Programowanie funkcyjne – laboratorium 4
28.10.2009 r.

Listy

1. Przy pomocy funkcji `fold_right` lub `fold_left` zdefiniuj procedury:
   (a) `append`
   (b) `sum` liczącą sumę liczb na liście
   (c) `sum_function`, która oblicza funkcję będącą sumą funkcji z danej listy
   (d) `cat` łączącą listę napisów w jeden napis

   ```
   cat ["jeden"; "dlugi"; "napis"] = "jeden dlugi napis "
   ```

   (e) `flatten` łączącą elementy listy list w jeden napis

   ```
   flatten [[1;2;3];[];[4;5]] = [1;2;3;4;5]
   ```

   (f) `len` liczącą długość listy
2. Zaimplementuj `fold_left` przy pomocy `fold_right`, a następnie `fold_right` przy pomocy `fold_left`.
3. Napisz procedurę `sumy: int list -> int list`, której wynikiem dla danej listy `[x_1,..., x_n]` jest lista: `[x_1, x_1 + x_2, x_1+x_2+x_3, ..., x_1+x_2+...+x_n]`.

   ```
   sumy [1; 5; 2; 7; 12; 10; 5] = [1; 6; 8; 15; 27; 37; 42].
   ```

4. Napisz procedurę `podzial: int list -> int list list`, która dla danej listy liczb całkowitych `l = [x_1; x_2; ...; x_n]` podzieli ją na listę list `[l_1; ...; l_k]`, przy czym:
   - `l = l_1 @ ...@ l_k`,
   - każda z list `l_i` jest ściśle rosnąca,
   - `k` jest najmniejsze możliwe.

   ```
   podzial [1;3;0;-2;-2;4;9] = [[1; 3]; [0]; [-2]; [-2;4;9]].
   ```

5. Napisz procedurę `podzial2: int list -> int list list`, która dla danej listy liczb całkowitych `l = [x_1; x_2; ...; x_n]` podzieli ją na listę list `[l_1; ...; l_k]`, przy czym:
   - `l = l_1 @ ...@ l_k`,
   - dla każdej listy `l_i` wszystkie elementy na takiej liście są tego samego znaku,
   - `k` jest najmniejsze możliwe.

   ```
   podzial2 [1;3;0;-2;-2;-4;9] = [[1; 3]; [0]; [-2;-2;-4]; [9]].
   ```

6. Korzystając z funkcji `filter` zaimplementuj algorytm quicksort.
7. Zaimplementuj algorytm Eratostenesa, zwracający listę liczb pierwszych zawartych w przedziale `[1;...;n]` (`n` jest parametrem algorytmu). Algorytm działa następująco:
   - generuje początkową listę `[2;...;n]`;
   - w pętli (tzn. rekurencyjnie) przekształca listę w następujący sposób: przy założeniu, że pierwszym elementem listy jest liczba pierwsza dołącza ją do listy liczb pierwszych, usuwa z listy wszystkie jej wielokrotności (użyj do implementacji tego kroku funkcji `filter`) i powstałej w ten sposób listy używa w następnym kroku;
   - pętla jest powtarzana dopóki lista jest niepusta.

Drzewa

1. Przerób funkcję `bst_insert_list`, wstawiającą do drzewa listę elementów tak, aby używała `fold` i `bst_insert`.
2. Napisz funkcję `fold_tree`, działającą podobnie jak `fold` działa na listach.
3. Przy pomocy funkcji `fold_tree` zaimplementuj jeszcze raz:
   - `elem_list` tworzącą posortowaną listę elementów drzewa BST
   - `mirror` tworzącą lustrzane odbicie drzewa
   - `size` obliczającą wielkość drzewa
   - `contains` sprawdzającą czy drzewo (niekoniecznie BST) zawiera dany element
4. Napisz przy pomocy `fold_tree` funkcję sprawdzającą, czy drzewo jest pełne (tzn. czy wszystkie liście są jednakowo odległe od korzenia).
5. Zdefiniuj funkcję `tree_to_string` przekształcającą drzewo na napis. Na przykład

   ```
   tree_to_string (Node (1, Leaf, Node (2, Leaf, Leaf))) = "<1<2>>"
   ```

   Użyj `fold_tree` i `string_of_int`.
6. Wypisz drzewo ładniej. Zamiast `<<<1>2<4>>3<<1>2<4>>>` wypisz na przykład:

   ```
     3
   .....
   2   2
  ... ...
  1 4 1 4
   ```

   Wskazówka: napisz funkcję przekształcającą drzewo na listę napisów, po jednym dla każdej linii rysunku. Do wypisania pojedynczej linii użyj `print_string`. Do wypisania listy – `fold`.

Liczebniki Churcha

Liczebniki Churcha to funkcje reprezentujące liczby naturalne:

```
c0 = function f -> function x -> x;;          (* albo: c0 f x = x *)
c1 = function f -> function x -> f x;;        (* albo: c1 f x = f x *)
c2 = function f -> function x -> f (f x);;    (* albo: c2 f x = f (f x) *)
```

1. Napisz funkcję `church_to_int` konwertujący liczebnik Churcha na liczbę naturalną, którą reprezentuje.
2. Napisz funkcję następnika spelniajaca:

   ```
   church_to_int (succ c) = 1 + church_to_int c
   ```

3. Napisz funkcję dodawania spełniającą:

   ```
   church_to_int (add c1 c2)  = (church_to_int c1) + (church_to_int c2)
   ```

4. Napisz funkcje mnożenia i potęgowania.