Madhuri Pyreddy
13 November 2019

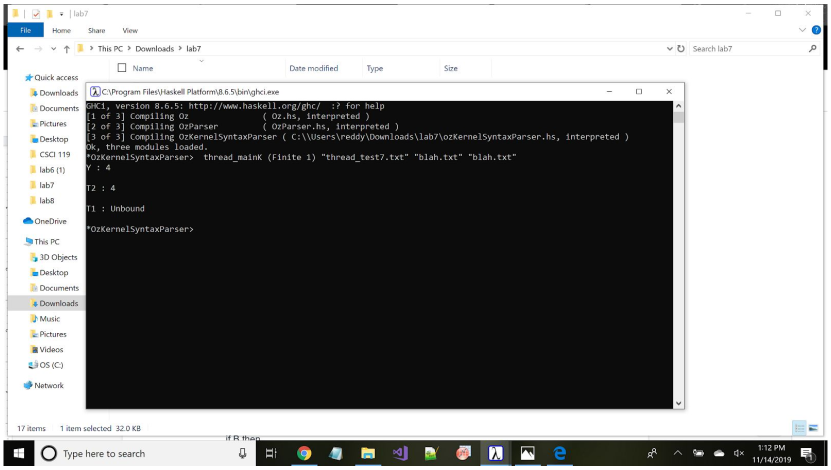
CSCI 117 Lab 12

Part 1:

1. Quantum specifies how many statements of a thread are executed before it moves to the next statement. It is the number that will create a suspension in the thread program because the suspension occurred due to the syntax sugar being converted to kernel syntax.

   Example of quantum:

```
local X Y T in

  thread Y = X end

  X = 4

  skip Browse Y

end

local T1 T2 in

 T2 = thread 4 end

 T1 = thread (5+4) end

 skip Browse T2

 skip Browse T1

end
```

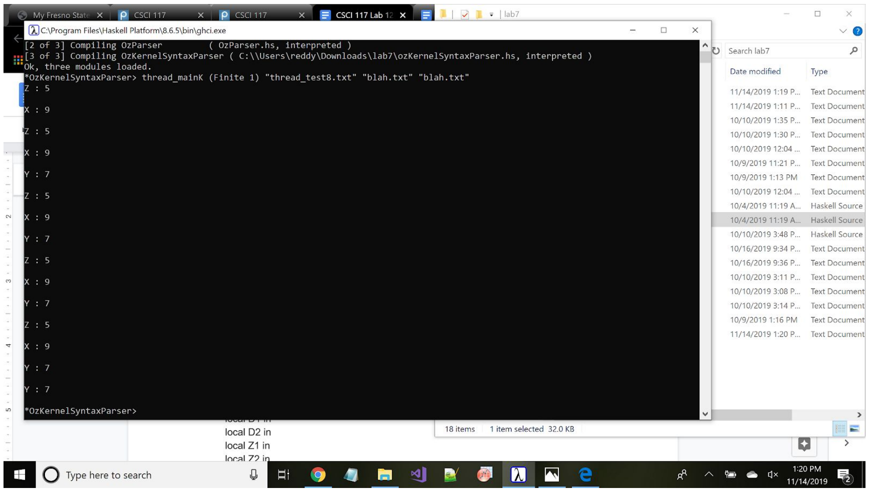


```
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
[1 of 3] Compiling Oz               ( Oz.hs, interpreted )
[2 of 3] Compiling OzParser         ( OzParser.hs, interpreted )
[3 of 3] Compiling OzKernelSyntaxParser ( C:\\Users\reddy\Downloads\lab7\ozKernelSyntaxParser.hs, interpreted )
Ok, three modules loaded.
*OzKernelSyntaxParser>  thread_mainK (Finite 1) "thread_test7.txt" "blah.txt" "blah.txt"
Y : 4

T2 : 4

T1 : Unbound

*OzKernelSyntaxParser>
```

Thread creation executes the statement and suspends immediately in the case statement since the list is unbound. It allows us to run parts of the code at the same time.

