DM D'INFORMATIQUE (Option) n°1

Corrigé

⋆ ⋆ ⋆

I Mots de Dyck

Question 1. On a $ABAABB = uv$ où $u = AB$ et $v = AABB$. $u = A\varepsilon B$ donc $u \in \mathcal{L}$. $v = AuB$ donc $v \in \mathcal{L}$. On en déduit que $ABAABB \in \mathcal{L}$.

Question 2. On augmente la valeur de 1 chaque fois qu'un A est rencontré :

```
let rec compteA = function
   | [] -> 0
   | A :: q -> 1 + compteA q
   | B :: q -> compteA q;;
```

Question 3. On montre la double implication :

- ($\Rightarrow$) Le mot vide vérifie bien la propriété sur le nombre de A et de B. Soit u et v des mots de Dyck. Alors :
  - Pour uv : $|uv|_A = |u|_A + |v|_A = |u|_B + |v|_B = |uv|_B$
    De plus, soit w un préfixe de uv. Deux cas se présentent : soit w est un préfixe de u, et comme u est un mot de Dyck, alors $|w|_A \geq |w|_B$. Sinon, $w = ux$ où x est un préfixe de v. Alors $|w|_A = |u|_A + |x|_A = |u|_B + |x|_A \geq |u|_B + |x|_B = |w|_B$, car v est un mot de Dyck.
  - Pour AuB : $|AuB|_A = |AuB|_B$.
    De plus, soit w un préfixe de AuB. Alors soit $w = A$, soit $w = AuB$ auxquels cas la propriété est immédiate. Sinon, $w = Ax$ où x est un préfixe de u. Alors $|w|_A = 1 + |x|_A \geq 1 + |x|_B > |w|_B$.

  On conclut par induction.
- ($\Leftarrow$) Dans un premier temps, il est clair qu'un tel mot est de taille paire. On raisonne ensuite par récurrence sur la taille $2n$ du mot u. Si $n = 0$, alors $u = \varepsilon$ et la propriété est vérifiée. Sinon, supposons que le résultat est vrai pour tout mot de taille inférieure ou égale à $2n$ pour $n \in \mathbb{N}$ fixé. Soit $u = u_1u_2\dots u_{2n+2}$ un mot de taille $2n+2$ vérifiant la propriété. On distingue alors deux cas :
  - Pour tout $k \in [\![1; 2n+1]\!]$, $|u_1\dots u_k|_A > |u_1\dots u_k|_B$. Alors nécessairement, pour tout $k \in [\![2; 2n+1]\!]$, $|u_2\dots u_k|_A \geq |u_2\dots u_k|_B$. De plus, il est clair que $u_1 = A$ et $u_{2n+2} = B$. On en déduit que $|u_2\dots u_{2n+1}|_A = |u_2\dots u_{2n+1}|_B$. Par hypothèse de récurrence, $u = AvB$ où v est un mot de Dyck.
  - Sinon, il existe $k \in [\![1; 2n+1]\!]$, $|u_1\dots u_k|_A = |u_1\dots u_k|_B$. Alors, $u = vw$ où $v = u_1\dots u_k$ et $w = u_{k+1}\dots u_{2n+2}$. Les mots v et w ont donc bien le même nombre de A et de B et vérifie la propriété sur les préfixes. On en déduit par hypothèse de récurrence que ce sont des mots de Dyck.

Question 4. On utilise un compteur `acc` qui compte le nombre $|v|_A - |v|_B$ sur le préfixe v du mot u testé. On augmente ou diminue de 1 ce compteur selon qu'on rencontre un A ou un B. On renvoie alors faux si ce compteur doit devenir négatif, et lorsque le parcours du mot est terminé, il faut que ce compteur soit égal à 0.

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

Question 5. On utilise un compteur `acc` qui compte les parenthèses ouvertes non fermées. Si ce compteur vaut 0 et qu'on rencontre un B, il faut rajouter un A juste avant. Lorsqu'on arrive à la fin du mot, on rajoute autant de B que ce compteur.

