Corrigé : automates en ligne (X 1998)

Partie I. Motifs et automates

Question 1.

a) Le motif `cl?ou` filtre les mots "cou" et "clou" ; le motif `c?ou*cou` filtre tous les mots débutant par "ou" ou par "cou" et se terminant par "cou" à l'exception de "cou".

b) Le motif `**` filtre le mot m = "abc" de deux façons différentes, avec m_1 = "a" et m_2 = "bc" ou m_1 = "ab" et m_2 = "c".

c) Raisonnons par récurrence sur la longueur $|p|$ du motif p.

- Si $|p| = 0$, p est le motif vide, donc p_1 et p_2 aussi. Le mot m est nécessairement le mot vide, et il suffit de poser $m_1 = m_2 = \varepsilon$ pour conclure.
- Si $|p| > 0$, supposons le résultat acquis aux rangs inférieurs. Si p_1 est le motif vide il suffit de poser $m_1 = \varepsilon$ et $m_2 = m$ pour conclure. Si p_2 est le motif vide on pose cette fois $m_1 = m$ et $m_2 = \varepsilon$. On suppose désormais que ni p_1 ni p_2 n'est le motif vide, avec pour conséquence que p n'est pas un motif simple. Il se décompose donc sous la forme $p = p_3p_4$ où ni p_3 ni p_4 n'est le motif vide, et puisque p filtre m il existe une factorisation $m = m_3m_4$ de m telle que p_3 filtre m_3 et p_4 filtre m_4.

  Supposons $|p_1| \leqslant |p_3|$. D'après le lemme de Levi il existe un motif p' tel que $p_3 = p_1p'$ et $p_2 = p'p_4$. Par hypothèse de récurrence appliquée à p_3 il existe une factorisation de m_3 sous la forme : $m_3 = m_1m'$ telle que p_1 filtre m_1 et p' filtre m'. Mais alors p_2 filtre le mot $m'm_4$. Posons $m_2 = m'm_4$. Alors p_1p_2 filtre le mot m_1m_2 et $m_1m_2 = m_1m'm_4 = m_3m_4 = m$.

  Le cas où $|p_3| \leqslant |p_1|$ se traite de la même façon.

d) Les motifs `*?` et `**` sont équivalents à `*` qui est de poids minimal puisque le motif vide ne filtre que le mot vide.

Le motif `a???` est équivalent à `a?` car ces deux motifs ne filtrent que les mots ε et "a", et le second motif est de poids minimal car aucun motif de poids égal à 1 ne peut filtrer ces deux mots et uniquement ceux-ci.

Question 2.

a) Si p est un motif simple il est équivalent à α, $*$ ou à $\alpha?$ où α désigne une lettre quelconque de l'alphabet, et les automates associés sont représentés ci-dessous.

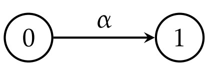


Figure 1 – L'automate $\mathcal{A}(\alpha)$.

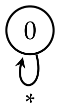


Figure 2 – L'automate $\mathcal{A}(*)$.

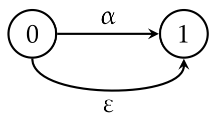


Figure 3 – L'automate $\mathcal{A}(\alpha?)$.

b) Si p est le motif vide, l'automate $\mathcal{A}(\emptyset)$ qui le reconnait est l'automate vide.

c) Si $p = p_1p_2$, soit $\mathcal{A}(p_1)$ et $\mathcal{A}(p_2)$ deux automates qui reconnaissent respectivement p_1 et p_2 et dont les états sont distincts. On construit l'automate $\mathcal{A}(p)$ en identifiant l'état final de $\mathcal{A}(p_1)$ avec l'état initial de $\mathcal{A}(p_2)$ (ou en les reliant par une transition instantanée).

d) Enfin, l'automate associé au motif `c?ou*cou` est représenté ci-dessous.

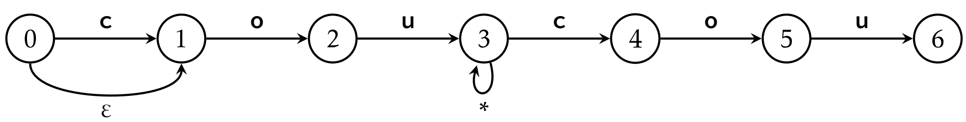


Figure 4 – L'automate $\mathcal{A}$(`c?ou*cou`).

