Sujet Mines-Ponts 2025 : corrigé

1.

```
let cmp_letter x1 x2 = if x1=x2 then 0 else if x1=A then -1 else 1;;
```

2.

```
let rec cmp_word w1 w2 = match w1,w2 with
   | [],[] -> 0
   | [],_ -> -1
   | _,[] -> 1
   | x1::w3,x2::w4 -> if x1=x2 then cmp_word w3 w4 else cmp_letter x1 x2;;
```

3. La structure d'arbre binaire de recherche permet de le faire. Elle peut s'implémenter ainsi

```
type 'a dico = Vide | Noeud of word * 'a * 'a dico * 'a dico;;
```

Les clés d'un dictionnaire doivent être des mots deux à deux différents. La clé du noeud d'un dictionnaire doit être strictement supérieur aux clés de son sous-arbre gauche et strictement inférieur aux clés de sons sous-arbre droit.
4.

```
let delta_a = WordMap [] [A] (WordMap [A] [] WordMap.empty);;
let delta_b = WordMap [] [] (WordMap [A] [A] WordMap.empty);;
let un_a = WordMap [] false (WordMap [A] true WordMap.empty);;
```

5. Il s'agit du langage des mots qui ont un nombre impair de lettres a. Il est dénoté par l'expression régulière $b^*a(b|ab^*a)^*$.
6. Dans un automate de Glushkov, un état ne peut être le but que de transitions par une même lettre. Or chaque état est le but de transitions par a et b. Ce n'est donc pas l'automate de Glushkov d'une expression régulière.
7.

```
let automaton_example = { initial = [] ;
    transitions = (fun q x -> WordMap.find q (if x=A then delta_a else delta_b));
    finals = (fun q -> WordMap.find q un_a };;
```

8.

```
let rec delta_star a q w = match w with
   | [] -> q
   | x::y -> delta_star a (a.transitions q x) y;;
```

9.

```
let accepts a w = a.finals (delta_star a (a.initial) w);;
```

10.

```
let separated_by u1 u2 v = not(Oracle.mem (u1@v) = Oracle.mem (u2@v));;
```

11. Soit u_1 un mot. Pour tout mot v, $u_1v = u_1v$ donc $\mathbb{1}_\Lambda(u_1v) = \mathbb{1}_\Lambda(u_1v)$ donc v ne sépare pas u_1 et u_1 donc u_1 et u_1 sont inséparables. On a donc $u_1 \equiv_\Lambda u_1$.
Soit u_2 un mot. Supposons que $u_1 \equiv_\Lambda u_2$. Pour tout mot v, $\mathbb{1}_\Lambda(u_1v) = \mathbb{1}_\Lambda(u_2v)$ donc $\mathbb{1}_\Lambda(u_2v) = \mathbb{1}_\Lambda(u_1v)$ donc v ne sépare pas u_2 et u_1 donc $u_2 \equiv_\Lambda u_1$.
Soit u_3 un mot tel que $u_2 \equiv_\Lambda u_3$. Pour tout mot v, $\mathbb{1}_\Lambda(u_1v) = \mathbb{1}_\Lambda(u_2v)$ et $\mathbb{1}_\Lambda(u_2v) = \mathbb{1}_\Lambda(u_3v)$ donc $\mathbb{1}_\Lambda(u_1v) = \mathbb{1}_\Lambda(u_3v)$ donc v ne sépare pas u_1 et u_3 donc $u_1 \equiv_\Lambda u_3$.
12. Soit v un mot. Alors $\mathbb{1}_{L_A}(u_1v) = 1$ si et seulement si $u_1v \in L_A$ si et seulement si $\delta^*(q_0, u_1v) \in F$ si et seulement si $\delta^*(\delta^*(q_0, u_1)v) \in F$ si et seulement si $\delta^*(\delta^*(q_0, u_2), v) \in F$ si et seulement si $\delta^*(q_0, u_2v) \in F$ si et seulement si $u_2v \in L_A$ si et seulement si $\mathbb{1}_{L_A}(u_2v) = 1$. Par conséquent, u_1 et u_2 sont inséparables par rapport au langage L_A.
13. D'après le théorème de Kleene, les langages réguliers sont les langages reconnus par automate. L est régulier donc est reconnu par un automate $A = (Q, q_0, \delta, \mathbb{1}_F)$. Soit $n = \text{Card}(Q)$. Supposons qu'il existe au moins $n+1$ classes d'équivalences. Soient $u_1, \cdots, u_{n+1}$ des mots deux à deux séparables. Par contraposition du résultat de la question précédente, pour tout $1 \leq i \neq j \leq n+1$, $\delta^*(q_0, u_i) \neq \delta^*(q_0, u_j)$. On en déduit que $\text{Card}\{\delta^*(q_0, u_i) \,/\, 1 \leq i \leq n+1\} = n+1$. Or $\{\delta^*(q_0, u_i) \,/\, 1 \leq i \leq n+1\} \subset Q$ donc $\text{Card}(\{\delta^*(q_0, u_i) \,/\, 1 \leq i \leq n+1\} \leq \text{Card}(Q) = n$. On en déduit que les classes d'équivalence forment un ensemble fini de cardinal au plus $\text{Card}(Q)$.
14.

```
let states_of_disctree t =
     let rec aux acc t = match t with
        | Leaf(q) -> q::acc
        | Node(t1,_,t2) -> aux (aux acc t2) t1
     in aux [] t;;
```

