L2 biologie

TD d'Informatique No. 1

Ce TD de révisions reprend un examen typique du L1.

Exercice 1 : On considère la session ML ci-dessous ; indiquez la réponse que donnerait l'ordinateur pour chacune des commandes successives :

1. `2 + 3 ; ;`
2. `2 + 4 / 1 + 1 ; ;`
3. `let "deux" = 1 + 1 ; ;`
4. `"deux" + "trois" ; ;`
5. `let p x = x > 0 ; ;`
6. `"bla " ^ "bla " ; ;`
7. `"bla " @ "bla " ; ;`
8.
```
let rien l = match l with [] -> true
                     | x : :l' -> false ; ;
```
9. `let g n = if n < 2 then 1 else n * (g (n - 1)) ; ;`
10. `let rec f n = if n < 2 then 1 else n * (f (n - 1)) ; ;`
11. `f 3 ; ;`
12.
```
let taille l = match l with [] -> 0
                       | x : :l' -> 1 + (taille l') ; ;
```

Exercice 2 : Écrivez une fonction `centuple` qui prend un nombre entier en entrée, et retourne sa valeur multipliée par 100.
Donnez ensuite le type de votre fonction.

Exercice 3 : Écrivez une fonction `divise` qui prend deux nombres entiers `m` et `n` en entrée, et retourne un booléen : `true` si `m` est un diviseur de `n`, `false` sinon.
Donnez ensuite le type de votre fonction.

Exercice 4 : Écrivez une fonction récursive `puissance` qui prend un nombre flottant `x` et un nombre entier `n` en entrée, et retourne `x` à la puissance `n`.
Donnez ensuite le type de votre fonction.

Dans les 3 questions qui suivent, on représente une couleur quelconque, comme c'est le cas sur un écran couleur, par un mélange de trois couleurs : `rouge`, `vert` et `bleu`. On utilise une approximation entière des intensités de chacune de ces trois couleurs, comprise entre 0 et 100. Par exemple, le noir est représenté par trois intensités égales à 0 ; le blanc est représenté par trois intensités égales à 100 ; un bleu pur est représenté par des intensités de `rouge` et de `vert` égales à 0 et une intensité de `bleu` non nulle ; etc.

Exercice 5 : Déclarez un type de données `couleur` pour représenter ces couleurs à trois composantes.

Exercice 6 : Écrivez la fonction `estNoir` qui prend une couleur `c` en argument et retourne un booléen : `true` si `c` est la couleur noire, `false` sinon.

Exercice 7 : Écrivez la fonction `correcte` qui prend en argument une donnée `c` de type `couleur`, et retourne un booléen indiquant si chacune des trois composantes est effectivement comprise entre 0 et 100.

Dans toutes les questions qui suivent, on considère le type suivant : `type element = A | T | G | C | N ; ;` qui représente les différentes lectures qui peuvent être faites d'un nucléotide durant un séquençage, le « `N` » représentant une lecture douteuse, donc un nucléotide indéterminé.

Exercice 8 : Écrivez la fonction `complementaire` qui prend un élément (de type `element`) en entrée et retourne son complementaire, sachant que le complémentaire de `N` est lui même.
Donnez ensuite le type de votre fonction.

Exercice 9 : Écrivez une fonction récursive `complet` qui prend en entrée une liste d'éléments (de type `element list`, représentant donc une séquence de nucléotides lue par séquençage) et qui retourne un booléen qui dit si la lecture est totalement réussie (c'est-à-dire que `N` n'apparait jamais dans la liste).
Donnez ensuite le type de votre fonction.

Exercice 10 : Écrivez une fonction récursive `supprime` qui prend un élément `e` quelconque (de type `element`) et une liste d'éléments `l` en entrée, et retourne la liste obtenue en supprimant toutes les occurences de `e` dans `l`.