Question 3. Commençons par observer que si a est un entier qui code un ensemble d'états A et α une lettre alors $b_\alpha = (a \wedge \mathrm{N}_\alpha) \ll 1$ code l'ensemble des états atteints après une transition étiquetée par α.
Observons ensuite que $b_* = a \wedge \mathrm{N}_*$ code l'ensemble des états atteints après une transition étiquetée par $*$.
$\alpha(\mathrm{A})$ est donc représenté par l'entier $b = b_\alpha \vee b_*$.
Observons enfin que si b_i est un ensemble d'états, $b_{i+1} = b_i \vee ((b_i \wedge \mathrm{N}_\varepsilon) \ll 1)$ code l'ensemble des états atteints après une transition instantanée. Considérons donc la suite définie par :

$$b_0 = b \quad \text{et} \quad b_{i+1} = b_i \vee ((b_i \wedge \mathrm{N}_\varepsilon) \ll 1)$$

Cette suite est croissante donc stationnaire puisque majorée par $2^n - 1$ où n désigne le nombre d'états de l'automate. Sa limite a' représente l'ensemble des états $\varepsilon(\alpha(\mathrm{A})) = \mathrm{A}'$.

Partie II. Reconnaissance de motifs

Question 4. Pour calculer plus aisément le code d'un caractère c dans le tableau **alpha** il peut être utile de définir la fonction :

```
let indice c = int_of_char c - int_of_char 'a' ;;
```

qui indique l'indice i du tableau **alpha** où se trouve rangé l'entier N_c.

a) D'après la question 3 les deux fonctions demandées peuvent être définies comme suit :

```
let mange etats c =
  let b1 = (etats land (alpha.(indice c))) lsl 1 and b2 = etats land !etoile in
  b1 lor b2 ;;
```

```
let rec ferme etats =
  let e = etats lor ((etats land !epsilon) lsl 1) in
  if etats = e then e else ferme e ;;
```

b) Toujours d'après cette même question, on obtient le code de l'état $\mathrm{A}' = \varepsilon(\alpha(\mathrm{A}))$ à l'aide de la fonction :

```
let etape etats c = ferme (mange etats c) ;;
```

c) Sachant que les opérations sur les nombres entiers sont de coût constant, la fonction **mange** est de coût constant. Lorsque le motif p ne contient pas le caractère **?** le coût de la fonction **etape** est donc lui aussi constant.
Le coût de la fonction **ferme** est linéaire en le nombre de transitions instantanées consécutives présentes dans le motif p. Ce nombre est majoré par $|p|$ donc dans le cas général le coût de la fonction **etape** est un $\mathrm{O}(|p|)$.

d) La présence de l'état e_{n-1} parmi l'ensemble d'états A représenté par l'entier $b = (b_i)$ est caractérisé par l'égalité $b_{n-1} = 1$. Sachant que $1 \ll (n-1) = (00\cdots001\underbrace{00\cdots00}_{n-1})_2$ on définit la fonction :

```
let etat_final etats = etats land (1 lsl (!n_etats - 1)) <> 0 ;;
```

Question 5. Pour respecter les spécifications de l'énoncé on peut définir :

```
exception Echec of string ;;
let echoue s = raise (Echec s) ;;
```

a) Pour construire l'automate $\mathcal{A}(p)$ on parcourt les différents caractères qui composent p en gardant en mémoire le dernier état e_i créé.
Un caractère alphabétique α entraîne la création d'un nouvel état e_{i+1} et d'une transition alphabétique (e_i, e_{i+1}, α). Un caractère $*$ entraîne la création d'une transition étoile $(e_i, e_i, *)$. Un caractère ?, s'il est précédé d'un caractère alphabétique, entraîne la création d'une transition instantanée $(e_i, e_{i-1}, \varepsilon)$ (ce qui suppose $i \geqslant 1$) ; dans les autres cas il est ignoré. Tout autre caractère déclenche l'exception **Echec**.
Enfin, le nombre d'états est égal au nombre de caractères alphabétiques présents dans p donc la référence **n_etats** est incrémentée à chaque fois qu'on en rencontre un.

```
let construire_auto p =
  let rec aux i k =
    if k < string_length p then match p.[k] with
    | c when indice c >= 0 && indice c < 26 ->
                incr n_etats ;
                alpha.(indice c) <- alpha.(indice c) lor (1 lsl i) ;
                aux (i+1) (k+1)
    | `*` -> etoile := !etoile lor (1 lsl i) ; aux i (k+1)
    | `?` when k = 0 -> echoue "motif_invalide"
    | `?` when p.[k-1] = `?` || p.[k-1] = `*` -> aux i (k+1)
    | `?` -> epsilon := !epsilon lor (1 lsl (i-1)) ; aux i (k+1)
    | _ -> echoue "motif_invalide"
  in fill_vect alpha 0 26 0 ;
     etoile := 0 ; epsilon := 0 ; n_etats := 1 ;
     aux 0 0 ;;
```

b) Chacune des opérations qui précède les appels récursifs à la fonction **aux** est de coût constant donc le coût de cette construction est un $O(|p|)$.

