Mines-Ponts 2011 — corrigé

Partie I — Exercice sur les automates

Question I.1
L_1 est l'ensemble des mots sur $\{a, b\}$ d'au moins 2 lettres, dont la première et la dernière sont identiques. Ainsi `a((a|b)*)a|b((a|b)*)b` est une expression régulière décrivant L_1.

Question I.2
On propose l'automate de la figure 1.

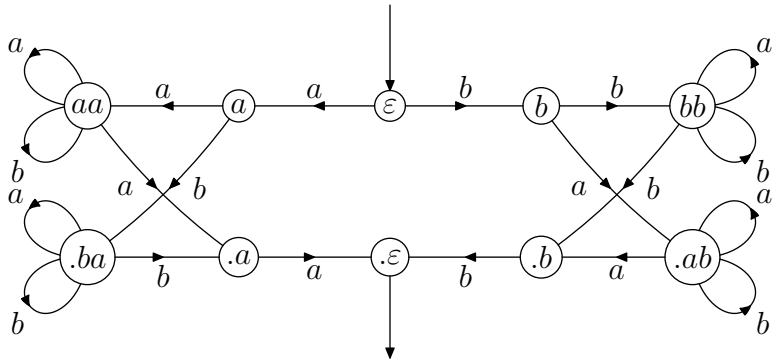

Figure 1 un automate non déterministe reconnaissant L_2

Question I.3
Le déterminisé de l'automate précédent compte 15 états. On trouvera son schéma en figure 2.
(On pourrait vérifier que l'automate obtenu est minimal.)

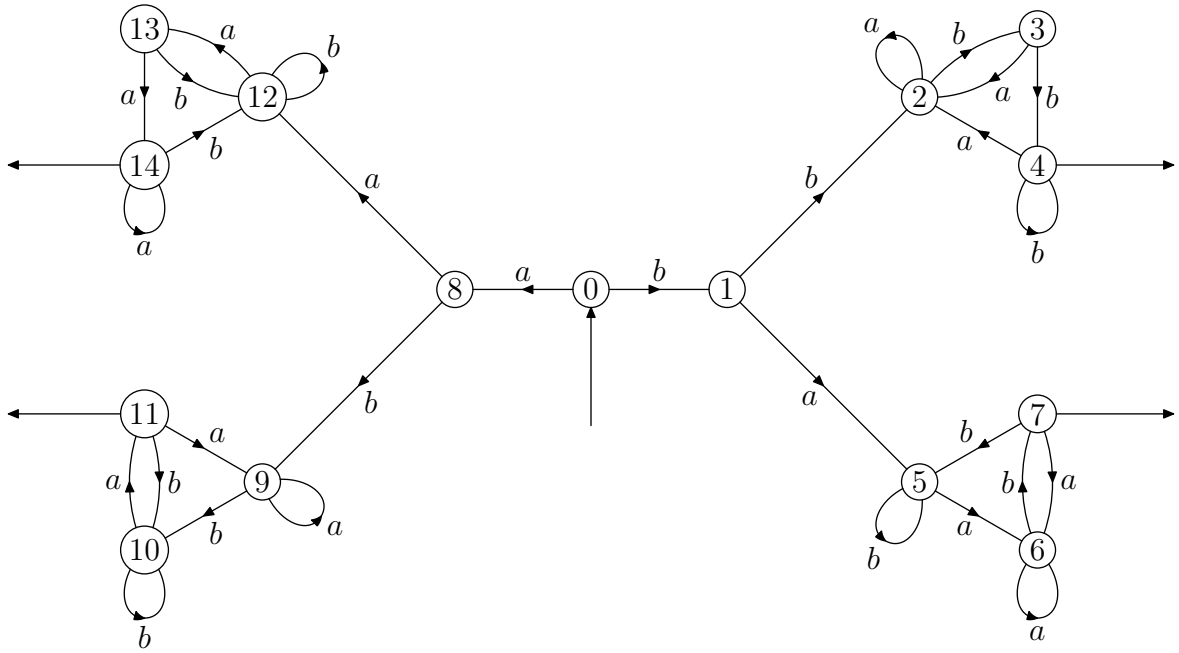

Figure 2 un automate déterministe reconnaissant L_2

Question I.4
Il suffit, pour montrer que L_n est rationnel, de remarquer que $L_n = \bigcup_{u\in\Sigma^n} u.\Sigma^*.\bar{u}$, réunion finie de langages rationnels.

Question I.5

Le plus simple est de remarquer que $L'_n = L_n \cup \bigcup_{k=0}^{2n-1} \Sigma^k$ est évidemment une réunion finie de langages rationnels, donc est également rationnel.