Examples of thread creation:

```
local Z in

  Z = 5

  thread local X in

    X = 9

    skip Browse X

    skip Browse X

    skip Browse X

    skip Browse X

    skip Browse X

    end

  end

  thread local Y in
  Y = 7
  skip Browse Y
  skip Browse Y
  skip Browse Y
  skip Browse Y
  skip Browse Y
  end
 end
 skip Browse Z
 skip Browse Z
 skip Browse Z
 skip Browse Z
 skip Browse Z
end
```



```
[2 of 3] Compiling OzParser         ( OzParser.hs, interpreted )
[3 of 3] Compiling OzKernelSyntaxParser ( C:\\Users\reddy\Downloads\lab7\ozKernelSyntaxParser.hs, interpreted )
Ok, three modules loaded.
*OzKernelSyntaxParser> thread_mainK (Finite 1) "thread_test8.txt" "blah.txt" "blah.txt"
Z : 5

X : 9

Z : 5

X : 9

Y : 7

Z : 5

X : 9

Y : 7

Z : 5

X : 9

Y : 7

Z : 5

X : 9

Y : 7

Y : 7

*OzKernelSyntaxParser>
```

Date modified	Type
11/14/2019 1:19 P...	Text Document
11/14/2019 1:11 P...	Text Document
10/10/2019 1:35 P...	Text Document
10/10/2019 1:30 P...	Text Document
10/10/2019 12:04 ...	Text Document
10/9/2019 11:21 P...	Text Document
10/9/2019 1:13 PM	Text Document
10/10/2019 12:04 ...	Text Document
10/4/2019 11:19 A...	Haskell Source
10/4/2019 11:19 A...	Haskell Source
10/10/2019 3:48 P...	Haskell Source
10/16/2019 9:34 P...	Text Document
10/16/2019 9:36 P...	Text Document
10/10/2019 3:11 P...	Text Document
10/10/2019 3:08 P...	Text Document
10/10/2019 3:14 P...	Text Document
10/9/2019 1:16 PM	Text Document
11/14/2019 1:20 P...	Text Document

18 items   1 item selected 32.0 KB

```
local D2 in
local Z1 in
local Z2 in
```

Kernel syntax translations (in relation to quantum) are basically converted from sugar syntax to kernel syntax.

Example of kernel syntax translations:

```
local X in
local Y in
local T in
  thread
    Y = X
  end
  X = 4
  skip Browse Y
end end end

local T1 in
local T2 in
  thread
    T2 = 4
    end
  thread
    local UniqueName11 in
    local UniqueName21 in
      UniqueName11 = 5
      UniqueName21 = 4
     {IntPlus UniqueName11 UniqueName21 T1}
    end
   end
    end
  skip Browse T2
  skip Browse T1
end end
```

Interleavings switches the threads when it is done translating the Oz code from sugar syntax to kernel syntax.

Example of interleavings:

```
local B B1 in


  thread         // 1


   B=true        // 2


   B1=false      // 3
  end
  thread           // 4
   B1=true        // 5
    B=false        // 6
  end
 thread           // 7
   if B then       // 8
     skip Browse B // 9
   end
 end
  if B1 then       // 10
    skip Browse B   // 11
  end
end
```

