Feuille d'exercice 1

Exercice 1 – $\Sigma = \{Nil : liste, Cel : \mathbb{N} \times liste \to liste\}$

1.
$$longueur(l) = \begin{cases} longueur(Nil) = 0 \\ longueur(Cel(x,l)) = 1 + longueur(l) \end{cases}$$

2.
$$concat(l1, l2) = \begin{cases} concat(Nil, l2) = l2 \\ concat(Cel(x,l), l2) = Cel(x, concat(l, l2)) \end{cases}$$

3.
$$\forall l_1, l_2 \in lists, \quad longueur(concat(l_1, l_2)) = longueur(l_1) + longueur(l_2)$$

On résone par récurrence sur l_1

- **Cas de base** $l_1 = Nil$

$$\begin{aligned} longueur(l_1, l_2) &= longueur(l_2) \\ &= longueur(Nil) + longueur(l_2) \\ &= longueur(l_1) + longueur(l_2) \end{aligned}$$

- **Cas récursif** $l_1 = Cel(x, l_1')$
  Hypothèse de récurence : $longueur(concat(l_1', l_2)) = longueur(l_1') + longueur(l_2)$

$$\begin{aligned} longueur(concat(Cel(x, l_1'), l_2)) &= longueur(Cel(x, concat(l_1', l_2))) \\ &= 1 + longueur(concat(l_1', l_2)) \\ &= 1 + longueur(l_1') + longueur(l_2) \\ &= longueur(Cel(x, l_1')) + longueur(l_2) \\ &= longueur(l_1) + longueur(l_2) \end{aligned}$$

4.
$$\forall l_1, l_2, l_3 \in lists, \quad concat(l_1, concat(l_2, l_3)) = concat(concat(l_1, l_2), l_3)$$

On résonne par récurence sur l_1

- **Cas de base** $l_1 = Nil$

$$\begin{aligned} concat(l_1, concat(l_2, l_3)) &= concat(l_2, l_3) \\ &= concat(concat(Nil, l_2), l_3) \end{aligned}$$

- **Cas récursif** $l_1 = Cel(x, l_1')$
  Hypothèse de récurence : $concat(l_1, concat(l_2, l_3)) = concat(concat(l_1, l_2), l_3)$

$$\begin{aligned} concat(l_1, concat(l_2, l_3)) &= concat(Cel(x, l_1'), concat(l_2, l_3)) \\ &= Cel(x, concat(l_1', concat(l_2, l_3))) \\ &= Cel(x, concat(concat(l_1', l_2), l_3)) \\ &= concat(Cel(x, concat(l_1', l_2)), l_3) \\ &= concat(concat(Cel(x, l_1'), l_2), l_3) \\ &= concat(concat(l_1, l_2), l_3) \end{aligned}$$

Exercice 2 –

$$rev : list \to list$$

$$rev(l) = \begin{cases} rev(Nil) & = Nil \\ rev(Cel(x,l)) & = concat(rev(l), Cel(x, Nil)) \end{cases}$$

1.

$$\forall l \in lists, \quad longueur(rev(l)) = longueur(l)$$

Par récurence sur l

- **Cas de base** $l = Nil$

$$longueur(rev(Nil)) = longueur(Nil)$$

- **Cas récursif** $l = Cel(x, l')$

  Hypothèse de récurence : $longueur(rev(l')) = longueur(l')$

$$\begin{aligned} longueur(rev(l)) &= longueur(rev(Cel(x,l'))) \\ &= longueur(concat(rev(l'), Cel(x, Nil))) \\ &= longueur(rev(l')) + longueur(Cel(x, Nil)) && \text{vue ex(1.3)} \\ &= longueur(rev(l')) + 1 \\ &= longueur(l') + 1 && \text{par HP} \\ &= longueur(l) \end{aligned}$$

2.

$$\forall l_1, l_2 \in lists, \quad rev(concat(l_1, l_2)) = concat(rev(l_2), rev(l_1))$$

