

Examen Final

IFT-2035

December 4, 2007

Directives

- Documentation autorisée: aucune.
- Répondre sur le questionnaire dans l'espace libre qui suit chaque question. Utiliser le verso de la page si nécessaire.
- Chaque question vaut 5 points pour un total maximum de 25 points.
- Les questions ne sont pas placées par ordre de difficulté.

0 Nom et prénom (1 point de bonus)

Écrire son nom et prénom et son code permanent en haut de chaque page.

1 TP2 - Gestion mémoire

Soit le code C suivant:

```
typedef struct list list;
struct list {
    int n;
    void *elem;
    list *next;
}

list *list_cons (void *elem, list *tail)
{ list *l = malloc (sizeof (list));
  l->n = 1;
  l->elem = elem;
  l->next = tail;
  return l;
}

void *list_head (list *l)
{ return l->elem; }

list *list_tail (list *l)
{ return l->next; }

list *list_copy (list *l)
{ l->n++; return l; }

list *list_free (list *l)
{ l->n--;
  if (l->n == 0) {
    list_free (l->next);
    free (l);
  } }
}
```

1. Quel style de gestion mémoire est-il utilisé par cette librairie? Justifier.
2. Décrire les conditions que ce code est censé préserver pour s'assurer que la gestion de la mémoire est correcte.
3. Il y a un bug dans le code ci-dessus, qui a à voir avec la gestion de la mémoire; lequel? Quel genre de problème peut-il engendrer? Corriger le bug.
4. Si on essaie de manipuler des listes de listes, on se heurte à une limitation du code ci-dessus. Laquelle? Comment peut-on la corriger?

2 Macros

Soit la macro `dotimes` en Scheme:

```
(define-macro dotimes (varlim body)
  (let ((var (car varlim))
        (lim (cadr varlim)))
    '(letrec ((loop (lambda (,var)
                      (if (< ,var ,lim)
                          (begin ,body
                                  (loop (+ ,var 1)))))))
      (loop 0))))
```

qui s'utilise de la manière suivante: `(dotimes (VAR LIM) BODY)` et qui est une boucle qui exécute *BODY* de manière répétitive avec la variable *VAR* qui va de 0 à la valeur de *LIM*.

1. Montrer trois exemples d'usage de cette macro naïve qui souffrent du problème de capture de nom: un dans *VAR*, un dans *LIM*, et un dans *BODY*.
2. Cette définition naïve souffre d'un autre problème. Lequel? Corrigez-le.
3. Écrire une autre définition qui évite la capture de nom en utilisant `gensym`.
4. Écrire encore une autre définition qui évite aussi la capture de nom mais sans utiliser `gensym`.

3 Prolog

Supposons qu'on a déjà défini les relations suivantes:

`pere(X,Y)` X est le père de Y
`mere(X,Y)` X est la mère de Y

Définissez les relations suivantes, en évitant autant que possible les solutions redondantes et les boucles infinies. Vous pouvez supposer qu'il n'y a pas d'enfants issus d'un couple consanguin.

`parent(X,Y)` X est un parent de Y
`grandmere(X,Y)` X est une grand-mère de Y
`freresoeur(X,Y)` X est un frère ou une soeur de Y
`oncletante(X,Y)` X est un oncle ou une tante de Y
`cousins(X,Y)` X et Y sont cousins
`demifreere(X,Y)` X est le demi-frère (ou demi-soeur) de Y
`distance(X,Y,N)` La distance minimum entre X et Y dans l'arbre généalogique est N

4 Types

Dans les morceaux de code Haskell ci-dessous, donner le **type** que doit avoir chaque trou (dénotés (•)) pour que le type de l'expression dans son ensemble soit celui indiqué. Par exemple, la réponse du numéro 0 est $(\text{Int} \rightarrow \text{Int}, \alpha)$.

0. $\text{fst } (\bullet) : \text{Int} \rightarrow \text{Int}$
1. $1 + (\bullet) : \text{Int}$
2. $(+) 1 (\bullet) : \text{Int}$
3. $\text{snd } (\bullet) : (\text{Int}, (\text{Int}, \text{Int})) \rightarrow \text{Int}$
4. $(\bullet) (3 + 4) : \text{Int} \rightarrow \text{Int}$
5. $(\bullet) (3 + 4) (5, 2) : \text{Int}$
6. $\text{fst } (\bullet) (\text{fst}, 2) : \text{Int}$
7. $\lambda x \rightarrow (\bullet) : \text{Int} \rightarrow (\text{Int}, \text{Int})$
8. $\lambda x \rightarrow (\bullet) : \text{Int}$
9. $\text{let } x = (\bullet) \text{ in } x (+) 1 \text{ end} : \text{Int} \rightarrow \text{Int}$
10. $(-) (\bullet) : [\text{Int}]$

Rappel, les fonctions *map*, *fst*, et *snd* sont (pré)définies comme suit:

$$\begin{aligned} \text{map } f [] &= [] \\ \text{map } f (x : xs) &= f x : \text{map } f xs \\ \text{fst } (x, y) &= x \\ \text{snd } (x, y) &= y \end{aligned}$$

Et $x + y$ n'est rien de plus que du sucre syntaxique pour $(+) x y$.

5 Concurrency

Soient les types et opérations suivants:

```
mutex *new_mutex();  
void lock(mutex *m);  
void unlock(mutex *m);  
  
condvar *new_condvar();  
void wait(mutex *m, condvar *c);  
void signal(condvar *c);
```

1. Définir les types et fonctions nécessaires pour implanter des *mutex_rw* qui distinguent les accès en lecture des accès en écriture (i.e. qui permettent plusieurs accès simultanés en lecture) en utilisant les *mutex* et *condvars* ci-dessus.
2. Décrire les conditions nécessaires pour que le code que vous avez écrit (et tout autre code qui accède aux données qu'il manipule) ne souffre pas de condition de course.
3. Réécrire le code en Haskell en utilisant le monad STM.