Correction - X 2013

Merci de signaler les erreurs à l'adresse suivante : mickaelpechaud (arobase) protonmail (point) com.

Partie I. Préliminaires

Question 1

1. $(u, 4) \models \mathbf{G}(p_a \vee p_b)$: c'est vrai : à partir de la lettre d'indice 4, u ne contient que des lettres a ou b.
2. $(u, 2) \models \mathbf{X}(\mathbf{G}(p_a \vee p_c))$: c'est faux. En effet, c'est équivalent à $(u, 3) \models \mathbf{G}(p_a \vee p_c)$, qui est faux, étant donné la présence de b après la position 3.
3. $(u, 1) \models \mathbf{F}(\mathbf{G}(p_a \vee p_b))$: c'est vrai : cela découle immédiatement du premier point et de la sémantique de F.
4. $u \models (p_a \vee p_b)\mathbb{U}(p_a \vee p_c)$: c'est vrai car $(u, j) \models p_b$ pour $j \leqslant 2$ et $(u, 3) \models p_c$.

Question 2

La formule $\mathbf{F}(p_a \wedge \mathbf{F}(p_b))$ convient.

Question 3

On utilise le fait que le mot vide ne satisfait aucune formule, pas même **VRAI**.
$Fin = \mathbf{VRAI} \wedge \neg(\mathbf{X}(\mathbf{VRAI}))$ convient donc.

Question 4

En utilisant la question précédente, $\mathbf{F}(Fin \wedge p_a)$ répond à la question.

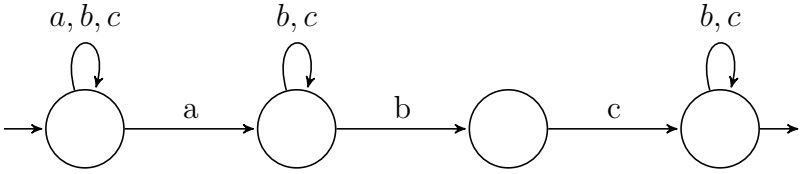
Question 5

Une solution parmi d'autres : on impose que la première lettre soit a, la dernière b, et que les facteurs de longueur 2 du mot soient ab ou ba (une idée typique des langages locaux). On obtient donc la formule suivante :
$p_a \wedge \mathbf{G}((p_a \wedge \mathbf{X}(p_b)) \vee (p_b \wedge \mathbf{X}(p_a)) \vee (Fin \wedge p_b))$.

Question 6

L_φ est l'ensemble des mots contenant une lettre a à une position telle que :
- dans la suite du mot, il n'y ait plus de lettre a ;
- dans la suite du mot, il y ait (au moins) un facteur bc.

L'automate suivant contient donc :



Question 7

Procédons par double implication.

- Supposons $u \models (\varphi \mathbb{U} \psi)$.
  Dans la définition de $\mathbb{U}$, soit $j = i$, et alors $u \models \psi$.
  Soit $j > i$. On a alors $u \models \varphi$, pour tout k tel que $i \leqslant k < j$, on a $(u, k) \models \varphi$ (ce résultat reste donc vrai pour tout k vérifiant $i + 1 \leqslant k < j$) et $(u, j) \models \psi$. Par définition, on a donc $u \models \mathbf{X}(\varphi \mathbb{U} \psi)$.
  Finalement, on a bien $u \models \varphi \wedge \mathbf{X}(\varphi \mathbb{U} \psi)$.
  La première implication est donc démontrée.
- La réciproque est claire d'après le raisonnement ci-dessus.

