté est de la forme 1^{2m+3} où $m < 2^{n-1}$ et le j-ème bit de m est 0. Il en résulte que le n-ème bit de $m+1$ est le même que celui de m, soit 0. Comme $m+1 \leq 2^{n-1}$, il en résulte que $m+1 < 2^{n-1}$.

En conclusion, les mots acceptés à partir de tous les états $\overline{q_i}$ (avec $i \leq n$) sont de la forme 1^{2m+1} où les $n-1$ bits de poids faible valent 1 et $m < 2^{n-1}$. Un unique mot satisfait cette condition : 1^{2^n-1}. Il en résulte qu'un mot au plus est accepté à partir de $q_0 : 1^{2^n}$.

Ainsi, on a montré que l'automate alternant construit admet comme langage $\{1^{2^n}\}$. Il est de taille $O(n^2)$ puisqu'il comporte $O(n^2)$ états et on a :

$$\begin{aligned}
\sum_{q \in Q} |\delta(q)| &= \sum_{i=0}^{n-1} |\delta(q_i)| + \sum_{i=1}^{n} |\delta(\overline{q_i})| + \sum_{i=1}^{n-2} \sum_{k=i+1}^{n-1} |\delta(q_{i,k})| + \sum_{k=1}^{n} |\delta(q_{k,k})| \\
&\quad + \sum_{i=1}^{n-1} \sum_{k=i+1}^{n} |\delta(\overline{q_{i,k}})| + \sum_{k=1}^{n-1} |\delta(\overline{q_{k,k}})| \\
&= \left(n + \sum_{i=1}^{n} i \right) + \sum_{i=1}^{n-2} 2(n-i+1) + \sum_{k=1}^{n} k + \sum_{i=1}^{n-1} 2(n-i) + \sum_{k=1}^{n-1} k \\
&= O(n^2)
\end{aligned}$$

Ainsi, la taille de l'automate est en $O(n^2)$.