Assignment 4 TDT4165 Programming Languages

Nicolai Molstad

Task 1

a)

- 30 gets printes first as it is the first in the local block
- a thread block for A is created
- a thread block for B is created
- C * 100 is printed
- A is printed
- B is printed
- A * 10 is printed
- B * 10 is printed
- A * 10 and B * 10 are printed in any order as both are delayed by 100ms

b) As each thread is independent of the other, the order of the output is order by the delay befor the print statement

based on that we should se C, A, B, A * 10, B * 10, C * 100

c)

- 2 is printed first as it is the first in the local block
- a thread block for A is created
- A is printed
- a thread block for B is created
- B is printed
- C is declared as A+B
- C is printed

d)

A is set in a thread, and both next thread and the ending is dependent on A, so it has to be printed first.

The end is dependent on B, so it has to be printed last.

B will only be printed after A is printed, and A is printed after the thread is done.

We can see that the threads are executed in parallel as A and B are printed before C is declared and that C is dependent on A and B

Task 2

```
fun {Enumerate Start End}
    local Tail in
        if Start =< End then

                Tail = thread {Enumerate (Start+1) End} end


            Start | {List.take Tail End-Start}
        else
            nil
        end
    end
end

fun {GenerateOdd Start End} List in
    List = {Enumerate Start End}

    {Filter List Int.isOdd}
end
```

Task 3

```
fun {ListDivisorsOf Number}
    local PosibleDivisors in
        PosibleDivisors = {Enumerate 1 Number }
        {Filter PosibleDivisors fun {$ Var}

            (Number mod Var) == 0 end}
    end
end

fun {ListPrimesUntil N}
    local Numbers in
        Numbers = {Enumerate 2 N}

        {Filter Numbers fun {$ Var}
            {ListDivisorsOf Var} == [1 Var] end}
    end
end
```

Task 4

```
fun lazy {EnumerateLazy} EnumerateInner in
    fun lazy {EnumerateInner N}
        N | {EnumerateInner N + 1}
    end

    {EnumerateInner 1}
end

fun lazy {PrimesLazy}
    local FilterLazy in
        fun lazy {FilterLazy Numbers Function}
            case Numbers of Head|Tail then
                if {Function Head} then
                    Head | {FilterLazy Tail Function}
                else
                    {FilterLazy Tail Function}
                end
            else nil
            end
        end

        {FilterLazy {EnumerateLazy} fun {$ Var}
            {ListDivisorsOf Var} == [1 Var] end}
    end

end
```