Pour chaque noeud, il y a deux appels récursifs et pour chaque feuille un ajout d'élément en tête d'une liste. On en déduit, en notant n le nombre de noeud et f le nombre de feuilles, que la complexité est $2n + f$. En outre, $f = n + 1$ donc la complexité est $3f + 2 = O(f)$ donc est linéaire en le nombre de feuilles.
15. Le criblat de ba est b.
16.

```
let rec sift t u = match t with
    | Leaf(f) -> f
    | Node(t1,w,t2) -> if Oracle.mem (u@w) then sift t1 u else sift t2 u;;
```

17. La concaténation d'un mot u et d'un mot w est de complexité $O(|u|)$ si $|u|$ est la taille de u.
A chaque appel récursif de la fonction pour un mot u, l'appel sera sur un arbre de hauteur au moins un de moins et on effectuera une concaténation de u et d'un mot de taille au plus M ainsi qu'un appel à "Oracle.mem" sur un mot de taille au plus $|u| + M$.
Par conséquent, si t est un arbre de hauteur $h \geq 1$ et u un mot de taille $|u|$, la complexité $C(h, u)$ dans le pire des cas vérifie $C(h, u) = C(h - 1, u) + |u| + \mu(|u| + M)$. On en déduit que cette complexité est au pire $O(h(|u| + \mu(|u| + M)))$.
18. Supposons que u_1 et u_2 ont des criblats distincts. Il existe donc un discriminant de l'arbre v tel que $\mathbb{1}_L(u_1 v) \neq \mathbb{1}_L(u_2 v)$ donc u_1 et u_2 sont séparables.
19. Par contraposition, si deux mots sont inséparables, ils ont les mêmes criblats.
20. Pour chaque classe d'équivalence, il y a au plus un criblat donc une feuille. Le nombre de feuilles est donc majoré par le nombre de classes d'équivalence.
21. Supposons que u_1 et u_2 ont le même criblat. Pour tout discriminant v de l'arbre, $\mathbb{1}_L(u_1 v) = \mathbb{1}_L(u_2 v)$. Comme ϵ est le discriminant de la racine, $\mathbb{1}_L(u_1) = \mathbb{1}_L(u_2)$.
22.

```
let automaton_of_disctree t =
    { initial = [] ;
      transitions = fun q x -> sift t (q@[x]) ;
      finals = Oracle.mem };;
```

