Interopérabilité : TD et TME

LU3IN032 : Programmation comparée

Benjamin Canou, Basile Pesin

19 mars 2021

Table des matières

1 Travaux Dirigés

1.1 Exercice 1 : Manipulation de listes

1.1.1 Question 1 (Rappel)

Comment une liste est elle représentée dans la mémoire ?

1.1.2 Question 2

En vous référant au cours sur l'interfaçage avec C, écrivez la fonction C `my_list_length`, équivalente à la fonction `List.length` (de type OCaml `'a list -> int`).

1.1.3 Question 3

Ecrivez maintenant l'équivalent de la fonction `List.map` (de type OCaml `('a -> 'b) -> 'a list -> 'b list`).

1.1.4 Question 4

Donnez le code OCaml permettant de manipuler ces fonctions C, un exemple d'utilisation ainsi que la commande de compilation.

1.2 Exercice 2 : Entiers bornés

On veut implémenter en C une bibliothèque d'entiers dont la valeur est surveillée à l'exécution pour rester entre deux bornes de données. L'interface OCaml qu'on implémentera dans cette exercice est la suivante :

```
type ranged

external ranged_new : int -> int -> int -> ranged = "ml_ranged_new"
external ranged_add : ranged -> ranged -> ranged = "ml_ranged_add"
external int_of_ranged : ranged -> int = "ml_int_of_ranged"
```

Avec un exemple d'utilisation :

```
let () =
  let a = ranged_new 3 10 4 in
  let b = ranged_new 3 10 3 in
  let c = ranged_add a b in
  print_int (int_of_ranged c); (* 7 *)
  ranged_add c c
```

1.2.1 Question 1

On va utiliser les blocs *custom* pour implanter les entiers bornés. Choisissez une représentation mémoire pour le contenu de ces blocs.

1.2.2 Question 2

Rappelez le rôle des structures `custom_ops`, et donnez celle correspondant à la représentation choisie. On se limitera au minimum d'opérations prises en charge. Quelles sont-elles ?

1.2.3 Question 3

Donnez la fonction `ml_int_of_ranged`.

1.2.4 Question 4

Donnez la fonction `ml_ranged_new`. La fonction devra lancer `Invalid_arg "ranged_new"` si les arguments sont mal formés, et `Failure "out of range"` en cas de dépassement.

1.2.5 Question 5

Même question pour `ml_ranged_add`. On n'additionnera que des entiers dans les même bornes.

1.2.6 Question 6

Donnez la commande de compilation.

2 Travaux sur Machines Encadrés

2.1 Exercice 3 : Afficheur générique

On cherche à implémenter une fonction d'affichage générique avec la signature suivante :

```
external generic_print : 'a -> unit
```

Vous pouvez vous aider de la documentation officielle : [1].

2.1.1 Question 1

Dans un premier temps, donnez une fonction prenant en charge les valeurs entières et les blocs simples. Pour un entier, la fonction affichera simplement sa valeur. Pour un bloc simple, la fonction affichera son étiquette et ses éléments sous le format suivant : `[tag | elt0, elt1, ... ]`

Pour tous les cas non gérés, votre fonction devra imprimer `<unsupported>`.

2.1.2 Question 2

Améliorez la fonction pour prendre en charge les chaînes, les flottants et les tableaux de flottants.

2.1.3 Question 3

Améliorez la fonction pour prendre en charge les fermetures (vous pourrez afficher simplement `<fun>`, ou bien afficher en plus l'environnement de la fermeture), les valeurs abstraites et custom.

2.1.4 Question 4

Définissez divers types (sommes, enregistrements, types de la bibliothèque standard, etc), et observez le résultat de l'impression de valeurs de ces types afin de vérifier que vous avez bien compris la représentation des données.

2.1.5 Question 5

Transformez votre fonction pour qu'elle renvoie une chaîne plutôt que d'afficher à l'écran.

2.2 Exercice 4 : Adaptation d'un algorithme en Cython

La distance de Leveinshtein est une distance entre deux chaînes de caractères : elle indique le nombre minimum d'opérations à effectuer pour transformer l'une des chaînes en l'autre. Les opérations possibles sont :

— Supprimer un caractère
— Ajouter un caractère
— Substituer un caractère à un autre

Dans le cas général, on peut attribuer des couts différents à ces opérations ; ici on considérera que chaque opération à un coût de 1. La distance peut être calculée efficacement via un algorithme de programmation dynamique : pour deux chaines `s` et `t` de longeurs respectives `m` et `n`, on calcule une matrice `M` où `M[i][j]` indique la distance entre les préfixes `s[0:i]` et `t[0:j]`. La dernière case `M[m][n]` contient donc la distance entre les deux chaînes. Une version plus efficace (en espace) de l'algorithme ne conserve en mémoire que les deux dernières lignes calculées de la matrice. On donne ci-dessous (et dans le fichier `levenshtein_py.py` une implémentation en python de cet algorithme.

```
def levenshtein(s, t):
    m = len(s)
    n = len(t)

    v0 = [0 for _ in range(n+1)]
    v1 = [0 for _ in range(n+1)]

    # initialisation de v0. v0[i] correspond a M[0][i]
    # la distance d'edition entre s vide et t
    # c'est simplement le nombre de caracteres a effacer dans t
    for i in range(n+1):
        v0[i] = i
```

```
    for i in range(m):
        # calculer v1 a partir de v0

        # le premier element de v1 est M[i+1][0]
        # i+1 caracteres a effacer dans s pour matcher une chaine vide t
        v1[0] = i + 1

        # pour le reste, one utilise la formule
        for j in range(n):
            deletionCost = v0[j + 1] + 1 # Supprimer un caractere
            insertionCost = v1[j] + 1 # Ajouter un caractere
            substitutionCost = v0[j]
            if s[i] != t[j]:
                substitutionCost = substitutionCost + 1 # Si on a besoin de substituer

            v1[j + 1] = min(deletionCost, insertionCost, substitutionCost)

        # on remplace v0 par v1 pour l'iteration suivante
        v0, v1 = v1, v0
    print("Distance : " + str(v0[n]))
    return v0[n]
```

Listing 1 – Distance de Levenshtein, en Python

On fourni également dans `levenshtein_c.c` une version C du même programme.

