Devoir surveillé n° 05 — corrigé

I Langages reconnaissables

Q 1) Soit $L \subseteq \Sigma^*$ un langage régulier. Alors il existe $N \in \mathbb{N}^*$ tel que pour tout mot $u \in L$ avec $|u| \geq N$, il existe $x, y, z \in \Sigma^*$ tel que $xyz = u$ avec $|xy| \leq N$ et $y \neq \epsilon$ et tel que $xy^*z \subseteq L$.

Q 2)

1. L_1 est dénoté par l'expression régulière a^*b^2 et est donc régulier.
2. Supposons que L_2 soit régulier, notons N la constante du lemme de l'étoile et considérons le mot $u = a^N b^{2N}$. Il existe une décomposition $u = xyz$ avec $|xy| \leq N$ et $y \neq \epsilon$ et donc $i \in \mathbb{N}$ et $j \in \mathbb{N}^*$ tels que $x = a^i$, $y = b^j$ et $z = a^{N-i-j}b^{2N}$. La conclusion du lemme de l'étoile indique que $xz = a^{N-j}b^{2N} \in L_2$ ce qui est absurde puisque $2(N-j) \neq 2N$ puisque $j \neq 0$. Le langage L_2 n'est donc pas régulier.
3. $L_3 = \Sigma^*\Sigma^* = \Sigma^*$ qui est bien sûr régulier.
4. On applique le lemme de l'étoile avec le mot $u = a^N b a^N b$ où N est la constante donnée par le lemme. On en déduit que $a^{N-j}ba^Nb$ est un carré, avec $j \neq 0$, ce qui est absurde. Le langage n'est donc pas régulier.

II Déduction naturelle

Q 3)

1. En logique minimale.

$$\dfrac{\dfrac{\dfrac{}{A \wedge B, B \wedge C \vdash A \wedge B}\,\text{ax}}{A \wedge B, B \wedge C \vdash A}\,\wedge_e^g \qquad \dfrac{\dfrac{}{A \wedge B, B \wedge C \vdash B \wedge C}\,\text{ax}}{A \wedge B, B \wedge C \vdash C}\,\wedge_e^d}{A \wedge B, B \wedge C \vdash A \wedge C}\,\wedge_i$$

2. En logique minimale. On note $\Gamma = \{A \to (B \vee C), \neg B, \neg C, A\}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash A \to (B \vee C)}\,\text{ax} \qquad \dfrac{}{\Gamma \vdash A}\,\text{ax}}{\Gamma \vdash B \vee C}\,\to_e \qquad \dfrac{\dfrac{}{\Gamma, B \vdash B}\,\text{ax} \qquad \dfrac{}{\Gamma, B \vdash \neg B}\,\text{ax}}{\Gamma, B \vdash \bot}\,\neg_e \qquad \dfrac{\dfrac{}{\Gamma, C \vdash C}\,\text{ax} \qquad \dfrac{}{\Gamma, C \vdash \neg C}\,\text{ax}}{\Gamma, C \vdash \bot}\,\neg_e}{\Gamma \vdash \bot}\,\vee_e}{A \to (B \vee C), \neg B, \neg C \vdash \neg A}\,\neg_i$$

3. En logique intuitionniste. On note $\Gamma = \{(A \vee C) \to (B \vee C), \neg C, A\}$.

$$(A \vee C) \to (B \vee C), \neg C \vdash A \to B$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash (A \vee C) \to (B \vee C)}\,\text{ax} \qquad \dfrac{\dfrac{}{\Gamma \vdash A}\,\text{ax}}{\Gamma \vdash A \vee C}\,\vee_i^g}{\Gamma \vdash B \vee C}\,\to_e \qquad \dfrac{}{\Gamma, B \vdash B}\,\text{ax} \qquad \dfrac{\dfrac{}{\Gamma, C \vdash C}\,\text{ax} \qquad \dfrac{}{\Gamma, C \vdash \neg C}\,\text{ax}}{\Gamma, C \vdash \bot}\,\neg_e}{\Gamma, C \vdash B}\,\bot_e}{\Gamma \vdash B}\,\vee_e}{(A \vee C) \to (B \vee C), \neg C \vdash A \to B}\,\to_i$$

4. En logique classique. On note $\Gamma = \{C \to B, \neg C \to \neg A, A\}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, C \vdash C \to B}\,\text{ax} \qquad \dfrac{}{\Gamma, C \vdash C}\,\text{ax}}{\Gamma, C \vdash B}\,\to_e \qquad \dfrac{\dfrac{}{\Gamma, \neg C \vdash A}\,\text{ax} \qquad \dfrac{\dfrac{}{\Gamma, \neg C \vdash \neg C \to \neg A}\,\text{ax} \qquad \dfrac{}{\Gamma, \neg C \vdash \neg C}\,\text{ax}}{\Gamma, \neg C \vdash \neg A}\,\to_e}{\Gamma, \neg C \vdash \bot}\,\neg_e}{\Gamma, \neg C \vdash B}\,\bot_e}{\Gamma \vdash B}\,\text{t.e.}}{C \to B, \neg C \to \neg A \vdash A \to B}\,\to_i$$

III Chemin de largeur maximale

Q 4) Le chemin Maldon-Clacton-Tiptree-Harwich-Blaxhall-Feering est un chemin de largeur maximale de largeur 29.

