Madhuri Pyreddy
9/26/19

CSCI 117 Lab 5

Part 1:

```
// 1) nested if, nested case
local A B in
  A = false()
     local D1 in
       D1 = true
         if D1 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
     case A of true() then
      skip Basic
     else
      skip Basic
     end
    end
  end
end
// 2) expression in if condition
local A One Three in
 A = 2
 One = 1
 Three = 3
  local F1 in
```

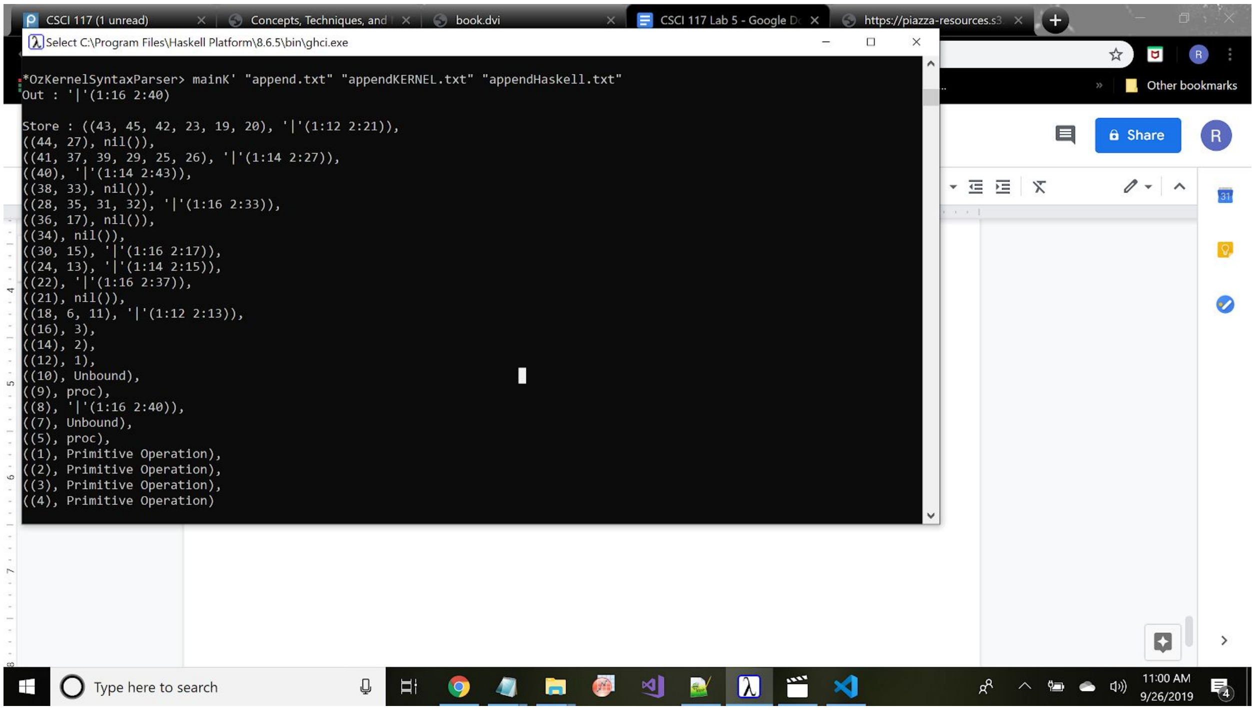
```
   {Eq A One F1}
   if F1 then
    skip Basic
   else
    skip Basic
   end
  end
  local In F3 in
   {IntMinus Three One In}
   {Eq A In F3}
   if F3 then
    skip Browse A
   else
    skip Basic
   end
  end
end

// 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
  end

// 4) expressions in place of statements
local Fun R in
 Fun = proc {$ X ProcOut0}
 ProcOut0 = 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
```

Explanation: What I noticed is that there are more local statements in the compiler “sugar2kern.txt” file than the translated “my_sugar2kern.txt” file. In the “ my_sugar2kern.txt” file, you can bind two variables in a local statement. For instance, in the “sugar2kern.txt” file, it does “local A in local B in”. Instead, the “my_sugar2kern.txt” file declares local statements like “local A B in”.   Also, you cannot have an “elseif” nested loops without putting “ends” in conditional statements in the “my_sugar2kern.txt” translated file. You have to do “if then else if then” and then add the right amount of closing “ends” to run the program. The new “my_sugar2kern.txt” file requires more “else” statements in pattern matching statements “case X of pattern1 then ___ else case X of pattern1 then____” and also requires right amount of “end” statements to run the program. The new translated file has more “ends” because it creates a new environment for every single variable and requires multiple “end” statements to close each variable at the end.