2.2.1 Question 1 : Tests de performances

En utilisant les fichiers `lorem.txt` et `lorem_noisy.txt`, testez les deux programmes, et comparez leur vitesse d'exécution. Vous pouvez ajouter d'autres fichiers textes de différentes tailles pour faire plus de tests. Etant donné un fichier, vous pouvez générer une version du fichier avec du "bruit" en utilisant le script `makesomenoise.py` fourni.

2.2.2 Question 2 : Implémentation Cython

Adaptez l'implémentation Python pour tirer partie des types de Cython et améliorer les performances. Si écrivez votre programme dans le fichier `levenshtein.pyx`, vous pouvez compiler et exécuter avec la commande `make run_pyx`. Vous devriez pouvoir arriver à des performances proches de celles du programme C.

2.3 Exercice 5 : Bindings pour une bibliothèque en Js_of_ocaml

Notre objectif est de visualiser les structures d'expressions manipulées et décrites pendant le TME7. Pour cela, on souhaite utiliser la bibliothèque Javascript Treant.js, qui permet de visualiser des structures d'arbres. On va voir qu'on peut utiliser Js_of_ocaml pour interfacer le code OCaml de la séance précédante et cette bibliothèque.

2.3.1 Question 1 : L'objet `node`

Le fonctionnement de la bibliothèque est très simple : on fourni une structure (encodée dans le format JSON) représentant l'arbre, qu'on passe à un constructeur 'Treant' exposé par la bibliothèque. La structure est décrite en détail sur la page du projet, mais on va s'intéresser à une version très simplifiée. L'objet central est la `node`, qui contient un tableau d'enfants, ainsi qu'un champ `text`. Celui ci contient le contenu textuel de la `node` (qui sera affiché), en particulier un champ `name`.

On a déjà donné une représentation OCaml de l'objet `text` simplifié dans le fichier `treantml.ml`. Dans ce même fichier, ajoutez une représentation OCaml de l'objet `node`.

Indications :

— un array Javascript est représentable par le type `'a Js.js_array Js.t`.

— un champ d'objet OCaml correspondant à un champ Javascript doit être de type `'a Js.prop` ou `'a Js.readonly_prop` pour un champ immutable. Ici on ne considérera que des champs immutables.

2.3.2 Question 2 : Fonction de création de noeud

Donnez une fonction permettant de créer une `node`, de prototype `val mknode : string -> node Js.t list -> node Js.t` où le premier paramètre contient le "texte" du noeud, et le second la liste de ses enfants.

Indication :

— le constructeur `Js.array_empty` permet de construire un tableau vide (n'oubliez pas la notation `new%js`).

— la fonction `val Js.array_set : 'a Js.array -> int -> 'a -> unit` permet de placer un élément dans une case d'un tableau.

2.3.3 Question 3 : L'objet `chart_config`

Une autre structure centrale à définir est l'objet `chart_config`, qui contient :

— une `chart` : qui contient les paramètres globaux de l'arbre utilisés pour l'affichage, entre autres le `container` indicant l'identifiant de l'élément HTML dans lequel afficher l'arbre, ainsi que la classe CSS par défaut à utiliser pour les noeuds. Cette structure est déjà définie.

— une `nodeStructure`, qui contient la `node` racine de l'arbre.

2.3.4 Question 4 : Folder produisant des `node`

La semaine dernière, on a écrit un `Folder` générant une représentation textuelle d'une expression (`Show`). Dans le fichier `exprtree.ml`, implémenter un nouveau `Folder` qui renvoie une `node Js.t` contenant l'arbre représentant l'expression.

2.3.5 Question 5 : Création de la `chart_config`

En utilisant le Folder défini dans la fonction ci-dessus, écrire une fonction `val create_config : expr -> chart_config` qui crée une `chart_config` contenant l'arbre correspondant à l'expression passée en entrée.

2.3.6 Question 6 : Initialisation de l'arbre

Compléter la fonction `init`, qui est appelée dès que la page est chargée. Celle-ci doit

— générer une nouvelle expression : vous utiliserez le générateur d'expression aléatoires écrit à la fin du TME précédant.

— ajouter le texte de l'expression dans l'élément `"currentexpr"` (voir l'exemple de `appendChild` donné en cours).

— utiliser la fonction `treant`, et la fonction écrite dans la question précédante pour construire et afficher l'arbre.

2.3.7 Question 7 : Implémentation d'un event pour recharger une nouvelle expression

On veut aussi pouvoir générer de nouvelles expressions sans avoir à recharger la page.

Pour cela, compléter la fonction `on_generate`, qui comme la fonction précédante, doit générer une nouvelle expression, afficher son texte et l'arbre correspondant. Attention, on veut remplacer le texte et non pas l'ajouter. Vous pourrez utiliser la fonction `replace_child` donnée.

On veut que `on_generate` soit appelée quand l'utilisateur clique sur le bouton marqué "Nouvelle expression". Pour cela, vous changerez l'attribut `onclick` du bouton, dont l'id est `"generate"` (vous pourrez vous référer à l'exemple du cours).

Références

[1] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The OCaml system : Documentation and user's manual - interfacing c with ocaml. August 2020.