Assignment 2

Dor Lebel 205872807 Nitzan Hochman 316264845

Q1.1 Give an example for each of the following categories in L3:

●Primitive atomic expression - 3

●Non-primitive atomic expression – x (when x is defined beforehand)

●Non-primitive compound expression – (* 7 3)

●Primitive atomic value – the numerical value of 3

●Non-primitive atomic value – the symbol ‘green

●Non-primitive compound value – a list (3, 5, 8)

Q1.2 What is a special form? Give an example [2 points]

A special form is an expression that follows *special* evaluation rules, for example: `(define x 3)` .

Q1.3 What is a free variable? Give an example [2 points]

Free variable is a variable that isn’t bound within the given scope.

(+ x x) → x is a free variable within the scope of the expression .

Q1.4 What is Symbolic-Expression (s-exp)? Give an example [2 points]

Is a general structure that is defined recursively and is a composition of atomic values (of certain language) which forms an expression. For example:

‘(+ 1 (* 2 3))’

Q1.5 What is ‘syntactic abbreviation’? Give two examples [5 points]

A syntactic abbreviation is an expression that is equivalent to a combination of other syntactic constructs that mean the same thing. For example:

- Let can be defined with ‘ define ‘ and ‘ Lambda’ expressions.

  ```
  (let
          ((x 1) (y 2))(+ x y))
  ```

  Is the same as:

  ```
  ((lambda
          (x y)(+ x y)) 1 2)
  ```

- If can be defined with ‘cond’ expressions.

```
(if ( = x 3)
        7 8)
```

Is the same as:

```
(cond ((= x 3) 7)
        (else 8))
```

Q1.6 Let us define the L30 language as L3 excluding the list primitive operation and the literal expression for lists with items (there is still a literal expression for the empty list ‘()).Is there a program in L3 which cannot be transformed to an equivalent program in L30? Explain or give a contradictory example.[5 points]

We can transform any program in L3 that uses the list primitive operation and the literal expression for lists to a program in L30 because list is a syntactic abbreviation, we can use the operations cons and ()’ to create any list. For example, instead of using “ list 1 2 3” we can use “ “(cons 1 (cons 2 (cons 3 ‘())))”.

Q1.7 In practical session 5, we dealt with two representations of primitive operations: PrimOpand Closure. List an advantage for each of the two methods [2 points].

Closure: easier addition of new primitive operations.

PrimOp: faster return value because there’s no need to go through the environment list.

Q1.8 In class, we implemented map in L3, where the given procedure is applied on the first item of the given list, then on the second item, and so on. Would another implementation which applies the procedure in the opposite order (from the last item to the first one), while keeping the original order of the items in the returned list, be equivalent? Would this be the case also for: reduce, filter, compose [6 points]

Because L3 is functional, none of the functions above have side-affects, therefore, functions g and g^r are equivalent iff:

- the range and the domain are the same

-for every x in the domain , $g^r(x) = g(x)$

For the function map, the reverse function is equivalent because the domain and range are the same, and each value is computed individually with no relation to the other values. The same goes for filter.

In contrast, functions reduce and compose are computing a value based on the previous computations, and thus in some cases, they would return different values based on the order of the given values.

Q2 contracts:

; Signature: last-element (lst)

```
; Type:[ list(T) -> T ]

; Purpose: returns the last element of the given list

; Pre-conditions: lst != '()

; Tests: (last-element (list 1 3 4)) → 4


; Signature: power (n1 n2)

; Type: [Number*Number -> Number]

; Purpose: returns n1 to the power of n2 (n1^n2).

; Pre-conditions: n1 >= 0 , n2 >= 0

; Tests:(power 2 4) → 16


; Signature: sum-lst-power (lst n)

; Type: [Number*List(Number) -> Number]

; Purpose: returns the sum of all its elements in the power of N.

; Pre-conditions: n is natural

; Tests: (sum-lst-power (list 1 2 3) 2) → 14


; Signature:num-from-digits(lst)

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

; Purpose: returns the number consisted from the digits of the list

; Pre-conditions:n is natural

; Tests:(num-from-digits (list 1 2 3)) → 123


; Signature: remove-last(lst)

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

; Purpose: remove the last element in lst

; Pre-conditions:

; Tests:(remove-last (list 1 2 3)) → '(1 2)


; Signature:is-narcissistic(lst)
```

; Type: [list(Number) -> boolean]

; Purpose:returns true if the number represented in lst is a number that is the sum

; of its own digits, each raised to the power of the number of the digits.

; Pre-conditions:

; Tests: (is-narcissistic (list 1 2 3)) → false

; Signature:num-of-digits(lst)

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

; Purpose: returns the number of elements in a list

; Pre-conditions:

; Tests: (num-of-digits((list 1 2 3)))-> 3