Exercice 1. Évaluation de Horner

Question 1 – Pour montrer la terminaison de la fonction `eval`, il suffit de montrer la terminaison de la fonction `aux`. Montrons le par récurrence sur la taille de la liste `p` en entrée.

Initialisation Si `p` est vide alors la fonction termine (ligne 3).

Hérédité Soit `p` une liste non vide et supposons que la fonction termine pour toute liste de taille strictement inférieure à `len(p)`. À la ligne 4, il y a un appel récursif sur `q` la queue de `p`, qui vérifie `len(q) < len(p)`. Comme l'appel sur `q` termine par l'hypothèse de récurrence, l'appel sur `p` termine.

Question 2 – Soit `n` la taille de `p` et `p[i]` l'élément d'indice `i` de `p`. Il suffit de montrer que « `aux y p` » renvoie :

$$\texttt{y} \times \sum_{\texttt{i=0}}^{\texttt{n-1}} \texttt{x}^{\texttt{i}}\texttt{p[i]}$$

Montrons le par récurrence sur la taille de `p`.

Initialisation Si `p` est vide alors « `aux y p` » renvoie `0` qui est bien égal à :

$$\texttt{y} \times \sum_{\texttt{i=0}}^{\texttt{-1}} \texttt{x}^{\texttt{i}}\texttt{p[i]} \qquad \text{(somme vide)}$$

Hérédité Soit $\texttt{n} \in \mathbb{N}$. On suppose la propriété vraie pour toute liste de taille strictement inférieure à `n`. Soit `p` une liste de taille `n`, alors par la ligne 4 et l'hypothèse de récurrence, « `aux y p` » s'évalue en :

$$\begin{aligned}\texttt{p[0]} \times \texttt{y} + \texttt{y} \times \texttt{x} \sum_{\texttt{i=0}}^{\texttt{n-2}} \texttt{x}^{\texttt{i}}\texttt{q[i]} &= \texttt{p[0]} \times \texttt{y} + \texttt{y} \times \sum_{\texttt{i=1}}^{\texttt{n-1}} \texttt{x}^{\texttt{i}}\texttt{p[i]} \\ &= \texttt{y} \sum_{\texttt{i=0}}^{\texttt{n-1}} \texttt{x}^{\texttt{i}}\texttt{p[i]}\,.\end{aligned}$$

Question 3 – Soit n la taille de `p`. Soient $A(n)$ et $M(n)$ le nombre d'additions et de multiplications effectuées lors d'un appel à « `eval p x` ». On a :

$$\begin{cases} A(0) = 0 \\ A(n+1) = 1 + A(n) \text{ pour tout } n \geqslant 0 \end{cases} \qquad \begin{cases} M(0) = 0 \\ M(n+1) = 2 + M(n) \text{ pour tout } n \geqslant 0 \end{cases}$$

Par une récurrence immédiate sur n :

$$A(n) = n \qquad M(n) = 2n$$

Question 4 – Soit $C(n)$ le temps d'exécution de l'appel à `aux` avec une liste de taille n et T_n le temps d'exécution sans compter les appels récursifs. On a :

$$T_n \underset{n\to+\infty}{=} \Theta(1) \qquad \begin{cases} C(0) = T(0) \\ C(n) = T_n + C(n-1) \qquad \text{pour } n > 0 \end{cases}$$

Ainsi, pour tout $n \geqslant 0$:

$$C(n) = \sum_{k=0}^{n} T_k \underset{n\to+\infty}{=} \Theta\left(\sum_{k=0}^{n} 1\right) \underset{n\to+\infty}{=} \Theta(n)$$

Question 5 – Montrons le par récurrence sur la taille de `p` :

Initialisation Si `p` est la liste vide alors « `eval_horner p x` » renvoie 0 ce qui est correct