Question I.6
Soit P le langage des palindromes. Supposons qu'il soit rationnel : il existe un automate fini déterministe qui le reconnaît, d'état initial q_0.
Pour tout $n \geqslant 0$, $(ab)^n(ba)^n \in P$, donc il existe un état $q_n = q_0.(ab)^n$. L'automate a un nombre fini d'états, donc il existe $i < j$ tels que $q_i = q_j$. Alors $q_0.(ab)^i(ba)^j = q_i.(ba)^j = q_j.(ba)^j = q_0.((ab)^j(ba)^j)$ est un état final (car $(ab)^j(ba)^j \in P$), alors que $(ab)^i(ba)^j$ n'est pas un palindrome : c'est la contradiction espérée.
Ainsi P n'est pas rationnel.

Question I.7
Montrons que P est l'intersection des L'_n pour conclure.
Si u est un palindrome de longueur k, $u \in L'_n$ dès que $k < 2n$. Et si $1 \leqslant n \leqslant \dfrac{k}{2}$, le préfixe de longueur n de u est le transposé du suffixe de longueur n : on a bien l'inclusion $P \subset \bigcap_{n \geqslant 1} L'_n$.
Réciproquement, soit u un mot de l'intersection des L'_n, de longueur p. Si $|u| \leqslant 1$, c'est bien un palindrome. Si p est pair, $p = 2k \geqslant 2$, $u \in L'_k$ donc son préfixe v de longueur k est le transposé de son suffixe de longueur k, et $u = v\bar{v}$ est bien un palindrome. Enfin, si p est impair, $p = 2k + 1 \geqslant 3$, $u \in L'_k$ donc son préfixe v de longueur k est le transposé de son suffixe de longueur k, et $u = vc\bar{v}$ (où c est une lettre) est bien un palindrome.

Partie II — Ordres pour un tournoi

Méthode de Copeland

Question II.8
Les scores des 6 joueurs sont respectivement 3, 1, 3, 1, 4 et 3. Un classement de T_6 est donc $(4, 0, 2, 5, 1, 3)$.

Question II.9
La fonction `calculer_scores` proposée ci-dessous a clairement une complexité quadratique $O(n^2)$.

```
let calculer_scores T =
  let n = vect_length T in
  let s = make_vect n 0 in
    for i = 0 to n-1 do for j = 0 to n-1 do
      if T.(i).(j) then s.(i) <- s.(i) + 1
    done done ;
    s ;;
```

Programme 1 la fonction `calculer_scores`

Question II.10
Les algorithmes de tri sont au programme.
On peut supposer connue une fonction `tri : (int -> int -> bool) -> int vect -> int vect` telle que l'appel `tri compare v`, où `v` désigne un vecteur de n entiers, renvoie, par un calcul de complexité $O(n \lg n)$, un nouveau vecteur v' contenant les mêmes éléments rangés en ordre décroissant, pour l'ordre défini par $v_i \prec v_j$ si et seulement si `compare i j` est `true`.
Alors il suffit d'écrire :

```
let classement_Copeland T =
  let n = vect_length T in
  let v = init_vect n (function i -> i) in
  let s = calculer_scores T in
    tri (fun i j -> s.(i) < s.(j)) v ;;
```

Programme 2 la fonction `classement_Copeland`

La complexité est $O(n^2 + n \lg(n)) = O(n^2)$.

Valeur de Slater d'un classement

Question II.11
Je trouve $Slater(T_5, \sigma_5) = 4$; les parties contredites opposaient 0 et 2, 1 et 2, 3 et 4, 0 et 3.

Question II.12
Je trouve $Slater(T_6, \sigma_6) = 4$; les parties contredites opposaient 2 et 4, 0 et 2, 2 et 3, 2 et 5.

Question II.13
On écrit sans difficulté le programme 3, page 4.
Sa complexité est clairement $O(n^2)$.

```
let valeur_Slater T sigma =
  let n = vect_length sigma in
  let v = ref 0 in
    for i = 0 to n-2 do for j = i+1 to n-1 do
      if not T.(sigma.(i)).(sigma.(j)) then incr v
    done done ;
    !v ;;
```

Programme 3 la fonction `valeur_Slater`

Indice de Slater d'un tournoi

Question II.14
On vérifiera que $s(T_4) = 1$ et qu'un classement de Slater est $(3, 1, 2, 0)$.

Question II.15
Soit σ un classement quelconque d'un tournoi T.
Si $(i_1, i_2, \ldots, i_p)$ est un circuit de T, soit $k \in \{1, \ldots, p\}$ l'indice tel que i_k intervienne en premier dans le classement σ : mais (i_{k-1}, i_k) est un arc de T (avec la convention $i_0 = i_p$), et correspond à une partie qui contredit le classement σ.
On peut faire de même pour chaque circuit, et si les circuits considérés sont arc-disjoints, on obtient une nouvelle partie contredisant le classement σ.
Ainsi, pour tout classement σ, $Slater(T, \sigma) \geqslant m$ et donc $s(T) \geqslant m$.

Question II.16
Bien sûr, le raisonnement précédent montre que s'il existe un classement σ tel que $Slater(T, \sigma) = m$ c'est que σ est un classement de Slater et que $s(T) = m$.

