Lista zagadnień nr 3

Wnioskowanie

Ćwiczenie 1.

Zakładając, że dla dwóch wartości `v1` i `v2`, wyrażenie `(cons v1 v2)` samo w sobie jest wartością, zaproponuj reguły w naszym modelu ewaluacji wyrażeń racketowych dla `cons`, `car`, `cdr` i `pair?`. Czy umiesz przy ich użyciu udownić, że:

- `(car (cons x y))` $\equiv$ `x`
- Jeśli `(pair? x)` $\equiv$ `true`, to `(cons (car x) (cdr x))` $\equiv$ `x`

Ćwiczenie 2.

Rozważ następującą procedurę:

```
(define (mult a b)
  (if (= b 0)
      0
      (+ a (mult a (- b 1)))))
```

Pokaż, że dla wszystkich naturalnych `a` i `b` zachodzi `(mult a b)` $\equiv$ `(* a b)`.

Pary

Ćwiczenie 3.

Zdefiniuj procedury:

- `(define (swap p) ...)` – zamienia miejscami elementy pary `p`, np. `(swap (cons 2 5))` $\equiv$ `(cons 5 2)`.
- `(define (fun-product f g) ...)` – zwraca procedurę, która aplikuje procedury `f` i `g` „po współrzędnych”, np. `((fun-product inc square) (cons 3 5))` $\equiv$ `(cons 4 25)`.

Ćwiczenie 4.

Korzystając z faktu, że jeśli $\langle a, b\rangle$ to para zawierająca kolejne liczby Fibonacciego, to para $\langle b, a+b\rangle$ też zawiera kolejne liczby Fibonacciego, uzupełnij poniższą definicję procedury obliczającej `n`-tą liczbę Fibonacciego, gdzie `repeated` to procedura z poprzedniej listy zadań składająca daną procedurę samą ze sobą daną liczbę razy:

```
(define (fib n)
  (define (step p) ... )
  (car ((repeated step n) (cons 0 1))))
```

Czy umiesz zmodyfikować to rozwiązanie, by obliczało liczby Tribonacciego dane wzorem $T_0 = 0$, $T_1 = 0$, $T_2 = 1$, $T_{n+3} = T_{n+2} + T_{n+1} + T_n$? A zgeneralizować na liczby k-bonacciego, dane wzorem $T_0 = 0, \ldots, T_{k-1} = 0$, $T_k = 1$, $T_{n+k} = T_{n+k-1} + \cdots + T_n$, gdzie k jest dodatkowym argumentem procedury?

Macierze

Ćwiczenie 5.

Zdefiniuj typ danych macierzy o wymiarze 2×2 poprzez uzupełnienie implementacji następującego interfejsu:

- `(define (matrix a b c d) ... )`

  Wynikiem jest macierz $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

- `(define (matrix-at m x y) ... )`

  Wynikiem jest element macierzy `m` w `x`-owym wierszu (licząc od 1) i `y`-owej kolumnie, np. `(matrix-at (matrix 10 20 30 40) 2 1)` $\equiv$ `30`.

- `(define (matrix?) ... )`

  Predykat definiujący kształt reprezentacji.

Macierze reprezentuj jako parę wierszy, gdzie każdy wiersz jest parą dwóch wartości.

Ćwiczenie 6.

Korzystając z interfejsu z poprzedniego zadania zdefiniuj:

- `(define (matrix-mult m n) ... )` – iloczyn dwóch macierzy
- `(define matrix-id ...)` – macierz identycznościowa

Zadbaj o to, by nie złamać konwencjonalnej abstrakcji danych!

Ćwiczenie 7.

Użyj naszej mikroskopijnej biblioteki macierzowej z poprzednich dwóch zadań, by zdefiniować procedury:

- `(define (matrix-expt m k) ... )` – podnosi macierz `m` do `k`-tej potęgi (naturalnej)
- `(define (fib k) ... )` – oblicza `k`-tą liczbę Fibonacciego F_k korzystając z zależności $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^k = \begin{bmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{bmatrix}$

Ćwiczenie 8.

Zdefiniuj procedury `fast-matrix-expt` i `fast-fib` analogiczne do tych z poprzedniego zadania, ale stosujące algorytm szybkiego potęgowania. Czy umiesz zaimplementować szybkie potęgowanie jako proces iteracyjny? Użyj racketowej procedury `time`, żeby sprawdzić, czy rzeczywiście jest szybciej.

Ćwiczenie 9.

Alternatywną reprezentacją macierzy $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ jest dwuargumetnowa procedura zwracająca element macierzy o danych współrzędnych. Np. macierz $\begin{bmatrix} 10 & 10 \\ 20 & 30 \end{bmatrix}$ można reprezentować przy użyciu procedury

```
(lambda (x y) (cond [(= x 1) 10]
                    [(= y 1) 20]
                    [(= y 2) 30]))
```

Podmień implementację z Zadania 5. tak, by używała reprezentacji w formie procedury. Czy po zmianie reprezentacji kod z Zadań 6–8 nadal działa poprawnie?