Exercice 11 : Écrivez une fonction récursive `lectureInverse` qui prend en entrée une liste d'éléments (chacun de type `element`) et qui donne en sortie la liste obtenue en inversant l'ordre de la liste et en passant au complémentaire pour chaque élément. Par exemple `(lectureInverse [A ;A ;A ;T ;C ;G])` a pour résultat la liste `[C ;G ;A ;T ;T ;T]`

L2 biologie

TD d'Informatique No. 2

Gestion d'une bibliothèque de prêts

L'objectif de ce TD est d'écrire un type de données muni de fonctions de gestion du stock de livres et de l'ensemble des abonnés d'une bibliothèque.

Exercice 1 : Un abonné est fiché à la bibliothèque avec les informations suivantes : son `numero` d'inscription qui sert de clef, son `nom`, son `adresse` et son numero de `tel`éphone. Écrivez le type `abonne` correspondant.

Exercice 2 : Un livre est répertorié à la bibliothèque par son numéro d'`ISBN` qui sert de clef, la liste de ses `auteurs`, son `titre`, le `nombre` d'exemplaires qui appartiennent à la bibliothèque et la liste des numéros des abonnés qui détiennent actuellement un exemplaire de ce livre en `emprunts`. Écrivez de type `livre` correspondant. Déclarez ensuite trois livres différents quelconques.

Exercice 3 : La gestion d'une bibliothèque impose de connaître à tout moment deux choses : la liste des abonnés `inscrits` à la bibliothèque et la liste des livres du `stock` de la bibliothèque, empruntés ou non. Écrivez le type `biblio` correspondant. Déclarez ensuite une bibliothèque `vide` qui n'a aucun abonné ni aucun livre.

On gèrera les listes d'inscrits et le stock en les triant sur la clef de manière strictement *décroissante*. Par ailleurs les numéros des abonnés sont simplement incrémentés de 1 à chaque nouvel abonné.

Exercice 4 : Écrivez la fonction `inscrire` qui ajoute un nouvel abonné à une bibliothèque. On supposera sans le vérifier que le nouvel abonné n'est pas déjà inscrit.

Exercice 5 : Écrivez une fonction `desabonne` qui supprime un abonné (dont on connaît le numéro) de la liste des abonnés inscrits d'une bibliothèque. Ne rien changer si le numéro n'était déjà pas parmi les inscrits.

Exercice 6 : Écrivez une fonction `autorisé` qui vérifie si un numéro d'abonné fait réellement parti des abonnés inscrits d'une bibliothèque.

Exercice 7 : Écrivez la fonction `acheter` qui ajoute dans le stock de la bibliothèque `n` exemplaires d'un livre qui n'y est pas déjà. Laisser le stock inchangé si le numéro ISBN fourni était par erreur déjà en stock.

Exercice 8 : Écrivez la fonction `completer` qui ajoute dans le stock de la bibliothèque `n` exemplaires d'un livre qui y est déjà. Laisser le stock inchangé si par erreur le numéro ISBN fourni n'appartenait pas déjà au stock.

Exercice 9 : Écrivez la fonction `emprunter` qui permet à un abonné inscrit à la bibliothèque d'emprunter un exemplaire d'un livre du stock. Si le numéro de l'abonné n'est pas dans la liste des inscrits, ou si le numéro ISBN du livre n'appartient pas au stock, ou enfin si l'emprunteur a déjà un exemplaire de ce livre, ne pas préter le livre et donc laisser l'état de la bibliothèque inchangé.

Exercice 10 : Écrivez de même la fonction `rendre` qui permet à un abonné de rendre un exemplaire de livre qu'il a emprunté.

Exercice 11 : Écrivez une fonction qui fournit le nombre de livres empruntés par un abonné dont on connaît le numéro.

TD d'Informatique No. 3

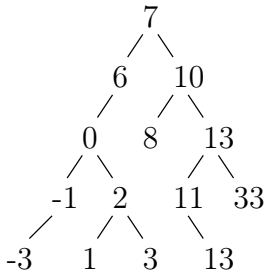
1 Arbres Binaires de Recherche

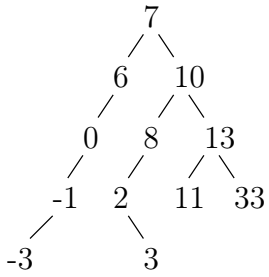
Dans toute la suite, on considère le type des arbres binaires suivant :

```
type 'a ABin = empty
             | node of 'a * ('a ABin) * ('a ABin) ;;
```

Un arbre binaire est dit *de recherche* si en tout nœud de l'arbre :
- la valeur qui est en ce nœud est supérieure ou égale à toutes les valeurs contenues dans son sous-arbre de gauche
- et cette valeur est strictement inférieure à toutes les valeurs contenues dans le sous-arbre de droite

Exercice 1 : Pour chacun des deux arbres ci-dessous, dites s'il est de recherche ou non :





Justifiez vos réponses.

