Logique temporelle (X-ENS 2013)

Partie I. Préliminaires

Question 1.

(a) On a $(u,4) \models \mathbf{G}(p_a \vee p_b)$ car à partir de la lettre d'indice 4 toutes les lettres sont des a ou des b ;

(b) on a $(u,2) \not\models \mathbf{X}(\mathbf{G}(p_a \vee p_c))$ car à partir du rang 3 toutes les lettres ne sont pas que des a ou des c ;

(c) on a $(u,1) \models \mathbf{F}(\mathbf{G}(p_a \vee p_b))$ car il existe un rang (par exemple 4) à partir duquel toutes les lettres sont des a ou des b ;

(d) on a $u \models (p_a \vee p_b)\,\mathbf{U}\,(p_a \vee p_c)$ car il existe un rang (ici 3) pour lequel la lettre correspondante est un a ou un c et toutes les lettres précédentes des a ou des b.

Question 2. La formule $\varphi = \mathbf{F}(p_a \wedge \mathbf{F}\,p_b)$ convient.

Question 3. La formule $\mathrm{Fin} = \mathrm{VRAI} \wedge \neg(\mathbf{X}\,\mathrm{VRAI})$ convient.

Question 4. De la question précédente il résulte immédiatement que $\varphi = \mathbf{F}(p_a \wedge \mathrm{Fin})$ convient.

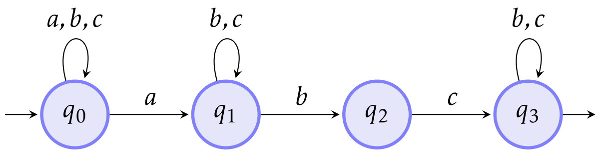
Question 5. J'ai choisi de traduire les propriétés suivantes : les mots doivent commencer par un a, finir par un b, tout a doit être suivi d'un b, tout b hormis le dernier suivi d'un a. Ceci nous donne :

$$\varphi = p_a \wedge \mathbf{G}((p_a \wedge \mathbf{X}\,p_b) \vee (p_b \wedge \mathbf{X}\,p_a) \vee (p_b \wedge \mathrm{Fin}))$$

Question 6. Soit u un mot de longueur $\ell \geqslant 1$. On a $u \models \varphi$ si et seulement s'il existe un rang $i \leqslant \ell - 1$ tel que :

- $u_i = a$;
- $j > i \Longrightarrow u_i \neq a$;
- il existe $i < j < \ell - 1$ tel que $u_j u_{j+1} = bc$.

Le langage L_φ est rationnel car dénoté par l'expression $(a+b+c)^* a(b+c)^* bc(b+c)^*$; il est reconnu par l'automate non déterministe suivant :



Question 7. Considérons un mot u. On a $u \models \varphi\,\mathbf{U}\,\psi$ si et seulement s'il existe $j \leqslant |u| - 1$ tel que $(u,j) \models \psi$ et $(u,k) \models \varphi$ pour $0 \leqslant k < j$. Cette équivalence recouvre deux cas :

- ou bien $j = 0$ et dans ce cas seule la condition $u \models \psi$ subsiste ;
- ou bien $j \geqslant 1$ et dans ce cas la condition énoncée peut aussi s'écrire $u \models \varphi$ et $u \models \mathbf{X}(\varphi\,\mathbf{U}\,\psi)$.

Cette disjonction de cas se traduit par l'équivalence : $\varphi\,\mathbf{U}\,\psi \equiv \psi \vee (\varphi \wedge \mathbf{X}(\varphi\,\mathbf{U}\,\psi))$.