Part 2:

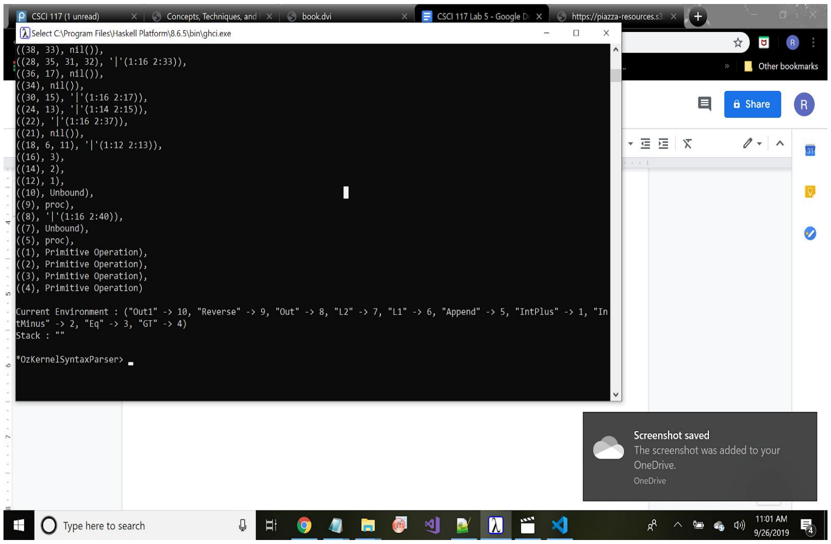
a. // Append function p 133

```
local Append L1 L2 Out Reverse Out1 in
        Append = fun {$ Ls Ms}
                case Ls
                of nil then Ms
                [] '|'(1:X 2:Lr) then Y in
                        Y = {Append Lr Ms}
                        //skip Full
                        (X|Y)
                end
        end

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

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

Reverse = fun{$ Xs}
case Xs of nil then nil
[] '|'(1:X 2:Xr) then Y in
        Y = (X|nil)
        {Append {Reverse Xr} Y}
     end
   end
   L1 = (1|(2|(3|(4|nil))))
   Out = {Reverse L1}
   skip Browse Out
   skip Full
end
```



CSCI 117 (1 unread)
Concepts, Techniques, and
book.dvi
CSCI 117 Lab 5 - Google D
https://piazza-resources.s3
Select C:\Program Files\Haskell Platform\8.6.5\bin\ghci.exe
Other bookmarks
Share
Type here to search
11:00 AM
9/26/2019

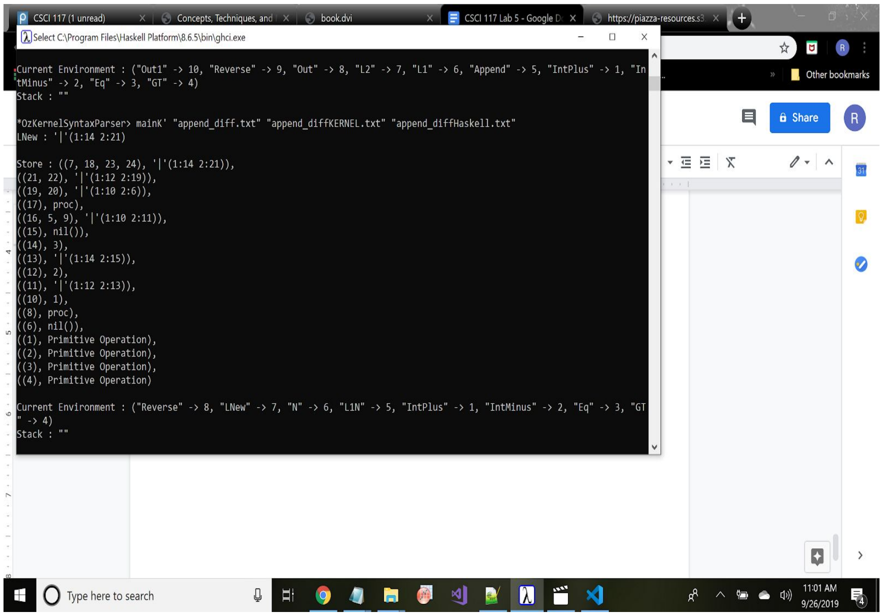
```
*OzKernelSyntaxParser> mainK' "append.txt" "appendKERNEL.txt" "appendHaskell.txt"
Out : '|'(1:16 2:40)

Store : ((43, 45, 42, 23, 19, 20), '|'(1:12 2:21)),
((44, 27), nil()),
((41, 37, 39, 29, 25, 26), '|'(1:14 2:27)),
((40), '|'(1:14 2:43)),
((38, 33), nil()),
((28, 35, 31, 32), '|'(1:16 2:33)),
((36, 17), nil()),
((34), nil()),
((30, 15), '|'(1:16 2:17)),
((24, 13), '|'(1:14 2:15)),
((22), '|'(1:16 2:37)),
((21), nil()),
((18, 6, 11), '|'(1:12 2:13)),
((16), 3),
((14), 2),
((12), 1),
((10), Unbound),
((9), proc),
((8), '|'(1:16 2:40)),
((7), Unbound),
((5), proc),
((1), Primitive Operation),
((2), Primitive Operation),
((3), Primitive Operation),
((4), Primitive Operation)
```


