Saishnu Ramesh Kumar

CSCI 117 – Lab 5

----------------------------------------------------------------------------------------------------------------

**Part 1 – Syntactic Sugar:**

// 1) nested if, nested case

local A B in

A = false

local C1 in

C1 = true

if C1 then

skip Browse A

else

if B then

skip Basic

else

skip Basic

end

end

end

case A of tree() then

skip Basic

else

case A of false() then

skip Basic

else

skip Basic

end

end

end

end

// 2) more expressions; note that applications of primitive binary operators

// ==, <, >, +, -, \*, mod must be enclosed in parentheses for hoz

local A One Three in

A = 2

One = 1

Three = 3

local F1 in {Eq A One F1}

if F1 then

skip Basic

else

skip Basic

end

end

local In F3 in

{IntMinus Three One In}

{Eq A in F3}

if F3 then

skip Browse A

else

skip Basic

end

end

end

// 3) "in" declaration

local T X Y Three in

Three = 3

T = tree(1:Three 2:T)

local T2 A B in

T2 = tree(1:A 2:B)

T2 = T

local One C in

One = 1

{Eq One One C}

if C then

local B Z H0 H1 in

H0 = 5

H1 = 2

{IntMinus H0 H1 B}

skip Browse B

end

else skip Basic

end

end

end

// 4) expressions in place of statements

local Fun R in

Fun = proc {$ X ProcOut()}

ProcOut() = X

end

local R1 in

R1 = 4

{Fun R1 R}

end

skip Browse R

end

// 5) Bind fun

local A B in

skip Basic

local Five Three Four E1 in

Five = 5

Three = 3

Four = 4

local P in

P = '#'(1:B 2:B)

A = rdc(1:Four 2:B 3:P)

{IntMinus Three Four E1}

{IntMinus Five E1 B}

skip Browse A

skip Browse B

skip store

end

end

end

/\*

\*\* Output from sugar2kern.txt \*\*

[local ["A","B"] [A = false(),local ["EXU1"] [EXU1 = true(),if EXU1 then [skip/BA] else [local ["EXU2"]

[EXU2 = B,if EXU2 then [skip] else [skip]]]],case A of tree() then [skip] else [case A of false() then [skip]

else [case A of true() then [skip] else [skip]]]],local ["A"] [A = 2,local ["EXU1"] [local ["EXU2","EXU3"]

[EXU2 = A,EXU3 = 1,"Eq" "EXU2" "EXU3" "EXU1"],if EXU1 then [skip] else [skip]],local ["EXU1"] [local ["EXU2","EXU3"]

[EXU2 = A,local ["EXU5","EXU6"] [EXU5 = 3,EXU6 = 1,"IntMinus" "EXU5" "EXU6" "EXU3"],"Eq" "EXU2" "EXU3" "EXU1"],

if EXU1 then [skip/BA] else [skip]]],local ["X","Y"] [local ["T"] [local ["EXU1","EXU2"]

[EXU1 = 3,EXU2 = T,T = tree(1:EXU1 2:EXU2)],local ["A","B","PTU0"] [PTU0 = tree(1:A 2:B),PTU0 = T,

local ["EXU1"] [local ["EXU2","EXU3"] [EXU2 = 1,EXU3 = 1,"Eq" "EXU2" "EXU3" "EXU1"],

if EXU1 then [local ["Z"] [local ["B"] [local ["EXU1","EXU2"] [EXU1 = 5,EXU2 = 2,"IntMinus" "EXU1" "EXU2" "B"],skip/BB]]]

else [skip]]]]],local ["Fun","R"] [Fun = proc {$ X EXU1} [EXU1 = X],local ["EXU1"] [EXU1 = 4,"Fun" "EXU1" "R"],skip/BR],

local ["A","B"] [skip,local ["EXU1","EXU2","EXU3"] [EXU1 = 4,EXU2 = B,local ["EXU4","EXU5"] [EXU4 = B,EXU5 = B,

EXU3 = '#'(1:EXU4 2:EXU5)],A = rdc(1:EXU1 2:EXU2 3:EXU3)],local ["EXU1","EXU2"] [EXU1 = 5,local ["EXU4","EXU5"]