Question 6.

a) La fonction qui suit construit l'automate associé au motif p, calcule à partir de l'ensemble d'états initiaux $A_0 = \varepsilon(\{e_0\})$ l'ensemble d'états auquel on aboutit à la lecture du mot m, puis détermine si l'état final se trouve parmi eux.

```
let filtre p m =
  let rec aux etats = function
    | k when k = string_length m -> etats
    | k  -> aux (etape etats m.[k]) (k+1)
  in construire_auto p ; etat_final (aux (ferme 1) 0) ;;
```

b) Nous avons vu que le coût de la construction de l'automate est un $O(|p|)$ et que le coût de la fonction **etape** est un $O(1)$ s'il n'y a pas de motif optionnel, un $O(|p|)$ sinon. On en déduit que le coût de la fonction **filtre** est un $O(|p|+|m|)$ s'il n'y a pas de motif optionnel et un $O(|p|\cdot|m|)$ s'il en existe.

Question 7.

a) Posons $q = *p*$; alors q filtre m si et seulement si p filtre un sous-mot de m. D'où la fonction :

```
let filtre_sous_mot p = filtre ("*" ^ p ^ "*") ;;
```

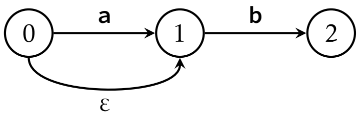
b) La lecture du préfixe m_i de m aboutit à l'ensemble d'états A_i donc p filtre m_i si et seulement si l'état final de $\mathcal{A}(p)$ est dans A_i.
Posons $B_0 = \varepsilon(\{e_0\})$ et $B_{i+1} = \varepsilon(\alpha_i(B_i)) \cup B_0$, et montrons par récurrence sur i que B_i est l'ensemble des états auquel peut aboutir la lecture d'un sous-mot $\alpha_j\alpha_{j+1}\cdots\alpha_{i-1}$.
   - Si $i = 0$ le sous-mot est vide et les états auquel on peut aboutir à partir de l'état e_0 sont $\varepsilon(\{e_0\}) = B_0$.
   - Si $i \geqslant 0$ supposons le résultat acquis au rang i. Par hypothèse de récurrence B_i est l'ensemble des états auquel aboutit la lecture d'un sous-mot $\alpha_j\cdots\alpha_{i-1}$ donc $\varepsilon(\alpha_i(B_i))$ est l'ensemble des états auquel aboutit la lecture d'un sous-mot *non vide* de la forme $\alpha_j\cdots\alpha_i$. Sachant que $\varepsilon(\{e_0\}) = B_0$ est l'ensemble des états auquel aboutit la lecture du sous-mot vide, on en déduit le résultat au rang $i+1$.

   De ceci il résulte immédiatement qu'un sous-mot $\alpha_j\cdots\alpha_{i-1}$ est filtré par p si et seulement si l'état final de $\mathcal{A}(p)$ appartient à B_i.

c) On construit l'automate associé au mot (qui est aussi un motif) s puis pour chaque valeur de i on détermine si un sous-mot de m de la forme $m_j\cdots m_{i-1}$ est reconnu par le motif s (notons que puisque s est uniquement formé de caractères alphanumériques, un seul sous-mot de cette forme a une chance d'être reconnu, celui pour lequel $i-j=|s|$).
Notons que puisqu'il n'y a pas de transition instantanée on a $B_0 = \{e_0\}$ et $B_{i+1} = \alpha_i(B_i) \cup \{e_0\}$.

```
let compte_sous_mots s m =
  construire_auto s ;
  let b = ref 1 in
  let nb = ref 0 in
  for k = 0 to string_length m - 1 do
    b := etape !b m.[k] lor 1 ;
    if etat_final !b then incr nb ;
  done ;
  !nb ;;
```

