TD termes et récurrences

Exercice 1 (Concaténation de listes)**.** On représente des listes d'entiers par les termes définis par la signature $\Sigma = \{\mathsf{Nil} : \mathsf{liste}, \mathsf{Cel} : \mathbb{N} \times \mathsf{liste} \longrightarrow \mathsf{liste}\}$.

1. Définir une fonction $\mathsf{longueur} : \mathsf{liste} \longrightarrow \mathbb{N}$ donnant la longueur d'une liste.
2. Définir une fonction $\mathsf{concat} : \mathsf{liste} \times \mathsf{liste} \longrightarrow \mathsf{liste}$ effectuant une concaténation de listes.
3. Démontrer que la concaténation additionne les longueurs, c'est-à-dire que

$$\forall l_1, l_2 \in \mathsf{liste}, \quad \mathsf{longueur}(\mathsf{concat}(l_1, l_2)) = \mathsf{longueur}(l_1) + \mathsf{longueur}(l_2)$$

4. Démontrer que la concaténation est associative, c'est-à-dire que

$$\forall l_1, l_2, l_3 \in \mathsf{liste}, \quad \mathsf{concat}(l_1, \mathsf{concat}(l_2, l_3)) = \mathsf{concat}(\mathsf{concat}(l_1, l_2), l_3)$$

Exercice 2 (Retournement de liste)**.** Voici une définition pour une fonction $\mathsf{rev} : \mathsf{liste} \longrightarrow \mathsf{liste}$ de retournement de liste.

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

1. Démontrer que rev préserve la longueur :

$$\forall l \in \mathsf{liste}, \quad \mathsf{longueur}(\mathsf{rev}(l)) = \mathsf{longueur}(l)$$

2. Démontrer que rev se distribue sur concat de la manière suivante :

$$\forall l_1, l_2 \in \mathsf{liste}, \quad \mathsf{rev}(\mathsf{concat}(l_1, l_2)) = \mathsf{concat}(\mathsf{rev}(l_2), \mathsf{rev}(l_1))$$

3. Démontrer que rev est involutive, c'est-à-dire que

$$\forall l \in \mathsf{liste}, \quad \mathsf{rev}(\mathsf{rev}(l)) = l$$

Exercice 3 (Correction de retournement récursif terminal)**.** La définition précédente de rev est adaptée pour spécifier le retournement de liste et démontrer ses propriétés, mais elle est assez éloignée des réalisations réelles. On va maintenant faire le lien entre les deux.

1. Donner une définition pour une fonction $\mathsf{rev\_rt} : \mathsf{liste} \longrightarrow \mathsf{liste}$ correspondant à une réalisation récursive terminale et efficace du retournement de liste.
   Indication : il faudra utiliser une fonction auxiliaire.
2. Démontrer que rev_rt est correcte, c'est-à-dire que

$$\forall l \in \mathsf{liste}, \quad \mathsf{rev\_rt}(l) = \mathsf{rev}(l)$$

   Indication : il faudra démontrer quelque chose par récurrence, mais quoi exactement ?

Exercice 4 (Expressions arithmétiques et variables libres)**.** Prenons les expressions arithmétiques avec variables locales définies par la signature $\{\mathsf{n}, \mathsf{x}, \mathsf{Add}, \mathsf{Mul}, \mathsf{Let}\}$.

1. Démontrer que, si $x \notin \mathsf{fv}(e)$, alors pour tout v on a $e[x := v] = e$.
2. Les variables introduites par Let étant locales, leur nom importe peu et peut être modifié, tant qu'on met bien à jour chaque occurrence. Ainsi l'expression $\mathsf{Let}(x, e_1, e_2)$ est équivalente à l'expression $\mathsf{Let}(y, e_1, e_2[x := y])$ obtenue en remplaçant le nom local x par un autre nom local y.
   (a) Donner un contre-exemple.

(b) Énoncer la condition qui doit être vérifiée pour que les deux expressions soient effectivement équivalentes.

(c) Démontrer l'équivalence sous la condition citée.

Exercice 5 (Arbres binaires de recherche)**.** On représente des arbres binaires dont les nœuds sont étiquetés par des entiers avec les termes définis par la signature $\Sigma = \{\mathsf{F} : \text{arbre}, \mathsf{N} : \text{arbre} \times \mathbb{N} \times \text{arbre} \longrightarrow \text{arbre}\}$

1. Définir une fonction infixe : arbre $\longrightarrow$ liste qui étant donné un arbre binaire t, renvoie la liste composée des entiers contenus dans l'arbre dans l'ordre donné par un parcours infixe (l'entier étiquetant un nœud est donc pris en compte entre les deux sous-arbres de ce nœud).
2. Définir une fonction appartient : $\mathbb{N} \times$ arbre $\longrightarrow \mathbb{B}$ testant l'appartenance d'un entier à un arbre et renvoyant un booléen.

On s'intéresse aux arbres binaires de recherche qui vérifient que dans chaque sous-arbre de la forme $\mathsf{N}(t_1, n, t_2)$, tous les entiers stockés dans le sous-arbre t_1 sont inférieurs ou égaux à n qui est lui-même inférieur ou égal à tous les entiers stockés dans le sous-arbre t_2.

3. Les termes suivants sont-ils des arbres binaires de recherche ?
   (a) N(N(F,1,F),2,N(F,3,F))
   (b) N(F,2,N(N(F,1,F),3,F))
4. Écrire deux termes différents qui correspondent à des arbres binaires de recherche contenant les valeurs 1, 2, 3 et 4. Donner la valeur de la fonction infix pour ces arbres.
5. Démontrer que si l'arbre t est un arbre binaire de recherche, alors le mot infix(t) est trié.

La structure d'arbre binaire de recherche permet de chercher efficacement un élément.

6. Lorsque l'on recherche un élément n dans un arbre binaire de recherche de la forme $\mathsf{N}(t_1, m, t_2)$, est-il utile de fouiller l'ensemble de l'arbre ? dans quel cas faut-il chercher dans t_1 et dans quel cas dans t_2 ?
7. Définir une fonction appartient_abr : $\mathbb{N} \times$ arbre $\longrightarrow \mathbb{B}$ telle que inabr(n, t) teste si l'entier n est présent dans l'arbre t, en supposant que t est un arbre binaire de recherche.
8. Montrer que si appartient_abr(n, t) = vrai alors appartient(n, t) = vrai.
9. Montrer que si appartient(n, t) = vrai et t est un arbre binaire de recherche, alors appartient_abr(n, t) = vrai.

On peut également définir une procédure efficace pour vérifier qu'un arbre binaire est bien un arbre binaire de recherche.

10. Définir une fonction verif : $\mathbb{N} \times \mathbb{N} \times$ arbre $\longrightarrow \mathbb{B}$ telle que verif(min, max, t) renvoie vrai si et seulement si t est un arbre binaire de recherche dont tous les entiers x vérifient $min \leq x \leq max$.
11. Démontrer que la fonction verif est correcte.

Comme nous l'avons déjà dit, les termes et fonctions récursives sur les termes sont facilement représentables en caml.

12. Définir en caml un type des arbres binaires, ainsi que les fonctions infix, appartient, appartient_abr et verifie.