Question II.17
Si $\sigma = (4, 0, 5, 3, 2, 1)$, on trouve $Slater(T_6, \sigma) = 3 < Slater(T_6, \sigma_6) = 4$. Ceci prouve déjà qu'un classement de Copeland n'est pas toujours de Slater.
Une étude soigneuse montre qu'on ne peut trouver que deux circuits arc-disjoints dans T_6, par exemple $(0, 5, 2)$ et $(2, 4, 3)$.
Or si $\sigma = (2, 4, 0, 5, 1, 3)$, on trouve $Slater(T_6, \sigma) = 2$, et l'étude précédente conclut que ce classement est un classement de Slater et que $s(T_6) = 2$.

Question II.18
Les triangles de T_5 sont $(0, 1, 3)$, $(0, 2, 3)$, $(0, 2, 4)$, $(1, 2, 4)$ et $(1, 3, 4)$.
On vérifie que $E = \{(0, 1, 3), (0, 2, 4)\}$ répond à la question.

Question II.19
On procède ainsi : on considère tour à tour les triangles du tournoi, par exemple dans l'ordre lexicographique, et on se rappelle les arcs déjà utilisés. Si le nouveau triangle examiné utilise un arc déjà pris, on l'abandonne, et sinon on marque les trois nouveaux arcs utilisés.
Le tableau `dispo` permet de savoir quels sont les arcs disponibles à chaque instant : on l'initialise en recopiant le tournoi lui-même.
Bien sûr, la complexité de `compter_triangles`, dans le programme 4 page 5, est $O(n^3)$.

Méthode de Slater

Question II.20
Après $(1, 2, 5, 4, 0, 3)$ viennent successivement $(1, 2, 5, 4, 3, 0)$, $(1, 3, 0, 2, 4, 5)$, $(1, 3, 0, 2, 5, 4)$, $(1, 3, 0, 4, 2, 5)$, $(1, 3, 0, 4, 5, 2)$, $(1, 3, 0, 5, 2, 4)$, $(1, 3, 0, 5, 4, 2)$ et $(1, 3, 2, 0, 4, 5)$.

Question II.21
On propose le programme 5, qui mérite quelques explications.
On commence par chercher (en ligne 4) l'index i tel que

$$\sigma_{i-1} < \sigma_i > \sigma_{i-1} > \sigma_{i-2} > \cdots > \sigma_{n-1}$$

c'est-à-dire qui correspond au plus grand suffixe strictement décroissant de la permutation considérée.

```
let copy_matrix m =
  init_vect (vect_length m) (function i -> copy_vect m.(i)) ;;

let compter_triangles T =
  let n = vect_length T in
  let dispo = copy_matrix T and compteur = ref 0 in
    for i = 0 to n-1 do for j = 0 to n-1 do for k = 0 to n-1 do
      if dispo.(i).(j) && dispo.(j).(k) && dispo.(k).(i) then
       ( incr compteur ;
          dispo.(i).(j) <- false ;
          dispo.(j).(k) <- false ;
          dispo.(k).(i) <- false )
    done done done ;
    !compteur ;;
```

Programme 4 la fonction `compter_triangles`

Il convient alors d'échanger $p = \sigma_{i-1}$ et σ_j où j est choisi de sorte que

$$\sigma_i > \cdots > \sigma_{j-1} > \sigma_j > p = \sigma_{i-1} > \sigma_{j+1} > \cdots > \sigma_{n-1}$$

ce dont se charge la ligne 8 (c'est facile car on se déplace dans une portion triée du tableau).
Les éléments de la fin du tableau sont alors triés en ordre décroissant et il suffit de retourner cette fin de tableau pour trouver la permutation cherchée : c'est le rôle de la boucle des lignes 10 à 12.

```
1   let permutation_suivante sigma =
2     let n = vect_length sigma in
3     let i = ref (n-1) in
4       while !i>0 && sigma.(!i) < sigma.(!i-1) do decr i done ;
5       if !i = 0 then false
6       else begin
7         let j = ref !i in
8          while !j < n-1 && sigma.(!j+1) > sigma.(!i-1) do incr j done ;
9          échange sigma (!i-1) !j ;
10         for k = !i to (n + !i-1)/2 do
11           échange sigma k (n + !i-1 - k)
12         done ;
13         true
14      end ;;
```

Programme 5 la fonction `permutation_suivante`

Question II.22
On propose le programme 6.
Le tableau `sigma` étant modifié à chaque appel de `permutation_suivante`, on garde trace du minimum courant et du classement qui réalise ce minimum courant, qu'on conserve sous forme d'une liste (on aurait pu également utiliser `copy_vect`).

```
let classement_Slater T =
  let n = vect_length T in
  let sigma = init_vect n (function i -> i) in
  let rec boucle meilleur m =
    if permutation_suivante sigma then
      let m' = valeur_Slater T sigma in
       if m' < m then boucle (list_of_vect sigma) m'
       else boucle meilleur m
    else vect_of_list meilleur
  in
    boucle (list_of_vect sigma) (valeur_Slater T sigma) ;;
```

Programme 6 la fonction `classement_Slater`