Numéro d'anonymat :

Examen - Janvier 2010
LI101
Durée 2 heures

Aucun document ni machine électronique n'est permis à l'exception de la carte de référence de Scheme.

Le sujet comporte 15 pages. Ne pas désagrafer les feuilles.

Répondre sur la feuille même, dans les boîtes appropriées. La taille des boîtes suggère le nombre de lignes de la réponse attendue. Le barème apparaissant dans chaque boîte n'est donné qu'à titre indicatif. Le barème total est lui aussi donné à titre indicatif : 66 points.

La clarté des réponses et la présentation des programmes seront appréciées. Les questions peuvent être résolues de façon indépendante. Il est possible, voire utile, pour répondre à une question, d'utiliser les fonctions qui sont l'objet des questions précédentes.

Pour vous faire gagner du temps, il ne vous est pas systématiquement demandé, en plus de la définition, la spécification entière d'une fonction. Bien lire ce qui est demandé : seulement la définition ? la signature et la définition ? la spécification et la définition ? seulement la spécification ?

Lorsque la signature d'une fonction vous est demandée, vous veillerez à bien préciser, avec la signature, les éventuelles hypothèses sur les valeurs des arguments.

Première partie

Exercices simples

Exercice 1

Question 1.1

Donner *la signature et une définition* de la fonction `somme-fractions` qui, étant donné un nombre entier naturel `n` rend la valeur de la somme

$$\sum_{i=0}^{n} \frac{i}{i+1}$$

.

```
(somme-fractions 0) → 0.0
(somme-fractions 1) → 0.5
(somme-fractions 2) → 1.1666666666666665
(somme-fractions 7) → 5.2821428571428575
```

Numéro d'anonymat :

[2/66]

Question 1.2

Donner *la signature et une définition* de la fonction `somme-divisibles` qui, étant donné trois nombres entiers naturels `n`, `m`, et `p` rend la somme des entiers compris entre `n` et `m` inclus qui sont divisibles par `p`. Par hypothèse, on considèrera que `p` est non nul, et que `n`≤`m`.

```
(somme-divisibles 1 1 2) → 0
(somme-divisibles 2 2 2) → 2
(somme-divisibles 1 7 2) → 12
(somme-divisibles 1 10 7) → 7
```

[4/66]

Question 1.3

Donner *la signature et une définition* de la fonction `fixe-taille` qui, étant donné un entier naturel `n`, ainsi qu'une liste d'entiers `L` rend la liste contenant `n` éléments, obtenue soit en prenant les `n` premiers éléments de `L` si elle contient plus de `n` éléments, soit en complétant la liste des éléments de `L` par la valeur 0 autant de fois que nécessaire.

```
(fixe-taille 0 '(1 2 3 4 5)) → ()
(fixe-taille 4 '(1 2 3 4 5)) → (1 2 3 4)
(fixe-taille 4 '(1 2)) → (1 2 0 0)
```

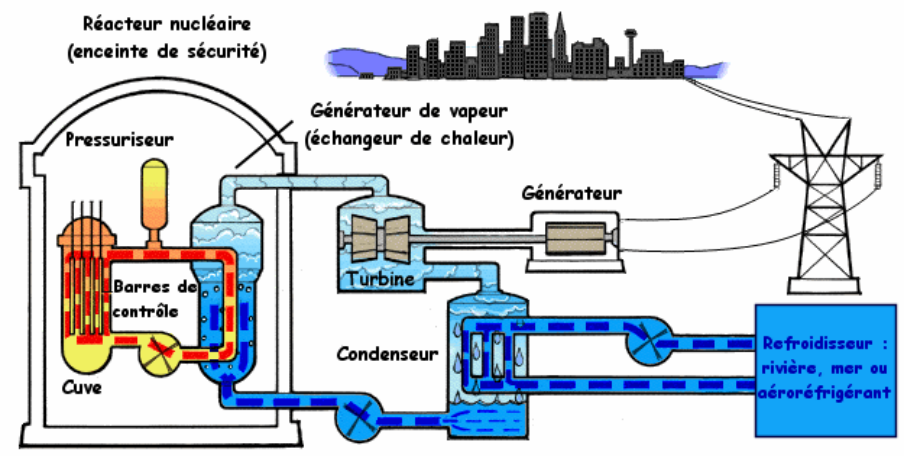
Numéro d'anonymat :

