**עבודה 2 PPL**

שאלה 1:

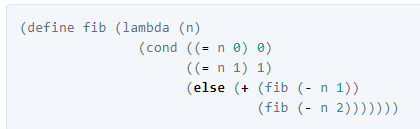
א)

ב)

* תכנית שניתנת לחישוב באופן מקבילי:
* תכנית שאינה מוגנת עבור חישוב באופן מקבילי:

ג) לא

ד) כן. דוגמה מהתרגול:



בעצם לא נוכל למצוא מספר פיבונאצ'י בלי להגדיר פונקציה כלשהי שתקרא ברקורסיה.

לא ניתן לעשות גם פתרון איטרטיבי כיוון שלא ניתן להגדיר משתנה כי אין יותר define.

ה)

|  |  |  |
| --- | --- | --- |
| explanation | answer | function |
| because | Yes | Map |
| because | No | Reduce |
| because | Yes | filter |
| because | Yes | All |
| because | No | compose |

ו) 12

Append:

; Signature: append(l1 l2)

; Type: [List(T) \* List(T) -> List(T)]

; Purpose: two lists and returns their concatenation

; Pre-conditions: none

; Tests: (append '(1 2) '(3 4)) → '(1 2 3 4)

Reverse:

; Signature: reverse(ls)

; Type: [List(T) -> List(T)]

; Purpose: gets a list and reverses it

; Pre-conditions: none

; Tests: (reverse '(1 2 3)) → '(3 2 1)

Duplicate-items:

; Signature: duplicate-items(ls dup-count)

; Type: [List(T) \* List(T) -> List(T)]

; Purpose: gets two lists - lst , dup-count - and duplicates each item of lst according to the number defined in the same position in dup-count. In case dups-count length is smaller than lst, dup-count should be treated as a cyclic list;Pre-conditions: none

; Tests: (duplicate-items '(1 2 3) '(1 0))→ '(1 3)

(duplicate-items '(1 2 3) '(2 1 0 10 2))→ '(1 1 2)

Payment:

; Signature: payment(n coins-lst)

; Type: [number \* List(number) -> number]

; Purpose: gets a sum of money and list of available coins, and returns the number of possible ways to pay the money with these coins.

; Pre-conditions: none

; Tests: (payment 10 ‘(5 5 10)) → 2

(payment 5 ‘(1 1 1 2 2 5 10) → 3

Compose-n:

; Signature: compose(f n)

; Type: [closure<T> \* number -> closure<T>]

; Purpose: h gets an unary function f and a number n (>0) and returns the closure of the n-th self-composition of f

; Pre-conditions: f is an unary function, n>0.

; Tests:

(define mul8 (compose-n (lambda (x) (\* 2 x)) 3))

(mul8 3) → 24