Correction du devoir à faire pour le Lundi 5 Janvier

Exercice 1 (Mots et pliages)

1. (a) Nous avons :

$$\overline{aababa} = \overline{ababab} = \overline{babab}\overline{bb} = \overline{aba}\overline{abb} = \overline{ba}\overline{babb} = \overline{a}\overline{ababb} = bababb$$

(b) Soit s appartenant à Σ^*. Notons pour tout entier naturel n, la propriété $\mathscr{P}(n)$: "pour tout mot r de longueur n, $\overline{rs} = \overline{s}\,\overline{r}$".

Ini. Soit r un mot de longueur 0. Donc $r = \varepsilon$ et on a :

$$\overline{rs} = \overline{s} = \overline{s}\,\varepsilon = \overline{s}\,\overline{\varepsilon} = \overline{s}\,\overline{r}.$$

Donc $\mathscr{P}(0)$ est vraie.

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

Soit r un mot de longueur $n+1$. On a deux cas :

— Si la première lettre de r est a, alors $r = ar'$ avec r' appartenant à Σ^* de longueur n et par hypothèse de récurrence :

$$\overline{rs} = \overline{ar's} = \overline{r's}\;b = \overline{s}\,\overline{r'}\;b = \overline{s}\,\overline{ar'} = \overline{s}\,\overline{r}.$$

— Si la première lettre de r est b, alors $r = br'$ avec r' appartenant à Σ^* de longueur n et par hypothèse de récurrence :

$$\overline{rs} = \overline{br's} = \overline{r's}\;a = \overline{s}\,\overline{r'}\;a = \overline{s}\,\overline{br'} = \overline{s}\,\overline{r}.$$

Dans tous les cas, $\mathscr{P}(n+1)$ est vraie.

Ccl. Par le principe de récurrence, on a donc démontré $\mathscr{P}(n)$ pour tout $n \in \mathbb{N}$.

Comme ce résultat est vrai pour tout $s \in \Sigma^*$, on a donc démontré que :

$$\forall (r,s) \in (\Sigma^*)^2, \quad \overline{rs} = \overline{s}\,\overline{r}.$$

(c) On définit en OCaml les types `alphabet` et `mot` ainsi :

```
type alphabet = A | B ;;
type mot = alphabet list ;;
```

(d) Voici la fonction demandée :

```
let rec barre m = match m with
  | [] -> []
  | a :: q -> (barre q) @ [b]
  | b :: q -> (barre q) @ [a]
;;
```

2. (a) i. A chaque pliage, nous doublons le nombre de faces, donc pour tout n appartenant à $\mathbb{N}$, après n pliages, nous avons 2^n faces et donc $2^n - 1$ plis. Le mot u_n est donc de longueur $2^n - 1$.

Soit n appartenant à $\mathbb{N}$. Imaginons la feuille dépliée après $n+1$ pliages. Ayant commencé par plier la feuille en deux, le pli central est en creux symbolisé par la lettre a.

Le premier pliage effectué, il reste n pliages à effectuer :

— ces n pliages vont être effectués sur la moitié gauche à l'endroit donc générés le mot u_n.
— ces n pliages vont être effectués sur la moitié droite à l'envers donc générés le mot $\overline{u_n}$.

donc, par concaténation, $u_{n+1} = u_n a \overline{u_n}$.

ii. Voici la fonction demandée :

```
let rec suiteu n = match n with
  | 0 -> []
  | _ -> let u = suiteu (n-1) in u @ ([a] @ (barre u))
;;
```

(b) i. On remarque que, pour tout $n \geqslant 2$, $2^n - 1 \equiv 3[4]$, c'est-à-dire qu'il existe k appartenant à $\mathbb{N}$ tel que $2^n - 1 = 4k + 3$.

Montrons alors par récurrence que, pour tout entier $n \geqslant 2$,

$$u_n = a w_2 b w_4 \dots b w_{2^n-4} a w_{2^n-2} b.$$

Ini. $u_2 = aab$ et u_2 est bien de la forme voulue donc $\mathscr{P}(2)$ est vraie.