[EXU4 = 3,EXU5 = 4,"IntMinus" "EXU4" "EXU5" "EXU2"],"IntPlus" "EXU1" "EXU2" "B"],skip/BA,skip/BB,skip/s]]

\*\* Observation and Explanation \*\*

After taking a look at the output by the sugar2kern.txt file, it can be seen that the number of local statements that were declared in the sugarKernel.txt file was more as compared to the sugar2kern file. In the sugar2kern file, you are allowed to bind two variables together within the local statement that was created. For example, as seen above, you are able to put in in a way where it's: local A B in as compared to the sugar2kern file which does it separately, local A in local B in. I have also realised that you must put an 'end' must be added at the end of conditional statements like if-else.

\*/

**Command Prompt Output for Part 1:**

Text

Description automatically generated

**Part 2 – Lists and Difference Lists:**

**Section 2A:**

/\*

\*\* Output from append.txt \*\*

(Before Reverse)

Out : [ 1 2 3 4 5 6 ]

Store : ((37, 39, 35, 31, 27, 10), '|'(1:20 2:21)),

((38, 19), nil()),

((36, 18), 3),

((34, 17), '|'(1:18 2:19)),

((32, 16), 2),

((33), '|'(1:36 2:37)),

((30, 15), '|'(1:16 2:17)),

((28, 14), 1),

((29), '|'(1:32 2:33)),

((26, 9), '|'(1:14 2:15)),

((24), 6),

((25), nil()),

((22), 5),

((23), '|'(1:24 2:25)),

((20), 4),

((21), '|'(1:22 2:23)),

((8), proc(["Ls","Ms","EXU1"],[case Ls of nil() then [EXU1 = Ms] else [case Ls of '|'(1:X 2:Lr) then [local ["EXU2","EXU3"] [EXU2 = X,local ["EXU4","EXU5"] [EXU4 = Lr,EXU5 = Ms,"Append" "EXU4" "EXU5" "EXU3"],EXU1 = '|'(1:EXU2 2:EXU3)]] else [skip]]],[("Append",8)])),

((11), '|'(1:28 2:29)),

((12), Unbound),

((13), Unbound),

((1), Primitive Operation),

((2), Primitive Operation),

((3), Primitive Operation),

((4), Primitive Operation),

((5), Primitive Operation),

((6), Primitive Operation),

((7), Primitive Operation)

Mutable Store: Empty

Current Environment : ("Append" -> 8, "L1" -> 9, "L2" -> 10, "Out" -> 11, "Reverse" -> 12, "Out1" -> 13, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT" -> 5, "Mod" -> 6, "IntMultiply" -> 7)

Stack : "Reverse = proc {$ Xs EXU1} [case Xs of nil() then [EXU1 = nil()] else [case Xs of '|'(1:X 2:Xr) then [local [\"EXU2\",\"EXU3\"] [local [\"EXU4\"] [EXU4 = Xr,\"Reverse\" \"EXU4\" \"EXU2\"],local [\"EXU4\"] [EXU4 = X,local [\"EXU5\",\"EXU6\"] [EXU5 = EXU4,EXU6 = nil(),EXU3 = '|'(1:EXU5 2:EXU6)]],\"Append\" \"EXU2\" \"EXU3\" \"EXU1\"]] else [skip]]]local [\"EXU1\"] [EXU1 = L1,\"Reverse\" \"EXU1\" \"Out1\"]skip/BOut1skip/f"

(After Reverse)

Out1 : [ 3 2 1 ]