Partie II. Normalisation de formules

Question 8

```
let rec taille form = match form with
  | VRAI | Predicat (_) -> 1
  | NON (f) | X(f) | G(f) | F(f) -> 1 + taille f
  | ET (fg,fd) | OU (fg,fd) | U (fg,fd) -> 1 + taille fg + taille fd
 ; ;
```

Question 9

On a clairement $\mathbf{F}(\varphi) \equiv \mathbf{VRAI}\ \mathbb{U}\ \varphi$, qui ne contient pas de $\mathbf{F}$ si φ n'en contient pas. On en déduit la fonction suivante :

```
let rec normaliseF form = match form with
  | F(f) -> U(VRAI,normaliseF f)
  | NON(f) -> NON (normaliseF f)
  | X(f) -> X (normaliseF f)
  | G(f) -> G (normaliseF f)
  | ET (fg,fd) -> ET (normaliseF fg, normaliseF fd)
  | OU (fg,fd) -> OU (normaliseF fg, normaliseF fd)
  | U (fg,fd) -> U (normaliseF fg, normaliseF fd)
  | _ -> form
 ; ;
```

La complexité est linéaire en la taille de la formule - i.e. en le nombre de noeuds de l'arbre correspondant.

Question 10

Il faut éliminer les G.
Pour ce faire, on constate que pour toute formule φ, on a $\mathbf{G}(\varphi) \equiv \neg(F(\neg\varphi))$.
On commence donc par appliquer cette transformation à tous les G qui apparaissent dans la formule, puis on supprime les F à l'aide de la question précédente.

```
let rec normaliseG form = match form with
  | F(f) -> F(normaliseG f)
  | NON(f) -> NON (normaliseG f)
  | X(f) -> X (normaliseG f)
  | G(f) -> NON(F (NON (normaliseG f)))
  | ET (fg,fd) -> ET (normaliseG fg, normaliseG fd)
  | OU (fg,fd) -> OU (normaliseG fg, normaliseG fd)
  | U (fg,fd) -> U (normaliseG fg, normaliseG fd)
  | _ -> form
 ; ;

let normalise f = normaliseF (normaliseG f) ; ;
```

La fonction obtenue est de complexité linéaire en la taille de la formule pour la même raison que ci-dessus.

Question 11

En étant attentif au cas du mot vide dans les cas de base, on obtient la fonction suivante :

```
let rec veriteN u i form = match form with
  | NON(f) -> not (veriteN u i f)
  | X(f) ->  veriteN u (i+1) f
  | ET (fg,fd) -> veriteN u i fg && veriteN u i fd
  | OU (fg,fd) -> veriteN u i fg || veriteN u i fd
  | U (fg,fd) -> (veriteN u i fd) || (veriteN u i fg && (veriteN u (i+1) form))
  | VRAI -> i<string_length u
  | Predicat (c) -> (i<string_length u) && (u.[i]=c)
 ; ;
```

La terminaison est évidente : chaque appel récursif se fait
- soit sur une formule de taille strictement inférieure
- soit avec un indice strictement supérieur (dans le cas de $\mathbb{U}$).

La taille et les indices étant des entiers respectivement minoré par 0 et majorés par la longueur de la chaîne de départ, l'algorithme termine nécessairement.
Si l'opérateur $\mathbb{U}$ n'intervenait pas, la complexité serait immédiatement linéaire en la taille de la formule – chaque appel récursif portant sur un noeud différent de celle-ci. Avec l'opérateur $\mathbb{U}$: chaque appel récursif porte sur un couple (noeud,indice) différent. La complexité est donc $O(l_f l_u)$, où l_f et l_u sont respectivement la taille de la formule et la longueur de la chaîne.
La complexité est donc polynomiale.

Partie III. Rationalité des langages décrits par des formules

Question 12

Il s'agit simplement de parcourir l'arbre de la formule en ajoutant des drapeaux (en bon franglais informatique, on dirant *flags*) faux.

```
let rec initialise f = match f with
  | VRAI -> AVRAI,false
  | Predicat (c) -> APredicat (c),false
  | NON(f) -> ANON(initialise f),false
  | X(f) ->  AX(initialise f),false
  | ET (fg,fd) -> AET(initialise fg,initialise fd),false
  | OU (fg,fd) -> AOU(initialise fg,initialise fd),false
  | U (fg,fd) -> AU(initialise fg,initialise fd),false
 ; ;
```

Question 13

On montre par induction sur l'ensemble des formules normalisées la propriété demandée.