Héré. Soit un entier $n \geqslant 2$. Supposons $\mathscr{P}(n)$ vraie. Donc u_n est de la forme :

$$u_n = a w_2 b w_4 \dots b w_{2^n-4} a w_{2^n-2} b.$$

On a alors (avec les question 2.(a)i. puis 1.(b)) :

$$\begin{aligned} & u_{n+1} = u_n a \overline{u_n} \\ = \; & \underbrace{a w_2 b w_4 \dots b w_{2^n-4} a w_{2^n-2} b}_{u_n} a \underbrace{a \overline{w_{2^n-2}} b \overline{w_{2^n-4}} a \dots \overline{w_4} a \overline{w_2} b}_{\overline{u_n}} \\ = \; & a w_2 b w_4 \dots b w_{2^n-4} a w_{2^n-2} b w_{2^n} a w_{2^n+2} b w_{2^n+4} a \dots w_{2^{n+1}-4} a w_{2^{n+1}-2} b \end{aligned}$$

donc u_{n+1} est de la forme attendue et $\mathscr{P}(n+1)$ est vraie.

Ccl. Par le principe de récurrence, pour tout $n \geqslant 2$, on a :

$$u_n = a w_2 b w_4 \dots b w_{2^n-4} a w_{2^n-2} b.$$

Soit k appartenant à $\mathbb{N}$. Il existe alors n supérieur ou égal à 2 tel que : $4k + 3 \leqslant 2^n - 1$. Donc w_{4k+1} est la $(4k+1)$-ième lettre de u_n et w_{4k+3} la $(4k+3)$-ième lettre de u_n. Donc avec le résultat précédent, $w_{4k+1} = a$ et $w_{4k+3} = b$.

Ainsi, pour tout $k \in \mathbb{N}$, $w_{4k+1} = a$ et $w_{4k+3} = b$.

ii. Formons le mot $v_n = w_2 w_4 \dots w_{2^n-2}$ à partir des lettres d'indices pairs de u_n (on rappelle que u_n est de longueur $2^n - 1$).

Alors $v_{n+1} = v_n a \overline{v_n}$ (car il y a un nombre impair de lettres dans u_n).

Puisque $v_1 = \varepsilon$, une récurrence immédiate montre que, pour tout $n \geqslant 1$, $v_n = u_{n-1}$.

Ainsi, pour tout entier naturel k, il existe n appartenant à $\mathbb{N}\backslash\{0,1\}$ tel que $k \leqslant 2^{n-1} - 1$ et on a :

$$\begin{aligned} v_n &= w_2 w_4 \dots w_{2k} \dots w_{2^n-2} \\ u_{n-1} &= w_1 w_2 \dots w_k \dots w_{2^{n-1}-1}. \end{aligned}$$

Donc $w_{2k} = w_k$.

iii. Voici la fonction demandée :

```
let rec suitew n = match n mod 2 with
  | 0 -> suitew (n / 2)
  | _ -> if (n mod 4) = 1 then a else b
;;
```

iv. $w_{2000} = w_{1000} = w_{500} = w_{250} = w_{125} = a$ car $125 = 4 \times 31 + 1$.

Exercice 2 (Arbres gauchers)

1. Dans l'ordre des arbres :
   — Les sommets $2, 3$ et 4 ont pour rang 1. Les sommets 5 et 9 ont pour rang 2. Le sous-arbre de racine 5 n'est pas gaucher car son fils gauche (vide) a un rang inférieur à son fils droit. L'arbre n'est donc pas gaucher.
   — Les sommets $1, 2, 3$ et 7 sont de rang 1. Les sommets 5 et 6 sont de rang 2. Il n'y a pas de problème sur les rangs, mais l'arbre ne vérifie pas la première condition car le fils du sommet 2 est 7. L'arbre n'est donc pas gaucher.
   — Tous les sommets sont de rang 1. Chaque sous-arbre est bien gaucher, car le fils droit est vide. L'arbre est donc gaucher.
   — Tous les sommets sont de rang 1, sauf 6 qui est de rang 2. L'arbre est gaucher car il respecte bien les deux conditions.

