Correction du Devoir Surveillé du Samedi 14 Mars

Exercice 1

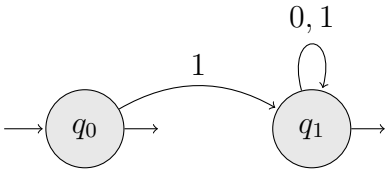
1. (a) 41 s'écrit 101001 en base 2 (en utilisant par exemple la méthode par divisions successives par 2).

   (b) 10101010 représente l'entier $2^7 + 2^5 + 2^3 + 2 = 170$.

   (c) Un automate est local si deux transitions étiquetées par la même lettre aboutissant au même état.

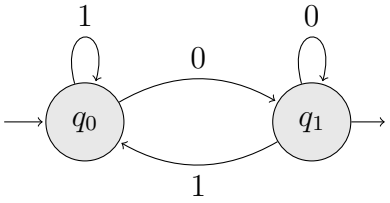
   Un automate est standard s'il n'a qu'un seul état initial et qu'aucun transition n'aboutit sur cet état initial.

   (d) Voici l'automate $\mathcal{A}_1$ demandé :



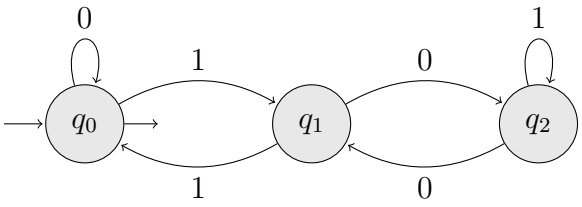
   (e)
```
let langage_1 m = match m with
  | [] -> true
  | a::_ -> a
;;
```

2. (a) Voici l'automate $\mathcal{A}_2$ demandé :



   (b)
```
let rec langage_2 m = match m with
  | [] -> false
  | [a] -> not a
  | a::u -> langage_2 u
;;
```

3. (a) Voici l'automate $\mathcal{A}_3$ demandé :



(b)
```
let langage_3 m =
  let rec delta_etoile etat mot = match mot with
    | [] -> etat = 0
    | a :: u -> delta_etoile ((2*etat + (if a then 1 else 0)) mod 3) u
  in delta_etoile 0 m
;;
```

(c) Par récurrence :

Ini. $\mathscr{P}(1)$ signifie : "$\forall w_1 \in A,\ \delta^\star(0, w_1) = w_1 \bmod 3$" ce qui est vrai car $\delta^\star(0, w_1) = \delta(\delta^\star(0, \varepsilon), w_1) = \delta(0, w_1) = (2 \times 0 + w_1) \bmod 3 = w_1 \bmod 3$.

Héré. Soit $n \in \mathbb{N}^*$. Supposons $\mathscr{P}(n)$ et montrons $\mathscr{P}(n+1)$.

Soient $w_1, w_2, \ldots, w_{n+1} \in A$. Alors :

$$\begin{aligned}
\delta^\star(0, w_1 w_2 \ldots w_{n+1}) &= \delta(\delta^\star(0, w_1 w_2 \ldots w_n), w_{n+1}) \\
&= \delta\left(\sum_{k=1}^{n} w_k 2^{n-k}, w_{n+1}\right) \quad \text{(d'après } \mathscr{P}(n)\text{)} \\
&= \left(2\sum_{k=1}^{n} w_k 2^{n-k} + w_{n+1}\right) \bmod 3 \\
&= \sum_{k=1}^{n+1} w_k 2^{n+1-k} \bmod 3
\end{aligned}$$

On a donc bien montré $\mathscr{P}(n+1)$.

Ccl. D'après le principe de récurrence, $\mathscr{P}(n)$ est donc vraie pour tout $n \in \mathbb{N}^*$.

4. (a) Clairement L_2 est l'ensemble des écritures en base 2 des entiers pairs. D'après 3.(c), L_3 est l'ensemble des écritures en base 2 des entiers congrus à 0 modulo 3 (c'est-à-dire multiples de 3).

D'après le théorème chinois, comme 2 et 3 sont premiers entre eux :

$$(k \equiv 0 \bmod 2) \text{ et } (k \equiv 0 \bmod 3) \quad \Leftrightarrow \quad k \equiv 0 \bmod 6.$$

Donc $L_4 = L_1 \cap L_2 \cap L_3$ est l'ensemble des écritures en base 2 des entiers multiples de 6.

(b) D'après le cours, l'intersection de langages reconnaissables est reconnaissable. Comme L_1, L_2, L_3 sont tous les trois reconnaissables (on a donné des automates pour chacun d'entre eux), $L_4 = L_1 \cap L_2 \cap L_3$ est reconnaissable.