Par récurence sur l_1

- **Cas de base** $l_1 = Nil$

$$\begin{aligned} rev(concat(l_1, l_2)) = rev(l_2) &= concat(rev(l_2), Nil) \\ &= concat(rev(l_2), rev(l_1)) \end{aligned}$$

- **Cas récursif** $l_1 = Cel(x, l_1')$

  Hypothèse de récurence : $rev(concat(l_1', l_2)) = concat(rev(l_2), rev(l_1'))$

$$\begin{aligned} rev(concat(l_1, l_2)) &= rev(concat(Cel(x, l_1'), l_2)) \\ &= rev(Cel(x, concat(l_1', l_2))) \\ &= concat(rev(concat(l_1', l_2)), Cel(x, Nil)) \\ &= concat(concat(rev(l_2), rev(l_1'), Cel(x, Nil))) && \text{par HP} \\ &= concat(rev(l_2), concat(rev(l_1')), Cel(x, Nil)) \\ &= concat(rev(l_2), rev(Cel(x, l_1'))) \\ &= concat(rev(l_2), rev(l_1)) \end{aligned}$$

3.

$$\forall l \in lists, \quad rev(rev(l)) = l$$

Par récurence sur l

- **Cas de base** $l = Nil$

$$rev(rev(l)) = Nil = l$$

- **Cas récursif** $l = Cel(x, l')$
  Hypothèse de récurence : $rev(rev(l')) = l'$

$$\begin{aligned} rev(rev(l)) &= rev(rev(Cel(x, l'))) \\ &= rev(concat(rev(l'), Cel(x, Nil))) \\ &= concat(rev(Cel(x, Nil)), rev(rev(l'))) \\ &= concat(rev(Cel(x, Nil)), l') && \text{par HP} \\ &= concat(Cel(x, Nil), l') \\ &= l \end{aligned}$$

Exercice 3 – $rev\_rt : list \to list$

1.

$$rev\_acc(l, acc) = \begin{cases} rev\_acc(Nil, acc) & = acc \\ rev\_acc(Cel(x, l), acc) & = rev\_acc(l, Cel(x, acc)) \end{cases}$$

$$rev\_rt(l) = rev\_acc(l, Nil)$$

2.

$$\forall l \in lists, rev\_rt(l) = rev(l)$$

Pour cela nous allons montrer par récurence :

$$rev\_acc(l, acc) = concat(acc, rev(l))$$

- **Cas de base** $l = Nil$

$$\begin{aligned} rev\_acc(l, acc) = acc &= concat(Nil, acc) \\ &= concat(rev(l), acc) \end{aligned}$$

- **Cas récursif** $l = Cel(x, l')$
  Hypothèse de récurence : $rev\_acc(l', acc) = concat(rev(l'), acc)$

$$\begin{aligned} rev\_acc(l, acc) &= rev\_acc(Cel(x, l'), acc) \\ &= rev\_acc(l', Cel(x, acc)) \\ &= concat(rev(l'), Cel(x, acc)) && \text{Par HP} \\ &= concat(rev(l'), concat(Cel(x, Nil), acc)) \\ &= concat(concat(rev(l'), Cel(x, Nil), acc)) \\ &= concat(concat(rev(Cel(x, l')), acc)) \\ &= concat(rev(l), acc) \end{aligned}$$

Exercice 4 – varaible locales définies par la signature $\{n, x, Add, Mul, Let\}$

1. Montrons que, si $x \notin fv(e)$, alors pour tout v on a $e[x := v] = e$
   On va donc procédé par récurence sur e
   - **Cas de base** — $e = n \Rightarrow e[x := v] = e$
     — $e = (y \neq x) \Rightarrow e[x := v] = e$
     — $e = x$ impossible
   - **Cas récursif** Supossons $x \notin fv(e_1) \wedge x \notin fv(e_2) \Rightarrow (e_1[x := v] = e_1) \wedge (e_2[x := v] = e_2)$
     On sait que $x \notin fv(e)$
     — $e = Add(e_1, e_2)$ Par definition de Add :
     $fv(e) = fv(e_1) \cup fv(e_2)$ donc $x \notin fv(e) \Rightarrow x \notin fv(e_1) \wedge x \notin fv(e_2)$