2. On ne s'intéresse ici qu'au sous-arbre droit pour effectuer la formule inductive :

```
let rec rang a = match a with
  | Vide -> 0
  | N(_, _, d) -> 1 + rang d
;;
```

3. On vérifie que l'arbre respecte les deux conditions pour être gaucher.

```
let rec gaucher a = match a with
  | Vide -> true
  | N(_, Vide, Vide) -> true
  | N(_, Vide, _) -> false
  | N(x, g, Vide) -> let N(y, _, _) = g in (y <= x) && (gaucher g)
  | N(x, g, d) -> let N(y, _, _) = g and N(z, _, _) = d in
    (y <= x) && (z <= x) && (rang d <= rang g) && (gaucher g) && (gaucher d)
;;
```

4. Montrons que $0 \leqslant r(a) \leqslant \log_2(1 + |a|)$.
   La première inégalité est directe car l'arbre vide est de rang 0.
   La seconde inégalité revient à montrer que $2^{r(a)} - 1 \leqslant |a|$, que l'on montre par induction :
   - Si a est vide, alors $|a| = r(a) = 0$ et l'inégalité est une égalité.
   - Si a est un arbre gaucher de la forme $N(x, g, d)$ et que le résultat est prouvé pour g et d, alors :
     — $r(a) = r(d) + 1$;
     — $|a| = |g| + |d| + 1$;
     — $r(d) \leqslant r(g)$.

On obtient donc avec toutes ces informations :

$$|a| = 1 + |g| + |d| \geqslant 1 + 2^{r(g)} - 1 + 2^{r(d)} - 1 \geqslant 2 \times 2^{r(d)} - 1 = 2^{r(a)} - 1.$$

L'induction est donc terminée. Cette seconde borne est atteinte pour des arbres complets.

5. Il suffit de lire l'information :

```
let rang a = match a with
  | Vide -> 0
  | N(r, _, _, _) -> r
;;
```

6. On applique ce qui est décrit en prenant garde aux différentes valeurs de rang possibles.

```
let rec fusion a b = match a, b with
  | _, Vide -> a
  | Vide, _ -> b
  | N(_, xa, _, _), N(_, xb, _, _) when xb > xa -> fusion b a
  | N(_, xa, ga, da), _ -> let c = fusion da b in
                           let rg = rang ga and rc = rang c in
                           if rg >= rc then N(rc + 1, xa, ga, c)
                           else N(rg + 1, xa, c, ga)
;;
```

7. L'insertion est juste la fusion de l'arbre et d'une feuille.

```
let insertion x a = fusion a (N(1, x, Vide, Vide)) ;;
```

8. On enlève la racine et on fusionne les deux fils.

```
let extraire_max a = match a with
  | Vide -> failwith "Arbre vide"
  | N(_, x, g, d) -> x, fusion g d
;;
```

9. La complexité temporelle de la fonction `fusion` vérifie la relation de récurrence (sur les rangs) suivante :

$$C(r_a, r_b) = \begin{cases} C(r_a - 1, r_b) + O(1) \\ C(r_a, r_b - 1) + O(1) \end{cases}$$

selon les racines des arbres fusionnés. On constate ainsi que la somme des deux rangs diminue exactement de 1 à chaque appel récursif. On en déduit que $C(r_a, r_b) = O(r_a + r_b) = O(\log|a| + \log|b|)$.

On en déduit ainsi que la fonction d'insertion a une complexité en $O(r) = O(\log|a|)$. Pour un arbre de la forme $a = N(x, g, d)$, la fonction d'extraction a une complexité en $O(r(g) + r(d)) = O(\log|a|)$.

10. Comme pour le tri par tas, on commence par insérer tous les éléments dans un arbre gaucher, puis on les extrait un par un.

```
let tri_gaucher l =
  let rec creer_gaucher l = match l with
    | [] -> Vide
    | x :: q -> insertion x (creer_gaucher q) in
  let rec tri acc a = match a with
    | Vide -> acc
    | _ -> let x, b = extraire_max a in tri (x :: acc) b in
  tri [] (creer_gaucher l)
;;
```

Pour une liste de taille n, chacune de ces fonctions auxiliaires effectue n fusions. On en déduit que la complexité temporelle de cette fonction de tri est en $O(n\log(n))$, ce qui est optimal pour un tri par comparaison.
