TD1 - Langages formels (éléments de réponses)

Exercice 1

Question 1. On a les résultats suivants.
- ☐ **1.1.** $abc \in L \cdot L'$, $abc \in L^* \cdot L'$, $abc \in (L \cdot L')^*$.
- ☐ **1.2.** $aacb \notin L \cdot L'$, $aacb \in L^* \cdot L'$, $aacb \notin (L \cdot L')^*$.
- ☐ **1.3.** $abccb \notin L \cdot L'$, $abccb \in L^* \cdot L'$, $abccb \in (L \cdot L')^*$.
- ☐ **1.4.** $abc \notin L \cdot L'$, $abc \notin L^* \cdot L'$, $abc \notin (L \cdot L')^*$.

Question 2. Pour répondre à cette question, procéder par double inclusion. Considérer un mot $w \in (L' \cdot L)^* \cdot L'$. Montrer alors qu'il appartient à $L' \cdot (L \cdot L')^*$. Puis montrer que tout mot $w \in L' \cdot (L \cdot L')^*$ appartient à $(L' \cdot L)^* \cdot L'$.

Question 3.

- ☐ **3.1.** $(L^+ = L^*) \iff (\varepsilon \in L)$ - vrai
- ☐ **3.2.** $L^2 \cdot L^+ = L^+$ - faux
- ☐ **3.3.** $\varnothing \cdot L = L$ - faux
- ☐ **3.4.** $L^* \cdot L^* = L^*$ - vrai
- ☐ **3.5.** $(L = L^*) \implies (L = \{\varepsilon\})$ - faux
- ☐ **3.6.** $\varnothing^* = \varnothing$ - faux
- ☐ **3.7.** $\varnothing^+ = \varnothing$ - vrai
- ☐ **3.8.** $L^+ \cdot L^+ = L^+$ - faux
- ☐ **3.9.** $L^+ \cdot L^* = L^+$ - vrai
- ☐ **3.10.** $\{\varepsilon\}^* = \{\varepsilon\}$ - vrai

Question 4. Considérer trois langages A, B, C sur Σ puis tout mot w de $(A \cup B) \cdot C$. Montrer que c'est un mot de $(A \cdot C) \cup (B \cdot C)$.
Pour la deuxième partie de la question, choisir $A = \{a\}$, $B = \{ab\}$ et $C = \{a, ab\}$. Construire $(A \cap B) \cdot C$ et $(A \cdot C) \cap (A \cdot C)$ puis conclure.

Question 5. Si $L = \{a, b\}$, le mot ab appartient à LL^{R} mais n'est pas un palindrome.

Question 6. Qu'en pensez-vous ?

Exercice 2

Question 1. On procède par double inclusion.

- $(L_1 \cup L_2)L_3 \subseteq (L_1L_3) \cup (L_2L_3)$
  Soit $w \in (L_1 \cup L_2)L_3$. Par définition de la concaténation, il existe $uv = w$ tels que $u \in (L_1 \cup L_2)$ et $v \in L_3$. Comme $u \in (L_1 \cup L_2)$, par définition de l'union, $u \in L_1$ ou $u \in L_2$. Par cas :
  $u \in L_1$: dans ce cas $w = u\,v \in L_1L_3$ et donc $w \in (L_1L_3) \cup (L_2L_3)$.
  $u \in L_2$: symétrique au précédant avec $u\,v \in L_2L_3$
- $(L_1 \cup L_2)L_3 \supseteq (L_1L_3) \cup (L_2L_3)$
  Soit $w \in (L_1L_3) \cup (L_2L_3)$. Par définition de l'union, et par cas :
  $w \in L_1L_3$: il existe $uv = w$ avec $u \in L_1$ et $v \in L_3$ par définition de la concaténation. Comme $u \in L_1$, $u \in L_1 \cup L_2$ et ainsi $uv \in (L_1 \cup L_2)L_3$.
  $w \in L_2L_3$: symétrique au précédant avec $u \in L_2$.

Question 2. L'affirmation est fausse. Il suffit de considérer $\Sigma = \{a\}$, et $L_1 = L_2 = \{\varepsilon, a\}$. On a $L_1L_2 = \{\varepsilon, a, aa\}$ qui ne contient que trois mots. De façon plus générale, l'affirmation est fausse dès que l'on peut créer par concaténation le même mot de deux façons différentes.