Exercice 2

1. (a) $P(L) = \left\{ a \in \Sigma \mid a\,\Sigma^* \cap L \neq \varnothing \right\}$

$S(L) = \left\{ a \in \Sigma \mid \Sigma^*\, a \cap L \neq \varnothing \right\}$

$F(L) = \left\{ u \in \Sigma^2 \mid \Sigma^*\, u\, \Sigma^* \cap L \neq \varnothing \right\}$, $N(L) = \Sigma^2 \backslash F(L)$.

(b) Nous avons : $L\backslash\{\varepsilon\} \subset \Big(P(L)\Sigma^* \cap \Sigma^* S(L)\Big)\backslash\Big(\Sigma^* N(L) \Sigma^*\Big)$.

Soit ω appartenant à $L\backslash\{\varepsilon\}$, en notant n la longueur du mot ω,

n est non nul et il existe $a_1, \ldots, a_n$ appartenant à Σ tels que : $\omega = a_1\, a_2\, \ldots\, a_n$.

En notant : $u = \begin{cases} a_2\, \ldots\, a_n & , n \geqslant 2 \\ \varepsilon & , n = 1 \end{cases}$, nous avons : $\omega = a_1\, u$,

d'une part, u appartient à Σ^* donc $\omega = a_1\, u$ appartient à $a_1\, \Sigma^*$,

d'autre part, ω appartient à L,
donc ω appartient à $a_1\,\Sigma^* \cap L$, donc $a_1\,\Sigma^* \cap L$ est non vide, donc a_1 appartient à $P(L)$,
donc $\omega = a_1\,u$ appartient à $P(L)\,\Sigma^*$.
En notant : $v = \begin{cases} a_1 \ldots a_{n-1} & , n \geqslant 2 \\ \varepsilon & , n = 1 \end{cases}$, le lecteur montre que $\omega = v\,a_n$ appartient à $\Sigma^*\,S(L)$.
Donc ω appartient à $P(L)\Sigma^* \cap \Sigma^* S(L)$.
Supposons que ω appartient à $\Sigma^* N(L)\Sigma^*$, il existe donc u appartenant à $N(L)$, v et w appartenant à Σ^* tels que : $\omega = v\,u\,w$,
alors u appartient à Σ^2 et $\omega = v\,u\,w$ appartient à $\Sigma^* u \Sigma^* \cap L$, donc $\Sigma^* u \Sigma^* \cap L$ est non vide, donc u appartient à $F(L)$: CONTRADICTION !
Donc ω n'appartient pas à $\Sigma^* N(L)\Sigma^*$.
Donc ω appartient à $\Big(P(L)\Sigma^* \cap \Sigma^* S(L)\Big) - \Big(\Sigma^* N(L)\Sigma^*\Big)$.
Donc : $L\backslash\{\varepsilon\} \subset \Big(P(L)\Sigma^* \cap \Sigma^* S(L)\Big)\backslash\Big(\Sigma^* N(L)\Sigma^*\Big)$.

(c) Dans ce cas, le langage L est dit local.

2. (a) $P(\{\varepsilon\}) = \varnothing$, $S(\{\varepsilon\}) = \varnothing$, $F(\{\varepsilon\}) = \varnothing$.
   (b) $P(\{a\}) = \{a\}$, $S(\{a\}) = \{a\}$, $F(\{a\}) = \varnothing$.
   (c) $P(L_1 \cup L_2) = P(L_1) \cup P(L_2)$, $S(L_1 \cup L_2) = S(L_1) \cup S(L_2)$,
   $F(L_1 \cup L_2) = F(L_1) \cup F(L_2)$.
   (d) $P(L_1.L_2) = \begin{cases} P(L_1) & , \varepsilon \notin L_1 \\ P(L_1) \cup P(L_2) & , \varepsilon \in L_1 \end{cases}$,
   $S(L_1.L_2) = \begin{cases} S(L_2) & , \varepsilon \notin L_2 \\ S(L_1) \cup S(L_2) & , \varepsilon \in L_2 \end{cases}$,
   $F(L_1.L_2) = F(L_1) \cup F(L_2) \cup S(L_1)P(L_2)$.
   (e) $P(L^*) = P(L)$, $S(L^*) = S(L)$, $F(L^*) = F(L) \cup S(L)P(L)$.
3. Voici la représentation de $(a+b)^*.a$ en OCaml :

```
let er_ex= Concat (Kleene (Sum (Const "a", Const "b")), Const "a") ;;
```