Hérédité Soit $\texttt{n} \in \mathbb{N}$. On suppose la propriété vraie pour toute liste de taille strictement inférieure à `n`. Soit `p` une liste de taille `n`. Par la ligne 3 et l'hypothèse de récurrence, « `eval_horner p x` » renvoie :

$$\texttt{p[0] + x}\sum_{\texttt{i=0}}^{\texttt{n-2}}\texttt{q[i]x}^{\texttt{i}} = \texttt{p[0] +}\sum_{\texttt{i=1}}^{\texttt{n-1}}\texttt{p[i]x}^{\texttt{i}} = \sum_{\texttt{i=0}}^{\texttt{n-1}}\texttt{p[i]x}^{\texttt{i}}$$

Question 6 – Soit n la taille de `p`. Soient $A(n)$ et $M(n)$ le nombre d'additions et de multiplications utilisées lors d'un appel à « `eval_horner p x` ». Alors :

$$\begin{cases} A(0) = 0 \\ A(n+1) = 1 + A(n) \end{cases} \qquad \begin{cases} M(0) = 0 \\ M(n+1) = 1 + M(n) \end{cases}$$

Par une récurrence immédiate sur n :

$$A(n) = n \qquad M(n) = n$$

Exercice 2. Fonction 91 de McCarthy

Idée. Faire des exemples à la main pour comprendre le comportement de la fonction. On pourra calculer « `f 100` », puis « `f 99` », puis « `f 98` », ..., « `f 89` », « `f 88` », ...

Preuve. On remarque tout d'abord que pour `n > 100`, un appel à « `f n` » termine et renvoie `n - 10`. Pour tout $\texttt{n} \leqslant \texttt{101}$, soit $\mathcal{P}\texttt{(n)}$ la propriété :

$\mathcal{P}\texttt{(n)}$: un appel à « `f n` » termine et renvoie `91`.

Montrons $\mathcal{P}\texttt{(n)}$ par récurrence "descendante" sur `n`, c'est à dire :

→ Initialisation : $\mathcal{P}\texttt{(101)}$ est vérifiée.

→ Hérédité : soit $\texttt{n} \in \mathbb{Z}$ tel que $\texttt{n} \leqslant \texttt{100}$. Il s'agit de montrer $\mathcal{P}\texttt{(n)}$ à partir des hypothèses $\mathcal{P}\texttt{(n + 1)}$, $\mathcal{P}\texttt{(n + 2)}$, ..., $\mathcal{P}\texttt{(101)}$.

On pourra alors en conclure que $\mathcal{P}\texttt{(n)}$ est vraie pour tout $\texttt{n} \leqslant \texttt{101}$.

Initialisation Un appel à « `f 101` » correspond au cas de base de la fonction car `101 > 100`. Cet appel termine et renvoie `91`.

Hérédité Soit $\texttt{n} \in \mathbb{Z}$ tel que $\texttt{n} \leqslant \texttt{100}$. Supposons $\mathcal{P}\texttt{(m)}$ pour tout $\texttt{m} \in [\![\texttt{n + 1; 101}]\!]$ et montrons $\mathcal{P}\texttt{(n)}$. On a deux cas :

⋆ Si $\texttt{90} \leqslant \texttt{n} \leqslant \texttt{100}$ alors l'appel à « `f n` » fait un premier appel récursif à « `f (n+11)` ». Comme $\texttt{n + 11} \geqslant \texttt{101}$, l'appel à `f (n+11)` termine et renvoie `n+1`. On fait ensuite un deuxième appel récursif à « `f (n+1)` ». Comme $\texttt{n + 1} \leqslant \texttt{101}$, on peut utiliser l'hypothèse de récurrence qui garantit que l'appel à « `f (n+1)` » termine et renvoie `91`. Donc « `f n` » termine et renvoie 91.

⋆ Si `n < 90` alors l'appel à « `f n` » fait un premier appel récursif à `f (n+11)`. Comme `n + 11 < 101`, on peut utiliser l'hypothèse de récurrence qui garantit que l'appel à « `f (n+11)` » termine et renvoie `91`. Comme `91 > n`, on peut utiliser l'hypothèse de récurrence qui garantit que l'appel à « `f 91` » termine et renvoie `91`. Donc « `f n` » termine et renvoie 91.