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

Question 6. Par taille des mots de Dyck :

- $\mathcal{L}_0$: Un seul mot : ε. Soit $C_0 = 1$.
- $\mathcal{L}_1$: Un seul mot : AB. Soit $C_1 = 1$.
- $\mathcal{L}_2$: $AABB$ et $ABAB$, soit $C_2 = 2$.
- $\mathcal{L}_2$: $AAABBB$, $AABABB$, $AABBAB$, $ABAABB$ et $ABABAB$, soit $C_3 = 5$.

Question 7. Un tel k existe toujours (il vaut au pire $2n$). On peut alors écrire $u = vw$ où $v = u_1 \dots u_k$ et $w = u_{k+1} \dots u_{2n}$. Les mots v et w sont des mots de Dyck, et de plus, $v \neq \varepsilon$ (car $n > 0$). On peut donc écrire $v = AxB$ où x est un mot de Dyck. Soit au final, $u = AxBw$ avec x et w des mots de Dyck de taille strictement inférieure à u.
Dès lors, on en déduit que pour créer un mot de Dyck de taille $2n$, il faut choisir deux mots de Dyck x et w de taille $2p$ et $2n-2p-2$ et construire un mot de la forme $AxBw$. Par ce partitionnement, on en déduit la relation de récurrence : $C_n = \displaystyle\sum_{2p=0}^{2n-2} C_p C_{n-p-1} = \sum_{k=0}^{n-1} C_k C_{n-k-1}$.

Question 8.

a) Supposons qu'il existe $k_1 < k_2 < \dots < k_m$ tels que $\rho_{k_1}(\alpha) = \dots = \rho_{k_m}(\alpha)$, avec m maximal pour cette propriété. Alors on a $\displaystyle\sum_{i=k_1}^{k_2-1} a_i = \sum_{i=k_2}^{k_3-1} a_i = \dots = \sum_{i=k_m}^{k_1-1} a_i = S$, avec la convention que $a_i = a_{1+(i \mod 2n+1)}$. On en déduit que $mS = \displaystyle\sum_{i=1}^{2n+1} a_i = 1$, puis que $m = S = 1$.

b) Si $n = 0$, le résultat est immédiat. Supposons alors le résultat vrai pour $n \in \mathbb{N}$ fixé.
Soit α un scrutin de taille $2n+3$. Il existe alors k tel que $a_k = 1$ et $a_{k+1} = -1$ (indices modulo $2n+1$). En supprimant ces deux éléments, on obtient un scrutin β de taille $2n+1$ qui, par hypothèse de récurrence, possède une rotation stricte, qu'on notera $b_1 \dots b_{2n+1}$. Dès lors, il existe j tel que $b_1 \dots b_j a_k a_{k+1} b_{j+1} \dots b_{2n+1}$ est une rotation de α.
Dès lors, toutes les sommes partielles sont strictement positives jusqu'à l'indice j (par HR), puis jusqu'à a_k et a_{k+1} car ces valeurs sont 1 et -1 (dans cet ordre), puis jusqu'à l'indice $2n+1$ (par HR). Cette rotation de α est donc bien un scrutin strict. On conclut par récurrence.

c) Quitte à renommer les indices, supposons que α est une rotation stricte et que $\rho_k(\alpha)$ est une rotation stricte, avec $k \neq 1$. Alors $\displaystyle\sum_{i=1}^{k-1} a_i > 0$ et $\displaystyle\sum_{i=k}^{2n+1} a_i > 0$. C'est absurde, car les deux sommes doivent être égales à 1.

d) Le nombre de scrutins de taille $2n+1$ est $\binom{2n+1}{n}$ car choisir un scrutin revient à choisir la position des n valeurs -1 dans la suite α. Comme un scrutin possède $2n+1$ rotations, on en déduit qu'il existe $\dfrac{1}{2n+1}\binom{2n+1}{n}$ classes d'équivalences par rotation. Pour chacune de ces classes, on a montré qu'il n'existe qu'une seule rotation stricte. Cette valeur est donc bien le nombre de rotations strictes de taille $2n+1$.

e) Il existe une bijection entre les scrutins stricts et les mots de Dyck, en remplaçant les 1 par des A, les -1 par des B et en supprimant le premier élément (qui est nécessairement un A). On en déduit :