CSCI 117 (1 unread)
Concepts, Techniques, and
book.dvi
CSCI 117 Lab 5 - Google D
https://piazza-resources.s3
Select C:\Program Files\Haskell Platform\8.6.5\bin\ghci.exe
Other bookmarks
Share
Screenshot saved
The screenshot was added to your OneDrive.
OneDrive
Type here to search
11:01 AM
9/26/2019

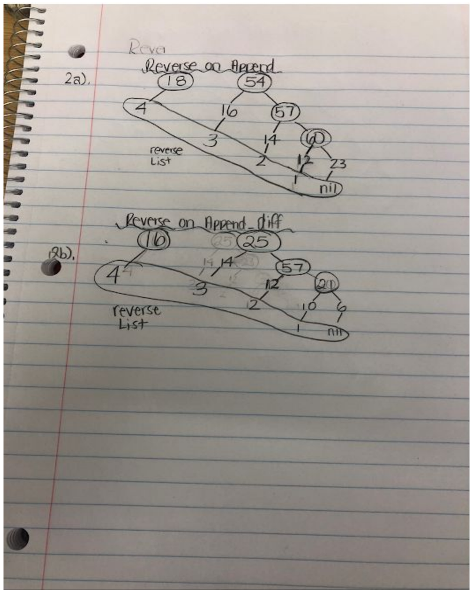
```
((38, 33), nil()),
((28, 35, 31, 32), '|'(1:16 2:33)),
((36, 17), nil()),
((34), nil()),
((30, 15), '|'(1:16 2:17)),
((24, 13), '|'(1:14 2:15)),
((22), '|'(1:16 2:37)),
((21), nil()),
((18, 6, 11), '|'(1:12 2:13)),
((16), 3),
((14), 2),
((12), 1),
((10), Unbound),
((9), proc),
((8), '|'(1:16 2:40)),
((7), Unbound),
((5), proc),
((1), Primitive Operation),
((2), Primitive Operation),
((3), Primitive Operation),
((4), Primitive Operation)

Current Environment : ("Out1" -> 10, "Reverse" -> 9, "Out" -> 8, "L2" -> 7, "L1" -> 6, "Append" -> 5, "IntPlus" -> 1, "In
tMinus" -> 2, "Eq" -> 3, "GT" -> 4)
Stack : ""

*OzKernelSyntaxParser> _
```

```
b. //local T1 T2 L1N L2N D1 D2 D1a D2a LNew in

        //L1N = ((1|(2|T1))#T1)
        //L2N = ((3|(4|T2))#T2)

        //L2N = (D1#D2)
        //L1N = (D1a#D2a)
        //T1 = D1

        //LNew = (D1a#D2)

        //skip Browse LNew
        //skip Full
//end
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
```


CSCI 117 (1 unread)
Concepts, Techniques, and
book.dvi
CSCI 117 Lab 5 - Google D
https://piazza-resources.s3
Select C:\Program Files\Haskell Platform\8.6.5\bin\ghci.exe
Other bookmarks
Share
Type here to search
11:01 AM
9/26/2019

```
Current Environment : ("Out1" -> 10, "Reverse" -> 9, "Out" -> 8, "L2" -> 7, "L1" -> 6, "Append" -> 5, "IntPlus" -> 1, "In
tMinus" -> 2, "Eq" -> 3, "GT" -> 4)
Stack : ""

*OzKernelSyntaxParser> mainK' "append_diff.txt" "append_diffKERNEL.txt" "append_diffHaskell.txt"
LNew : '|'(1:14 2:21)

Store : ((7, 18, 23, 24), '|'(1:14 2:21)),
((21, 22), '|'(1:12 2:19)),
((19, 20), '|'(1:10 2:6)),
((17), proc),
((16, 5, 9), '|'(1:10 2:11)),
((15), nil()),
((14), 3),
((13), '|'(1:14 2:15)),
((12), 2),
((11), '|'(1:12 2:13)),
((10), 1),
((8), proc),
((6), nil()),
((1), Primitive Operation),
((2), Primitive Operation),
((3), Primitive Operation),
((4), Primitive Operation)

Current Environment : ("Reverse" -> 8, "LNew" -> 7, "N" -> 6, "L1N" -> 5, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT
" -> 4)
Stack : ""
```



c. For the recursive reverse function in (a), there are 15 cons. For the iterative reverse function in (b), there are 6 cons. What I noticed is that there are more cons in the recursive reverse function than in the iterative reverse function. The reason is that the iterative reverse function time complexity is O(n). In other words, the iterative reverse function is going through the append list once, returning n time. However, the recursive reverse function is going to traverse through the append list twice, using the append function to reverse the list. Therefore, it will return O(n*n) times.