VRAI : $a.u \models$ **VRAI**, indépendemment de u (et même de $\mathcal{S}$), donc le résultat est vrai si la formule est réduite à **VRAI**.

p_c **:** $a.u \models p_c$ Ssi $a = c$ – ce qui est également indépendant de u (et de $\mathcal{S}$).

$\wedge$ **:** on suppose le résultat vrai pour deux formules φ_1 et φ_2. On note $\varphi = \varphi_1 \wedge \varphi_2$. On veut donc montrer que la valeur de $a.u \models \varphi$ ne dépend pas de u (c'est immédiatement le cas pour toutes les autres formules par hypothèse, puisque ce sont des sous formules de φ_1 ou φ_2).
$a.u \models \varphi$ Ssi ($a.u \models \varphi_1$ et $a.u \models \varphi_2$), ce qui par hypothèse ne dépend que de a, $\mathcal{S}$ (plus précisément des ensembles des sous-formules vraies de φ_1 et φ_2 – qui sont trivialement inclus dans celles de φ).
Le résultat est donc prouvé dans ce cas ;

$\vee, \neg$ **:** la preuve est la même que dans le cas précédent.

X : on suppose le résultat vrai pour une formule φ. On a $a.u \models X(\varphi)$ Ssi $u \models \varphi$.
Or l'information "$u \models \varphi$" est dans $\mathcal{S}$: il s'agit de savoir si la sous-formule constituée de $X(\varphi)$ privée du X est vraie pour u. Déterminer $a.u \models X(\varphi)$ ne dépend donc que de $\mathcal{S}$.

$\mathbb{U}$ **:** supposons le résultat vrai pour φ et ψ. D'après la question 7, $a.u \models \varphi \mathbb{U} \psi$ Ssi ($(a.u \models \psi)$ ou ($a.u \models \varphi$ et $u \models \varphi \mathbb{U} \psi$)). Par hypothèse, les valeurs de $a.u \models \varphi$ et $a.u \models \psi$ ne dépendent pas de u. L'information "$u \models \varphi \mathbb{U} \psi$" est dans $\mathcal{S}$ (il s'agit de savoir si u satisfait $\varphi \mathbb{U} \psi$, qui est bien une sous-formule d'elle même).

Le résultat est donc prouvé par induction.

Question 14

On traduit cas par cas les raisonnements de la question 13 :

```
let rec maj e a = match e with
  | AVRAI,_ -> AVRAI,true
  | APredicat (c),_ -> APredicat (c),c=a
  | ANON(f),_ -> let (e2,b)=maj f a in ANON(e2,b),(not b)
```

```
  | AET(fg,fd),_ -> let (e2g,bg)=maj fg a
                    and (e2d,bd)=maj fd a
      in AET((e2g,bg),(e2d,bd)),(bg && bd)
  | AOU(fg,fd),_ -> let (e2g,bg)=maj fg a
                    and (e2d,bd)=maj fd a
      in AOU((e2g,bg),(e2d,bd)),(bg || bd)
  | AX((f,b)),_ ->  AX(maj (f,b) a),b
  | AU (fg,fd),b -> let (e2g,bg)=maj fg a
                    and (e2d,bd)=maj fd a
      in AU((e2g,bg),(e2d,bd)),(bd || (bg && b))
; ;
```

A noter que l'exemple fourni par l'énoncé est curieux. Sauf erreur, aucun mot ne vérifie exactement l'ensemble des sous-formules indiquées : $u \models p_a$ donc commence par a. $u \models \mathbf{X}\mathbb{U}(p_a, p_b)$, donc à partir de la deuxième lettre, u est une séquence de 'a' suivie d'un 'b'. Donc on devrait avoir $u \models \mathbb{U}(p_a, p_b)$, ce qui n'est pas le cas.