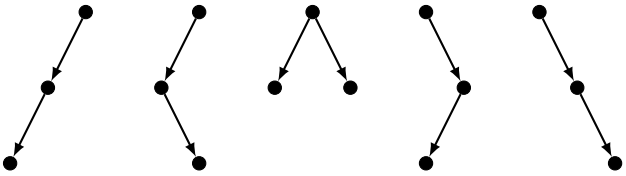
$$C_n = \frac{1}{2n+1}\binom{2n+1}{n} = \frac{1}{n+1}\binom{2n}{n}$$

II Arbres binaires

Question 9. Un grand classique.

```
let rec taille = function
   | Vide -> 0
   | N(g, d) -> 1 + taille g + taille d;;
```

Question 10. On classe par taille :

Taille	Arbres
0	Arbre Vide
1	
2	
3	

Soit alors $n \in \mathbb{N}^*$. Alors pour construire un arbre de taille n, il faut choisir un entier $k \in [\![0; n-1]\!]$, un fils gauche à de taille k et un fils droit de taille $n-1-k$ (d'après la formule d'induction sur la taille). On en déduit la formule du nombre d'arbres de taille n :

$$T_n = \sum_{k=0}^{n-1} T_k T_{n-1-k}$$

De plus, on remarque que $T_0 = C_0$. La formule de récurrence étant la même, on a bien $T_n = C_n$.

Question 11. Il n'y a pas de problème pour associer l'arbre vide à un mot de Dyck vide.
Pour un mot non vide, on cherche l'entier k de la question 7, et on écrit un mot de Dyck de la forme $u = AxBv$. Dès lors, l'arbre associé à u, sera le nœud ayant pour fils gauche l'arbre associé à x et pour fils droit l'arbre associé à v.
Dans la fonction suivante, la fonction `aux` sert à trouver l'entier k et renvoie le couple x, v des mots cherchés (en ayant enlevé le A et le B au bon endroit). On se sert toujours du même compteur qui compte la différence de A et de B.
Une fois les deux mots trouvés, on peut lancer des appels récursifs pour créer l'arbre cherché.

```
let rec arbre_dyck u =
   let rec aux acc = function
      | [] -> [], []
      | A :: q -> let v, w = aux (acc + 1) q in A :: v, w
      | B :: q when acc = 1 -> [], q
      | B :: q -> let v, w = aux (acc - 1) q in B :: v, w in
   if u = [] then Vide
   else (let v, w = aux 1 (List.tl u) in
      N(arbre_dyck v, arbre_dyck w));;
```

Question 12. La réciproque se fait de manière simple, en utilisant toujours la même bijection :

```
let rec dyck_arbre = function
   | Vide -> []
   | N(g, d) -> (A :: (dyck_arbre g)) @ (B :: (dyck_arbre d));;
```

Pour éviter la concaténation, on utilise un accumulateur qui représente le mot en train d'être formé. On commence du côté droit de l'arbre pour remplir le mot par la fin :

```
let dyck_arbre =
   let rec aux acc = function
      | Vide -> acc
      | N(g, d) -> let u = aux acc d in A :: (aux (B :: u) g) in
   aux [];;
```

Question 13. Dans un premier temps, notons que l'utilisation de `::` se fait en temps constant. On en déduit que la complexité ne dépend pas de la taille de l'accumulateur mais uniquement de la structure de l'arbre. On a les relations d'induction suivantes :

- $C(Vide) = O(1) = k_0$.
- $C(N(g,d)) = C(g) + C(d) + O(1) = C(g) + C(d) + k_1$.

On reconnait ici une relation inductive similaire à celle de la taille d'un arbre. On vérifie facilement qu'on a $C(a) = |a| \times k_1 + m \times k_0$ où m est le nombre de fois que l'arbre vide est le fils d'un nœud de a. Par une récurrence similaire à la relation entre feuilles et nœuds internes dans un arbre binaire entier, on montre que $m = |a| + 1$. On conclut finalement que $C(a) = |a|$.

⋆ ⋆ ⋆