[4/66]

Deuxième partie

Exercices sur les listes

Dans cette série d'exercices, nous concevons une petite application de surveillance pour une centrale nucléaire. La figure ci-dessous décrit les composantes principales d'une centrale nucléaire (source Wikipedia) :



Nous n'allons pas décrire le fonctionnement d'une telle centrale nucléaire mais plutôt son système informatisé de surveillance.

Exercice 2

Evénements et flux

Le système de surveillance est basé sur la notion d'événement et de flux d'événements. Des sondes sont disposées à des endroits stratégiques de la centrale, et émettent des informations lorsqu'un événement anormal se produit. Un événement est une petite structure de données indiquant :

1. la source de l'événement (réacteur, turbine, condenseur, etc.) sous la forme d'une chaîne de caractères

Numéro d'anonymat :

2. un entier naturel indiquant l'instant de l'événement (la centrale démarre à l'instant t=0)
3. les informations décrivant l'événement sous la forme d'une chaîne de caractères

En Scheme, on choisit de représenter un tel événement par un n-uplet : `NUPLET[String Nat String]`.

Pour gagner en concision, on parlera dans la suite du sujet du type `Evenement` pour de tels n-uplets.

Question 2.1

Donner *la signature et une définition* de trois fonctions accesseurs `evt-source`, `evt-instant` et `evt-infos`, qui permettent d'accéder aux données d'un événement `e`.

Exemples :

`(evt-source (list "pressuriseur" 12 "haute temperature"))` → `"pressuriseur"`

`(evt-instant (list "pressuriseur" 12 "haute temperature"))` → `12`

`(evt-infos (list "pressuriseur" 12 "haute temperature"))` → `"haute temperature"`

[2/66]

Dans les questions qui suivent, vous devrez utiliser les accesseurs si vous avez besoin d'accéder aux données d'un événement.

Question 2.2

Les événements de surveillance émis par les sondes de la centrale sont collectés dans des flux de surveillance dont le type est : `Flux::=LISTE[Evenement]` Dans cette question, on s'intéresse à la détection d'une anomalie au sein d'un flux d'événements. Pour cela, on souhaite définir un semi-prédicat `detection` qui, à partir d'un flux d'événements, recherche la première occurrence d'un événement portant une information (chaîne de caractères) passée en argument. La spécification de cette fonction est la suivante :

```
;;; detection: string * LISTE[Evenement] -> Evenement + #f
;;; (detection info L) retourne le premier événement
;;; associé à info dans la liste L, ou retourne #f
;;; s'il n'y pas d'événement correspondant dans L.
```

Numéro d'anonymat :

Par exemple :

```
(detection "alerte" (list (list "pressuriseur" 12 "haute temperature")
                          (list "reacteur" 24 "surchauffe")
                          (list "turbine" 41 "alerte")))

  →("turbine" 41 "alerte")

(detection "alerte" (list (list "pressuriseur" 12 "haute temperature")
                          (list "reacteur" 24 "alerte")
                          (list "turbine" 41 "alerte")))

  →("reacteur" 24 "alerte")

(detection "alerte" (list (list "pressuriseur" 12 "haute temperature")
                          (list "reacteur" 24 "surchauffe")))

  →#f
(detection "alerte" (list)) → #f
```

Donner *une définition* de la fonction `detection`

Rappel : on utilise le prédicat `equal?` pour tester si des chaînes de caractères sont identiques

[3/66]

Question 2.3

On souhaite, à l'aide d'une fonction `filtrer-source`, filtrer les événements provenant d'une certaine source dans un flux d'événements donné. Par exemple :

```
(filtrer-source "reacteur" (list (list "pressuriseur" 12 "haute température")
     (list "reacteur" 24 "surchauffe")
     (list "turbine" 41 "alerte")
     (list "reacteur" 57 "alerte")
     (list "refroidisseur" 61 "avarie")
     (list "reacteur" 79 "explosion")))
```

Numéro d'anonymat :

→`(("reacteur" 24 "surchauffe") ("reacteur" 57 "alerte") ("reacteur" 79 "explosion"))`

Donner *la signature et une définition* de la fonction `filtrer-source` sans utiliser de fonctionnelle.