Store : ((68, 70, 66, 42), '|'(1:61 2:62)),

((69, 55), nil()),

((67, 54, 53, 32, 16), 2),

((65, 57, 59, 45), '|'(1:54 2:55)),

((63, 56, 51, 50, 36, 18), 3),

((64), '|'(1:67 2:68)),

((61, 60, 28, 14), 1),

((62), nil()),

((58, 52), nil()),

((44, 48), '|'(1:51 2:52)),

((49, 38, 19), nil()),

((47), nil()),

((46, 34, 17), '|'(1:18 2:19)),

((43, 30, 15), '|'(1:16 2:17)),

((41), '|'(1:56 2:57)),

((40, 26, 9), '|'(1:14 2:15)),

((37, 39, 35, 31, 27, 10), '|'(1:20 2:21)),

((33), '|'(1:36 2:37)),

((29), '|'(1:32 2:33)),

((24), 6),

((25), nil()),

((22), 5),

((23), '|'(1:24 2:25)),

((20), 4),

((21), '|'(1:22 2:23)),

((8), proc(["Ls","Ms","EXU1"],[case Ls of nil() then [EXU1 = Ms] else [case Ls of '|'(1:X 2:Lr) then [local ["EXU2","EXU3"] [EXU2 = X,local ["EXU4","EXU5"] [EXU4 = Lr,EXU5 = Ms,"Append" "EXU4" "EXU5" "EXU3"],EXU1 = '|'(1:EXU2 2:EXU3)]] else [skip]]],[("Append",8)])),

((11), '|'(1:28 2:29)),

((12), proc(["Xs","EXU1"],[case Xs of nil() then [EXU1 = nil()] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3"] [local ["EXU4"] [EXU4 = Xr,"Reverse" "EXU4" "EXU2"],local ["EXU4"] [EXU4 = X,local ["EXU5","EXU6"] [EXU5 = EXU4,EXU6 = nil(),EXU3 = '|'(1:EXU5 2:EXU6)]],"Append" "EXU2" "EXU3" "EXU1"]] else [skip]]],[("Reverse",12),("Append",8)])),

((13), '|'(1:63 2:64)),

((1), Primitive Operation),

((2), Primitive Operation),

((3), Primitive Operation),

((4), Primitive Operation),

((5), Primitive Operation),

((6), Primitive Operation),

((7), Primitive Operation)

Mutable Store: Empty

Current Environment : ("Append" -> 8, "L1" -> 9, "L2" -> 10, "Out" -> 11, "Reverse" -> 12, "Out1" -> 13, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT" -> 5, "Mod" -> 6, "IntMultiply" -> 7)

Stack : ""

\*\* Output from append.out \*\*

[local ["Append","L1","L2","Out","Reverse","Out1"] [Append = proc {$ Ls Ms EXU1} [case Ls of nil() then [EXU1 = Ms] else [case Ls of '|'(1:X 2:Lr) then [local ["EXU2","EXU3"] [EXU2 = X,local ["EXU4","EXU5"] [EXU4 = Lr,EXU5 = Ms,"Append" "EXU4" "EXU5" "EXU3"],EXU1 = '|'(1:EXU2 2:EXU3)]] else [skip]]],local ["EXU1","EXU2"] [EXU1 = 1,local ["EXU3","EXU4"] [EXU3 = 2,local ["EXU5","EXU6"] [EXU5 = 3,EXU6 = nil(),EXU4 = '|'(1:EXU5 2:EXU6)],EXU2 = '|'(1:EXU3 2:EXU4)],L1 = '|'(1:EXU1 2:EXU2)],local ["EXU1","EXU2"] [EXU1 = 4,local ["EXU3","EXU4"] [EXU3 = 5,local ["EXU5","EXU6"] [EXU5 = 6,EXU6 = nil(),EXU4 = '|'(1:EXU5 2:EXU6)],EXU2 = '|'(1:EXU3 2:EXU4)],L2 = '|'(1:EXU1 2:EXU2)],local ["EXU1","EXU2"] [EXU1 = L1,EXU2 = L2,"Append" "EXU1" "EXU2" "Out"],skip/BOut,skip/f,Reverse = proc {$ Xs EXU1} [case Xs of nil() then [EXU1 = nil()] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3"] [local ["EXU4"] [EXU4 = Xr,"Reverse" "EXU4" "EXU2"],local ["EXU4"] [EXU4 = X,local ["EXU5","EXU6"] [EXU5 = EXU4,EXU6 = nil(),EXU3 = '|'(1:EXU5 2:EXU6)]],"Append" "EXU2" "EXU3" "EXU1"]] else [skip]]],local ["EXU1"] [EXU1 = L1,"Reverse" "EXU1" "Out1"],skip/BOut1,skip/f]]

