Série d'exercices #7

IFT-2035

May 29, 2025

7.1 Types et classes de types

Donner le types des expressions Haskell ci-dessous.

1. $(+)$
2. $(+)\ (3 :: Int)$
3. $\lambda x \rightarrow fst\ x + snd\ x$
4. $\lambda x\ y \rightarrow \mathsf{if}\ fst\ x = snd\ x\ \mathsf{then}\ y\ \mathsf{else}\ y + 1$
5. $map\ (\lambda x \rightarrow \mathsf{if}\ fst\ x < 0\ \mathsf{then}\ snd\ x\ \mathsf{else}\ snd\ x + 1)$

7.2 Keep trying

Soit une macro `until` en ELisp, qui prend la forme suivante:

```
(until (COND EXP))
```

Une telle expression devrait répéter `EXP`, jusqu'à ce que `COND` soit différent de `nil`. De plus elle renvoie comme valeur la dernière valeur de `COND`.
Une implantation naïve pourrait être

```
(defmacro until (ce)
  (let ((cond (car ce))
        (exp (cadr ce)))
    `(letrec ((loop (lambda ()
                      (if ,cond
                          ,cond
                        (progn ,exp (funcall loop))))))
       (funcall loop))))
```

Montrer le résultat de l'expansion de la macro pour les usages suivants:

1. `(until (exp (setq exp (read-expression))))`
2. ```
   (until ((progn (message "new iteration") cond)
     (setq cond (read-answer))))
   ```
   ```

3.
```
(let ((loop nil))
     (until (loop
       (setq loop (read-answer)))))
```

Certaines de ces expansions sont incorrectes.

4. Indiquer précisément chacun des problèmes.

5. Montrer quelles seraient des expansions correctes.

6. Corriger la définition de la macro pour éliminer ces problèmes.

7.3 Déterminer le passage de paramètres

Écrire un programme (dans un langage hypothétique avec une syntaxe similaire à celle de C), qui donnerait un résultat différent dans chacune des 4 méthodes de passage de paramètres:

- passage de paramètres par valeur
- passage de paramètres par référence
- passage de paramètres par valeur-résultat
- passage de paramètres par nom

Bonus portée Modifier le programme de sorte qu'il produise des résultats différents selon que la portée des variables est lexicale ou dynamique (pour un total de 8 résultats différents pour les 8 combinaisons possibles).

7.4 Programmation fonctionelle en C

Soit la fonction C suivante:

```
void main (void)
{ /* Élimine les caractères répétés et
   * stoppe après la première ligne vide.  */
  int c;
  int last = EOF;
  while ((c = getchar ()) != EOF) {
    if (c == last) {
      if (c == '\n') {
        break;
      } else {
        continue;
      }
    }
    putchar (last = c);
  }
}
```

D'abord, réécrire le code dans un style de programmation structurée stricte, c'est à dire sans utiliser de continue, break, ou goto.

Ensuite, réécrire le code à nouveau mais cette fois dans un style fonctionnel, c'est à dire sans opération d'affectation (sauf bien sûr dans les initialisations, e.g. `int last = EOF`). Il faudra introduire une fonction récursive auxiliaire et éliminer while, break, et continue.

Essayer de faire ces réécritures en les subdivisant en plusieurs petits pas simples, où on peut facilement voir que chaque pas préserve la sémantique du code précédent, sans avoir besoin de comprendre le code dans son ensemble.