Question 15

L'arbre construit à la question 12 correspond à l'ensemble vide, ce qui est cohérent avec le fait que le mot vide ne satisfait aucune formule.
Il suffit ensuite de "rajouter" les lettres de u les unes après les autres (en partant de la dernière) – on construira ainsi successivement les ensembles de formules satisfaites par des suffixes de plus en plus longs de u, jusqu'à arriver à l'ensemble des formules satisfaites par u.
Si l'on suppose la formule déjà normalisée, on peut écrire la fonction suivante :

```
let sousFormulesVraies f u =
  let e = ref (initialise f) in
  for i=string_length u-1 downto 0 do
    e :=maj  !e (u.[i])
  done ;
   !e
; ;
```

Question 16

C'est immédiat après la question précédnte : il suffit de vérifier si le drapeau de la racine a pour valeur *vrai*.

```
let veriteBis f u =
  match sousFormulesVraies f u with
      (_,b)-> b
; ;
```

Question 17

Soit φ une formule.
La question 14 montre qu'il est possible, si l'on connait l'ensemble $\mathcal{S}$ des sous-formules satisfaites par u, de construire l'ensemble des formules $\mathcal{S}'$ satisfaites par $a.u$ pour toute

lettre a. L'idée est de considérer cette construction comme une transition dans un automate – où chaque état correspondra à un ensemble de sous-formules de φ.
On peut donc construire l'automate déterministe (Q, A, Δ, q_0, F) suivant :
- Q est l'ensemble des sous-ensembles de sous-formules de φ.
- pour tout couple d'états (q, q') et toute lettre a, $(q, a, q') \in \Delta$ Ssi $majqa = q'$ (avec les abus de notation évidents...)
- $q_0 = \varnothing$ (aucune sous-formule n'est satisfaite par u)
- F est l'ensemble des sous-ensemble de sous-formules de $\underbrace{\varphi}$ contenant φ.

Par construction, u est reconnu par cet automate Ssi $u \in \widetilde{L_\varphi}$.
Le nombre d'état est $2^{taille(\varphi)}$ – et probablement beaucoup moins si l'on se restreint aux états accessibles.

Question 18

C'est quasiment une question de cours. Si l'on dispose d'un automate $\mathcal{A}$ (déterministe ou pas) reconnaissant $\widetilde{L_\varphi}$, on construit un automate $\mathcal{A}'$ (en général non-déterministe) reconnaissant L_φ en
- inversant toutes les transitions ;
- permutant les états initiaux et acceptants.

Par construction, un calcul étiqueté par w est réussi dans $\mathcal{A}'$ Ssi le calcul obtenu en inversant toutes les flêches est réussi dans $\mathcal{A}$ – ce calcul étant alors étiqueté dans $\tilde{w}$.
$\mathcal{A}'$ reconnait donc bien L_φ, et son nombre d'états est toujours $O(2^{taille(\varphi)})$.

Partie IV. Satisfiabilité et expressivité

Question 19

Il s'agit d'un algorithme de parcours de graphe.
On utilise une structure par exemple de file, que l'on initialise en y mettant l'état initial. On "marque" l'état initial comme déjà atteint (on peut par exemple utiliser un tableau de booléens à cet effet).
Tant que la file est non-vide :
- on retire un élément q de la file ;
- pour chaque état accessible *non-marqué* q' en une transition à partir de q
  - on marque q' ;
  - on l'ajoute à la file.

On renvoie finalement l'ensemble des états marqués.

Le langage reconnu par l'automate est alors non-vide Ssi l'ensemble des états accessibles contient au moins un état acceptant.

