Ren Hao Wong
CSCI 117 Lab 5

Part 1

<Terminal output>:

```
A : false()

A : 2

B : 3

R : 4

A : rdc(1:40 2:41 3:42)

B : 4

Store : ((47), 3),
((48), 4),
((45), 5),
((46), -1),
((44, 43, 41, 39), 4),
((40), 4),
((42), '#'(1:43 2:44)),
((38), rdc(1:40 2:41 3:42)),
((36, 37), 4),
((35), proc(["X","EXU1"],[EXU1 = X],[])),
((33), 5),
((34), 2),
((32), 3),
((31), Unbound),
((29), 1),
((30), 1),
((28), true()),
((26, 27, 24, 22), tree(1:25 2:26)),
((25, 23), 3),
((20), Unbound),
((21), Unbound),
((18), 3),
((19), 1),
((16, 13, 11), 2),
((17), 2),
((15), true()),
((14), 1),
((12), false()),
((10), true()),
((8), false()),
((9), Unbound),
((1), Primitive Operation),
((2), Primitive Operation),
((3), Primitive Operation),
((4), Primitive Operation),
((5), Primitive Operation),
((6), Primitive Operation),
((7), Primitive Operation)
```

sugar2kern.txt	kernel.txt
1) nested if, nested case	
The kernel translation is for the most part accurate, except for the missing introductions of temporary variables for each of the if-statement conditions	
`local ["A","B"] [`	`local A B in`

```
	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
			]
		]
	]
]
```

```
    A = false

    if true then
        skip Browse A
    else
        if B then
            skip Basic
        else
            skip Basic
        end
    end

    case A of tree() then
        skip Basic
    else
        case A of false then
            skip Basic
        else
            case A of true then
                skip Basic
            else
                skip Basic
            end
        end
    end
end
```

2) more expressions

For the first if-statement, the kernel translation is missing the else-statement. It also did not correctly replicate the level of temporary variables. The second if-statement also has incorrect temporary variables introduction.

```
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 A in
    A = 2
    local One IsOne in
        One = 1
        {Eq A One IsOne}
        if IsOne then
            skip Basic
        end
    end

    local Three One Difference IsEqualToDifference
in
        Three = 3
        One = 1
        {IntMinus Three One Difference}
        {Eq A Difference IsEqualToDifference}
        if IsEqualToDifference then
            skip Browse A
        else
            skip Basic
        end
    end
end
```

3) "in" declaration

The kernel translation got the declaration of variable T wrong by having it declared in the first line, but instead should have been a nested local. It also inaccurately translated the further nested variable T and its unification with tree(1:A 2:B). The condition variable for the if-statement should have been declared one level above the condition arguments, whereas the condition arguments also need one more variable declared even if both arguments contain the same value. Like the variable T in the first line, B also should have been nested instead of being declared alongside Z.

```
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 T X Y in
    T = tree(1:3 2:T)
    local T in
       local A B in
              T = tree(1:A 2:B)
       end
        local One IsEqual in
            One = 1
            {Eq One One IsEqual}
            if IsEqual then
                local B Z in
                    local Five Two in
                        Five = 5
                        Two = 2
                        {IntMinus Five Two B}
                    end
                    skip Browse B
                end
            end
        end
    end
end
```

4) expressions in place of statements

The kernel translation is for the most part accurate, except for the missing introductions of a temporary variable to wrap around the value 4 before passing it as argument into Fun.

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

```
local Fun R in
    Fun = proc {$ X Out}
        Out = X
    end

    {Fun 4 R}
    skip Browse R
end
```

5) Bind fun

The kernel translation is inaccurate for the first section, where a temporary variable for EXU2 is missing. It is also lacking temporary variables for each of the members in the tuple declaration. The translation for the second section however has been accurate.

```
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 A B in
    skip Basic
    local Four Pattern in
        Four = 4
        Pattern = '#'(1:B 2:B)
        A = rdc(1:Four 2:B 3:Pattern)
    end

    local Five Difference in
        Five = 5
        local Three Four in
```

```
        ],
        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
]
```

```
            Three = 3
            Four = 4
            {IntMinus Three Four Difference}
        end
        {IntPlus Five Difference B}
    end
    skip Browse A
    skip Browse B
    skip Store
end
```