Exercice 2 : Écrivez en ML le sous-arbre du premier arbre de l'exercice précédent dont la racine est 0.

Exercice 3 : Tracez graphiquement chacun des trois arbres ci-dessous et dites s'il est complet puis s'il est de recherche.

```
let a1 = node(7,node(6,node(0,empty,empty),empty),
                node(33,empty,empty)) ;;
let a2 = node(10,node(8,empty,empty),
                 node(13,node(11,empty,empty),
                         node(33,empty,empty))) ;;
let a3 = node(2,node(-3,empty,empty),
                node(2,node(-1,empty,empty),
                       node(3,empty,empty))) ;;
```

2 Opérations sur les A.B.R.

Exercice 4 : Écrivez une fonction `maximum` qui prend en entrée un arbre binaire `a` *que l'on suppose de recherche*, et fournit en sortie la valeur maximale contenue dans cet arbre. Écrivez de même une fonction `minimum`. Ces fonctions devront échouer sur les arbres vides. On prendra soin de produire des algorithmes qui exploitent efficacement le fait que `a` soit de recherche.

Exercice 5 : Écrivez une fonction `deRecherche` qui prend un arbre binaire quelconque en entrée et retourne un booléen qui dit si l'arbre est de recherche.

Exercice 6 : Développez à la main les calculs effectués par la fonction `deRecherche` sur les arbres `a1` et `a3` précédents.

Exercice 7 : Écrivez une fonction `appartient` qui dit si une valeur `v` appartient à un arbre binaire `a` *supposé de recherche*. On prendra soin de produire un algorithme qui exploite efficacement le fait que `a` soit de recherche.

Exercice 8 : Écrivez une fonction `insere` qui insère une valeur `v` dans un arbre binaire `a` de telle sorte que si `a` est de recherche, alors le résultat aussi. *Indication* : faire l'insertion en une feuille de l'arbre.

Exercice 9 : Réécrire la fonction `insere` pour qu'elle insère l'élément `v` à la racine de l'arbre. *Indication* : il faut d'abord écrire une fonction de `coupe` de l'arbre, qui produit un couple d'A.B.R., l'un ne contenant que des éléments inférieurs ou égaux à `v` et l'autre des éléments strictement supérieurs.

3 Stratégies de suppression dans un A.B.R.

Il s'agit d'écrire une fonction `supprime` qui supprime une valeur `v` d'un arbre binaire `a` *supposé de recherche*, de telle sorte que le résultat est encore de recherche.

Si `v` n'est pas dans l'arbre, le résultat est conventionnellement l'arbre `a` tel quel plutôt que de produire une erreur. Sinon, on est confronté au problème de supprimer un nœud d'arbre puis de « raccrocher » d'une manière ou d'une autre les deux sous-arbres qui sont ses fils. Ce n'est pas si simple et plusieurs stratégies sont possibles.

Pour chacune des stratégies proposées ci-dessous, il est conseillé de se faire un dessin des opérations à effectuer avant de programmer.

Exercice 10 : On peut remplacer le nœud par la valeur maximale du sous-arbre de gauche. *Indication* : on peut pour cela écrire d'abord une fonction `extraitMax` qui fournit à la fois la valeur maximale d'un A.B.R. et l'arbre obtenu en la supprimant.

Exercice 11 : On peut faire de même avec la valeur minimale du sous-arbre de droite.

Exercice 12 : On peut raccrocher le sous-arbre de gauche sous la feuille la plus à gauche du sous-arbre de droite. *Indication* : écrire d'abord une fonction `raccrocheGauche`.