\*/

**Section 2B:**

/\*

\*\* Output from append.txt \*\*

LNew : '#'(1:35 2:36)

Store : ((36, 24, 28, 11, 33, 15), Unbound),

((35, 8, 31), '#'(1:17 2:18)),

((18, 22, 9, 30, 13, 23, 32, 14), '|'(1:25 2:26)),

((10, 34), '#'(1:23 2:24)),

((17, 29, 12), '|'(1:19 2:20)),

((27), 4),

((25), 3),

((26), '|'(1:27 2:28)),

((21), 2),

((19), 1),

((20), '|'(1:21 2:22)),

((16), '#'(1:35 2:36)),

((1), Primitive Operation),

((2), Primitive Operation),

((3), Primitive Operation),

((4), Primitive Operation),

((5), Primitive Operation),

((6), Primitive Operation),

((7), Primitive Operation)

Mutable Store: Empty

Current Environment : ("L1" -> 8, "End1" -> 9, "L2" -> 10, "End2" -> 11, "H1" -> 12, "T1" -> 13, "H2" -> 14, "T2" -> 15, "LNew" -> 16, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT" -> 5, "Mod" -> 6, "IntMultiply" -> 7)

Stack : "local [\"Reverse\",\"L1\",\"Out1\"] [Reverse = proc {$ Xs EXU1} [local [\"Y1\",\"ReverseD\"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local [\"EXU2\",\"EXU3\",\"EXU4\"] [EXU2 = Xr,EXU3 = Y1,local [\"EXU5\",\"EXU6\"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],\"ReverseD\" \"EXU2\" \"EXU3\" \"EXU4\"]] else [skip]]],local [\"EXU2\",\"EXU3\",\"EXU4\"] [EXU2 = Xs,EXU3 = Y1,EXU4 = nil(),\"ReverseD\" \"EXU2\" \"EXU3\" \"EXU4\"],EXU1 = Y1]],local [\"EXU1\",\"EXU2\"] [EXU1 = 1,local [\"EXU3\",\"EXU4\"] [EXU3 = 2,local [\"EXU5\",\"EXU6\"] [EXU5 = 3,local [\"EXU7\",\"EXU8\"] [EXU7 = 4,EXU8 = nil(),EXU6 = '|'(1:EXU7 2:EXU8)],EXU4 = '|'(1:EXU5 2:EXU6)],EXU2 = '|'(1:EXU3 2:EXU4)],L1 = '|'(1:EXU1 2:EXU2)],local [\"EXU1\"] [EXU1 = L1,\"Reverse\" \"EXU1\" \"Out1\"],skip/BOut1,skip/f]"

Out1 : [ 4 3 2 1 ]

Store : ((39, 70, 65, 60, 55, 52, 49, 71), '|'(1:72 2:73)),

((73, 66), '|'(1:67 2:68)),

((72, 46), 4),

((69, 47), nil()),

((68, 61), '|'(1:62 2:63)),

((67, 44), 3),

((64, 45), '|'(1:46 2:47)),

((63, 56), '|'(1:57 2:58)),

((62, 42), 2),

((59, 43), '|'(1:44 2:45)),

((58, 53), nil()),

((57, 40), 1),

((54, 41), '|'(1:42 2:43)),

((51, 48, 38), '|'(1:40 2:41)),

((50), proc(["Xs","Y1","Y"],[case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],[("ReverseD",50)])),

((37), proc(["Xs","EXU1"],[local ["Y1","ReverseD"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],local ["EXU2","EXU3","EXU4"] [EXU2 = Xs,EXU3 = Y1,EXU4 = nil(),"ReverseD" "EXU2" "EXU3" "EXU4"],EXU1 = Y1]],[])),

((36, 24, 28, 11, 33, 15), Unbound),

((35, 8, 31), '#'(1:17 2:18)),

((18, 22, 9, 30, 13, 23, 32, 14), '|'(1:25 2:26)),

((10, 34), '#'(1:23 2:24)),

((17, 29, 12), '|'(1:19 2:20)),

((27), 4),

((25), 3),

((26), '|'(1:27 2:28)),

((21), 2),

((19), 1),

((20), '|'(1:21 2:22)),

((16), '#'(1:35 2:36)),

((1), Primitive Operation),

((2), Primitive Operation),

((3), Primitive Operation),

((4), Primitive Operation),

((5), Primitive Operation),

((6), Primitive Operation),

((7), Primitive Operation)