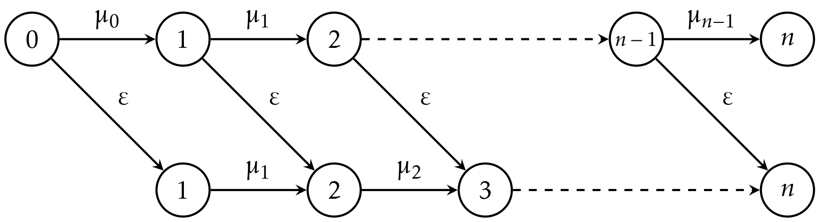
d) Considérons le motif $p =$ **a?b** et le mot $m =$ **ab**. L'automate $\mathcal{A}(p)$ est représenté ci-dessous :



À la lecture du mot m on a $B_0 = \{0,1\}$, $B_1 = \{0,1\}$ et $B_2 = \{0,1,2\}$ et seul B_2 contient l'état final alors que deux sous-mots de m sont filtrés par le motif p : **b** et **ab**.

Question 8.

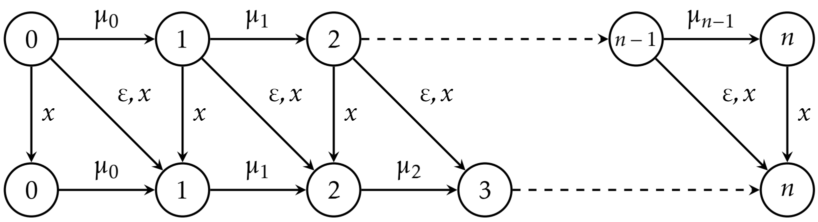
a) Pour détecter au plus un oubli il faut autoriser une transition instantanée *et une seule* entre deux états consécutifs de $\mathcal{A}(\mu)$. Ceci peut être représenté graphiquement en dupliquant les états $e_1, \dots, e_{n-1}$ comme ci-dessous :



où on a noté $\mu = \mu_0\mu_1\cdots\mu_{n-1}$ les lettres de μ.

Considérons maintenant les lettres $\alpha_0, \alpha_1, \dots, \alpha_k$ de m, et la suite d'états (A_i) définie par $A_0 = \{e_0\}$ et $A_{i+1} = \alpha_i(A_i)$. Le mot m est reconnu sans erreur si et seulement si $e_n \in A_k$ (avec nécessairement $k = n$).
On pose ensuite $A_0^1 = \{e_1\}$ et $A_{i+1}^1 = \alpha_i(A_i^1) \cup \operatorname{succ}(A_i)$ où $\operatorname{succ}(A_i)$ représente les successeurs des états contenus dans A_i. Alors le mot m est reconnu avec un oubli si et seulement si $e_n \in A_k^1$ (avec cette fois $k = n-1$).
Le mot m est donc reconnu avec au plus un oubli si et seulement si $e_n \in A_k \cup A_k^1$.

b) Le cas d'une erreur quelconque peut être représenté par le diagramme ci-dessous :



où x représente un caractère quelconque. Les transitions (e_i, e_i, x) représentent des insertions, les transitions (e_i, e_{i+1}, x) des substitutions. La suite (A_i^1) doit être modifiée en posant $A_0^1 = \{e_1\}$ et $A_{i+1}^1 = \alpha_i(A_i^1) \cup \operatorname{succ}(A_i) \cup A_i \cup \operatorname{succ}(A_{i+1})$ (le terme $\operatorname{succ}(A_i)$ autorise un oubli, A_i une insertion, $\operatorname{succ}(A_{i+1})$ une substitution).

c) Si l'ensemble d'états A est représenté par l'entier n, on a $\operatorname{succ}(A) = n \ll 1$. D'où la fonction :

```
let filtre_erreur mu m =
  construire_auto mu ;
  let a = ref 1 and a1 = ref 2 in
  for k = 0 to string_length m - 1 do
    let x = !a in
    a := etape x m.[k] ;
    a1 := (etape !a1 m.[k]) lor (x lsl 1) lor x lor (!a lsl 1)
  done ;
  etat_final (!a lor !a1) ;;
```

d) Enfin, en combinant la démarche établie aux questions 7.b-c avec celle établie aux questions 8.b-c on obtient la fonction :

```
let sous_mot_erreur mu m =
  construire_auto mu ;
  let a = ref 1 and a1 = ref 2 in
  let rep = ref false in
  for k = 0 to string_length m - 1 do
    let x = !a in
    a := etape x m.[k] lor 1 ;
    a1 := (etape !a1 m.[k]) lor (x lsl 1) lor x lor (!a lsl 1) lor 2 ;
    rep := !rep || etat_final (!a lor !a1)
  done ;
  !rep ;;
```

♦ ♦

♦