Question 20

Soit φ une formule satisfiable. En utilisant les questions 17, et 18, on obtient un automate $\mathcal{A}$ à au plus $2^{taille(\varphi)}$ états reconnaissant u Ssi $u \models \varphi$.
Soit u_{min} un mot de longueur minimale reconnu par $\mathcal{A}$. Montrons que sa longueur est inférieure à $2^{taille(\varphi)} - 1$.
Raisonnons par l'absurde, et supposons que la longueur de u_{min} est supérieure à $2^{taille(\varphi)}$.
On considère un calcul réussi

$$c = q_0 \xrightarrow[*]{u_{min}} q_f$$

étiqueté par u_{min} (q_0 étant un état initial et q_f un état acceptant). Ce calcul passe par au moins $2^{taille(\varphi)}$ transitions, et donc par au moins $2^{taille(\varphi)} + 1$ états.
D'après le lemme des tiroirs, il passe au moins deux fois par un même état q'. Il existe donc une décomposition $u_{min} = uvw$, avec $v \neq \epsilon$ tel que le calcul réussi c s'écrive

$$q_0 \xrightarrow[*]{u} q' \xrightarrow[*]{v} q' \xrightarrow[*]{w} q_f$$

Alors,

$$q_0 \xrightarrow[*]{u} q' \xrightarrow[*]{w} q_f$$

est un calcul réussi dans $\mathcal{A}$ étiqueté par uw. uw est donc accepté par l'automate, mais est de longueur strictement inférieure à celle de u_{min}, ce qui est absurde.

u_{min} est donc de longueur inférieure à $2^{taille(\varphi)} - 1$.

Question 21

La question précédente permet d'éviter la construction explicite de l'ensemble des états accessibles de l'automate correspondant à la formule φ : il "suffit" en effet de tester tous les mots de longueur inférieure à $2^{taille(\varphi)} - 1$ – et de voir s'il en existe un satisfaisant la formule.
Ce n'est pas optimal – le même ensemble d'état peut être parcouru étudié plusieurs fois – mais c'est beaucoup plus simple à écrire...
On commence donc par écrire une fonction auxiliaire qui prend en argument une chaîne de caractères s, un ensemble de sous-formules correspondant e, la longueur de l et une longueur limite *llim* a ne pas dépasser, et qui renvoie s'il existe un mot de suffixe l et de longueur inférieure à *llim* satisfaisant la formule, et faux sinon (le résultat renvoyé est donc en fait un couple (booléen,chaîne)).
Pour se faire, on commence par regarder si le mot courant satisfait la formule. Sinon, on essaye successivement de lui ajouter un 'a' ou un 'b'. On appelle alors récursivement la fonction avec la nouvelle chaîne et les ensembles de sous-formules et longueurs correspondantes. Vu les complexités en jeu, il est indispensable de renvoyer un mot satisfaisant la formule dès qu'il est trouvé.

```
let rec satisfiable_aux e s l llim =
  if (l>llim) then (false,"") else
  match e with
    | (_,true) -> (true,s)
    | (f,false) ->
      let (ba,ma) = satisfiable_aux (maj e 'a') (s^"a") (l+1) llim in
      if ba then (true,ma)
      else satisfiable_aux (maj e 'b') (s^"b") (l+1) llim
 ; ;
```

(On a utilisé l'opérateur ^ pour concaténer des chaînes de caractères – et on ajoute les nouveaux caractères à la fin de la chaîne pour éviter d'avoir à appeler une fonction *miroir* à la fin.)
La fonction demandée peut alors s'écrire de la façon suivante :

```
let satisfiable f =
  let llim = puiss2 (taille(f))-1 in
  let (b,s) = satisfiable_aux (initialise f) "" 0 llim in
  if b then s else "Formule non satisfiable"
 ; ;
```

Dans le cas le pire (la formule n'est pas satisfiable), *satisfiable_aux* va être appelé sur tous les mots possibles – il y en a $O(2^{llim}) = O(2^{2^{|\varphi|}-1})$.
Dans chaque appel, on va effectuer au pire 2 appels à *maj* – de complexité linéaire en la taille de la formule.
La complexité est donc

$$O(|\varphi|2^{2^{|\varphi|}-1})$$

– ce qui n'est pas le résultat demandé par l'énoncé. Il doit donc être possible de gagner sur la majoration peu subtile de $|u_{min}|$ à la question 20 – i.e. sur le nombre d'états des automates des question 17 et 18.

