Matthieu Boyer
Planche A

On the banks of Babylon

On s'intéresse à une représentation en OCamL de suites dites *paresseuses*. Pour ça on va représenter une suite paresseuse comme suit :

```
type 'a seq =
        | Nil
        | Cons of 'a * (unit -> 'a seq)
```

1. Représentez la suite $u_n = n$ comme une `int seq` .
2. Donnez une fonction `int -> 'a seq -> 'a list` qui renvoie les n premiers termes d'une suite paresseuse.
3. Donnez une fonction `'a -> ('a -> 'a) -> 'a seq` qui à x et f renvoie la séquence des itérées de f en x $(f^k(x))_k$
4. Donnez une fonction `'a seq -> 'a seq -> 'a seq` qui intercale deux suites en affichant tour à tour un élément de chaque suite.
5. Donnez une fonction `'a seq -> ('a -> bool) -> 'a seq` qui filtre par un prédicat une suite paresseuse.
6. Donnez une fonction `'a seq -> ('a -> 'b) -> 'b seq` qui applique une fonction à une suite paresseuse.

Solution : On the banks of Babylon

`www.cl.cam.ac.uk/teaching/1920/FoundsCS/FoCS-201920-9.pdf`

CCC

On s'intéresse à une représentation en OCamL d'un type pour les types, utilisable pour le typage.

1. Donnez un type somme, dit type de base, qui peut contenir un entier, une chaîne de caractère ou une valeur booléenne.
2. Donnez un type `('a, 'b) prod` qui représente le type des couples d'un élément de type a et d'un élément de type b.
3. Donnez un type `('a, 'b) coprod` qui représente le type d'un élément qui est soit de type a soit de type b.
4. Donnez un type `('a, 'b) exp` qui représente le type des fonctions de $a \to b$.
5. Donnez un type récursif `type` qui regroupe tous les types précédents et permet de décrire les types OCamL.
6. Donnez une fonction `type -> type -> type option` qui vérifie si son premier élément peut être appliqué au second élément et qui renvoie `Some x` où `x` est le type de retour ou la valeur `None`.

Matthieu Boyer
Planche B : Corrigés

Il faut cultiver son jardin

On s'intéresse à l'interprétation d'expression arithmétiques en OCamL. On se donne n variables $x_1, \dots, x_n$, stockées dans un tableau `x: int array` . On peut accéder à la valeur de x_i avec l'expression OCamL `x.(i)` .

1. Donnez un type récursif `arith` qui permet de représenter les expressions arithmétiques entières $+, \times, -, /,$ mod , et permet de faire appel à une variable par son indice.
2. Donnez une fonction `arith -> bool` vérifiant si une expression est bien formée. Il suffit pour ça de vérifier si les variables sont bien définies.
   Dans la suite on ne vérifiera plus que les expressions sont bien définies.
3. Donnez une fonction `arith -> int` qui va renvoyer le résultat de l'expression en argument.
4. Donnez un type `expr` qui permet de modéliser soit les assignations de variables, soit de renvoyer une valeur. Comment définir un type `prog` qui fonctionne comme une liste d'expressions.
5. Donnez une fonction `prog -> int` qui renvoie la première valeur de renvoi du programme.

Solution : Il faut cultiver son jardin