     On a donc :

     $$\begin{aligned} e[x := v] &= Add(e_1, e_2)[x := v] \\ &= Add(e_1[x := v], e_2[x := v]) \\ &= Add(e_1, e_2) && \text{Par HP} \end{aligned}$$

     — $e = Mul(e_1, e_2)$ Même méthode que pour Add

     $$\begin{aligned} e[x := v] &= Mul(e_1, e_2)[x := v] \\ &= Mul(e_1[x := v], e_2[x := v]) \\ &= Mul(e_1, e_2) && \text{Par HP} \end{aligned}$$

     — $e = (\text{let } y = e_1 \text{ in } e_2), y \neq x$

     $fv(e) = fv(e_1) \cup (fv(e_2) \setminus \{x\})$
     Si $y = x$ **:** On peut dir :
     $x \notin fv(e) \Rightarrow x \notin fv(e_1)$ car $y = x$ ($\bigtriangledown$)

     Par definition

     $$\begin{aligned} (\text{let } y = e_1 \text{ in } e_2)[x := v] &= (\text{let } y = e_1[x := v] \text{ in } e_2) \\ &= (\text{let } y = e_1 \text{ in } e_2) && \text{Par HP et } \bigtriangledown \end{aligned}$$

     Si $y \neq x$ **:** On peut dir :
     $x \notin fv(e) \Rightarrow x \notin fv(e_1) \wedge x \notin fv(e_2)$ car $y \neq x$ ($\bigtriangleup$)

     Par definition

     $$\begin{aligned} (\text{let } y = e_1 \text{ in } e_2)[x := v] &= (\text{let } y = e_1[x := v] \text{ in } e_2[x := v]) \\ &= (\text{let } y = e_1 \text{ in } e_2) && \text{Par HP et } \bigtriangleup \end{aligned}$$

2. $Let(x, e_1, e_2) \Leftrightarrow Let(y, e_1, e_2[x := y])$
   On va modifier cette propriété pour qu'elle soit vrai
   (a) On prend $Let(x, Add(1, 2), Add(x, y)) \rightarrow 3 + y$
   Si on subsitue x par y on aurra $Let(y, Add(1, 2), Add(y, y)) \rightarrow 6$
   (b) Pour que les expressions soit équivalentes il faut que $x \notin fv(e_2)$
   (c) Montrons :
   $$x \notin fv(e_2) \Rightarrow Let(x, e_1, e_2) \Leftrightarrow Let(y, e_1, e_2[x := y])$$
   Autrement dit en notant $v_1 = eval(e_1, \rho)$:
   $$x \notin fv(e_2) \Rightarrow eval(e_2, \rho[x \mapsto v_1]) = eval(e_2[x := y], \rho[y \mapsto v_1])$$
   Supossons que $x \notin fv(e_2)$
   Par récurence sur n_2

- Cas n : immédiat
- Cas z : Si
  - $z = x$: Alors $eval(x, \rho[x \mapsto v1]) = v_1$ et $eval(x[x := y], \rho[y \mapsto v1]) = eval(y, \rho[y \mapsto v1]) = v1$
  - $z = y$: interdit car $y \in fv(e_2)$
  - Sinon : résultat immédiat
- Cas $Add(e'_1, e'_2)$: Alors :

$$\begin{aligned} eval(Add(e'_1, e'_2), \rho[x \mapsto v_1]) &= eval(e'_1, \rho[x \mapsto v_1]) + eval(e'_2, \rho[x \mapsto v_1]) \\ &= eval(e'_1[x := y], \rho[x \mapsto v_1]) + eval(e'_2[x := y], \rho[x \mapsto v_1]) \end{aligned}$$

  avec $y \notin fv(e'_1) \wedge y \notin fv(e'_2)$

$$\begin{aligned} &= eval(Add(e'_1[x := y], e'_2[x := y]), \rho[y \mapsto v_1]) \\ &= eval(Add(e'_1, e_2)[x := y], \rho[y \mapsto v_1]) \end{aligned}$$