Q 5) On considère le graphe G^- obtenu en prenant l'opposé des poids des arêtes de G, c'est-à-dire que $G^- = (S, A, p^-)$ avec $p^- : A \to \mathbb{R}$ définie par $\forall a \in A$, $p^-(a) = -p(a)$. Chercher un arbre couvrant de poids maximal de G revient à chercher un arbre couvrant de poids minimal pour G^-. On peut en fait appliquer directement l'algorithme de KRUSKAL en triant la liste d'arêtes par poids *décroissants*. La complexité est en $O(|A| \log |S|)$ comme nous l'avons vu en cours. On considère les arêtes de poids : 53 (garde), 46 (garde), 46 (garde), 40 (garde), 31 (garde), 29 (garde) et on s'arrête là car on a ajouté 6 arêtes pour 7 sommets, sans jamais rejeter d'arêtes.

Q 6) Raisonnons par l'absurde et considérons $x, y \in S$ reliés par un chemin c de largeur maximale dans G de poids strictement plus grand que le chemin reliant x et y dans T. Notons $a \in B$ l'arête de poids minimal du chemin (unique puisque T est un arbre) entre x et y dans T. L'arête a a donc un poids strictement inférieure à toute arête de c. Considérons $T' = (S, B')$ avec $B' = B \setminus \{a\}$). T' comporte alors deux composantes connexes, l'une contenant x et l'autre y. Comme x et y sont reliés par c, il existe une arête $a' \notin B'$ sur le chemin c qui

comporte une extrémité dans chacune des deux composantes connexes de T'. Le sous-graphe $T'' = (S, B'')$ avec $B' \cup \{a'\}$ est donc un arbre couvrant de G. On a $p(T'') = p(T) - p(a) + p(a')$ mais par hypothèse $p(a') > p(a)$ et donc $p(T'') > p(T)$ ce qui est absurde puisque T est un arbre couvrant de poids maximal.

Q 7) Il suffit donc de chercher l'arbre couvrant de poids maximal, ce que l'on peut faire en $O(|A| \log |S|)$, puis de chercher l'unique chemin entre x et y, ce qui peut se faire par un simple parcours en $O(|A|)$ (on a ici $|S| = O(|A|)$ car le graphe est connexe). On obtient donc un algorithme en $O(|A| \log |S|)$.

Q 8) Si on connaît la largeur d'un chemin de largeur maximale entre x et y, on peut commencer par supprimer toutes les arêtes du graphe de poids strictement inférieur à cette largeur, ce qui se fait en temps linéaire. Il suffit ensuite de chercher un chemin entre x et y, qui existe par hypothèse, par un simple parcours en temps linéaire. On obtient bien un chemin de largeur maximale en $O(|A|)$.

IV Problème 2-SAT (en CAML)

Q 9) Aucune difficulté.

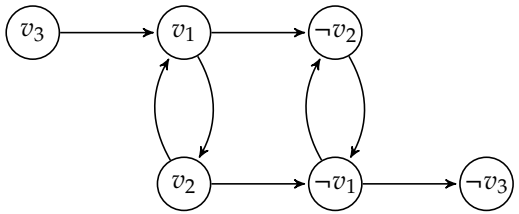
OCAML
```
let label = function Pos i | Neg i -> i
let neg = function
  | Pos i -> Neg i
  | Neg i -> Pos i
```

Q 10) On suit l'indication de l'énoncé.

OCAML
```
let rec indice_max formule =
  match formule with
  | [] -> 0
  | (l1, l2) :: suite ->
     max (max (label l1) (label l2)) (indice_max suite)

let nb_variable = indice_max
```

Q 11) On obtient le graphe suivant :



Q 12) Le graphe G_φ comporte $2n$ sommets et au plus $2m$ arêtes. Sa taille est donc bien linéaire en la taille de la formule $n + m$.

Q 13) Comme les clauses sont supposées distinctes, on ne peut jamais ajouter deux fois le même arc, sauf pour une clause comportant deux fois le même littéral. Il n'est donc pas nécessaire de faire de test avant l'ajout dans la liste d'adjacence.

OCAML
```
let indice = function
  | Pos i -> 2 * i - 2
  | Neg i -> 2 * i - 1

let graphe_implication f =
  let n = nb_variables f in
  let g = Array.make (2 * n) [] in
  let ajoute_clause (l1, l2) =
    let i = indice (neg l1) in
    g.(i) <- indice l2 :: g.(i);
    if l1 <> l2 then
      let j = indice (neg l2) in
      g.(j) <- indice l1 :: g.(j)
  in
  List.iter ajoute_clause f;
  g
```

Q 14) Si $(\ell_1, \ell_2) \in A$ alors $(\overline{\ell_1} \vee \ell_2)$ est une clause de φ. Une valuation qui satisfait φ satisfait donc cette clause $(\overline{\ell_1} \vee \ell_2)$ et donc $\ell_1 \to \ell_2$ qui lui est sémantiquement équivalente. Tout valuation qui satisfait φ satisfait donc $\ell_1 \to \ell_2$.