Question 22

Montrons le résultat intermédiaire demandé par induction sur l'ensemble des formules (normalisées).

VRAI : cette formule est satisfaite par tout mot non-vide – et la propriété est donc vraie avec $N = 1$.

p_a **:** l'alphabet étant réduit à la lettre a, p_a est équivalent à **VRAI**

$\wedge$ **:** on suppose le résultat vrai pour deux formules φ_1 et φ_2. Soient N_1 et N_2 telles
- pour tout $n \geqslant N_1$, $u \models \varphi$ ou pour tout $n \geqslant N_1$, $u \not\models \varphi_1$
- pour tout $n \geqslant N_2$, $u \models \varphi$ ou pour tout $n \geqslant N_2$, $u \not\models \varphi_2$

Posons $N = \max\{N_1, N_2\}$. Pour $n \geqslant N$, on a donc soit toujours $n \models \varphi$, soit toujours $n \not\models \varphi$.

$\vee$ **:** de même !

$\neg$ **:** si tout mot de longueur supérieure à N satisfait une formule φ, tout mot de longueur supérieur à N ne satisfait pas $\neg(\varphi)$. De mêle pour l'autre cas.

X : on suppose le résultat vrai pour une formule φ. Soit N tel que tout mot de longueur supérieure N satisfasse φ (l'autre cas est symétrique). Alors tout mot de longueur supérieure à $N + 1$ satisfait $X(\varphi)$.

$\mathbb{U}$ **:** supposons le résultat vrai pour φ_1 et φ_2. Distinguons plusieurs cas :
- φ_2 est satisfaite pour tout mot assez long (on note N_2 la longueur correspondante). Alors, tout mot de longueur plus grande que N_2 satisfait également $\varphi_1 \mathbb{U} \varphi_2$ par définition de $\mathbb{U}$.
- φ_2 n'est pas satisfaite pour tout mot assez long (on note N_2 la longueur correspondante).
  - si φ_1 n'est pas satisfaite pour tout mot assez long (on note N_1 la longueur correspondante) : alors tout mot de longueur plus grande que $N = \max\{N_1, N_2\}$ ne satisfait ni φ_1 ni φ_2, et donc pas non plus $\varphi_1 \mathbb{U} \varphi_2$.
  - si φ_1 est satisfaite pour tout mot assez long (on note N_1 la longueur correspondante).
    S'il n'existe aucun mot de longueur supérieure à N_1 satisfaisant $\varphi_1 \mathbb{U} \varphi_2$, c'est terminé.

Sinon, notons N la longueur d'un tel mot. Soit $u = a^n$, avec $n \geqslant N$. On peut donc écrire $u = a^{n-N}a^N$. $a^N \models \varphi_1 \mathbb{U} \varphi_2$ par hypothèse, i.e. $(u, n-N) \models \varphi_1 \mathbb{U} \varphi_2$. De plus, pour tout $j < n - N$, $(u, j) \models \varphi_1$: en effet il reste au moins $N \geqslant N_1$ lettre à partir de la $j^{\text{ème}}$, et donc le suffixe de u correspondant satisfait φ_1 par définition de N_1. On en déduit que tout mot de longueur supérieure à N satisfait $\varphi_1 \mathbb{U} \varphi_2$, ce qui achève la preuve.

Finalement, étant donné qu'il existe des mots arbitrairement longs dans $L = \{a^{2i} | i \geqslant 1\}$ et des mots arbitrairement longs dans son complémentaire, il n'existe aucune formule φ telle que $L = L_\varphi$.