[4/66]

Question 2.4

Proposer une autre définition de `filtrer-source` en utilisant la fonctionnelle `filter` .

[3/66]

Exercice 3

Gestion des flux

Dans cet exercice, nous reprenons le système de surveillance de centrale nucléaire en ajoutant des fonctionnalités permettant de manipuler les flux d'événements.

Question 3.1

Tout d'abord il est important de vérifier que les flux respectent l'ordre entre les instants où les événements sont détectés. Pour cela, nous souhaitons définir un prédicat `flux-ordonne?` qui vérifie que les instants des événements sont bien observés dans l'ordre croissant à l'intérieur d'un flux. Par exemple :

```
(flux-ordonne? (list (list "pressuriseur" 12 "haute temperature")
```

Numéro d'anonymat :

```
                    (list "reacteur" 24 "alerte") (list "turbine" 41 "alerte")))

  →#t

(flux-ordonne? (list (list "pressuriseur" 12 "haute temperature")
                     (list "reacteur" 24 "alerte")
                     (list "générateur" 24 "baisse production")
                     (list "turbine" 41 "alerte")))

  →#t

(flux-ordonne? (list (list "pressuriseur" 12 "haute temperature")
                     (list "reacteur" 44 "alerte")
                     (list "turbine" 41 "alerte")))

  →#f
(flux-ordonne? (list (list "pressuriseur" 12 "haute temperature"))) → #t

(flux-ordonne? (list)) → #t
```

Proposer *la signature et une définition* du prédicat `flux-ordonne?`

[4/66]

Question 3.2

Pour des raisons de sécurité, des flux indépendants d'événements ont été mis en place dans la centrale. Au premier niveau de surveillance, les flux sont analysés de façon indépendante. Au deuxième niveau, le système effectue une analyse de la fusion des flux. Lorsque les flux sont fusionnés, il faut faire attention à ce que les événements qu'ils contiennent soient correctement ordonnés dans le flux résultant. La fonction `fusion-flux` effectue la fusion de deux flux d'événements ordonnés. Par exemple :

```
(fusion-flux (list (list "pressuriseur" 12 "haute temperature")
                     (list "reacteur" 24 "alerte") (list "turbine" 41 "alerte"))
               (list (list "condensateur" 17 "manque eau")
                     (list "générateur" 33 "baisse voltage")))
```

Numéro d'anonymat :

→

```
(("pressuriseur" 12 "haute temperature") ("condensateur" 17 "manque eau")
 ("reacteur" 24 "alerte") ("générateur" 33 "baisse voltage")
 ("turbine" 41 "alerte"))
```

Donner *la signature et une définition* de la fonction `fusion-flux` qui effectue la fusion de deux flux en respectant l'ordre entre les événéments.

Remarque : on n'oubliera pas de préciser l'hypothèse que les deux flux à fusionner sont correctement ordonnés.

[5/66]

Question 3.3

On souhaite généraliser la fusion de n flux indépendants avec la fonction suivante :

```
;;; fusion-nflux: LISTE[LISTE[Evenement]] -> LISTE[Evenement]
;;; (fusion-nflux LL) retourne la fusion ordonnée de tous les flux de LL
;;; HYPOTHESE: tous les flux de LL sont correctement ordonnés
```

Donner *une définition* de cette fonction.

Remarque : il est conseillé d'utiliser la fonction `fusion-flux` dans cette définition.