Partie II. Normalisation de formules

Question 8.

```
let rec taille = function
  | VRAI        -> 1
  | Predicat _  -> 1
  | NON f       -> 1 + taille f
  | ET (f, g)   -> 1 + taille f + taille g
  | OU (f, g)   -> 1 + taille f + taille g
  | X f         -> 1 + taille f
  | G f         -> 1 + taille f
  | F f         -> 1 + taille f
  | U (f, g)    -> 1 + taille f + taille g ;;
```

Question 9. On a $\mathbf{F}\,\varphi \equiv \text{VRAI}\,\mathbf{U}\,\varphi$, ce qui conduit à la fonction de normalisation suivante :

```
let rec normaliseF = function
  | NON f      -> NON (normaliseF f)
  | ET (f, g) -> ET (normaliseF f, normaliseF g)
  | OU (f, g) -> OU (normaliseF f, normaliseF g)
  | X f        -> X (normaliseF f)
  | G f        -> G (normaliseF f)
  | F f        -> U (VRAI, normaliseF f)
  | U (f, g)  -> U (normaliseF f, normaliseF g)
  | f          -> f ;;
```

À l'évidence la complexité de cette fonction est proportionnelle au nombre de nœuds de l'arbre associé à la formule, autrement dit à la taille de la formule.

Question 10. Compte tenu de la question précédente, il suffit pour montrer que toute formule est équivalente à une formule normalisée de trouver, pour une formule φ n'utilisant pas le connecteur $\mathbf{G}$, une formule équivalente à $\mathbf{G}\,\varphi$ n'utilisant pas le connecteur $\mathbf{G}$. Or à l'évidence $\mathbf{G}\,\varphi \equiv \neg(\mathbf{F}(\neg\varphi))$ soit, compte tenu de la question précédente, $\mathbf{G}\,\varphi \equiv \neg(\text{VRAI}\,\mathbf{U}\,(\neg\varphi))$. D'où la fonction :

```
let rec normalise = function
  | NON f      -> NON (normalise f)
  | ET (f, g) -> ET (normalise f, normalise g)
  | OU (f, g) -> OU (normalise f, normalise g)
  | X f        -> X (normalise f)
  | G f        -> NON (U (VRAI, NON (normalise f)))
  | F f        -> U (VRAI, normalise f)
  | U (f, g)  -> U (normalise f, normalise g)
  | f          -> f ;;
```

Question 11. La principale difficulté tient au traitement d'une formule de la forme $\varphi\,\mathbf{U}\,\psi$; j'utilise dans ce cas le résultat de la question 7, ce qui conduit à la définition suivante :

```
let rec veriteN u i = function
  | _ when i < 0 or i >= string_length u -> false
  | VRAI        -> true
  | Predicat c -> u.[i] = c
  | NON f       -> not (veriteN u i f)
  | ET (f, g)  -> (veriteN u i f) && (veriteN u i g)
  | OU (f, g)  -> (veriteN u i f) || (veriteN u i g)
  | X f         -> veriteN u (i+1) f
  | U (f, g)   -> veriteN u i g || (veriteN u i f) && veriteN u (i+1) (U (f, g))
  | _           -> failwith "formule non normalisée" ;;
```

Notons ℓ la longueur du mot u, $|\varphi|$ la taille de la formule φ, et posons $|(i,\varphi)| = (\ell - i, |\varphi|)$.
Munissons $\mathbb{N}^2$ de la relation d'ordre lexicographique : $(a,b) \leqslant (a',b')$ si et seulement si $a \leqslant a'$ ou $a = a'$ et $b \leqslant b'$. Il s'agit d'un bon ordre (toute partie non vide possède un plus petit élément) et chacun des appels récursifs (j,ψ) présent dans la fonction ci-dessus vérifie $|(j,\psi)| < |(i,\varphi)|$, ce qui assure la terminaison de l'algorithme.

Pour évaluer la complexité de cet algorithme, nous allons considérer le mot $u = a^p$ et la formule φ_q définie inductivement par les relations $\varphi_0 = \text{VRAI}$ et $\varphi_{n+1} = \neg\varphi_n\,\mathbf{U}\,\varphi_n$ (ce choix résulte du souhait de ne pas mettre fin prématurément à l'évaluation paresseuse). Notons $\mathrm{C}(p,q)$ le coût de l'évaluation de $(a^p, 0) \models \varphi_q$. On dispose de la relation :