Exercice 13 : On peut raccrocher le sous-arbre de droite sous la feuille la plus à droite du sous-arbre de gauche...

Exercice 14 : Enfin on peut choisir l'une des stratégies précédentes en fonction de la forme du sous-arbre de la valeur à supprimer, afin d'équilibrer au mieux l'arbre résultat. Ceci est une question subsidiaire.

L2 biologie

TD d'Informatique No. 4

On s'intéresse à la reconnaissance de motifs dans une séquence et l'on considère l'alphabet des nucléodites défini par :

```
type nucleotide = A | T | G | C ;;
```

Divers sans automates

Exercice 1 : Écrivez une fonction `prefixmax` qui prend en entrée deux mots (séquences de nucléotides) et qui retourne leur plus long préfixe commun.
Par exemple, `prefixmax([A;C;G;T;T;C;A],[A;C;G;T;A;C;T;A;C;T;A])` donne `[A;C;G;T]`.

Exercice 2 : Écrivez une fonction `repartition` qui calcule le pourcentage de AT dans une séquence.

Exercice 3 : Écrivez une fonction `quasiprefixe` qui prend en arguments un motif `m` et un mot `o`, et dit si `m` est un préfixe de `o` en autorisant qu'une lettre au plus ne coïncide pas.
Par exemple `AATTA` est un quasipréfixe de `AACTATGAAC`.

Exercice 4 : Écrivez une fonction `quasimotif` qui prend en arguments un motif `m` et un mot `o`, et dit si `m` est un sous-mot de `o` en autorisant qu'une lettre au plus ne coïncide pas.

Fonction de fabrication d'un automate

On a vu en cours comment fabriquer « à la main » un automate à partir d'un motif :
- On construit la table dont les colonnes sont les états de 0 à la longueur du motif et les lignes sont les nucléotides pouvant être lus (4 lignes donc). Dans chaque case on met l'état suivant résultant de la nouvelle lecture de nucléotide.
- Le numéro de l'état correspond au nombre de lettres reconnues et par conséquent l'état de chaque case est la longueur du plus grand suffixe de la séquence lue qui coïncide avec un préfixe du motif.
- La colonne de l'état *en succès*, correspondant à la longueur du motif, est traitée à part :
  - si l'on cherche simplement l'existence du motif alors l'état suivant reste toujours l'état en succès
  - si l'on accepte des motifs se chevauchant alors l'état suivant est traité comme toutes les autres colonnes
  - sinon l'état en succès est traité comme l'état 0
- On représente l'automate obtenu par la liste des triplets `(e1,c,e2)` où `e2` est l'état obtenu à partir de l'état `e1` après avoir lu le nucléotide `c`.

Exercice 5 : Construisez à la main la représentation de l'automate correspondant au motif `ATGATC`.

Ici, on se propose de programmer une fonction réalisant *automatiquement* cette construction de l'automate. Nous allons progresser en s'attaquant à des sous-problèmes successifs :

Exercice 6 : Écrivez une fonction `suffmax` qui prend en arguments deux mots `m1` et `m2` et retourne la taille du plus grand suffixe de `m1` qui soit un préfixe de `m2`.

Exercice 7 : Écrivez une fonction `extrait` qui prend en arguments un entier `n` et un motif `m`, et retourne le mot formé des `n` premiers nucléotides du motif `m`.
Par exemple `extrait (3,[A;A;T;T;A])` vaut `[A;A;T]` et `extrait (7,[A;A;T;T;A])` fournit un message d'erreur.

Exercice 8 : Écrivez une fonction `suivant` qui prend en arguments un état de départ `e1`, un nucléotide `c` et le motif `m`, et retourne l'état résultant de la lecture de `c` à partir de l'état `e1`.
Nota : partant de l'état en succès,on doit y rester.

Exercice 9 : Écrivez une fonction `listArcs` qui prend en argument un état `e1` de l'automate et le motif `m`, et fournit la liste des arcs de l'automate qui partent de cet état `e1` (c'est-à-dire les 4 triplets `(e1,c,e2)` correspondant aux 4 nucléotides).