En conclusion, l'appel à « `f n` » termine pour tout $\texttt{n} \in \mathbb{Z}$ et renvoie :

$$\begin{cases} n - 10 & \text{si } n \geqslant 102 \\ 91 & \text{si } n \leqslant 101 \end{cases}$$

Exercice 3. Fonction mystère

Question 1 – La fonction `f` prend en entrée deux entiers $n, k \geqslant 0$ et renvoie n^k à l'aide d'une exponentiation rapide.

Question 2 – Soit $k \in \mathbb{N}$ le deuxième argument de `f`.

- Si $k = 0$ alors la fonction termine (ligne 3).
- Sinon $k > 0$ et il y a un appel récursif sur $\lfloor k/2 \rfloor < k$. Si l'appel récursif termine alors l'appel principal termine.

Ainsi, par récurrence sur k, la fonction termine pour tout $k \in \mathbb{N}$.

Question 3 – On le montre par récurrence forte sur $k \in \mathbb{N}$:

Initialisation Si $k = 0$, la fonction renvoie 1 ce qui est correct

Hérédité Soit $k > 0$. Supposons la correction pour tout $k' < k$ et appelons la fonction avec k en entrée. On traite deux cas :

- ⋆ Si k est pair, par hypothèse de récurrence, l'appel récursif et l'appel principal renvoient $(n^2)^{k/2} = n^k$.
- ⋆ Si k est impair, par hypothèse de récurrence, l'appel récursif renvoie $(n^2)^{(k-1)/2}$, donc l'appel principal renvoie $n \times (n^2)^{(k-1)/2} = n^k$.

Question 4 – Soit $T(n)$ le temps d'exécution sans compter les appels récursifs. Alors :

$$T(n) \underset{n\to+\infty}{=} \Theta(1) \qquad \begin{cases} C(0) = T(0) \\ C(2j) = T(2j) + C(j) & \text{pour tout } j > 0 \\ C(2j+1) = T(2j+1) + C(j) & \text{pour tout } j \geqslant 0 \end{cases}$$

Ainsi, il existe des constantes $a, b > 0$ telles que :

$$C(2j) \leqslant a + C(j) \qquad\qquad C(2j+1) \leqslant b + C(j)$$

Avec $c_1 = \max(C(0), a, b)$:

$$C(0) \leqslant c_1 \qquad C(2j) \leqslant c_1 + C(j) \qquad C(2j+1) \leqslant c_1 + C(j)$$

Question 5 – Montrons l'inégalité par récurrence forte sur $k \in \mathbb{N}$:

Initialisation Si $k = 0$:

$$C(0) \leqslant c_1 \leqslant \frac{c_1}{\log_2(3/2)} = \frac{c_1}{\log_2(3/2)}\Big(\log_2(k+1) + 1\Big)$$

Hérédité Soit $k \in \mathbb{N}^*$. Supposons la propriété vraie pour tout $k' < k$:

⋆ Si $k = 2j$ est pair avec $j > 0$ alors $C(2j) \leqslant c_1 + C(j)$. Donc par hypothèse de récurrence :

$$\begin{aligned} C(2j) &\leqslant c_1 + \frac{c_1}{\log_2(3/2)}\Big(\log_2(j+1) + 1\Big) \\ &= \frac{c_1}{\log_2(3/2)}\Big(\log_2(j+1) + \log_2(3/2) + 1\Big) \\ &= \frac{c_1}{\log_2(3/2)}\Big(\log_2\Big(\frac{3}{2}(j+1)\Big) + 1\Big) \end{aligned}$$

On souhaite montrer que :

$$C(2j) \leqslant \frac{c_1}{\log_2(3/2)}\Big(\log_2(2j+1) + 1\Big)$$

Il suffit donc de montrer que $\frac{3}{2}(j+1) \leqslant 2j+1$ qui est équivalent à $j \geqslant 1$ et est donc vrai.