$$\mathrm{C}(p,q) = 2\mathrm{C}(p,q-1) + \mathrm{C}(p-1,q) + \Theta(1)$$

à partir de laquelle il est facile de tirer la minoration $\mathrm{C}(p,q) = \Omega(3^{p+q})$.
Sachant que $|\varphi_q| = \Theta(2^q)$ on en déduit que $\mathrm{C}(p,q) = \Omega(3^p|\varphi_q|^{\log_2 3})$. La complexité est bien polynomiale vis-à-vis de la taille de la formule φ_q, mais exponentielle vis-à-vis de la longueur du mot a^p.
Il faut donc en conclure que la complexité de la fonction **veriteN** peut être exponentielle en la taille de ses arguments.

Partie III. Rationalité des langages décrits par des formules

Question 12.

```
let initialise phi =
  let rec aux = function
    | VRAI        -> AVRAI, false
    | Predicat a -> APredicat a, false
    | NON f       -> ANON (aux f), false
    | ET (f, g)   -> AET (aux f, aux g), false
    | OU (f, g)   -> AOU (aux f, aux g), false
    | X f         -> AX (aux f), false
    | U (f, g)    -> AU (aux f, aux g), false
    | _           -> failwith "www.youtube.com/watch?v=BKQ9--_ZgB4"
  in aux (normalise phi) ;;
```

Question 13. Considérons une formule normalisée φ et montrons par induction structurelle que l'appartenance de φ à $\mathscr{S}'$ (autrement dit vérifie $a \cdot u \models \varphi$) dépend uniquement de a et de $\mathscr{S}$.

- Si $\varphi = \mathrm{VRAI}$, alors φ appartient à $\mathscr{S}'$;
- si $\varphi = p_x$ alors φ appartient à $\mathscr{S}'$ si et seulement si $x = a$;
- si $\varphi = \neg\psi$ alors φ appartient à $\mathscr{S}'$ si et seulement si ψ n'appartient pas à $\mathscr{S}'$;
- si $\varphi = \psi_1 \wedge \psi_2$, alors φ appartient à $\mathscr{S}'$ si et seulement si ψ_1 et ψ_2 appartiennent à $\mathscr{S}'$;
- si $\varphi = \psi_1 \vee \psi_2$, alors φ appartient à $\mathscr{S}'$ si et seulement si ψ_1 ou ψ_2 appartient à $\mathscr{S}'$;
- si $\varphi = \mathbf{X}\,\psi$, alors φ appartient à $\mathscr{S}'$ si et seulement si ψ appartient à $\mathscr{S}$;
- si $\varphi = \psi_1 \,\mathbf{U}\, \psi_2$, alors φ appartient à $\mathscr{S}'$ si et seulement ψ_2 appartient à $\mathscr{S}'$ ou ψ_1 appartient à $\mathscr{S}'$ et φ appartient à $\mathscr{S}$.

Question 14. De ces relations on déduit la fonction :

```
let rec maj s a = match s with
  | (AVRAI, _)          -> AVRAI, true
  | (APredicat c, _)    -> APredicat c, (a = c)
  | (ANON s1, _)        -> let (e1, b1) = maj s1 a in ANON (e1, b1), (not b1)
  | (AET (s1, s2), _) -> let (e1, b1) = maj s1 a and (e2, b2) = maj s2 a
                          in AET ((e1, b1), (e2, b2)), (b1 && b2)
  | (AOU (s1, s2), _) -> let (e1, b1) = maj s1 a and (e2, b2) = maj s2 a
                          in AOU ((e1, b1), (e2, b2)), (b1 || b2)
  | (AX s1, _)          -> let s = maj s1 a and _, b = s1
                          in AX s, b
  | (AU (s1, s2), b)  -> let (e1, b1) = maj s1 a and (e2, b2) = maj s2 a
                          in AU ((e1, b1), (e2, b2)), (b2 || b1 && b) ;;
```

On prouve la terminaison de cette fonction par induction structurelle sur φ. De plus, la fonction **`maj`** n'est appliquée qu'une fois à chacune des sous-formules de φ, donc la complexité de cette fonction est linéaire en la taille de la formule φ.