4. (a)
```
let rec mot_vide e = match e with
  | Epsilon -> true
  | Const _ -> false
  | Sum (e1,e2) -> (mot_vide e1) || (mot_vide e2)
  | Concat (e1,e2) -> (mot_vide e1) && (mot_vide e2)
  | Kleene _ -> true
;;
```
   (b)
```
let rec reunion l1 l2 = match l2 with
  | [] -> l1
  | t2::q2 -> if (List.mem t2 l1) then reunion l1 q2
                                  else t2::(reunion l1 q2)
;;
```
   (c)
```
let rec produit1 l x = match l with
  | [] -> []
  | t::q -> (t^x)::(produit1 q x)
;;
```

```
let rec produit l1 l2 = match l2 with
  | [] -> []
  | t2::q2 -> reunion (produit1 l1 t2) (produit l1 q2)
;;
```

5. (a)
```
let rec calculP e = match e with
  | Epsilon -> []
  | Const a -> [a]
  | Sum (e1,e2) -> reunion (calculP e1) (calculP e2)
  | Concat (e1,e2) when mot_vide e1 -> reunion (calculP e1)
                                               (calculP e2)
  | Concat (e1,e2) -> calculP e1
  | Kleene e1 -> calculP e1
;;
```

(b)
```
let rec calculS e = match e with
  | Epsilon -> []
  | Const a -> [a]
  | Sum (e1,e2) -> reunion (calculS e1) (calculS e2)
  | Concat (e1,e2) when mot_vide e2 -> reunion (calculS e1)
                                               (calculS e2)
  | Concat (e1,e2) -> calculS e2
  | Kleene e1 -> calculS e1
;;
```

(c)
```
let rec calculF e = match e with
  | Epsilon -> []
  | Const a -> []
  | Sum (e1, e2) -> reunion (calculF e1) (calculF e2)
  | Concat (e1, e2) -> let l = reunion (calculF e1) (calculF e2)
in reunion l (produit (calculS e1) (calculP e2))
  | Kleene e1 -> reunion (calculF e1) (produit (calculS e1)
                 (calculP e1))
;;
```

6. (a)
```
let recherche_iterative t x =
  let n = String.length t and m = String.length x in
  let deb = ref 0 and i = ref 0 and j = ref 0 in
    while (!deb + m <= n) && (!j < m) do
      if (t.[!i] = x.[!j])
        then
          begin
            i := !i + 1;
            j := !j + 1
          end
        else
          begin
            deb := !deb + 1;
            i := !deb;
            j := 0
          end
    done;
    !j = m
```

```
;;
```

(b) i. Soit (a,b) , (x,y) appartenant à $\mathbb{N}^2$:

$$(a,b) \leqslant (x,y) \Leftrightarrow (a < x) \text{ ou } \begin{cases} a = x \\ b \leqslant y \end{cases},$$

et :

$$(a,b) < (x,y) \Leftrightarrow (a < x) \text{ ou } \begin{cases} a = x \\ b < y \end{cases}.$$

L'ordre lexicographique sur $\mathbb{N}^2$ est un ordre bien fondé.

ii. Dans le cas où la condition `(!deb + m <= n) && (!j < m)`, le couple $(n - deb, m - j)$ appartient à $\mathbb{N}^2$,

et étant donné un tel couple $(n - deb, m - j)$, à l'itération suivante :

— soit `t.[!i] = x.[!j]`, alors i et j sont incrémentés de 1, la valeur de $n - deb$ est inchangée et la valeur de $m - j$ a diminué de 1,

donc : $(n - deb, m - j)_{nouveau} < (n - deb, m - j)_{ancien}$,

— la valeur de deb est incrémentée de 1 donc la valeur de $n - deb$ a diminué de 1,

donc : $(n - deb, m - j)_{nouveau} < (n - deb, m - j)_{ancien}$,

ainsi, dans les deux cas : $(n - deb, m - j)_{nouveau} < (n - deb, m - j)_{ancien}$.

En supposant que la condition `(!deb + m <= n) && (!j < m)` soit toujours vérifiée, la suite des $(n - deb, m - j)$ est une suite strictement décroissante pour l'ordre lexicographique de $\mathbb{N}^2$,

et l'ordre lexicographique sur $\mathbb{N}^2$ étant bien fondé, nous obtenons une contradiction.

Donc la condition `(!deb + m <= n) && (!j < m)` n'est pas toujours vérifiée et par conséquent, la répétitive conditionnelle nous conduit à un nombre fini d'itérations.

Suit alors une seule instruction simple, un test.

Donc l'algorithme se termine.

(c) Dans le pire des cas, deb varie de 0 à $n - m$ et à deb fixé, j varie de 0 à $m - 1$ (par exemple en recherchant le motif $aaab$ dans le texte $aaaaaaaaaaaaaaaaaaa$), soit un total de $\sum_{deb=0}^{n-m} \sum_{j=0}^{m-1} (1 \text{ comparaison}) = m\,(n - m + 1)$ comparaisons,

donc une complexité égale à $m\,(n - m + 1)$.