Exercice 3

Question 1. Soit $n \geqslant 3$. Par récurrence, si n est pair, f_4 est suffixe de f_n ; si n est impair, f_3 est suffixe de f_n. Or $f_3 = ba$ et $f_4 = bab$. Le résultat est donc vrai pour tout entier $n \geqslant 3$.

Question 2. On raisonne par récurrence. Tout d'abord, on a :

- Si $n = 3$, $g_3 = \varepsilon$ est un palindrome ;
- Si $n = 4$, $g_4 = b$ est un palindrome ;
- Si $n = 5$, $g_5 = bab$ est un palindrome.

Si $n \geqslant 6$, supposons le résultat acquis jusqu'au rang $n - 1$.

- Si n est pair, alors $f_n = f_{n-1}f_{n-2} = f_{n-2}f_{n-3}f_{n-2} = g_{n-2}abg_{n-3}bag_{n-2}ab$. Ainsi, $g_n = g_{n-2}abg_{n-3}bag_{n-2}$. Par hypothèse de récurrence, g_{n-2} et g_{n-3} sont des palindromes. Donc g_n est un palindrome.
- Si n est impair, on établit $g_n = g_{n-2}bag_{n-3}abg_{n-2}$. Le raisonnement est alors identique.

Exercice 4

Question 1. L'idée générale pour déterminer si un mot appartient à $\mathcal{D}$ ou pas est de parcourir séquentiellement le mot et d'ajouter ou de soustraire 1 à un compteur initialisé à 0 selon qu'une parenthèse est ouvrante ou fermante. Avant toute lecture d'une nouvelle parenthèse, il convient de contrôler la positivité de ce compteur. La fonction `dyck` suivante effectue ce parcours dans la fonction auxiliaire `aux` en contrôlant à chaque appel la positivité du compteur `acc`. Noter l'emploi du `&&` qui teste d'abord la positivité avant de faire un appel récursif (évaluation paresseuse).

```
let dyck m =
  let rec aux acc = function
    | [] -> acc = 0
    | A :: q -> acc >= 0 && aux (acc + 1) q
    | B :: q -> acc >= 0 && aux (acc - 1) q
  in aux 0 m
```

Question 2. Pour calculer le nombre de facteurs, le mot m est lu séquentiellement. Un premier facteur est déterminé dès qu'un préfixe p_1 du mot aboutit à $\sigma(p_1) = 0$. Un compteur du nombre de facteurs, initialement nul, est incrémenté et le parcours se poursuit pour déterminer la séquence p_2 suivante qui constitue un nouveau facteur. Et ainsi de suite jusqu'à lecture complète de m. Ainsi, le mot m est décomposé sous la forme $p_1 p_2 \dots p_{n_f}$ où chaque facteur p_i vérifie $\sigma(p_i) = 0$. Le nombre de facteurs est n_f, déterminé par la valeur finale du compteur.

```
let nb_fact m =
  let rec aux acc n_facteurs = function
    | [] -> n_facteurs
    | B :: q when acc = 1 -> aux 0 (n_facteurs + 1) q
    | B :: q -> aux (acc - 1) n_facteurs q
    | A :: q -> aux (acc + 1) n_facteurs q
  in aux 0 0 m
```

Question 3. Fonction `affiche_fact`.

```
let affiche_fact m =
  let rec aux acc = function
    | [] -> ()
    | [B] -> print_char ')'
    | B :: q when acc = 1 -> print_string ")."; aux 0 q
    | B :: q -> print_char ')'; aux (acc - 1) q
    | A :: q -> print_char '('; aux (acc + 1) q
  in aux 0 m
```

Par exemple :

```
let m = [A; A; B; B; A; B; A; A; B; A; B; B] in affiche_fact m
(()).().(()())- : unit = ()
```

Exercice 5

Question 1. Par récurrence, on établit que le nombre de pliages à l'étape n est $2^n - 1$.

Question 2. Considérons deux feuilles identiques et plions les i fois. Les deux feuilles présentent des creux et des bosses aux mêmes endroits.

- Si on déplie puis on retourne l'ensemble des deux feuilles superposées, sans les détacher, le mot w_i permet de les représenter.
- Si à présent on déplie l'ensemble des deux feuilles puis on retourne et on place la feuille du dessus à droite de la feuille du dessous, on obtient le résultat de $i + 1$ pliages d'une feuille initiale formée des deux feuilles. Le mot w_{i+1} permet de représenter cette grande feuille.