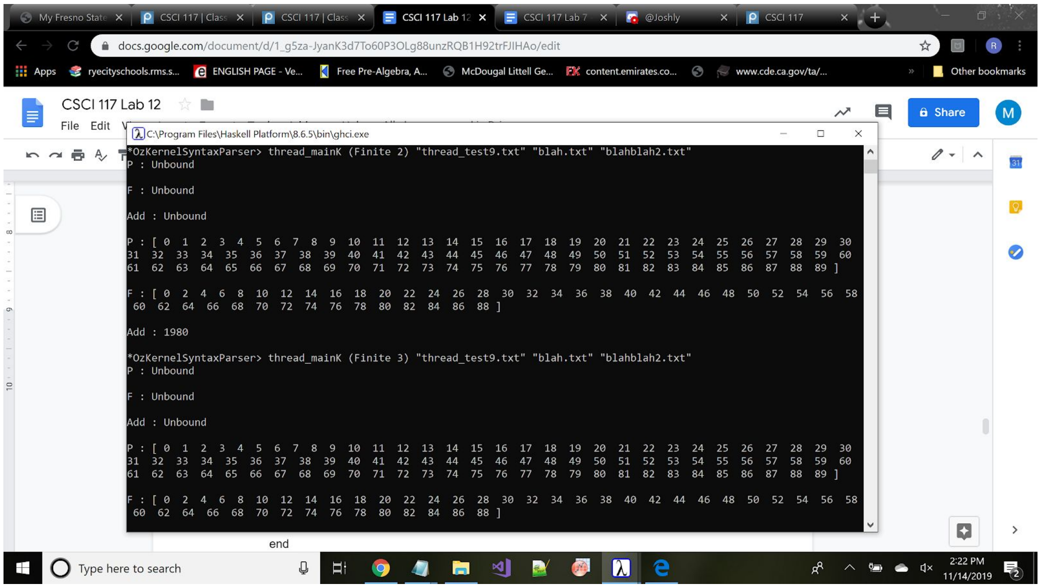
Interleavings: (1,4,7,8,2,6), (1,2,3,7,8,9,4,5), (1,4,5,6,7,8,9), (1,2,3,4,10,11,5), (1,4,7,3,5)

Part 2:

2. local Producer Consumer EvenFilter Filter N L P F Add in

```
   Producer = proc {$ N Limit Out}
  if (N<Limit) then T N1 in
    Out = (N|T)
    N1 = (N + 1)
    {Producer N1 Limit T}
  else Out = nil
  end
end
```

```
 EvenFilter = proc {$ P Out}
   Filter = fun {$ O1 T1}
       case O1 of nil then T1
            [] '|'(1:H 2:T) then S in
                if ((H mod 2) == 1) then
                      S = {Filter T T1}
                         S
                         else
                            S = {Filter T T1}
                            (H|S)
                        end
                     end
                  end
                  Out = {Filter P nil}
              end
Consumer = fun {$ P}
   case P of nil then 0
        [] '|'(1:H 2:T) then
            (H + {Consumer T})
         end
end
//Example Testing
  N = 0
  L = 90
  thread {Producer N L P}
     skip Browse P
  end
  thread {EvenFilter P F}
     skip Browse F
   end
  thread Add = {Consumer F}
      skip Browse Add
  end
      skip Browse P
      skip Browse F
      skip Browse Add
end
```



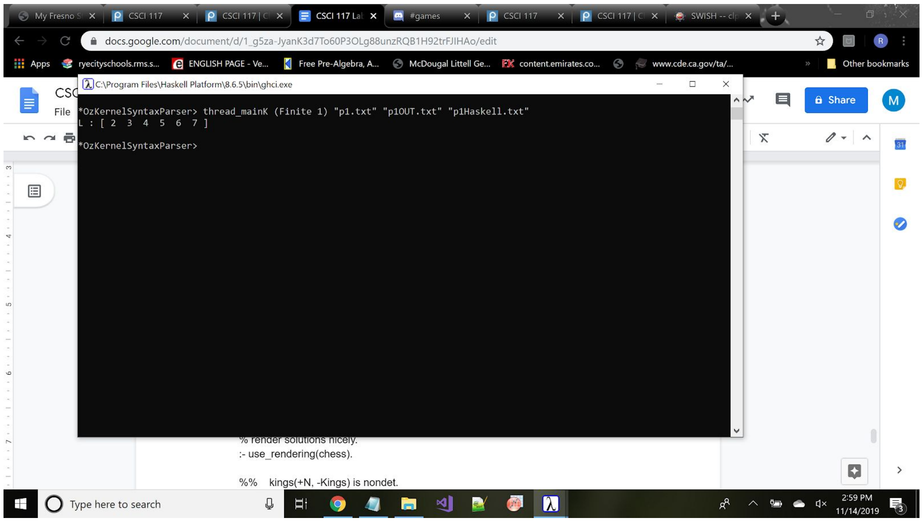
```
*OzKernelSyntaxParser> thread_mainK (Finite 2) "thread_test9.txt" "blah.txt" "blahblah2.txt"
P : Unbound

F : Unbound

Add : Unbound

P : [ 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89 ]

F : [ 0  2  4  6  8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38  40  42  44  46  48  50  52  54  56  58
 60  62  64  66  68  70  72  74  76  78  80  82  84  86  88 ]

Add : 1980

*OzKernelSyntaxParser> thread_mainK (Finite 3) "thread_test9.txt" "blah.txt" "blahblah2.txt"
P : Unbound

F : Unbound

Add : Unbound

P : [ 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89 ]

F : [ 0  2  4  6  8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38  40  42  44  46  48  50  52  54  56  58
 60  62  64  66  68  70  72  74  76  78  80  82  84  86  88 ]
```