Numéro d'anonymat :

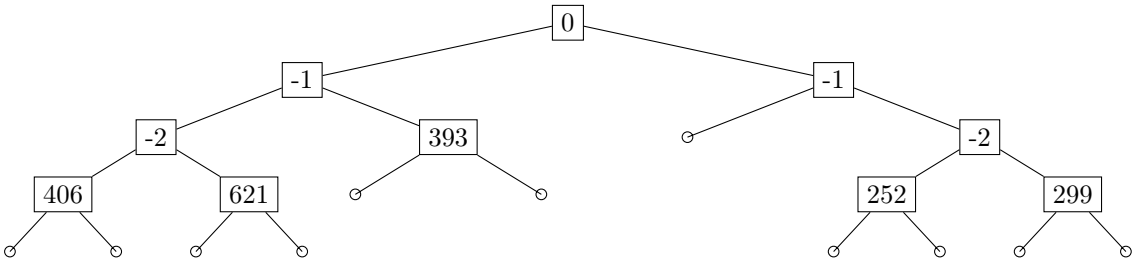
[5/66]

Troisième partie

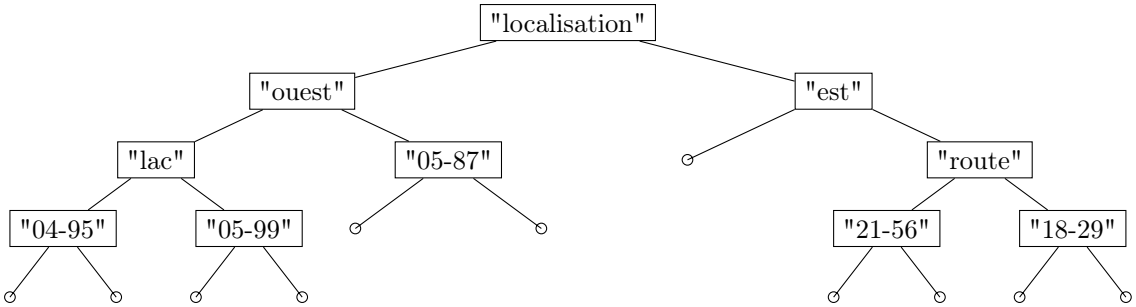
Exercices sur les arbres binaires

Exercice 4

Soit l'arbre binaire `(ab-lor)` dont les étiquettes sont des nombres entiers :



Et soit l'arbre binaire `(ab-mat)` dont les étiquettes sont des chaînes de caractères :



Dans ces deux arbres, tout au long de cet exercice, nous allons nous intéresser aux étiquettes qui se trouvent **dans les feuilles** de ces arbres, et seulement celles-là.

Question 4.1

On sait que l'expression Scheme `(ab-etiquette (ab-mat))` rend la chaîne `"localisation"`. Donner l'expression Scheme permettant d'obtenir la chaîne `"05-87"` à partir de l'arbre `(ab-mat)`.

Numéro d'anonymat :

[1/66]

Question 4.2

Donner *la signature et une définition* du prédicat `ab-feuille?` qui rend vrai si un arbre binaire est une feuille. (note : vous pourrez utiliser ce prédicat dans les réponses aux questions suivantes).

```
(ab-feuille? (ab-vide)) → #f
(ab-feuille? (ab-noeud 1 (ab-vide)(ab-vide))) → #t
(ab-feuille? (ab-lor)) → #f
```

[2/66]

Question 4.3

Donner *la signature et une définition* de la fonction `ab-compte` qui, étant donné un arbre binaire de type `ArbreBinaire[int]` rend la somme de toutes les valeurs qui se trouvent **dans les feuilles** de cet arbre ou la valeur 0 si l'arbre est vide.

```
(ab-compte (ab-vide)) → 0
(ab-compte (ab-noeud 375 (ab-vide) (ab-vide))) → 375
(ab-compte (ab-noeud -1 (ab-noeud 375 (ab-vide) (ab-vide)) (ab-vide))) → 375
(ab-compte (ab-lor)) → 1971
```

[4/66]

Numéro d'anonymat :

Question 4.4

Deux arbres binaires sont dits *jumeaux* s'ils possèdent la même structure, sans posséder forcément les mêmes étiquettes. Ainsi, les arbres donnés en exemple `(ab-lor)` et `(ab-mat)` sont des arbres jumeaux. On considèrera aussi que deux arbres binaires vides sont des arbres jumeaux.