Mutable Store: Empty

Current Environment : ("Reverse" -> 37, "L1" -> 38, "Out1" -> 39, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT" -> 5, "Mod" -> 6, "IntMultiply" -> 7)

Stack : ""

\*\* Output from append.out \*\*

[local ["L1","End1","L2","End2","H1","T1","H2","T2","LNew"] [local ["EXU1","EXU2"] [local ["EXU3","EXU4"] [EXU3 = 1,local ["EXU5","EXU6"] [EXU5 = 2,EXU6 = End1,EXU4 = '|'(1:EXU5 2:EXU6)],EXU1 = '|'(1:EXU3 2:EXU4)],EXU2 = End1,L1 = '#'(1:EXU1 2:EXU2)],local ["EXU1","EXU2"] [local ["EXU3","EXU4"] [EXU3 = 3,local ["EXU5","EXU6"] [EXU5 = 4,EXU6 = End2,EXU4 = '|'(1:EXU5 2:EXU6)],EXU1 = '|'(1:EXU3 2:EXU4)],EXU2 = End2,L2 = '#'(1:EXU1 2:EXU2)],local ["EXU1","EXU2"] [EXU1 = H1,EXU2 = T1,L1 = '#'(1:EXU1 2:EXU2)],local ["EXU1","EXU2"] [EXU1 = H2,EXU2 = T2,L2 = '#'(1:EXU1 2:EXU2)],T1 = H2,local ["EXU1","EXU2"] [EXU1 = L1,EXU2 = T2,LNew = '#'(1:EXU1 2:EXU2)],skip/BLNew,skip/f],local ["Reverse","L1","Out1"] [Reverse = proc {$ Xs EXU1} [local ["Y1","ReverseD"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],local ["EXU2","EXU3","EXU4"] [EXU2 = Xs,EXU3 = Y1,EXU4 = nil(),"ReverseD" "EXU2" "EXU3" "EXU4"],EXU1 = Y1]],local ["EXU1","EXU2"] [EXU1 = 1,local ["EXU3","EXU4"] [EXU3 = 2,local ["EXU5","EXU6"] [EXU5 = 3,local ["EXU7","EXU8"] [EXU7 = 4,EXU8 = nil(),EXU6 = '|'(1:EXU7 2:EXU8)],EXU4 = '|'(1:EXU5 2:EXU6)],EXU2 = '|'(1:EXU3 2:EXU4)],L1 = '|'(1:EXU1 2:EXU2)],local ["EXU1"] [EXU1 = L1,"Reverse" "EXU1" "Out1"],skip/BOut1,skip/f]]

\*/

**Section 2C:**

**// \*\*** New Code inserted in append\_diff \*\*

local L1N N LNew Reverse in

N = nil

Reverse = fun {$ Xs}

local ReverseD Y1 in

ReverseD = proc {$ Xs Y1 Y}

case Xs

of nil then Y1 = Y

[]'|'(1:X 2:Xr) then Z in

Z = (X|Y)

{ReverseD Xr Y1 Z}

end

end

{ReverseD Xs Y1 N}

Y1

end

end

L1N = (1|(2|(3|(4|nil))))

LNew = {Reverse L1N}

skip Browse LNew

skip Full

end

/\*

\*\* Output of updated append\_diff.txt \*\*

Out1 : [ 4 3 2 1 ]

Store : ((10, 41, 36, 31, 26, 23, 20, 42), '|'(1:43 2:44)),

((44, 37), '|'(1:38 2:39)),

((43, 17), 4),

((40, 18), nil()),

((39, 32), '|'(1:33 2:34)),

((38, 15), 3),

((35, 16), '|'(1:17 2:18)),

((34, 27), '|'(1:28 2:29)),

((33, 13), 2),

((30, 14), '|'(1:15 2:16)),

((29, 24), nil()),

((28, 11), 1),

((25, 12), '|'(1:13 2:14)),

((22, 19, 9), '|'(1:11 2:12)),

((21), proc(["Xs","Y1","Y"],[case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],[("ReverseD",21)])),