La première moitié de cette feuille est représentée par le mot w_i qui est donc préfixe de w_{i+1}.

Question 3.
□ **3.1.** Le résultat obtenu à la question précédente montre également que la deuxième feuille de droite présente des bosses et des creux symétriques de la feuille de gauche par rapport à un pli central, qui est un creux, représenté par un 0. Le mot représentant w_{i+1} pliages est la concaténation de w_i, de ce 0 et du miroir de l'inverse bit à bit de w_i, notée $\overline{w_i}$.
□ **3.2.** La fonction `barre` suivante calcule l'image miroir d'un mot.

```
let barre lst =
  let rec aux lst lst_acc = match lst with
    | [] -> lst_acc
    | x::q -> aux q ((1-x)::lst_acc)
  in aux lst []
```

La fonction `pliage` construit alors le mot associé à un pliage.

```
let rec pliage = function
  | 0 -> []
  | n -> let w = pliage (n-1) in w @(0::(barre w))
```

Pour $n = 5$, le résultat est le suivant.

```
# pliage 5;;
- : int list = [0;0;1;0;0;1;1;0;0;0;1;1;0;1;1;0;0;0;1;0;0;1;1;1;0;0;1;1;0;1;1]
```

Question 4.

□ **4.1.** La décomposition binaire d'un entier s'obtient par divisions entières successives par 2. La fonction suivante renvoie la liste des bits de la décomposition d'un tel entier ainsi que son nombre de bits.

```
let int2bin n =
  let rec aux lst_bin nb = function
    | 0 -> lst_bin, nb
    | p -> let q = p/2 and r = p mod 2 in aux (r::lst_bin) (nb+1) q
  in aux [] 0 n
```

□ **4.2.** Pour tout entier naturel i, tout mot w_i est préfixe de w. Par conséquent, le n-ième bit de w est aussi celui de tous les mots w_i dont la longueur est supérieure à n : $|w_i| \geqslant n$. Considérons un tel mot w_i. Il est possible de montrer qu'il se décompose sous la forme :

$$w_i = 0 \cdot \sigma_0 \cdot 1 \cdot \sigma_1 \cdot 0 \cdot \sigma_2 \cdot 1 \cdots 1 \cdot \sigma_{p-2} \cdot 0 \cdot \sigma_{q-1} \cdot 1$$

avec $q = 2^{i-1} - 2$. Ce qui permet alors d'établir que :

- si n est un multiple de 4, le bit recherché est un 0 ;
- si $n - 2$ est un multiple de 4, le bit recherché est un 1.

c'est-à-dire de déterminer les bits de rangs pairs de w. Les bits de rangs impairs sont donnés par les σ_k. La concaténation de ces caractères définit le mot w_{i-1} de taille $2^{i-1} - 1$, moitié entière de celle de w_i. Le bit recherché est alors le $\lfloor n/2 \rfloor$-ième bit de w_{i-1}. Un algorithme récursif permet ainsi de trouver tous les bits de w.

Par ailleurs, cette recherche peut être faite en parcourant les bits de n. En décomposant n sous la forme :

$$n = b_{N-1} b_{N-2} \dots b_1 b_0$$

avec $b_{N-1} = 1$ et pour tout $i \in [\![1, N-1]\!]$, $b_i \in \{0, 1\}$, on a :

$$\lfloor n/2 \rfloor = b_{N-1} b_{N-2} \dots b_1$$

D'où l'algorithme :

- tant que le dernier dernier bit de n est 0, remonter d'un bit ;
- si le dernier bit est 1, renvoyer le précédent bit s'il existe ;
- sinon renvoyer 0.

La fonction suivante met en œuvre cet algorithme.

```
let word_bit n =
  let lst_bits, nb_bits = int2bin n in
  let tab_bits = Array.of_list lst_bits in
  let i = ref (nb_bits - 1) in
  while (!i > 0) && (tab_bits.(!i) = 0) do decr i done;
  if !i = 0 then 0 else tab_bits.(!i-1)
```

Exercice 6

Question 1. Tout mot de plus de trois lettres sur l'alphabet $\{a, b\}$ et sans facteur carré possède l'un des deux préfixes suivants : aba ou bab (les six autres facteurs possibles comportent tous un carré). S'il possède au moins quatre lettres, le préfixe suivant sera l'un des quatre mots : $abaa$, $abab$, $baba$, $babb$, qui tous possèdent un facteur carré.