It will produce 90 even numbers added in a filter. The sum is 1980.

3. local X Y Generate Display DisplayH Merge Add in

```
fun {Generate N}
 fun {$} (N#{Generate (N+1)}) end
end
 fun {Add X Y}
 fun {$}
  (V#F) = {X} in
  ((V+Y)#{Add F Y})
 end
end
 Merge = fun {$ X Y}
 fun {$}
    (V#F) = {X}
    (U#H) = {Y} in
      if (V < U) then (V#{Merge F Y})
      else
          if (V > U) then (U#{Merge X H})
      else (V#{Merge F H})
          end
      end
    end
  end
 proc {Display X N}
   fun {DisplayH Z Num}
    if (Num == 0) then nil
    else
```

```
        (V#F) = {Z} in
        (V|{DisplayH F (Num-1)})
        end
       end
     local L in
       L = {DisplayH X N}
       skip Browse L
     end
    end

  //Testing
  X = {Generate 2}
  Y = {Generate 4}

  {Display {Merge X Y} 6}
   end
```



```
*OzKernelSyntaxParser> thread_mainK (Finite 1) "p1.txt" "p1OUT.txt" "p1Haskell.txt"
L : [ 2  3  4  5  6  7 ]

*OzKernelSyntaxParser>
```

```
% render solutions nicely.
:- use_rendering(chess).

%%    kings(+N, -Kings) is nondet.
```

C:\Program Files\Haskell Platform\8.6.5\bin\ghci.exe
docs.google.com/document/d/1_g5za-JyanK3d7To60P3OLg88unzRQB1H92trFJIHAo/edit
Share
Type here to search
2:59 PM
11/14/2019

Here, the function Merge is taking 2 generators X and Y. Generators X and Y are inputs. The function Merge merges both X and Y to return a generator where the values are merged through the sum.

Part 3:

4.
```
bubble_sort(List,Sorted):-b_sort(List,[],Sorted).
b_sort([],Acc,Acc).
b_sort([H|T],Acc,Sorted):-bubble(H,T,NT,Max),b_sort(NT,[Max|Acc],Sorted).

bubble(X,[],[],X).
bubble(X,[Y|T],[Y|NT],Max):-X>Y,bubble(X,T,NT,Max).
bubble(X,[Y|T],[X|NT],Max):-X=<Y,bubble(Y,T,NT,Max).

?-bubble_sort([10,40,2,5],L).

L = [2, 5, 10, 40]
```

5.
```
:- use_rendering(chess).
:- use_module(library(clpfd)).

n_kings(N, Qs) :-
        length(Qs, N),
        Qs ins 1..N,
        safe_kings(Qs).

safe_kings([]).
safe_kings([Q|Qs]) :-
        safe_kings(Qs, Q, 1),
        safe_kings(Qs).

safe_kings([], _).
safe_kings([Q|Qs], Q0) :-
        Q0 #\= Q+1,
        abs(Q0 - Q) #> 1,
        safe_kings(Qs, Q).


/** <examples>

?- n_kings(8, Qs), labeling([ff], Qs).

*/

?- n_kings(8, Qs), labeling([ff], Qs).

Qs = [1, 3, 1, 3, 1, 3, 1, 3]
```