<kernel.txt>

```
// 1) nested if, nested case

local A B in
    A = false

    if true then                    // expression in if-condition
        skip Browse A
    else
        if B then                   // elsif can be repeated 0 or more times
            skip Basic
        else                            // else is optional
            skip Basic
        end
    end

    case A of tree() then
        skip Basic
    else
        case A of false then
            skip Basic     // nesting symbol is [] followed by record
        else
            case A of true then
                skip Basic
            else                            // else is optional
                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 in
    A = 2
    local One IsOne in
        One = 1
        {Eq A One IsOne}
        if IsOne then
            skip Basic
        end
    end

    local Three One Difference IsEqualToDifference in
        Three = 3
        One = 1
        {IntMinus Three One Difference}
        {Eq A Difference IsEqualToDifference}
        if IsEqualToDifference then
            skip Browse A
        else
```

```
            skip Basic
        end
    end
end


// 3) "in" declaration

local T X Y in
    T = tree(1:3 2:T)
    local T in
        local A B in
            T = tree(1:A 2:B)
        end
        local One IsEqual in
            One = 1
            {Eq One One IsEqual}
            if IsEqual then
                local B Z in
                    local Five Two in
                        Five = 5
                        Two = 2
                        {IntMinus Five Two B}
                    end
                    skip Browse B
                end
            end
        end
    end
end


// 4) expressions in place of statements

local Fun R in
    Fun = proc {$ X Out}
        Out = X
    end

    {Fun 4 R}
    skip Browse R
end


// 5) Bind fun

local A B in
    skip Basic
    local Four Pattern in
        Four = 4
        Pattern = '#'(1:B 2:B)
        A = rdc(1:Four 2:B 3:Pattern)
    end

    local Five Difference in
        Five = 5
        local Three Four in
            Three = 3
            Four = 4
            {IntMinus Three Four Difference}
        end
        {IntPlus Five Difference B}
    end
    skip Browse A
    skip Browse B
    skip Store
end
```

Part 2

A)

<Terminal output>:

```
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"

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 : ""
```

Lists are stored in the memory locations in the format: '|'(1:A 2:B)
where "A" denotes the value at the root of a record tree, and "B" denotes the location of the following subtree.
A location at "B" may be bounded to nil() if the value at "A" is the final element in a record.

In "append.txt"'s first algorithm, "L1" and "L2" were declared as lists that will be appended together using the "Append()" function. According to the memory storage, the appended list is constructed on top of the existing "L2" list. Recursively, a new subtree is reconstructed for every element in "L1", with "Out" pointing to the location of the root in the appended list. When arriving at the final element of "L1", the second argument of its record is not terminated with nil(), but points to the location of the "L2" list to complete the append operation. For every recursion, new locations are introduced in the memory to bind to the recursion's current subtree in the "L1" and the "L2" lists.
Following the location of "Out" at 11, it is bounded to a record with locations 28 and 29, where 28 is bounded to the value 1. Location 29 is bounded to another record with locations 32 and 33, where 32 is bounded to the value 2. Location 33 then binds to another record with locations 36 and 37, where 36 is bounded to the value 3, and finally 37 is bounded to the record "L2".
Currently, the stack contains unexecuted operations for the next algorithm.

In the second algorithm, "L1" is again used with its original value intact because the append function does not modify its input lists. To create a reversed "L1" list for "Out1" using the "Reverse()" function, a new list had to be created. This function is an expensive call because a lot of records are created in the memory but then forgotten during the "Append()" function, which reconstructs its first argument to append to the front of its second argument. From the memory storage, we can notice that the location for "Out1" at 13, points to a record with arguments at locations 63 and 64. This indicates that a lot of locations were used during the recursive reverse operation to have a big increase in the location index of around 30 spots. During each recursion, a new location is used for binding to a section of the list in the process, some other locations to bind to each step in reconstructing the list for appending, and another location for the newly created singleton list as the second argument of "Append()".

B)

<Terminal output>:

```
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 : ""
```

In a difference list, rather than terminating a record with nil(), it is terminated with an unbound variable instead. The list is stored in a pair, with the first argument being the list itself, and the second argument being the unbound termination variable. The pair structure allows easy pattern matching to reference the tail of the list using a different variable. To append the difference list, the unbound termination variable of the first list will be bounded to the second list. This is a very efficient operation that does not reconstruct either list, but also creates an interesting structure for the data. On one hand, the first argument of the appended list is a continuous list tree connecting the first element to the last element and then its unbound termination variable. On the other hand, the pair structure created for pattern matching previously still persists, so there are still other pair variables with a second argument that is bounded to a subtree from the appended list.

In the iterative reverse algorithm, a helper function is used to provide an accumulator for building the reversed list in an O(n) complexity without constructing extra lists. The variable "Y1" remains unbound until the very end when it is ready to bind with the completed reverse list and serves as the return variable of the reverse function.

C)

<append.txt>:

```
local Append L1 L2 Out Reverse Out1 in

   // Append function on p 133 (modified for hoz)
   fun {Append Ls Ms}
      case Ls
      of nil then Ms
      [] '|'(1:X 2:Lr) then (X|{Append Lr Ms})
      end
   end

   L1 = (1|(2|(3|nil)))
   L2 = (4|(5|(6|nil)))

   Out = {Append L1 L2}
   skip Browse Out
   skip Full

   // O(n^2) Reverse function on p 135 (modified for hoz):
   fun {Reverse Xs}
      case Xs
      of nil then nil
      [] '|'(1:X 2:Xr) then
         {Append {Reverse Xr} [X]}
      end
   end

   Out1 = {Reverse Out}
   skip Browse Out1
   skip Full
end
```