Q 15) Montrons ce résultat par récurrence sur n la longueur du chemin entre ℓ_1 et ℓ_2. Si $n = 0$ alors $\ell_1 = \ell_2$ et on a $\vDash \ell_1 \to \ell_1$ et donc $\varphi \vDash \ell_1 \to \ell_1$. Supposons le résultat vrai au rang $n + 1$ et notons ℓ le deuxième sommet du chemin de ℓ_1 à ℓ_2, avec donc $(\ell_1, \ell) \in A$ et le chemin de ℓ à ℓ_2 de longueur n. Par hypothèse de récurrence on a $\varphi \vDash \ell \to \ell_2$ et d'après la question précédente on a $\varphi \vDash \ell_1 \to \ell$. On conclut par transitivité de l'implication.

Q 16) Supposons qu'un littéral ℓ et sa négation $\overline{\ell}$ apparaissent dans une même composante fortement connexe. On a alors, d'après la question précédente, $\varphi \vDash \ell \to \overline{\ell}$ et $\varphi \vDash \overline{\ell} \to \ell$, donc $\varphi \vDash (\ell \to \overline{\ell}) \wedge (\overline{\ell} \to \ell)$. Or $(\ell \to \overline{\ell}) \wedge (\overline{\ell} \to \ell) \equiv \overline{\ell} \wedge \ell \equiv \bot$. On a donc $\varphi \vDash \bot$ et φ n'est pas satisfiable.

Q 17) Considérons une composante fortement connexe C et une valuation μ qui satisfait φ. Supposons qu'il existe un littéral $\ell \in C$ tel que $\mu(\ell) = 1$. Si ℓ' est un littéral de C, il existe un chemin de ℓ à ℓ' et donc $\mu \vDash \ell \to \ell'$. On a donc $\mu(\ell') = 1$. Ainsi soit un littéral, et donc tous, se voit affecté la valeur de vérité 1, soit tous les littéraux ont pour évaluation 0.

Q 18) On peut utiliser l'algorithme de Sharir-Kosaraju qui permet de trouver les composantes fortement connexes en $O(|S| + |A|)$. L'idée est d'effectuer un parcours en profondeur du graphe miroir et de classer les sommets par ordre de dernière visite décroissante. On peut ensuite effectuer une simple recherche de composantes connexes en utilisant cet ordre pour trouver les composantes fortement connexes.

Q 19) On peut directement utiliser l'ordre donné par premier parcours en profondeur dans l'algorithme de Sharir-Kosaraju. On peut aussi faire un tri topologique du graphe acyclique des composantes fortement connexes. Il faut simplement le faire une fois pour toute au début et pas à chaque étape. Dans les deux cas, on reste linéaire en la taille du graphe.

Q 20) Le nombre de sommets restant est un variant de cet algorithme qui termine donc. De plus, à la fin de cet algorithme, tous les sommets auront été supprimés et donc toutes les variables auront reçu une valeur de vérité par μ.

Q 21) Considérons une clause $(\ell_1 \vee \ell_2)$ de φ. Si $\mu(\ell_1)$ a été affecté lors du traitement de la composante fortement connexe de ℓ_1, alors on a choisi $\mu(\ell_1) = 1$ et la clause $(\ell_1 \vee \ell_2)$ est satisfaite par μ. Sinon, c'était pendant le traitement de la composante fortement connexe C de $\mu(\overline{\ell_1})$ et on a affecté $\mu(\overline{\ell_1}) = 1$. Comme $(\overline{\ell_1}, \ell_2) \in A$, soit ℓ_2 est dans la composante fortement connexe C de $\overline{\ell_1}$, soit ℓ_2 est dans une composantes fortement connexe C' avec $C \to C'$. Dans les deux cas on a $\mu(\ell_2) = 1$ et $(\ell_1 \vee \ell_2)$ est satisfaite par μ.

Q 22) Il suffit de vérifier s'il n'existe pas deux littéraux opposés dans une même composante fortement connexe. On ne demande pas ici de renvoyer une valuation lorsque la formule est satisfiable.

OCaml

```
exception Unsat

let satisfiable f =
  let n = nb_variables f in
  let g = graphe_implication f in
  let c = composantes_fortement_connexes g in
  try
    for i = 0 to n - 1 do
      if c.(2 * i - 2) = c.(2 * i - 1) then raise Unsat
    done;
    true
  with Unsat -> false
```

La complexité est bien linéaire en la taille de la formule si la construction du graphe et la recherche des composantes fortement connexes l'est.