Programmation fonctionnelle et logique (INF6120)

Automne 2024 — Examen 2

3 heures

Samuele Giraudo

Université du Québec à Montréal

10 décembre 2024

Informations personnelles

Prénom : .....

Nom : .....

Code permanent : .....

Groupe : ☐ 20 (S. Giraudo)

Barème

Donné ici et dans les questions à titre indicatif uniquement.

1 : ___ /20 2 : ___ /10 3 : ___ /20 4 : ___ /30 5 : ___ /20

Instructions générales

1. Seules les notes manuscrites personnelles d'au plus huit pages au format A4 ou équivalent sont autorisées. Nom et prénoms doivent être écrits sur chacune de ces feuilles.
2. Tout appareil électronique est interdit (téléphones, montres, ordinateurs, calculatrices, *etc.*).
3. Il est interdit de dégrafer le sujet : aucun sujet avec des pages manquantes, détachées ou déchirées ne sera évalué.
4. Il est interdit de laisser des débris de gomme (ou analogue) dans le rendu.
5. Aucune sortie n'est permise durant l'examen.
6. La carte d'étudiant.e devra être présentée sur demande.
7. Les sacs et autres effets personnels non nécessaires à l'examen doivent être déposés sur le devant de la salle.
8. Les règles concernant le plagiat sont strictement appliquées.
9. Les questions sont majoritairement indépendantes et de difficulté non progressive. Le barème peut être généreux pour certaines questions dont le nombre de points est bien supérieur à leur difficulté.
10. Les espaces alloués pour les réponses ne constituent pas une indication sur la longueur de la réponse attendue.
11. Bien vérifier au début de l'épreuve que toutes les pages du sujet sont présentes (le nombre total de pages est spécifié en bas de cette page).
12. Sauf mention contraire, l'utilisation des fonctions, types et prédicats des modules et bibliothèques `Stdlib`, `List` et `Fun` d'OCaml et `clpfd` de Prolog sont autorisées. Tout matériel provenant d'un autre module ou bibliothèque est interdit.

Ce document est composé de 8 pages.

1 Listes en OCaml

Question 1.1. *(2 points)* — Soit `lst` une liste d'entiers non vide. Donner une **expression** dont la valeur est le plus grand élément de `lst`. Cette expression doit utiliser des fonctions d'ordre supérieur sur les listes.

.....

.....

Question 1.2. *(2 points)* — Soit `lst` une liste de chaînes de caractères. Donner une **expression** dont la valeur est la chaîne de caractères obtenue en concaténant de gauche à droite toutes les chaînes de caractères de `lst`. Cette expression doit utiliser des fonctions d'ordre supérieur sur les listes.

.....

.....

Question 1.3. *(3 points)* — Soit `lst` une liste d'entiers. Donner une **expression** dont la valeur est la somme des valeurs positives de `lst`. Cette expression doit utiliser des fonctions d'ordre supérieur sur les listes.

.....

.....

Question 1.4. *(4 points)* — Soit `lst` une liste de couples d'entiers. Donner une **expression** dont la valeur est la somme des 1$^{\text{res}}$ coordonnées des couples de la liste. Cette expression doit utiliser des fonctions d'ordre supérieur sur les listes.

.....

.....

Question 1.5. *(4 points)* — Soit `lst` une liste de listes de booléens. Donner une **expression** dont la valeur est la liste des listes de booléens obtenues par image miroir de celles de `lst`. Cette expression doit utiliser des fonctions d'ordre supérieur sur les listes.

.....

.....

Question 1.6. *(5 points)* — Soit `lst` une liste de listes d'entiers. Donner une **expression** s'évaluant en `true` si et seulement si tous les éléments de `lst` sont des listes non vides. Cette expression doit utiliser des fonctions d'ordre supérieur sur les listes.

.....

.....

2 Induction structurelle

Question 2.1. *(10 points)* — Soit le type

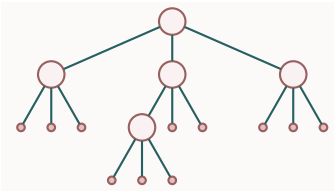
```
type arbres_ternaires =
    |Feuille
    |Noeud of arbres_ternaires * arbres_ternaires * arbres_ternaires
```

Si t est une valeur de ce type (c'est-à-dire que t est un arbre ternaire), notons par $\mathrm{f}(t)$ le nombre de feuilles de t et par $\mathrm{n}(t)$ le nombre de nœuds de t.

Par exemple, l'arbre ternaire t dont la construction en OCaml et la représentation graphique (où les `Noeud` sont les gros cercles et les `Feuille`, les petits) sont données respectivement par

```
Noeud (
    Noeud (Feuille, Feuille, Feuille),
    Noeud (
        Noeud (Feuille, Feuille, Feuille),
        Feuille,
        Feuille
    ),
    Noeud (Feuille, Feuille, Feuille)
)
```

et



vérifie $\mathrm{f}(t) = 11$ et $\mathrm{n}(t) = 5$.