- Cas Mul : similaire
- Cas $Let(z, e'_1, e'_2)$: On note $v'_1 = eval(e'_1, \rho[x \mapsto v_1])$
  On a aussi $y \notin fv(e'_1)$ donc $v'_1 = eval(e'_1[x := y], \rho[y \mapsto v_1])$
  - Si $z = x$:

$$\begin{aligned} eval(Let(z, e'_1, e'_2)[x := y], \rho[y \mapsto v_1]) &= eval(Let(z, e'_1[x := y], e'_2), \rho[y \mapsto v_1]) \\ &= eval(Let(e'_2, (\rho[y \mapsto v_1])[z \mapsto v'_1])) \end{aligned}$$

  - Si $z \neq x$ et $z \notin fv(y)$, on a

$$\begin{aligned} eval(Let(z, e'_1, e'_2)[x := y], \rho[y \mapsto v_1]) &= eval(Let(z, e'_1[x := y], e'_2[x := y]), \rho[y \mapsto v_1]) \\ &= eval(e'_2[x := y], (\rho[y \mapsto v_1])[z \mapsto v'_1]) \\ &= eval(e'_2[x := y], (\rho[z \mapsto v_1])[y \mapsto v'_1]) && (y \neq z) \\ &= eval(e'_2, (\rho[z \mapsto v_1])[x \mapsto v'_1]) && \text{H.P} \\ &= eval(e'_2, (\rho[x \mapsto v_1])[z \mapsto v'_1]) && (z \neq y) \\ &= eval(Let(z, e'_1, e'_2), \rho[x \mapsto v_1]) \end{aligned}$$

Exercice 5 – $\Sigma = \{F : arbre, N : arbre \times \mathbb{N} \times arbre \to arbre\}$

1. infixe : arbre → arbre

$$infixe(a) = \begin{cases} infixe(F) = Nil \\ infixe(N(a_1, x, a_2)) = concat(infixe(a_1)Cel(x, infixe(a_2))) \end{cases}$$

2. appartient : $\mathbb{N} \times arbre \to \mathbb{B}$

$$appartient(n, a) = \begin{cases} appartient(n, F) = False \\ appartient(n, N(a_1, x, a_2)) = (x = n) \vee appartient(n, a_1) \vee appartient(n, a_2) \end{cases}$$

3. (a) $N(N(F, 1, F), 2, N(F, 3, F))$ Vrai
   (b) $N(F, 2, N(N(F, 1, F), 3, F))$ Faux
4. $a_1 = N(F, 1, N(F, 2, N(F, 3, N(F, 4, F))))$ $\quad infixe(a_1) = Cel(1, Cel(2, Cel(3, Cel(4))))$
   $a_2 = N(N(F, 1, F), 2, N(F, 3, N(F, 4, F)))$ $\quad infixe(a_2) = Cel(1, Cel(2, Cel(3, Cel(4))))$
5. Montrons que si a est un ABR alors $infixe(a)$ est trié
   - **Cas de base** $a = F$
     On a donc $infixe(a) = Nil$ or la liste est vide est bien trié
   - **Cas récursif** $a = N(a_1, n, a_2)$ et a un ABR
     Hypothèse de récurence : $infixe(a_1) \wedge infixe(a_2)$ sont triées

$$infixe(a) = concat(infixe(a_1), Cel(n, infixe(a_2)))$$

     Par HP on sait que $infixe(a_1)$ et $infixe(a_2)$ sont bien triées or comme a un ABR n est plus grand que n'importe quel éléments de $infixe(a_1)$ et est plus petit que n'importe quel éléments de $infixe(a_2)$.

