Zadání započtového domácího úkolu pro KMI/YPP1

22.10.13

Celkový počet bodů je 45, z nich student, z nich student musí získat 30. Řešení odevzdávejte e-mailem (jako přílohu):
adresa: `jan.konecny@upol.cz`
předmět: `KMI/YPP1 ukol`
příloha: ◁*jmeno*▷`_`◁*prijmeni*▷`.scm`
V odevzdávaném souboru mějte jen definice požadovaných procedur, veškeré pomocné procedury definujte interně; v souboru nemějte příklady použití a testy, kód nemusí být komentován.

1. Naprogramujte proceduru `interlace`, která bere dva argumenty: seznam ◁*list*▷, libovolný element ◁*elem*▷, a vrací seznam, který vznikne ze seznamu ◁*list*▷ vložením elementu ◁*elem*▷ mezi každé dva prvky.

   Příklady použití:
   `(interlace () 'x)` ⇒ ()
   `(interlace (list 1) 'x)` ⇒ (1)
   `(interlace (list 1 2 3) 'x)` ⇒ (1 x 2 x 3)
   `(interlace (list 1 2 3) ())` ⇒ (1 () 2 () 3) (5 bodů)

2. Naprogramujte predikát `interlaced?`, který vrací #t, pokud je jeho argument seznam liché délky (nebo je prázdný) a prvky na sudých pozicích jsou si rovny. Predikát vlastně zjišťuje, jestli seznam mohl vzniknout použitím procedury `interlace` z předchozího příkladu.

   Příklady použití:
   `(interlaced? ())` ⇒ #t
   `(interlaced? (list 1))` ⇒ #t
   `(interlaced? (interlace (list 1 2 3) 'x))` ⇒ #t
   `(interlaced? '(1 x 2 x))` ⇒ #f
   `(interlaced? '(x 1 x 2 x))` ⇒ #f
   `(interlaced? '(x x x))` ⇒ #t (3 body)

3. Naprogramujte proceduru `collate`, která bere jako argument seznam ◁*list*▷ a vrací seznam párů `(` ◁*elem*▷ `.` ◁*count*▷ `)`, kde ◁*elem*▷ je prvek ze seznamu ◁*list*▷, ◁*count*▷ je počet výskytů ◁*elem*▷ v seznamu ◁*list*▷. Páry jsou uspořádány sestupně podle ◁*count*▷. Pro setřídění můžete použít `sort`

   Příklady použití:
   `(collate ())` ⇒ ()
   `(collate '(x))` ⇒ ((x . 1))
   `(collate (interlace (list 1 2 3) 'x))` ⇒ ((x . 2) (1 . 1) (2 . 1) (3 . 1))
   `(collate '(1 1 2 1 2))` ⇒ ((1 . 3) (2 . 2)) (6 bodů)

4. Naprogramujte proceduru `pick-combination`, která bere jako argumenty dva seznamy: seznam ◁*list*▷ a číselný seznam ◁*indexes*▷ – čísla v něm představují pozice. Procedura vrací seznam prvků ze seznamu list, které odpovídají pozicím číselném seznamu.

   Příklady použití:
   `(pick-combination '(a b c) '(1 1 3 1))` ⇒ (a a c a)
   `(pick-combination '(a b c) '(1 3 2))` ⇒ (a c b) (3 body)

5. Naprogramujte predikát `pairwise-test?`, která bere dva argumenty: seznam ◁*list*▷ a predikát ◁*pred?*▷ dvou argumentů. Predikát `pairwise-test?`, který vrací #t, právě když každé dva sousední prvky v seznamu ◁*list*▷ splňují predikát ◁*pred?*▷. Řešte pomocí rekurze.

   Příklady použití:
   `(pairwise-test? '(1 3 4) <=)` ⇒ #t
   `(pairwise-test? '(1 3 4) (lambda(a b) (= a (- b 1))))` ⇒ #f
   `(pairwise-test? '(1 2 3) (lambda(a b) (= a (- b 1))))` ⇒ #t

   (7 bodů)

6. Naprogramujte predikát `pairwise-test?` z předchozího příkladu pomocí akumulace. (6 bodů)

7. Uvažujme následující reprezentaci stromů: každý uzel je reprezentován jako struktura `(`◁*tag*▷ `(`◁*left*▷ ◁*right*▷`))`. ◁*tag*▷ je pravdivostní hodnota, která reprezentuje barvu (#t je černá,, #f je červená) a ◁*left*▷ a ◁*right*▷ jsou prázdné seznamy a nebo další uzly. Listy jsou representovány jako `(`◁*tag*▷ `())`.

   Naprogramujte predikát `redblack-tree?`, který zjišťuje, jestli je jeho argument reprezentací červeno-černého stromu. To znamená

   - kořen je černý,
   - na cestě od kořenu ke každému listu je stejný počet černých uzlů,
   - pokud je uzel červený, tak ◁*left*▷ a ◁*right*▷ jsou černé.

   (15 bodů)