Exercice 10 : Écrivez une fonction `enumere` qui prend en entrée un entier positif `n` et fournit la liste des entiers allant de `0` à `n` : `[0;1;...;n]`.

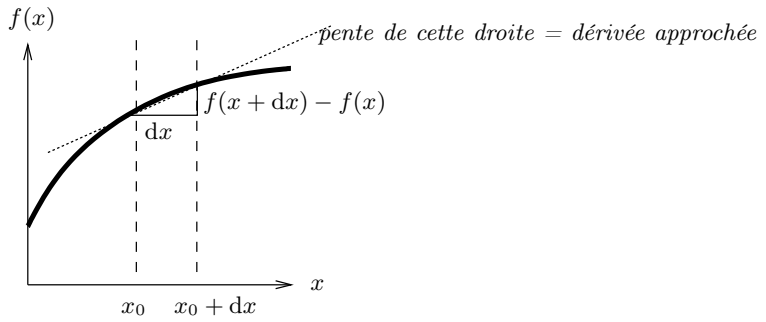
Exercice 11 : Écrivez une fonction `expande` qui prend en argument une liste d'états `l` et le motif `m`, et fournit la liste de tous les arcs qui partent des états contenus dans la liste `l`.

Exercice 12 : Écrivez une fonction `construit` qui prend en argument un motif `m` et construit l'automate correspondant. Testez-la avec le motif de l'exercice 5.

TD d'Informatique No. 5

1 Dérivées et intégrales.

Étant donnée une fonction $f : \mathbb{R} \to \mathbb{R}$ (c'est-à-dire de type `float→float` pour le langage ML), la dérivée de f en une valeur x_0, si elle existe, est la valeur vers laquelle tend $\frac{f(x_0+\mathrm{d}x)-f(x_0)}{\mathrm{d}x}$ lorsque $\mathrm{d}x$ tend vers 0. La phrase précédente signifie que l'on peut calculer la pente *locale* de la fonction f de manière de plus en plus précise en prenant une valeur de $\mathrm{d}x$ de plus en plus petite, comme le montre la figure ci-dessous.



Par exemple, on peut choisir $\mathrm{d}x = 0,01$ mais une approximation plus précise sera probablement obtenue avec $\mathrm{d}x = 0,001$, ou encore mieux $0,0001$, etc.

La dérivée d'une fonction f est elle même une fonction : celle qui associe à chaque valeur x_0 la dérivée de f en x_0. On la note souvent $f' : \mathbb{R} \to \mathbb{R}$.

Exercice 1 : Écrivez selon le principe décrit au-dessus la fonction d'ordre supérieur `derivee` qui prend en entrée une valeur d'approximation $\mathrm{d}x$ (de type `float`) et une fonction f (de type `float→float`, supposée dérivable), et retourne la fonction f' correspondante.
Calculez ensuite son type en faisant l'arbre de l'expression qui la définit.

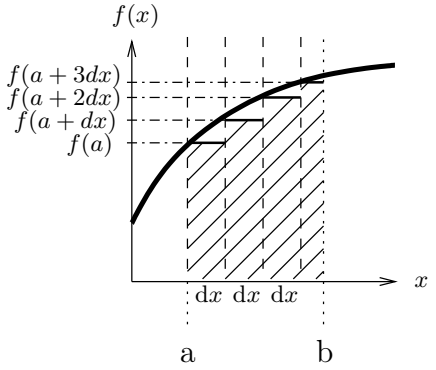
Exercice 2 : Écrivez une fonction `derivee_au_millieme` qui prend en entrée une fonction de type `float→float` et retourne sa dérivée approximée par un $\mathrm{d}x$ égal à 10^{-3}. On utilisera la question précédente. Calculez ensuite le type de cette fonction.

Exercice 3 : Testez votre fonction sur un polynome de degré 2. On constatera que l'approximation au millième en $\mathrm{d}x$ induit une petite erreur qui dépasse le millième : $\mathrm{d}x$ n'est pas l'erreur admise mais la largeur utilisée pour mesurer la pente.