     On peut donc dir que $l = Cel(n, infixe(a_2))$ est triée et que $concat(infixe(a_2), l)$ est trié aussi donc $infixe(a)$ est triée aussi.
6. Il faut chercher dans t_1 quand $m < n$ sinon si $m > n$ dans t_2 sinon $m = n$ fini.
7. appartient_abr : $\mathbb{N} \times arbre \to \mathbb{B}$

$$appartient\_abr(n, a) = \begin{cases} appartient\_abr(F) = False & \\ appartient\_abr(N(a_1, m, a_2)) = True & \text{si } n = m \\ appartient\_abr(N(a_1, m, a_2)) = appartient\_abr(a_2) & \text{si } x < n \\ appartient\_abr(N(a_1, m, a_2)) = appartient\_abr(a_1) & \text{si } x > n \end{cases}$$

8. Montrons : $appartient\_abr(n, t) \Rightarrow appartient(n, t)$
   Par récurrence sur **t**.
   — Cas F : $appartient\_abr(n, F) = False$ rien à montrer.
   — Cas $N(t_1, m, t_2)$ on regarde quand $appartient\_abr(n, N(t_1, m, t_2))$ est vrai.
      — Cas $n = m$. Alors $appartient(n, N(t_1, m, t_2)) = true$
      — Cas $n < m \wedge appartient\_abr(n, t_1)$. En particulier $appartient\_abr(n, t_1)$ donc par H.R $appartient(n, t_1) = True$, donc $appartient(n, N(t_1, m, t_2)) = true$.
      — Cas $m < n \wedge \ldots$ similaire.
9. Montrons : $t \in ABR \wedge appartient(n, t) \Rightarrow appartient\_abr(n, t)$
   Par récurrence sur **t**.
   — Cas F : $appartient(n, F) = False$ rien à faire.
   — Cas $N(t_1, m, t_2)$. Par disjonction de cas $appartient(n, N(t_1, m, t_2))$.
      — Cas $n = m$. Alors $appartient\_abr(n, N(t_1, m, t_2)) = true$

— Cas $appartient(n, t_1) = true$. Par H.R, $appartient\_abr(n, t_1) = true$.
En outre $N(t_1, m, t_2)$ étant un ABR et n étant déjà présent dans le sous-arbre gauche, on a $n \leq m$.
Or $n = m$ a déjà été traité. Donc $appartient\_abr(n, N(t_1, m, t_2)) = true$

— Cas $appartient(n, t_2) = true$ similaire.

10. verif : $\mathbb{N} \times \mathbb{N} \times arbre \rightarrow \mathbb{B}$

$$verif(m, M, t) = \begin{cases} verif(m, M, F) = true \\ verif(m, M, N(t_1, n, t_2)) = m \leq n \leq M \wedge verif(m, n, t_1) \wedge verif(n, M, t_2) \end{cases}$$

11. Montrons que $verif(m, M, t) \Rightarrow t \in ABR \wedge \max(t) \leq M \wedge \min(t) \geq m$ fonctionne
Récurence sur t
— Cas F : correcte car F est bien un ABR
— Cas $N(t_1, n, t_2)$.
Si $verif(m, M, N(t_1, n, t_2)) = true$ alors on a $m \leq n \leq N verif(m, M, t_1), verif(m, M, t_2)$. Par H.R on a t_1 et t_2 des ABR avec pour tous éléments n_1 de t_1 $m \leq n_1 \leq n$ et pour tous éléments n_2 de t_2 $n \leq n_2 \leq M$. Donc $N(t_1, n, t_2)$ est bien un ABR de min m et de max M

12. voir fichier *ex5_ 12.ml*