Question 15. Partant de l'ensemble vide des sous-formules de φ, on le met à jour en lisant une par une les lettres de u en commençant par la fin :

```
let rec sousFormulesVraies phi u =
  let rec aux acc = function
    | -1 -> acc
    | i  -> aux (maj acc u.[i]) (i-1)
  in aux (initialise phi) (string_length u - 1) ;;
```

Dans la fonction auxiliaire, l'accumulateur transporte l'ensemble des sous-formules successivement mis à jour au fur et à mesure de la lecture des lettres de u.

Sachant que la fonction **`maj`** a une complexité en $\mathrm{O}(|\varphi|)$, le coût de cette fonction est en $\mathrm{O}(|u| \times |\varphi|)$.

Question 16. De la fonction précédente il résulte immédiatement :

```
let veriteBis phi u = snd (sousFormulesVraies phi u) ;;
```

Question 17. On considère l'automate $\mathscr{A} = (A, Q, q_0, F, \delta)$ où :

- Q est l'ensemble des ensembles de sous-formules de φ ;
- q_0 est l'ensemble vide ;
- F est l'ensemble des ensembles de sous-formules de φ qui contiennent φ ;
- δ est la fonction `maj`.

On définit ainsi un automate déterministe complet à $2^{|\varphi|}$ états qui, d'après la question 16, reconnait le langage $\widetilde{L}_\varphi$ (puisque, on l'a dit à la question 15, la lecture du mot u doit se faire de la droite vers la gauche).

Question 18. Si on considère maintenant l'automate non déterministe $\mathscr{A}' = (A, Q, F, \{q_0\}, \delta')$ où δ' est défini par : $\delta'(q, a) = q' \iff \delta(q', a) = q$ on obtient un automate à $2^{|\varphi|}$ états qui reconnaît le langage miroir de $\widetilde{L}_\varphi$, à savoir L_φ.

Partie IV. Satisfiabilité et expressivité

Question 19. Considérons un automate fini déterministe $\mathscr{A} = (A, Q, q_0, F, \delta)$. Pour obtenir l'ensemble des états accessibles on réalise un parcours en profondeur à partir de l'état q_0 :

fonction ÉTATS_ACCESSIBLES(A, Q, q_0, F, δ)
  Acc $= \{q_0\}$
  Pile $\leftarrow q_0$
  tant que Pile $\neq \emptyset$ **faire**
    Pile $\rightarrow q$
    pour $c \in \{a, b\}$ **faire**
      $q' = \delta(q, c)$
      si $q' \notin$ Acc **alors**
        Pile $\leftarrow q'$
        Acc $\leftarrow q'$
  retourner Acc

(en ne tenant pas compte des transitions bloquantes lors du calcul de q').

Question 20. Nous avons vu à la question 18 l'existence d'un automate non déterministe $\mathscr{A}$ à $2^{|\varphi|}$ états qui reconnait L_φ. Il existe donc dans cet automate un chemin étiqueté par $u_{\min}$, que nous notons $q_0 \to q_1 \to \cdots \to q_\ell$ avec $\ell = |u_{\min}|$, qui mène d'un état initial q_0 à un état acceptant q_ℓ.
Si on avait $\ell \geqslant 2^{|\varphi|}$ il existerait $i < j$ tel que $q_i = q_j$. Mais alors il existerait un sous-mot v de $u_{\min}$, de longueur strictement inférieure à ℓ étiquetant le chemin $q_0 \to q_1 \to \cdots \to q_i = q_j \to q_{j+1} \to \cdots \to q_\ell$. Ce mot est reconnu par l'automate $\mathcal{A}$ donc appartient à L_φ, ce qui contredit le caractère minimal de $u_{\min}$. On en déduit que $|u_{\min}| \leqslant 2^{|\varphi|} - 1$.