23. On peut construire un automate reconnaissant tous les mots (l'automate à un seul état avec transitions étiquetées par a, b vers ce même état) et un automate reconnaissant le langage vide (l'automate à deux états, l'un initial, l'autre final, des transitions étiquetées par a, b restant sur le même état). A l'aide de ces deux automates et de la fonction "Oracle.equiv", on peut trouver un mot w_1 du langage L et un mot w_2 qui n'est pas dans le langage L. Si le mot vide est dans L (ce que l'on sait grâce à la fonction "Oracle.mem"), on construit l'arbre à Noeud d'étiquette ϵ, à feuille gauche ϵ, à feuille droite w_2. Sinon, on construit l'arbre à Noeud d'étiquette ϵ, à feuille gauche w_1, à feuille droite ϵ.
24. Par hypothèse, $\mathbb{1}_F(c) \neq \mathbb{1}_L(c)$. Supposons que $c \in L$ (resp. $c \notin L$) et $c \notin F$ (resp. $c \in F$). $c \in L$ (resp. $c \notin L$) donc le criblat de c sera dans le sous-arbre gauche (resp. droit) ; $c = \pi_\gamma$ donc $\underline{p_\gamma}$ est dans le sous-arbre gauche (resp. droit). Le criblat de $\underline{p_\gamma}$ est dans le sous-arbre gauche (resp. droit) donc $p_\gamma \in L$ (resp. $p_\gamma \notin L$).
Par ailleurs, $c \in F$ (resp. $c \notin F$) donc $\delta^*(q_\epsilon, c)$ n'est pas final (resp. est final) ; par définition des états finals de l'automate, c'est un état du sous-arbre droit (resp. gauche) donc $\underline{\hat{p_\gamma}}$ est dans le sous-arbre droit (resp. gauche). Par conséquent, le mot $\underline{\hat{p_\gamma}}$ a pour criblat $\underline{\hat{p_\gamma}}$ qui est dans le sous-arbre droit (resp. gauche) donc $\hat{p_\gamma} \notin L$ (resp. $\hat{p_\gamma} \in L$).
Enfin, $\tau_\gamma = p_\gamma$, $\hat{\tau_\gamma} = \hat{p_\gamma}$ donc $\mathbb{1}_L(\tau_\gamma) \neq \mathbb{1}_L(\hat{\tau_\gamma})$.
25.

```
let split t c = let a = automaton_of_disctree t in
   let rec aux d f = let pp = sift t d and p = delta_star a [] d in
      if pp = p then let c::f0 = f in aux (d@[c]) f0
      else (pp,p,f) in aux [] c;;
```

26.

```
let rec substitude t (pp,p,s) = match t with
      | Leaf(q) when q=p ->
          if Oracle.mem pp then Node( Leaf(pp) , s, Leaf(p))
          else Node (Leaf(p), s, Leaf(pp))
      | Leaf(_) -> t
      | Node( tg, q, td) -> Node( substitute tg (pp,p,s), q, substitute td (pp,p,s));;
```

27. Tester si L est $\emptyset$ ou L est Σ^* (on a vu à la question 23 comment construire des automates reconnaissant $\emptyset$ et Σ^*). Si c'est le cas, on a vu à la question 23 comment construire des automates reconnaissant ces langages.
Si ce n'est pas le cas, considérer le crible T construit à la question 23.
Tant que l'automate associé à T ne reconnaît pas L, considérer un contre-exemple c. Remplacer T par le crible T'.
A chaque passage de boucle, le nombre de feuilles de T augmente de 1. D'après la question 20, le nombre de feuilles de T est majoré par le nombre de classes d'équivalence de L. Le nombre de passage de boucle est donc fini (et majoré par le nombre de classes d'équivalence de L). Ceci assure la terminaison de l'algorithme.
28. D'après la question précédente, cet automate a au plus un nombre d'état égal au nombre de classes d'équivalence de L. De plus, si u_1, u_2 sont séparables, d'après la question 12, $\delta^*(q_0, u_1) \neq \delta^*(q_0, u_2)$. Le nombre d'états d'un automate reconnaissant le langage à apprendre est donc au moins son nombre de classes d'équivalence.
Ceci prouve que l'automate construit par le procédé décrit à la question 27 a un nombre d'états minimal.