עבודה PPL-2

1.1)על מנת שנוכל לבצע תהליכים מורכבים, מעבר לחישובים פרימיטיביים, נזדקק לצורות המיוחדות. דרך הצורות המיוחדות נוכל להגדיר פונקציות ופעולות שונות, בהתאם לצורה המיוחדת הספציפית, כאשר כל אחת פועלת אחרת, וגם באופן שונה מהצורות הפרימיטיביות. לדוגמה:
(define x 1) במקום לבצע חישוב מסוים בין x ל-1 ,הוא מבצע פעולה מיוחדת, של הגדרת משתנה x והשמה של הערך 1 בתוכו , דבר שאינו יכול לקרות ללא צורות מיוחדות, או אופרטור if שמאפשר לנו לבצע בדיקה האם משתנה שווה לערך כלשהו, ומה עושים במידה וכן ובמידה ולא, וגם הוא פועל בצורה שונה מפעולה רגילה.

1.2) תוכנית המאפשרת הרצה מקבילית: ((2 2 +) (1 1+)-) – מתאפשרת הפעולה של הרצה במקביל של 2 החיבורים
תוכנית שלא מאפשרת הרצה מקבילית: ((4 3 -) 2 -) 1-), כל פעם נוכל לעשות חישוב אחד בלבד(כל חישוב תלוי באחר)

1.3) כן, ניתן לכתוב כל תוכנית ב- L1 גם בשפה החדשה L0 שכן, נוכל במקום כל ביטוי בו הפעלנו define את הערך עצמו (לא נוח אבל תקין).

1.4)לא ניתן לכתוב כל תוכנית בשפה החדשה L20 שכן כאשר אנחנו בשפה L2 אנחנו יכולים לכתוב פונק' רקורסיביות על ידי הפעלת define ונתינת שם לפעולות, כאשר אנו מורידים את הצורה define בשפה החדשה, ונרצה לבנות רקורסיה אינסופית ,לא נוכל לכתוב כל פעם מחדש את הפונקציה, כי אנו לא יכולים לכתוב את הפונקציה בצורה אינסופית ולכן הפעולה לא תתבצע.

1.5)Map – כאשר אנו מפעילים מאפ על רשימה, הפעולה אינדיבידואלית עבור כל איבר ולא מתחשבת באיברים אחרים, לכן עם כל איבר לפני ואחרי המניפולציה נשאר במקומו, ניתן לבצע ע"י ריצה של תוכנית מקבילית.

Reduce- בפעולה זו קיים לנו צובר ויכולות לרוץ פונקציות שהרדיוסר מבצע שתלויות בתוצאות אחרות שיכולות לפגוע בנכונות הקוד אם נבצע אותו בצורה מקבילית, ולכן את פעולה זו לא ניתן לבצע בצורה מקבילית.

Filter- מפני שהבדיקה של פילטר מתבצעת עבור כל איבר בנפרד, ללא תלות באיברים אחרים ומחזירה את האיברים שמחזירים את הסינון, נוכל להריץ תוכנית מקבילית.

All-ניתן להריץ באופן מקבילי, שכן הבדיקה מתבצעת עבור כל אחד בנפרד ללא חשיבות לסדר ומכיוון שכולם צריכים להחזיר #t אין חשיבות לסדר הריצה על האיברים ברשימה, שכן בכל מקרה כשנסיים את הבדיקה עבור כולם נצטרך לראות שכולם מקיימים את התנאי.

Compose- לא ניתן להריץ באופן מקבילי, כיוון שבהרכבה של פרוצדורות שמורכבות אחת על השנייה יש חשיבות לסדר ההרכבה, והריצה יכולה להניב תוצאה שונה בחישובים אחרים שמקבלים מריצה מקבילית (למשל כפל ואז חיבור או חיבור ואז כפל).

1.6) התשובה שתתקבל היא 9, כאשר a=3,b=4,c=2. הסבר לערכים אלה הוא שa מקבל את הערך כהשמה בפעולת ההגדרה של הפונק' וכך גם b, כאשר הערך יהיה 4 וההשמה של הערך 1 לא מתקבלת לאחר שהגדרנו את הנקודה. הערך של c הוא 2 מפני שכאשר אנחנו לוקחים את ה-c הגלובלי, שהיה מוגדר ברגע שנכנסנו לפעולה, כאשר הוא אינו מוגדר בתוך הנקודה. ההפעלה מתעלמת מהערך 5 ובכך נקבל את חיבור הערכים שיוצר לנו את ערך ההחזרה 9.

2.1)

Signature: append (lst1 lst2)

Type: [list*list -> list]

Purpose: append two lists into one

Pre-conditions: none

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

2.2)

Signature: reverse (lst)

Type: [list -> list]

Purpose: change the elements to a reverse order

Pre-conditions: none

Tests: (reverse '(1 2 3)) → '(3 2 1)

2.3)

Signature: duplicate-items (lst dup-count)

Type: [list*list -> list]

Purpose: duplicate every item in lst by getting the number of duplications from dup-count list in a circular order

Pre-conditions: dup-count list of numbers and not empty

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

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

(2.4

Signature: payment (n coins-lst)

Type: [number*list -> number]

Purpose: get the number of possible ways to pay by the coins list.

Pre-conditions: coins-lst is not empty

Tests: (payment 5 ‘(1 1 1 2 2 5 10) → 3

(payment 10 ‘(5 5 10)) → 2

(2.5

Signature: compose-n (f n)

Type: [function*number -> number]

Purpose: get the closure of the n-th composition of f

Pre-conditions: n>0

Tests: (define mul8 (compose-n (lambda (x) (* 2 x)) 3)) (mul8 3) → 24