<Terminal output (Reverse only)>:

```
Out1 : [ 6  5  4  3  2  1 ]

Store : ((134, 136, 132, 128, 124, 120, 42), '|'(1:115 2:116)),
((135, 97), nil()),
((133, 96, 95, 32, 16), 2),
((131, 111, 113, 109, 105, 101, 45), '|'(1:96 2:97)),
((129, 110, 81, 80, 36, 18), 3),
((130), '|'(1:133 2:134)),
((127, 107), '|'(1:110 2:111)),
((125, 106, 91, 70, 69, 20), 4),
((126), '|'(1:129 2:130)),
((123, 103), '|'(1:106 2:107)),
((121, 102, 87, 76, 63, 62, 22), 5),
((122), '|'(1:125 2:126)),
((119, 99), '|'(1:102 2:103)),
((117, 98, 83, 72, 65, 60, 59, 24), 6),
((118), '|'(1:121 2:122)),
((115, 114, 28, 14), 1),
((116), nil()),
```

```
((112, 82), nil()),
((108, 92, 94, 90, 86, 48), '|'(1:81 2:82)),
((104, 88), '|'(1:91 2:92)),
((100, 84), '|'(1:87 2:88)),
((93, 71), nil()),
((89, 77, 79, 75, 51), '|'(1:70 2:71)),
((85, 73), '|'(1:76 2:77)),
((78, 64), nil()),
((74, 66, 68, 54), '|'(1:63 2:64)),
((67, 61), nil()),
((53, 57), '|'(1:60 2:61)),
((58, 25), nil()),
((56), nil()),
((55, 23), '|'(1:24 2:25)),
((52, 21), '|'(1:22 2:23)),
((50), '|'(1:65 2:66)),
((49, 37, 39, 35, 31, 27, 10), '|'(1:20 2:21)),
((47), '|'(1:72 2:73)),
((46, 33), '|'(1:36 2:37)),
((44), '|'(1:83 2:84)),
((43, 29), '|'(1:32 2:33)),
((41), '|'(1:98 2:99)),
((40, 11), '|'(1:28 2:29)),
((38, 19), nil()),
((34, 17), '|'(1:18 2:19)),
((30, 15), '|'(1:16 2:17)),
((26, 9), '|'(1:14 2:15)),
((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)])),
((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), '|'(1:117 2:118)),
((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 : ""
```

<append_diff.txt>:

```
// Append example with difference lists
local L1 End1 L2 End2 H1 T1 H2 T2 LNew in
   L1 = ((1|(2|End1)) # End1)        // List [1,2] as a difference list
   L2 = ((3|(4|End2)) # End2)        // List [3,4] as a difference list

   L1 = (H1 # T1)                    // Pattern match, name head and tail
   L2 = (H2 # T2)                    // Pattern match, name head and tail
   T1 = H2                           // Bind/unify tail of L1 with head of L2

   LNew = (L1 # T2)                  // Build a new difference list

   skip Browse LNew
   skip Full
end

// Testing iterative Reverse function
local Reverse L1 Out1 in

   // O(n) version of Reverse on p 148 (modified for hoz):
```

```
   fun {Reverse Xs} Y1 ReverseD in
      proc {ReverseD Xs Y1 Y}
         case Xs
         of nil then Y1 = Y
         [] '|'(1:X 2:Xr) then {ReverseD Xr Y1 (X|Y)}
         end
      end
      {ReverseD Xs Y1 nil}
      Y1
   end

   L1 = (1|(2|(3|(4|(5|(6|nil))))))
   Out1 = {Reverse L1}
   skip Browse Out1
   skip Full
end
```

<Terminal output (Reverse only)>:

```
Out1 : [ 6  5  4  3  2  1 ]

Store : ((39, 84, 79, 74, 69, 64, 59, 56, 53, 85), '|'(1:86 2:87)),
((87, 80), '|'(1:81 2:82)),
((86, 50), 6),
((83, 51), nil()),
((82, 75), '|'(1:76 2:77)),
((81, 48), 5),
((78, 49), '|'(1:50 2:51)),
((77, 70), '|'(1:71 2:72)),
((76, 46), 4),
((73, 47), '|'(1:48 2:49)),
((72, 65), '|'(1:66 2:67)),
((71, 44), 3),
((68, 45), '|'(1:46 2:47)),
((67, 60), '|'(1:61 2:62)),
((66, 42), 2),
((63, 43), '|'(1:44 2:45)),
((62, 57), nil()),
((61, 40), 1),
((58, 41), '|'(1:42 2:43)),
((55, 52, 38), '|'(1:40 2:41)),
((54), 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",54)])),
((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 : ""
```

I counted 21 record constructors from "append.txt" but only 6 record constructors from "append_diff.txt". "append_diff.txt" reverses its list without unnecessary record constructions, and only had to construct as many times as the size of the list, which is 6. The recursive reverse from "append.txt" however had to construct a lot of temporary lists, adding up to 21 times for a list of 6 elements.