Linguagens de Programação (CK0115)
Lista de Exercícios IV (Capítulo 4)

Fernanda Costa de Sousa - 485404

1. Thread semantics.

(a)

- 1,2,3,6,4,5,7
- 1,2,3,6,4,7
- 1,2,3,6,7
- 1,2,3,7,4,5,6
- 1,2,3,7,4,6
- 1,2,3,7,6
- 1,2,6,3,4,5,7
- 1,2,6,3,4,7
- 1,2,6,3,7

```
1.local B in <s> end
2.thread <s> end
3.thread <s> end
4.if B then <s> end
5.{Browse yes}
6.B=true
7.B=false
```

(b)
Para essa pequena mudança no programa basta fazer B=true.

2. Threads and garbage collection.

A operação Wait bloqueia até que o argumento seja ligado. Como o underline é como se fosse um identificador sem nome, o primeiro e segundo são diferentes.
O Collectibles é referenciado como argumento quando o procedimento B é chamado, mas como não há referência no momento do procedimento Wait, ele está condicionado ao Garbage Collectibles.

4. Order-determining concurrency.

Em que ordem são feitas as adições? A criação da thread está na ordem escrita. A adição está na ordem de A, B, C, D e não na ordem causal.

Qual é o resultado final? O resultado é 4.

Conclusão:

Mesmo que o cálculo seja parcialmente encadeado, o cálculo continua sob a restrição de ordem necessária e o resultado não muda.

5. The Wait operation.

proc {Wait X}
if X==unit then skip else skip end
end

Como posso definir a operação como acima, a verificação de compreensão `==` é bloqueada quando a expressão tem a mesma forma, e existem variáveis não ligadas. Então, a instrução if também é bloqueada, ou seja funciona.

6. Thread scheduling.

O Mozart escalona pela prioridade da thread. Então, produtor e consumidor são escalonados dessa maneira.
Como a operação skip está dentro da parte do cálculo da thread do consumidor, a etapa de cálculo que consome ou produz o fluxo unitário é bem maior para o consumidor. De maneira que os produtores produzem muito e por isso muitos não são consumidos, não são calculados, pois acabam sendo ignorados pelos consumidores.

9. Digital logic simulation.

```
declare
local
  fun {NotLoop Xs}
     case Xs of X|Xr then (1-X)|{NotLoop Xr} end
  end
in
  fun {NotG Xs}
     thread {NotLoop Xs} end
  end
end
fun {GenerateGate F}
  fun{$ Xs Ys}
     fun {FLoop Xs Ys}
    case Xs#Ys of (X|Xr)#(Y|Yr) then {F X Y}|{FLoop Xr Yr} end
     end
  in
     thread {FLoop Xs Ys} end
  end
end
AndG={GenerateGate fun {$ X Y} X*Y end}
OrG ={GenerateGate fun{$ X Y} X+Y-X*Y end}
```

```
XOrG ={GenerateGate fun{$ X Y} X+Y-2*X*Y end}

declare
proc {FullAdder X Y Z ?C ?S}
  K L M
in
  K={AndG X Y}
  L={AndG Y Z}
  M={AndG Z X}
  C={OrG {OrG K L} M}
  S={XOrG {XOrG X Y} Z}
end

declare
Z0=0|0|0|_
X0=1|1|0|_ Y0=0|1|0|_ C0 S0
X1=0|0|1|_ Y1=1|1|1|_ C1 S1
X2=0|0|0|_ Y2=0|0|0|_ C2 S2
in
{FullAdder X0 Y0 Z0 C0 S0}
{FullAdder X1 Y1 C0 C1 S1}
{FullAdder X2 Y2 C1 C2 S2}

{Browse input}
{Browse X0#Y0}
{Browse X1#Y1}
{Browse X2#Y2}
{Browse output}
{Browse S0}
{Browse S1}
{Browse S0}
```

10. Basics of laziness.

```
fun lazy {Three} {Delay 1000} 3 end
{Three}+0
{Three}+0
{Three}+0
```

Podemos traduzir e fica da seguinte maneira:

```
declare Three A B C
```

```
Three = proc {$ ?X} {Delay 1000} X=3 end
{ByNeed Three A}
A+0
{ByNeed Three B}
B+0
{ByNeed Three C}
C+0
```

Com isso, é fácil ver que é calculado três vezes.
Respondendo a pergunta, se o resultado não deveria ser calculado apenas uma vez, posso dizer que em um modelo estritamente declarativo é possível implementar armazenando o resultado em cache e calculando apenas uma vez. Mas para linguagens com estados, tal implementação não é interessante por causa dos efeitos que ela pode causar. OZ é uma linguagem que possui estado explícito e como disse acima deve ser essa a sua implementação.

11. Laziness and concurrency.

```
declare
fun lazy {MakeX} {Browse x} {Delay 3000} 1 end
fun lazy {MakeY} {Browse y} {Delay 6000} 2 end
fun lazy {MakeZ} {Browse z} {Delay 9000} 3 end
X={MakeX}
Y={MakeY}
Z={MakeZ}
{Browse thread X+Y end +Z}
```

O MakeX e o MakeY são necessários de imediato, o atraso vai começar 6 segundos depois do cálculo de X + Y, e Z será necessário. Então o resultado seria 15 segundos depois (6 + 9)10..
No segundo exemplo, após analisar a expressão, temos que Z é necessário logo após a criação do encadeamento.
Assim assumindo que leva aproximadamente 1 segundo para criar uma thread, o cálculo mais rápido é:
Thread ... Thread Thread i_1 + i_2 end + i_3 end ... + i_n end
Então, pode ser implementado da seguinte maneira:

```
{FoldL Ls fun{$ X Y} thread X+Y end end 0}
```

16. By-need execution.

```
proc {Wait X}
```

```
   thread if X==unit then skip else skip end end
end
```

19. Limitations of declarative concurrency.

A função Merge é uma função que pega um elemento por vez de cada um dos inputs e vai juntando, então não existe aqui o não determinismo. Os fluxos ficam esperando o momento em que serão “ligados”. Não é possível mesclar dois se um dos três não estiver vinculado, pois o resultado muda dependendo do momento da união, e aqui seria mostrado o não determinismo. As condições para o modelo cliente servidor são as mesmas e não podem ser atendidas no modelo declarativo.