((8), proc(["Xs","EXU1"],[local ["Y1","ReverseD"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],local ["EXU2","EXU3","EXU4"] [EXU2 = Xs,EXU3 = Y1,EXU4 = nil(),"ReverseD" "EXU2" "EXU3" "EXU4"],EXU1 = Y1]],[])),

((1), Primitive Operation),

((2), Primitive Operation),

((3), Primitive Operation),

((4), Primitive Operation),

((5), Primitive Operation),

((6), Primitive Operation),

((7), Primitive Operation)

Mutable Store: Empty

Current Environment : ("Reverse" -> 8, "L1" -> 9, "Out1" -> 10, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT" -> 5, "Mod" -> 6, "IntMultiply" -> 7)

Stack : "local [\"L1N\",\"N\",\"LNew\",\"Reverse\"] [N = nil(),Reverse = proc {$ Xs EXU1} [local [\"ReverseD\",\"Y1\"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local [\"Z\"] [local [\"EXU2\",\"EXU3\"] [EXU2 = X,EXU3 = Y,Z = '|'(1:EXU2 2:EXU3)],local [\"EXU2\",\"EXU3\",\"EXU4\"] [EXU2 = Xr,EXU3 = Y1,EXU4 = Z,\"ReverseD\" \"EXU2\" \"EXU3\" \"EXU4\"]]] else [skip]]],local [\"EXU2\",\"EXU3\",\"EXU4\"] [EXU2 = Xs,EXU3 = Y1,EXU4 = N,\"ReverseD\" \"EXU2\" \"EXU3\" \"EXU4\"],EXU1 = Y1]],local [\"EXU1\",\"EXU2\"] [EXU1 = 1,local [\"EXU3\",\"EXU4\"] [EXU3 = 2,local [\"EXU5\",\"EXU6\"] [EXU5 = 3,local [\"EXU7\",\"EXU8\"] [EXU7 = 4,EXU8 = nil(),EXU6 = '|'(1:EXU7 2:EXU8)],EXU4 = '|'(1:EXU5 2:EXU6)],EXU2 = '|'(1:EXU3 2:EXU4)],L1N = '|'(1:EXU1 2:EXU2)],local [\"EXU1\"] [EXU1 = L1N,\"Reverse\" \"EXU1\" \"LNew\"],skip/BLNew,skip/f]"

LNew : [ 4 3 2 1 ]

Store : ((47, 85, 79, 73, 67, 61, 59, 86, 81), '|'(1:82 2:83)),

((84, 56), nil()),

((83, 80, 75), '|'(1:76 2:77)),

((82, 55), 4),

((78, 54), '|'(1:55 2:56)),

((77, 74, 69), '|'(1:70 2:71)),

((76, 53), 3),

((72, 52), '|'(1:53 2:54)),

((71, 68, 63), '|'(1:64 2:65)),

((70, 51), 2),

((66, 50), '|'(1:51 2:52)),

((65, 62, 46), nil()),

((64, 49), 1),

((60, 57, 45), '|'(1:49 2:50)),

((58), proc(["Xs","Y1","Y"],[case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["Z"] [local ["EXU2","EXU3"] [EXU2 = X,EXU3 = Y,Z = '|'(1:EXU2 2:EXU3)],local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,EXU4 = Z,"ReverseD" "EXU2" "EXU3" "EXU4"]]] else [skip]]],[("ReverseD",58)])),

((48), proc(["Xs","EXU1"],[local ["ReverseD","Y1"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["Z"] [local ["EXU2","EXU3"] [EXU2 = X,EXU3 = Y,Z = '|'(1:EXU2 2:EXU3)],local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,EXU4 = Z,"ReverseD" "EXU2" "EXU3" "EXU4"]]] else [skip]]],local ["EXU2","EXU3","EXU4"] [EXU2 = Xs,EXU3 = Y1,EXU4 = N,"ReverseD" "EXU2" "EXU3" "EXU4"],EXU1 = Y1]],[("N",46)])),