Démontrer, par induction structurelle sur les arbres ternaires, que pour tout arbre ternaire t,

$$\mathrm{f}(t) = 2\,\mathrm{n}(t) + 1.$$

3 Unification et résolution

Question 3.1. *(5 points)* — Donner le **résultat de l'unification** des deux termes `b(X, c)` et `b(c, Y)`. Le résultat doit être donné en spécifiant les valeurs que doivent prendre les variables `X` et `Y` pour que l'unification ait lieu. La réponse doit être **justifiée**. La justification peut s'appuyer sur les dessins des termes.

Question 3.2. *(7 points)* — Donner le **résultat de l'unification** des deux termes `a(X, b(Y, X), Y)` et `a(c(Y), Z, c(T))`. Le résultat doit être donné en spécifiant les valeurs que doivent prendre les variables `X`, `Y`, `Z` et `T` pour que l'unification ait lieu. La réponse doit être **justifiée**. La justification peut s'appuyer sur les dessins des termes.

Question 3.3. *(8 points)* — Donner les **solutions** et dessiner l'**arbre de résolution complet** de la requête `?- a(Y).` au sein du programme

```
a(X) :-
    b(X).

b(X) :-
    X = f.
b(X) :-
    X = g.
```

4 Prédicats

Question 4.1. *(6 points)* — Écrire un **prédicat** `somme_liste/2` qui est tel que `somme_liste(L, R)` si la somme des éléments de la liste `L` est l'entier `R`. Aucun autre prédicat que `somme_liste/2` ne doit être utilisé, sauf éventuellement des prédicats intermédiaires (qu'il faut bien entendu définir aussi). Il est en revanche autorisé d'utiliser des éléments fournis par la bibliothèque `clpfd`.

Par exemple,

```
?- somme_liste([], R).
R = 0.
```

```
?- somme_liste([2, 1, 2, 3], R).
R = 8.
```

Question 4.2. *(8 points)* — Écrire un **prédicat** `croissante/1` qui est tel que `croissante(L)` si la liste d'entiers `L` est triée dans l'ordre croissant. Aucun autre prédicat que `croissante/1` ne doit être utilisé, sauf éventuellement des prédicats intermédiaires (qu'il faut bien entendu définir aussi). Il est en revanche autorisé d'utiliser des éléments fournis par la bibliothèque `clpfd`.

Par exemple,

```
?- croissante([7]).
true.
```

```
?- croissante([1, 2, 5]).
true.
```

```
?- croissante([1, 2, 2, 3, 5, 5, 5]).
true.
```

```
?- croissante([1, 2, 2, 5, 3, 5, 5]).
false.
```

Question 4.3. *(10 points)* — Écrire un **prédicat** `binomial/3` qui est tel que `binomial(N, K, R)` si `N`, `K` et `R` sont trois entiers positifs vérifiant $C($`N`$,$ `K`$) =$ `R`. La fonction C est définie récursivement par

$$C(n,k) = \begin{cases} 0 & \text{si } k > n, \\ 1 & \text{si } k = 0 \text{ ou } n = k, \\ C(n-1,k-1) + C(n-1,k) & \text{si } 0 < k < n. \end{cases}$$

Aucun autre prédicat que `binomial/3` ne doit être utilisé, sauf éventuellement des prédicats intermédiaires (qu'il faut bien entendu définir aussi). Il est en revanche autorisé d'utiliser des éléments fournis par la bibliothèque `clpfd`.

Par exemple,

```
?- binomial(5, 10, R).
R = 0.
```

```
?- binomial(10, 10, R).
R = 1.
```

```
?- binomial(10, 5, R).
R = 252.
```

Question 4.4. *(6 points)* — En utilisant le prédicat `binomial/3` spécifié à la question précédente (que l'on pourra admettre comme défini même si la question n'a pas été correctement traitée), écrire une **requête** qui permet de calculer les `N` et `K` tels que `binomial(N, K, 15)` lorsque `N` et `K` sont des entiers compris entre 0 et 15.

Par exemple,

```
?- % Requête à écrire
N = 15, K = 1 ;
N = 6, K = 2 ;
N = 6, K = 4 ;
N = 15, K = 14.
```

5 Programmation par contraintes

Question 5.1. *(10 points)* — Écrire un **prédicat** `solution/4` qui est tel que `solution(A, B, C, D)` si `A`, `B`, `C` et `D` sont des entiers compris inclusivement entre 1 et 5 et `A` + `B` est égal à `C` + `D`. Il est demandé ici d'utiliser la bibliothèque `clpfd` et de proposer une approche par contraintes.

Question 5.2. *(10 points)* — Écrire un **prédicat** `injection/3` qui est tel que `injection(N, K, L)` si la liste `L` est de longueur `N` et contient des éléments tous différents compris entre 1 et `K`. Il est demandé ici d'utiliser la bibliothèque `clpfd` et de proposer une approche par contraintes.

Par exemple,

```
?- injection(2, 3, L).
L = [1, 2] ;
L = [1, 3] ;
L = [2, 1] ;
L = [2, 3] ;
L = [3, 1] ;
L = [3, 2].
```

```
?- injection(3, 3, L).
L = [1, 2, 3] ;
L = [1, 3, 2] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
L = [3, 1, 2] ;
L = [3, 2, 1].
```

```
?- injection(3, 2, L).
false.
```