Question 2. Montrons par récurrence sur $n \in \mathbb{N}$ que $\sigma^n(a)$ est préfixe de $\sigma^{n+1}(a)$.

- Si $n = 0$, on a $\sigma^0(a) = a$ et $\sigma(a) = ab$. Donc $\sigma^0(a)$ est préfixe de $\sigma^1(a)$.
- Si $n \geqslant 1$, supposons que $\sigma^{n-1}(a)$ soit préfixe de $\sigma^n(a)$. On peut écrire $\sigma^n(a) = \sigma^{n-1}(a)u$. Alors $\sigma^{n+1}(a) = \sigma^n(a)\sigma(u)$. Donc $\sigma^n(a)$ est préfixe de $\sigma^{n+1}(a)$.

Question 3.

□ **3.1.** Chaque mot de Σ_1^* est construit par concaténation de motifs de la forme ab ou ba. Chaque fois qu'un motif est ajouté, un a et un b sont ajoutés. Par conséquent, tout mot de Σ_1^* contient autant de a que de b. Si $s \in \Sigma_1^*$, les mots asa ou bsb contiennent des nombres différents de a et de b. Ce ne sont donc pas des mots de Σ_1^*.

□ **3.2.** On établit le résultat par récurrence.

- Pour $n = 1$, on a $\sigma(a) = ab \in \Sigma_1^*$.
- Supposons $n \geqslant 2$ pour lequel il existe un $\sigma^{n-1}(a) \in \Sigma_1^*$ qui peut se décomposer sous la forme :

$$\exists p \in \mathbb{N} \quad \sigma^{n-1}(a) = u_1 \dots u_p$$

où pour tout entier $i \in [\![1, p]\!]$, $u_i \in \Sigma_1$. Alors :

$$\sigma^n(a) = \sigma(u_1) \dots \sigma(u_p)$$

Or :

$$\forall i \in [\![0, p]\!], \quad \sigma(u_i) \in \Sigma_1^*$$

Donc :

$$\sigma^n(a) \in \Sigma_1^*$$

□ **3.3.** Supposons que m possède un facteur de la forme r^2x où x est la première lettre de r. Choisissons ce facteur de longueur minimale, noté $r = xs$. Alors $xsxsx$ est facteur de m et donc de $\sigma^n(a)$ pour un certain entier n.
Puisque $\sigma^n(a)$ appartient à Σ_1^*, suivant la position du facteur $xsxsx$ dans le mot m, l'un des mots $xsxsxy$ ou $yxsxsx$ (y désignant la lettre qui succède ou qui précède ce facteur dans m) appartient à Σ_1^* et possède un antécédent par σ, lui-même facteur de m. Distinguons alors plusieurs cas suivant la parité de $|s|$.

- Si $xsxsxy \in \Sigma_1^*$ et si $|s|$ est pair, en regroupant les facteurs sous la forme $(xsx)s(xy)$, on déduit qu xsx et s appartiennent à Σ_1^*, ce qui est impossiblle d'après la question précédente.
- Si $xsxsxy \in \Sigma_1^*$ et si $|s|$ est impair, en regroupant les facteurs sous forme $(xs)(xs)xy$, il existe s' tel que $xs = \sigma(xs')$ et $xy = \sigma(x)$ puisque y est une lettre. Alors, $xs'xs'x$ est encore un facteur de m avec $|xs'| < |xs|$, inégalité qui contredit le caractère minimal de $|r|$.
- Si $yxsxsx \in \Sigma_1^*$, le raisonnement se mène comme dans les deux cas précédents et aboutit à deux contradictions.

Par conséquent, m ne possède pas de facteur de la forme r^2x où x est la première lettre de r.

Question 4.

□ **4.1.** D'après ce qui précède, bbb n'est pas facteur de m qui ne contient alors que des facteurs avec 0, 1 ou 2 lettres b. L'alphabet $\{0, 1, 2\}$ suffit donc à écrire le mot μ.
Supposons que μ possède deux facteurs consécutifs égaux. Ces derniers ont pour origine un facteur de m de la forme $asasa$, autrement dit de la forme r^2x où x est la première lettre de r. Cette situation est impossible. Donc μ ne possède pas deux facteurs consécutifs égaux.