Si `j1` et `j2` sont deux arbres jumeaux, toute étiquette d'une feuille de `j1` a une étiquette qui lui correspond dans l'arbre `j2` et qui est positionnée au même endroit de l'arbre, dans la feuille correspondante. Ainsi, par exemple, l'étiquette `"04-95"` de l'arbre `(ab-mat)` correspond à l'étiquette `406` de l'arbre `(ab-lor)`, etc.

Un des intérêts d'avoir des arbres jumeaux est qu'ils peuvent être parcourus simultanément dans une fonction.

Donner *la signature et une définition* de la fonction `abj-trouve` qui, étant donné deux arbres jumeaux `j1` et `j2`, ainsi qu'une valeur `e`, rend l'étiquette de `j2` qui correspond **à une feuille** de `j1` dont l'étiquette contient la valeur `e`. La fonction rendra `#f` s'il n'y a aucune feuille dans `j1` contenant la valeur `e`. Par hypothèse, on considèrera que les étiquettes de `j1` sont toutes différentes et, donc, que l'étiquette `e` ne peut y apparaître qu'une seule fois au plus.

```
(abj-trouve "42-83" (ab-noeud "42-83" (ab-vide) (ab-vide))
                    (ab-noeud 375 (ab-vide) (ab-vide))) →375
(abj-trouve "04-95"  (ab-mat) (ab-lor)) → 406
(abj-trouve "21-56"  (ab-mat) (ab-lor)) → 252
(abj-trouve "ankh-morpork"  (ab-mat) (ab-lor)) → #f
(abj-trouve "route"  (ab-mat) (ab-lor)) → #f
```

[5/66]

Question 4.5

À partir des étiquettes des **feuilles** de deux arbres jumeaux `j1` et `j2`, on peut construire des associations : la clé de l'association est une étiquette de feuille de `j1`, et la valeur de l'association est l'étiquette de la feuille correspondante dans `j2`.

Donner *la signature et une définition* de la fonction `abj->liste` qui, étant donné deux arbres jumeaux `j1` et `j2`, rend la liste des associations obtenues à partir des feuilles de `j1` et de `j2`.

Numéro d'anonymat :

```
(abj->liste  (ab-vide) (ab-vide)) → ()
(abj->liste (ab-noeud "42-83" (ab-vide) (ab-vide))
             (ab-noeud 375 (ab-vide) (ab-vide))) →(("42-83" 375))
(abj->liste  (ab-mat) (ab-lor)) →
        (("04-95" 406) ("05-99" 621) ("05-87" 393) ("21-56" 252) ("18-29" 299))
```

[5/66]

Quatrième partie

Exercices sur les arbres généraux

Exercice 5

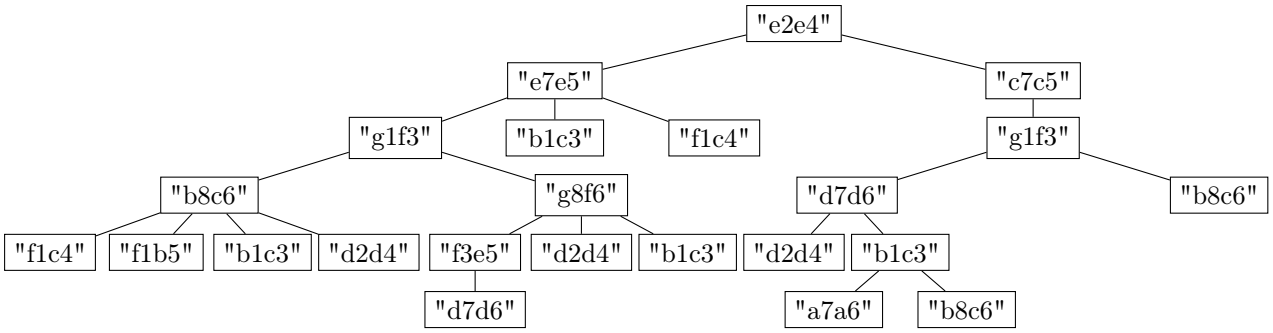
Lors de la conception d'un programme joueur d'échecs, il est commun de créer ce que l'on appelle une bibliothèque d'ouvertures et de la mémoriser sous la forme d'un arbre général.