⋆ Sinon, $k = 2j + 1$ est impair avec $j \geqslant 0$ alors $C(2j + 1) \leqslant c_1 + C(j)$. Avec le même calcul que pour le point précédent :

$$C(2j+1) \leqslant \frac{c_1}{\log_2(3/2)}\Big(\log_2\Big(\frac{3}{2}(j+1)\Big)+1\Big)$$

On souhaite montrer que :

$$C(2j+1) \leqslant \frac{c_1}{\log_2(3/2)}\Big(\log_2(2j+2)+1\Big)$$

Il suffit donc de montrer que $\frac{3}{2}(j+1) \leqslant 2j+2$ qui est équivalent à $j+1 \geqslant 0$ et est donc vrai.

Question 6 – D'après la question précédente, on a $C(k) \underset{k\to+\infty}{=} \mathcal{O}(\log(k))$. Montrons de manière similaire que $C(k) \underset{k\to+\infty}{=} \Omega(\log(k))$. Il existe une constante $c_2 > 0$ telle que :

$$C(0) \geqslant c_2 \qquad C(2j) \geqslant c_2 + C(j) \qquad C(2j+1) \geqslant c_2 + C(j)$$

On conclut par récurrence forte sur $k \in \mathbb{N}$ que :

$$C(k) \geqslant c_2\Big(\log_2(k+1)+1\Big)$$

Finalement $C(k) \underset{k\to+\infty}{=} \Theta(\log(k))$.

Exercice 4. Triangle de Pascal

Question 1 – On le montre par induction sur $n \in \mathbb{N}$:

Initialisation Si $n = 0$ alors l'appel à `binom1` termine

Hérédité Soit $n > 0$ tel que l'appel termine pour $n - 1$.

Lors de l'exécution de la fonction avec n en argument, il y a deux appels récursifs à « `binom1 (n-1) k` » et « `binom1 (n-1) (k-1)` ». Par hypothèse d'induction, ces deux appels récursifs terminent et donc l'appel principal termine.

Question 2 – On le montre par induction sur $n \in \mathbb{N}$:

Initialisation Si $n = 0$, alors :

- ⋆ Si $k = 0$ alors la fonction renvoie bien $1 = \binom{0}{0}$
- ⋆ Sinon, $k \neq n$ et la fonction renvoie bien $0 = \binom{0}{k}$

Hérédité Soit $n > 0$. On suppose que pour tout $k \in \mathbb{Z}$, l'appel à « `binom1 (n-1) k` » renvoie $\binom{n-1}{k}$. Soit $k \in \mathbb{Z}$. D'après la ligne 5 et l'hypothèse d'induction, « `binom1 n k` » renvoie :

$$\binom{n-1}{k-1} + \binom{n-1}{k}$$

On a plusieurs cas :

- ⋆ Si $k < 0$ alors :

$$\binom{n-1}{k-1} + \binom{n-1}{k} = 0 + 0 = \binom{n}{k}$$

- ⋆ Si $k = 0$ alors :

$$\binom{n-1}{k-1} + \binom{n-1}{k} = 0 + 1 = \binom{n}{k}$$

⋆ Si $1 \leqslant k < n$ alors d'après le triangle de Pascal :

$$\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$$

⋆ Si $k = n$ alors :

$$\binom{n-1}{k-1} + \binom{n-1}{k} = 1 + 0 = \binom{n}{k}$$

⋆ Si $k > n$ alors :

$$\binom{n-1}{k-1} + \binom{n-1}{k} = 0 + 0 = \binom{n}{k}$$

Donc un appel à « `binom1 n k` » renvoie bien $\binom{n}{k}$.

Question 3 – Pour $n \in \mathbb{N}$ on note $C(n)$ le nombre d'appels à la fonction `binom1` lors de l'exécution de « `binom1 n k` » où $k \in \mathbb{Z}$. Alors :

$$C(0) = 1 \qquad\qquad C(n+1) = 1 + 2C(n) \text{ pour } n \in \mathbb{N}.$$

Par une récurrence immédiate sur $n \in \mathbb{N}$, $C(n) = 2^{n+1} - 1$ et donc le temps d'exécution est en $\Theta(2^n)$.

Question 4 – Pour tout `n` $\in \mathbb{N}$, on pose `k_n = 0`. Lors de l'appel à « `binom2 n k_n` », la condition du `if` ligne 3 est vérifiée, donc la fonction s'arrête sans appel récursif. La complexité est en $\Theta(1)$.