Question 21. Pour cette question, il s'agit de rechercher un mot de longueur minimale qui satisfait φ. D'après la question précédente on peut se restreindre aux mots de longueurs strictement inférieures à $2^{|\varphi|}$, ce qui assure la terminaison de la recherche puisque l'alphabet est fini.
Cependant, un approche naïve conduit à un algorithme de complexité trop élevée : en effet, si on se contente de générer tous les mots possibles de longueur $\ell \leqslant 2^{|\varphi|} - 1$ en appliquant pour chacun d'eux la fonction `veriteBis`, la complexité totale sera en :

$$\sum_{\ell=1}^{2^{|\varphi|}-1} 2^\ell \times O(|\varphi| \times \ell) = O\Big(|\varphi| \sum_{l=1}^{2^{|\varphi|}-1} \ell \times 2^\ell\Big).$$

Sachant que $\displaystyle\sum_{\ell=1}^{n} \ell \times 2^\ell = (n-1)2^{n+1}$ ceci conduit à une complexité en $O\Big(|\varphi| 2^{|\varphi|} 2^{2^{|\varphi|}}\Big)$.

Pour obtenir un algorithme de la complexité demandée, il faut s'inspirer de la question 19 : déterminer s'il existe parmi les états accessibles d'un automate reconnaissant L_φ (ou plutôt $\widetilde{L}_\varphi$, ce qui revient au même) un état acceptant. Cet automate peut être construit en suivant la démarche initiée à la question 17.

```ocaml
let satisfiable phi =
  let borne = 1 lsl (taille phi) in
  let rec dfs dejavu = function
    | [] -> "Formule non satisfiable"
    | ((_, true), u)::_ -> u
    | (_, u)::q when string_length u = borne -> dfs dejavu q
    | (e, u)::q -> let s1 = maj e `a` and s2 = maj e `b` in
                   match (mem s1 dejavu, mem s2 dejavu) with
                   | (false, false) -> dfs (s1::s2::dejavu) ((s1, "a" ^ u)::(s2, "b" ^ u)::q)
                   | (false, true)  -> dfs (s1::dejavu)     ((s1, "a" ^ u)::q)
                   | (true, false)  -> dfs (s2::dejavu)     ((s2, "b" ^ u)::q)
                   | (true, true)   -> dfs dejavu            q
  in dfs [] [initialise phi, ""] ;;
```

La fonction **dfs** possède deux arguments : l'accumulateur **dejavu** (de type *ensemble list*) est la liste des ensembles de sous-formules de φ déjà rencontrés (autrement dit les états accessibles de l'automate). Le deuxième argument (de type *(ensemble * string) list*) est une liste d'ensembles de sous-formules en cours de traitement associées à des mots y menant. Le traitement d'un élément (e,u) de cette liste est le suivant :

- si cette liste est vide, la recherche est un échec ;
- si l'ensemble e contient la formule φ, c'est qu'on a $u \models \varphi$ (e est un état acceptant) ; la recherche s'achève ;
- si $|u| = 2^{|\varphi|}$, le mot est trop long, le couple (e,u) est supprimé ;
- dans les autres cas, on calcule les nouveaux ensembles de sous-formules associés aux mots $a\cdot u$ et $b\cdot u$. S'ils n'ont pas encore étés vus, ils sont ajoutés à la liste des éléments à traiter et le traitement se poursuit.

On sait qu'un parcours en profondeur a un coût en $O(n+p)$ où n est le nombre de sommets et p le nombre d'arêtes, mais à condition de pouvoir déterminer en coût constant si un sommet a déjà été vu, ce qui n'est pas le cas ici puisque j'ai représenté les éléments déjà vus par une liste. Le coût de la fonction **mem** étant linéaire, on obtient dans le cas qui nous intéresse un coût en $O(n(n+p))$.
Le nombre n de sommets est majoré par $2^{|\varphi|}$ et le nombres p d'arcs par $2\times 2^{|\varphi|}$ puisque l'alphabet est de cardinal 2, donc la complexité de cette fonction est en $O(2^{2|\varphi|})$.

