MP18 @ II UWr 4 czerwca 2018 r.

Lista zagadnień nr 14

Przed zajęciami

Tematem przewodnim jest stan mutowalny. Przed zajęciami należy zapoznać się z **Rozdziałami 3.1**, **3.3** i **3.5** podręcznika. Należy rozumieć pojęcia **stanu**, **tożsamości**, **mutatora**, **pary modyfikowalnej**, **wektora modyfikowalnego**. Uwaga! Język Racket nie wspiera mutacji list budowanych za pomocą `cons` i `list`, do rozwiązania poniższych zadań, jeśli jest to konieczne, użyj list mutowalnych opisanych w **The Racket Reference**, rozdział **4.10**. Mutowalne wektory są opisane w rozdziale **4.11**.

W trakcie zajęć

Ćwiczenie 1.

Rozszerz funkcję `make-account` z wykładu o zabezpieczenie hasłem. Niech ta funkcja przyjmuje dodatkowy parametr – symbol, który potraktujemy jako hasło dostępu do konta. Funkcja zwrócona z `make-account` również powinna zostać rozszerzona o dodatkowy argument, który będzie sprawdzany, czy jest równy z hasłem podanym do `make-account`. Jeśli nie, funkcja ta powinna nie wykonywać żądanej akcji i zwrócić symbol `'incorrect-password`. Przykład:

```
> (define acc (make-account 100 'secret-password))
> ((acc 'secret-password 'withdraw) 40)
60
> ((acc 'some-other-password 'deposit) 50)
'incorrect-password
```

Ćwiczenie 2.

Napisz funkcję `make-cycle`, która zmieni listę modyfikowalną podaną jako jedyny parametr na listę cykliczną, nadpisując `mcdr` ostatniej pary należącej do listy, aby wskazywała na pierwszą.

Ćwiczenie 3.

Napisz funkcję `has-cycle?`, która sprawdza, czy lista modyfikowalna podana jako argument zawiera cykl – tzn. czy przeglądając kolejne pary należące do listy trafimy w końcu na parę, którą już wcześniej odwiedziliśmy. Przykładowo, listy tworzone przez funkcję `make-cycle` z poprzedniego zadania są cykliczne. Uwaga: nie są to jedyne takie listy, przykładowo lista `(mcons 0 (make-cycle (mcons 1 null)))` zawiera cykl. *Opcjonalnie*: Czy potrafisz rozwiązać to zadanie tak, aby funkcja `has-cycle?` działała w stałej pamięci?

Ćwiczenie 4.

Napisz funkcję `make-monitored`, która rozszerzy funkcję podaną jako argument o funkcjonalność zliczania jak wiele razy ta funkcja została wywołana. Funkcja `make-monitored` ma zwracać parę dwóch funkcji. Pierwsza z nich powinna zachowywać się tak samo, jak funkcja podana w argumencie, a oprócz tego powinna (jako efekt uboczny) liczyć swoje wywołania. Druga z nich, wywołana z symbolem `'how-many?` jako argument, powinna zwrócić liczbę wywołań pierwszej funkcji, natomiast wywołana z symbolem `'reset` powinna wyzerować licznik wywołań. Przykład:

```
> (define p (make-monitored +))
> ((car p) 1 2 3)
6
> ((cdr p) 'how-many?)
1
> ((car p) 2 2)
4
> ((cdr p) 'how-many?)
2
> ((cdr p) 'reset)
> ((cdr p) 'how-many?)
0
```

Ćwiczenie 5.

Zaimplementuj **sortowanie kubełkowe** w wariancie uproszczonym – jeden kubełek na jedną wartość. Funkcja `bucket-sort` ma otrzymać jako jedyny parametr listę par klucz-wartość, w których kluczami są niewielkie liczby naturalne. Wynikiem ma być lista posortowana względem klucza. Sortowanie powinno być stabilne – tzn. kolejność par nie różniących się kluczem powinna być taka sama, jak w wejściu. Przykład:

```
> (bucket-sort (list (cons 2 'x) (cons 1 'y) (cons 2 'z)))
'((1 . y) (2 . x) (2 . z))
```

Do implementacji wykorzystaj mutowalny wektor, w komórkach wektora możesz przechowywać pary, których klucz jest równy indeksowi komórki.

Ćwiczenie 6.

Zdefiniuj ciąg kolejnych wartości silni wykorzystując funkcje z wykładu: `lcons`, `lmap` i `integers-starting-from`.

Ćwiczenie 7.

Napisz funkcję, która oblicza sumy częściowe leniwej listy podanej jako parametr i zwraca je w postaci leniwej listy. Wykorzystaj do tego funkcje `lcons` i `lmap`. Przetestuj ją, obliczając sumy częściowe leniwej listy zawierającej nieskończoną liczbę jedynek, oraz leniwej listy wszystkich liczb naturalnych. Czy spamiętywanie poprawia efektywność obliczeń?

Ćwiczenie 8.

Napisz funkcję `merge`, która otrzymuje dwa argumenty – rosnące strumienie liczb całkowitych, a zwraca rosnący strumień liczb znajdujących się w obu strumieniach wejściowych. Używając funkcji `lcons`, `lmap`, `integers-starting-from` i `merge` zdefiniuj `num235` – rosnący strumień wszystkich dodatnich liczb naturalnych, które posiadają wyłącznie 2, 3 i 5 jako swoje dzielniki. Przydatne spostrzeżenia:

- pierwszym elementem `num235` jest 1;
- elementy strumienia `num235` pomnożone przez 2 również należą do `num235`;
- analogicznie dla 3 i 5;
- żaden inny element nie należy do `num235`.