On rappelle qu'une ouverture aux échecs est une séquence de coups. Chaque coup est noté par une chaîne de caractères contenant des informations (case de départ et case d'arrivée de la pièce jouée) : par exemple `"e2e4"`, `"e7e5"`, `"g1f3"` sont trois coups possibles. Dans cet exercice, il n'est pas demandé de connaître les règles du jeu d'échecs.

Un arbre d'ouvertures est un arbre général, du type `ArbreGeneral[string]`. Chaque nœud d'un tel arbre contient un coup dans son étiquette, et sa forêt contient les nœuds des réponses possibles.

En exemple, vous avez l'arbre `(ag-kas)` qui représente un exemple de bibliothèque d'ouvertures.

Numéro d'anonymat :



Question 5.1

Une bibliothèque d'ouvertures est caractérisée par sa taille en nombre de variantes. Une variante d'ouverture est un chemin de la racine vers une feuille dans l'arbre d'ouvertures qui la représente. Ainsi, le chemin passant par `"e2e4", "e7e5", "g1f3", "b8c6"`, et `"f1c4"` est une des variantes d'ouverture contenues dans l'arbre `(ag-kas)`.

Il est facile de voir qu'il y a donc autant de variantes dans une bibliothèque d'ouvertures que de feuilles dans l'arbre général qui la représente.

Donner *la signature et une définition* de la fonction `ag-nombre-variantes` qui, étant donné un arbre d'ouvertures rend son nombre de feuilles.

`(ag-nombre-variantes (ag-kas))` → `13`

[4/66]

Question 5.2

On peut remarquer que la hauteur d'un arbre d'ouverture correspond à la taille, en nombre de coups, de la plus grande variante d'ouvertures qu'il contient.

Donner *la signature et une définition* de la fonction `ag-grande-variante` qui, étant donné un arbre d'ouvertures rend la taille en nombre de coups de la plus grande variante qu'il contient.

`(ag-grande-variante (ag-kas))` → `6`

Numéro d'anonymat :

[4/66]

Question 5.3

Une suite de coups quelconque se mémorise dans une `Liste[string]`. Ainsi, les listes ("e2e4" "e7e5" "g1f3" "b8c6" "f1c4"), ("e2e4" "c7c5" "g1f3") ou ("e2e4" "e7e5" "b1c3" "f8c5") sont des listes de coups possibles.

Une liste de coups est dite "théorique" si elle représente les coups contenus dans une variante de la bibliothèque d'ouvertures. Ainsi, selon l'arbre `(ag-kas)`, les deux premières listes données en exemple sont théoriques, mais la dernière liste ne l'est pas. Une liste de coups théorique est donc une liste de coups qui représente un chemin correct, éventuellement incomplet, depuis la racine, dans l'arbre d'ouvertures.

Plus généralement, on dira qu'une liste de coups `L` est théorique selon un arbre d'ouvertures `AG` si
- la liste `L` est vide
- la liste `L` n'est pas vide et dans ce cas, les deux conditions suivantes sont vérifiées :
  - le premier de coup de `L` doit être égal à l'étiquette de `AG`
  - le reste des coups de `L` doit être théorique pour l'un des arbres de la forêt de `AG`. Si la forêt de `AG` ne contient aucun arbre, alors `L` n'est théorique que si elle ne contient qu'un seul coup (celui qui se trouve dans l'étiquette de `AG`).

Donner *la signature et une définition* de la fonction `theorique?` qui, étant donné une liste de coups `L` et un arbre général `AG` rend vrai si la liste de coups est théorique selon l'arbre `AG`.

```
(theorique? '() (ag-kas)) → #t
(theorique? '("e2e4" "e7e5" "g1f3" "b8c6" "f1c4") (ag-kas)) → #t
(theorique? '("e2e4" "c7c5" "g1f3") (ag-kas)) → #t
(theorique? '("e2e4" "e7e5" "b1c3" "f8c5") (ag-kas)) → #f
```

Numéro d'anonymat :

[5/66]