□ **4.2.** La fonction `gen` est construite à l'aide de trois autres fonctions `sigma`, `itere_sigma` et `filtre`.

```
(* fonction sigma *)
let rec sigma = function
  | [] -> []
  | A :: q -> A :: B :: (sigma q)
  | B :: q -> B :: A :: (sigma q)
(* itérés de sigma *)
let rec itere_sigma = function
  | 0 -> [A]
  | n -> sigma (itere_sigma (n - 1))
(* fonction qui calcule le nombre de b entre deux a consécutifs *)
let rec filtre = function
  | A :: A :: q -> 0 :: (filtre (A :: q))
  | A :: B :: A :: q -> 1 :: (filtre (A :: q))
  | A :: B :: B :: A :: q -> 2 :: (filtre (A :: q))
  | _ -> []
(* génération de préfixes de longueur n de mu *)
let gen n = List.iter print_int (filtre (itere_sigma n));;
```

Par exemple :

```
gen 6;;
210201210120210201202101210201 2- : unit = ()
```

Exercice 7

Question 1. Notons $p = |u|$ et $q = |x|$ de sorte que $u = u_1 \dots u_p$ et $x = x_1 \dots x_q$. Supposons que $p \leqslant q$. Alors la relation $xy = uv$ indique de u est préfixe de x. En écrivant $x = x_1 \dots x_p x_{p+1} \dots x_q$ et en posant $z = x_{p+1} \dots x_q$, il vient $x = uz$. Puis $uv = xy = uzy$ et enfin $v = zy$.
Par symétrie, on montre la deuxième série d'égalités.

Question 2. Si $|x| \geqslant |y|$, on peut appliquer le lemme de Lévi. Il existe un mot t tel que $x = yt$ et $z = ty$. Dans ce cas, le résultat est bien acquis avec $u = y$, $v = t$ et $k = 0$.
Si $|x| \leqslant |y|$, raisonnons par récurrence sur la longueur de y.

- Si $|y| = 1$, on a aussi $|x| = 1$ car $x \neq \varepsilon$ et dans ce cas $x = y = z$. Le résultat est acquis avec $u = y$, $v = \varepsilon$ et $k = 0$.
- Si $|y| > 1$, supposons le résultat acquis pour tout mot de longueur inférieure, et appliquons le lemme de Levi. Il existe un mot t tel que $y = xt$ et $y = tz$. On a donc $xt = tz$ et puisque $x \neq \varepsilon$, $|t| < |y|$. On peut donc appliquer l'hypothèse de récurrence et affirmer l'existence de deux mots u et v tels que $x = uv$, $t = (uv)^k u = u(vu)^k$ et $z = vu$. Il reste à calculer $y = xt = tz$ et obtenir $y = (uv)^{k+1}u = u(vu)^{k+1}$.

Question 3. Par hypothèse, $x \neq \varepsilon$ et $y \neq \varepsilon$. Alors $|x| \geqslant 1$ et $|y| \geqslant 1$. On raisonne par récurrence sur $|xy| \geqslant 2$.

- Si $|xy| = 2$, $|x| = |y| = 1$ puis $x = y$. On choisit $w = x = y$ et $i = j = 1$.
- Soit un entier $n > 2$ et supposons la propriété vérifiée pour tous x et y vérifiant les hypothèses et satisfaisant $|xy| < n$. Considérons à présent deux mots x et y vérifiant les hypothèses et satisfaisant $|xy| = n$. D'après le résultat de la question précédente, il existe u et v dans Σ^* et un entier naturel k tels que $x = uv = vu$ et $y = (uv)^k u$.
  - Si $u = \varepsilon$, $x = v$ et $y = v^k$, alors $w = v$, $i = 1$ et $j = k$ conviennent.
  - Si $v = \varepsilon$, $x = u$ et $y = u^{k+1}$, alors $w = u$, $i = 1$ et $j = k + 1$ conviennent.
  - Sinon on a $uv = vu$ avec $|uv| = |x| < |xy| = n$. L'hypothèse de récurrence permet alors d'affirmer l'existence de w, i, j tels que $u = w^i$ et $v = w^j$. Donc $x = w^{i+j}$ et $y = w^{i+k(i+j)}$.