7. (a)
```
let rec est_prefixe_recursif t x = match t,x with
  | _,[] -> true
  | [],_ -> false
  | t1::t2,x1::x2 -> (t1=x1) && (est_prefixe t2 x2)
;;
```

(b)
```
let rec recherche_recursive t x = match t with
  | [] -> false
  | t1::t2 -> (est_prefixe_recursif t x) || (recherche_recursive t2 x)
;;
```

(c) i. L'appel récursif se fait par un motif x de longueur supérieure ou égale à 1 sur un motif $x2$ dont la longueur a diminué de 1 donc strictement inférieure à la longueur du motif appelant,

et l'ordre naturel sur $\mathbb{N}$ étant bien fondé, l'algorithme `est_prefixe_recursif` se termine.

ii. L'appel récursif se fait par un texte t de longueur supérieure ou égale à 1 sur un texte $t2$ dont la longueur a diminué de 1 donc strictement inférieure à la longueur du texte appelant,

et l'ordre naturel sur $\mathbb{N}$ étant bien fondé, l'algorithme `recherche_recursive` se termine.

8. (a) L'énoncé nous laisse penser que $k+s-1$ est inférieur ou égal à la longueur du mot v, $l+s-1$ est inférieur ou égal à la longueur du mot w.

```
let compare_sub_strings v k w l s =
let j = ref 0 in
    while (!j < s) && (v.[k+ !j] = w.[l+ !j]) do
      j := !j + 1
    done;
    !j
;;
```

(b) L'énoncé nous laisse penser que $k+s-1$ est inférieur ou égal à la longueur du mot v, $l+s-1$ est inférieur ou égal à la longueur du mot w.

```
let test_egalite_sub_strings v k w l s =
  (compare_sub_strings v k w l s) = s
;;
```

(c)
```
let border_length w =
  let n = String.length w and j = ref 1 in
    while not (test_egalite_sub_strings w 0 w !j (n - !j)) do
      j := !j + 1
    done;
    n - !j
;;
```

(d)
```
let borders x =
  let m = String.length x in
    let b = Array.make (m+1) (-1) in
      for j=1 to m do
        b.(j) <- border_length (String.sub x 0 j)
      done;
      b
;;
```

(e) Nous n'avons à nous justifier uniquement dans le second cas.

Nous avons donc : $t_{k+j-\beta(j)} \dots t_{k+j-1} = x_{j-\beta(j)} \dots x_{j-1}$,

et par définition de $\beta(j)$: $x_{j-\beta(j)} \dots x_{j-1} = x_0 x_1 \dots x_{\beta(j)-1}$,

donc : $t_{k+j-\beta(j)} \dots t_{k+j-1} = x_0 x_1 \dots x_{\beta(j)-1}$,

et il est alors légitime de reprendre la recherche en comparant t_{k+j} et $x_{\beta(j)}$.

Mais nous pouvons nous demander si nous n'avons pas oublié une apparition du motif dans ce "grand" décalage à droite.

Supposons donc qu'il existe ℓ appartenant à $\mathbb{N}$ tel que :

$$\begin{cases} k < \ell < k+j-\beta(j) \\ t_\ell t_{\ell+1} \dots t_{\ell+m-1} = x_0 x_1 \dots x_{m-1} \end{cases},$$

en particulier : $x_0x_1 \dots x_{k+j-1-\ell} = t_\ell t_{\ell+1} \dots t_{k+j-1}$,
et : $t_\ell t_{\ell+1} \dots t_{k+j-1} = x_{\ell-k} \dots x_{j-1}$, donc : $x_0x_1 \dots x_{k+j-1-\ell} = x_{\ell-k} \dots x_{j-1}$,
donc $x_0x_1 \dots x_{k+j-1-\ell}$ est un préfixe strict (en effet, comme $k - \ell$ est strictement négatif, alors : $k + j - \ell < j$) et un suffixe de $x_0x_1 \dots x_{j-1}$ de longueur $k + j - \ell$,
et : $k + j - \ell > \beta(j)$: CONTRADICTION !
Dons nous n'avons pas d'apparition du motif x entre les indices $k+1$ et $k+j-\beta(j)$.

(f)
```
let kmp t x =
  let n = String.length t and m = String.length x in
  let beta = borders x in
  let k = ref 0 and j = ref 0 in
    while (!k + m <= n) && (!j < m) do
      while (!j < m) &&  (t.[!k+ !j] = x.[!j]) do
          j := !j + 1
      done;
      if (!j < m)
        then
          begin
            k := !k + !j - beta.(!j);
            j := max 0 beta.(!j)
          end
      ;
    done;
    !j = m
;;
```