Question 22. Commençons par montrer le résultat intermédiaire suggéré par l'énoncé, en prouvant par induction structurelle sur φ qu'il existe un rang N à partir duquel l'une des deux alternatives est vraie :

(i) pour tout $n \geqslant \mathrm{N}$, $a^n \models \varphi$;
(ii) pour tout $n \geqslant \mathrm{N}$, $a^n \not\models \varphi$.

D'après la question 10 on peut restreindre l'étude aux formules normalisées.

- Si $\varphi = \mathrm{VRAI}$, on dispose de la propriété (i) avec $\mathrm{N}=1$.
- Si $\varphi = p_a$, on dispose de la propriété (i) avec $\mathrm{N}=1$.

On considère désormais deux formules φ_1 et φ_2 pour lesquelles l'une des deux propriétés est vraie, à partir d'un rang N_1 pour φ_1 et d'un rang N_2 pour φ_2.

- Si $\varphi = \neg\varphi_1$, φ vérifie la propriété (i) avec $\mathrm{N}=\mathrm{N}_1$ lorsque φ_1 vérifie (ii), et vérifie la propriété (ii) sinon.
- Si $\varphi = \varphi_1 \vee \varphi_2$, φ vérifie la propriété (i) avec $\mathrm{N}=\max(\mathrm{N}_1,\mathrm{N}_2)$ lorsque φ_1 ou φ_2 vérifie (i), et vérifie (ii) sinon.
- Si $\varphi = \varphi_1 \wedge \varphi_2$, φ vérifie la propriété (i) avec $\mathrm{N}=\max(\mathrm{N}_1,\mathrm{N}_2)$ lorsque φ_1 et φ_2 vérifient (i), et vérifie (ii) sinon.
- Si $\varphi = \mathbf{X}\,\varphi_1$, φ vérifie la même propriété que φ_1 avec $\mathrm{N}=\mathrm{N}_1+1$.
- Si $\varphi = \varphi_1\,\mathbf{U}\,\varphi_2$, commençons par traduire la propriété $a^n \models \varphi$:

$$\begin{aligned} a^n \models \varphi_1\,\mathbf{U}\,\varphi_2 &\iff \exists j<n \,\big|\, (a^n,j)\models\varphi_2 \text{ et } i<j \Rightarrow (a^n,i)\models\varphi_1 \\ &\iff \exists j<n \,\big|\, a^{n-j}\models\varphi_2 \text{ et } i<j \Rightarrow a^{n-i}\models\varphi_1 \\ &\iff \exists j\in[\![1,n]\!] \,\big|\, a^{j}\models\varphi_2 \text{ et } j<i\leqslant n \Rightarrow a^{i}\models\varphi_1 \end{aligned}$$

Traitons alors plusieurs cas.

- Si φ_2 vérifie (i) alors φ vérifie aussi (i) avec $\mathrm{N}=\mathrm{N}_2$ (il suffit de poser $j=n$).
- Si φ_2 et φ_1 vérifient (ii) alors φ vérifie (ii) avec $\mathrm{N}=\max(\mathrm{N}_1,\mathrm{N}_2)$.
- Reste le cas où φ_2 vérifie (ii) et φ_1 vérifie (i).
  Supposons par exemple que φ ne vérifie pas (ii). Il existe donc $\mathrm{N}\geqslant\mathrm{N}_1$ tel que $a^{\mathrm{N}}\models\varphi$, et donc un entier $j\leqslant\mathrm{N}$ tel que $a^j\models\varphi_2$ et $j<i\leqslant\mathrm{N}\Rightarrow a^i\models\varphi_1$. Mais φ_1 vérifie (i) à partir du rang N_1, donc pour tout $i>j$, $a^i\models\varphi_1$, ce qui montre que $a^n\models\varphi$ pour $n\geqslant\mathrm{N}$. Ainsi, φ vérifie la propriété (i).

Une conséquence de cette propriété est que le langage L_φ doit être ou bien fini, ou bien contenir tous les mots a^n à partir d'un rang N. Il ne peut donc exister de formule φ telle que $L_\varphi = \left\{a^{2i} \,\middle|\, i \geqslant 1\right\}$.