Question 5 – Pour tout `n` $\in \mathbb{N}$, on pose `k_n = 1`. Soit `C(n)` la complexité temporelle d'un appel à « `binom2 n k_n` ». Les valeurs de $C(0)$ et $C(1)$ sont fixées. De plus, pour `n` $\geqslant 2$, l'appel à « `binom2 n k_n` » fait un appel récursif à « `binom2 (n - 1) k_n-1` » et un appel récursif à « `binom2 (n-1) 0` ». Le deuxième appel récursif s'exécute en temps $\Theta(1)$. Donc :

$$C(n) = C(n-1) + T(n)$$

où $T(n) = \Theta(1)$. Ainsi :

$$C(n) = C(1) + \sum_{k=2}^{n} T(k) = \Theta(n)$$

Question 6 –

Idée.

- Les cas de bases renvoient 1 et se font en temps $\Theta(1)$.
- Lorsqu'on fait les 2 appels récursifs :
  - On renvoie la somme des appels récursifs.
  - La complexité est la somme des complexités des appels récursifs.

Par conséquent, la complexité de la fonction est de l'ordre de la valeur renvoyée (c'est à dire de l'ordre de $\binom{n}{k}$).

Preuve. Soit $n, k \in \mathbb{N}$ avec $0 \leqslant k \leqslant n$ et $C(n,k)$ le nombre d'appels à la fonction `binom2` lors de l'exécution de « `binom2 n k` ». On a :

$$C(n,0) = 1 \qquad\qquad C(n,n) = 1$$

Si $0 < k < n$:

$$C(n,k) = 1 + C(n-1,k) + C(n-1,k-1).$$

Par une récurrence immédiate sur n :

$$C(n,k) = 2\binom{n}{k} - 1$$

En particulier, on remarque que les arguments des appels récursifs vérifient $0 \leqslant k \leqslant n-1$ et $0 \leqslant k-1 \leqslant n-1$; ce qui permet d'utiliser l'hypothèse de récurrence. Ainsi, le temps d'exécution est en $\Theta(\binom{n}{k})$

Puisque les coefficients binomiaux sont maximaux lorsque $k = \lfloor n/2 \rfloor$, on pose $k_n = \lfloor n/2 \rfloor$. Par la formule de Stirling, lorsque n est pair (le cas impair se traite de la même façon) :

$$\binom{n}{k_n} = \frac{n!}{(n/2)!^2} = \Theta\left(\frac{n^n}{e^n}\sqrt{n}\left(\frac{e^{n/2}}{(n/2)^{n/2}\sqrt{n/2}}\right)^2\right) = \Theta\left(\frac{2^n}{\sqrt{n}}\right)$$

En conclusion la complexité d'un appel à « `binom2 n k`$_n$ » est en $\Theta(2^n/\sqrt{n})$.

Exercice 5. Algorithme d'Euclide

Question 1 –

```
let pgcd n0 m0 =
  if n0 = 0 && m0 = 0 then failwith "non defini";
  let rec aux n m = match m with
    | 0 -> n
    | m -> aux m (n mod m) in
  aux n0 m0;;
```

Question 2 – Il suffit de montrer que la fonction `aux` termine. Montrons le par récurrence forte sur son deuxième argument `m`.

Initialisation Si `m = 0` alors la fonction termine.

Hérédité Sinon, on fait un appel récursif à la fonction `aux` avec comme deuxième argument « `n mod m` » qui est strictement plus petit que `m`. Par l'hypothèse de récurrence, la fonction termine.

Question 3 – Il suffit de montrer la validité de la fonction `aux`. Montrons le par récurrence forte sur son deuxième argument `m`.

Initialisation Si `m = 0` alors la fonction renvoie `1` ce qui est correct.

Hérédité Sinon, la fonction renvoie « `aux m (n mod m)` ». Or le PGCD de `n` et `m` et le PGCD de `m` et `n mod m` sont égaux. Donc par hypothèse de récurrence, la fonction renvoie la bonne valeur.