Étant donnée une fonction $f : \mathbb{R} \to \mathbb{R}$ (c'est-à-dire de type `float→float` pour le langage ML), l'intégrale de f d'une valeur a à une valeur b, si elle existe, est la surface contenue sous la courbe de f au-dessus de l'intervalle $[a, b]$. On peut calculer cette intégrale, si elle existe, comme la surface vers laquelle tend la somme

$$\mathrm{d}x \times f(a) + \mathrm{d}x \times f(a + \mathrm{d}x) + \mathrm{d}x \times f(a + 2\mathrm{d}x) + \cdots + \mathrm{d}x \times f(a + (n-1)\mathrm{d}x) + (b - (a + n\mathrm{d}x)) \times f(a + n\mathrm{d}x)$$

(où $a + n\mathrm{d}x < b$ et $a + (n+1)\mathrm{d}x > b$) lorsque $\mathrm{d}x$ tend vers 0. La phrase précédente signifie que l'on peut calculer la surface sous la courbe de la fonction f de manière de plus en plus précise en prenant une valeur de $\mathrm{d}x$ de plus en plus petite, comme le montre la figure ci-dessous.



L'intégrale de la fonction f est elle même une fonction qui prend en arguments les deux réels a et b et retourne la surface qui est elle même un réel.

Exercice 4 : Écrivez selon le principe décrit au-dessus la fonction d'ordre supérieur `integrale` qui prend en entrée une valeur d'approximation dx (de type `float`) et une fonction f (de type `float→float`, supposée intégrable), et retourne la fonction intégrale de f correspondante. Cette intégrale prend elle même en entrée a et b et devra donc être soit de type `float*float→float`, soit de type `float→float→float` : programmez ces deux possibilités. Calculez ensuite le type de la fonction `integrale` en traçant l'arbre de l'expression qui la définit.

Exercice 5 : Écrivez une fonction `integrale_au_millieme` qui prend en entrée une fonction de type `float→float` et retourne son intégrale approximée par un dx égal à 10^{-3}. On utilisera la question précédente. Calculez ensuite le type de cette fonction.

Exercice 6 : Testez votre fonction sur le polynome $p(x) = x$ entre 0 et 1, puis entre 1 et 2, puis entre 2 et 3. Remarquer les approximations obtenues.

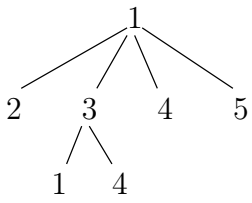
2 Les arbres généraux (dits planaires)

On considère le type suivant :

```
type 'a arbre = noeud of 'a * (('a arbre) list) ;;
```

Exercice 7 : Quel type de données peut-il intuitivement représenter ?

Exercice 8 : Déclarez `a0` de type `int arbre` représentant l'arbre suivant :



Dans toute la suite, on utilisera `a0` pour tester les fonctions écrites.

Exercice 9 : Écrivez une fonction `sup` qui prend en entrée un arbre planaire et fournit en sortie le plus grand élément de cet arbre. *Rappel :* la fonction `max` : $\alpha \to \alpha \to \alpha$ retourne le plus grand de ses deux arguments.

Exercice 10 : Écrivez une fonction `hauteur` qui calcule la hauteur d'un arbre planaire.

Exercice 11 : Écrivez une fonction `taille` qui calcule le nombre de nœuds d'un arbre planaire.

Exercice 12 : Ecrivez une fonction `nbfeuilles` qui calcule le nombre de feuilles d'un arbre planaire.

Exercice 13 : Écrivez avec un itérateur une fonction calculant la `somme` des éléments d'un arbre planaire de réels.

Exercice 14 : Écrivez une fonction booléenne `present` qui dit si un élément `e` est dans un arbre planaire `a`.

Exercice 15 : Écrivez une fonction `enumere` qui prend en entrée un arbre planaire `a` et retourne la liste de ses éléments (avec la racine en dernier).

Exercice 16 : QUESTION SUBSIDIAIRE : écrivez une fonction `range` qui range un arbre planaire dans un arbre binaire de recherche.