((10, 41, 36, 31, 26, 23, 20, 42), '|'(1:43 2:44)),

((44, 37), '|'(1:38 2:39)),

((43, 17), 4),

((40, 18), nil()),

((39, 32), '|'(1:33 2:34)),

((38, 15), 3),

((35, 16), '|'(1:17 2:18)),

((34, 27), '|'(1:28 2:29)),

((33, 13), 2),

((30, 14), '|'(1:15 2:16)),

((29, 24), nil()),

((28, 11), 1),

((25, 12), '|'(1:13 2:14)),

((22, 19, 9), '|'(1:11 2:12)),

((21), proc(["Xs","Y1","Y"],[case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],[("ReverseD",21)])),

((8), proc(["Xs","EXU1"],[local ["Y1","ReverseD"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],local ["EXU2","EXU3","EXU4"] [EXU2 = Xs,EXU3 = Y1,EXU4 = nil(),"ReverseD" "EXU2" "EXU3" "EXU4"],EXU1 = Y1]],[])),

((1), Primitive Operation),

((2), Primitive Operation),

((3), Primitive Operation),

((4), Primitive Operation),

((5), Primitive Operation),

((6), Primitive Operation),

((7), Primitive Operation)

Mutable Store: Empty

Current Environment : ("L1N" -> 45, "N" -> 46, "LNew" -> 47, "Reverse" -> 48, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT" -> 5, "Mod" -> 6, "IntMultiply" -> 7)

Stack : ""

\*\* New append\_diff.out file \*\*

[local ["Reverse","L1","Out1"] [Reverse = proc {$ Xs EXU1} [local ["Y1","ReverseD"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],local ["EXU2","EXU3","EXU4"] [EXU2 = Xs,EXU3 = Y1,EXU4 = nil(),"ReverseD" "EXU2" "EXU3" "EXU4"],EXU1 = Y1]],local ["EXU1","EXU2"] [EXU1 = 1,local ["EXU3","EXU4"] [EXU3 = 2,local ["EXU5","EXU6"] [EXU5 = 3,local ["EXU7","EXU8"] [EXU7 = 4,EXU8 = nil(),EXU6 = '|'(1:EXU7 2:EXU8)],EXU4 = '|'(1:EXU5 2:EXU6)],EXU2 = '|'(1:EXU3 2:EXU4)],L1 = '|'(1:EXU1 2:EXU2)],local ["EXU1"] [EXU1 = L1,"Reverse" "EXU1" "Out1"],skip/BOut1,skip/f],local ["L1N","N","LNew","Reverse"] [N = nil(),Reverse = proc {$ Xs EXU1} [local ["ReverseD","Y1"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["Z"] [local ["EXU2","EXU3"] [EXU2 = X,EXU3 = Y,Z = '|'(1:EXU2 2:EXU3)],local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,EXU4 = Z,"ReverseD" "EXU2" "EXU3" "EXU4"]]] else [skip]]],local ["EXU2","EXU3","EXU4"] [EXU2 = Xs,EXU3 = Y1,EXU4 = N,"ReverseD" "EXU2" "EXU3" "EXU4"],EXU1 = Y1]],local ["EXU1","EXU2"] [EXU1 = 1,local ["EXU3","EXU4"] [EXU3 = 2,local ["EXU5","EXU6"] [EXU5 = 3,local ["EXU7","EXU8"] [EXU7 = 4,EXU8 = nil(),EXU6 = '|'(1:EXU7 2:EXU8)],EXU4 = '|'(1:EXU5 2:EXU6)],EXU2 = '|'(1:EXU3 2:EXU4)],L1N = '|'(1:EXU1 2:EXU2)],local ["EXU1"] [EXU1 = L1N,"Reverse" "EXU1" "LNew"],skip/BLNew,skip/f]]

\*/

I counted 23 cons for section 2A, append, and as for section 2B, append\_diff I counted 18 cons. Part A has more cons due to it being a recursive method as compared to Part B that uses a iterative method. The iterative method would only go through the append list once whereas the recursive method would go through the list twice in order to reverse the list after outputting the non-reverse order first.