Nome: João Pedro Silveira e Silva

Matrícula: 00303397

Questão 1: (2,5 pontos) Prove que a união de uma quantidade infinita, porém contável de conjuntos contáveis produz um conjunto que é contável. Mais formalmente, demonstre que se A0, A1, A2, A3, A4 são todos conjuntos contáveis, então o conjunto $\cup_{i \in N} A_i$ é contável.

Resolução:

Dado A_i conjuntos contáveis, então há uma função injetora que mapeia cada A_i no conjuntos dos números naturais.

Sabendo da seguinte propriedade da decomposição em fatores primos:

- Todo o número natural **maior do que 1** pode ser escrito como um produto onde todos os fatores são primos. Retirando o número **1** do conjunto dos primos, esta fatoração se torna única.

E também do fato de que o Teorema de Euclides garante a existência de uma infinidade de números primos.

Podemos mapear os i conjuntos contáveis com suas i funções sobrejetoras **F_i** da seguinte forma:

Dados i conjuntos **A_i com n elementos, todos contáveis com funções injetoras Fi que mapeiam cada elementos de cada conjunto A nos naturais**. Dado um conjunto **P contendo todos os infinitos números primos, excluindo o 1**. E dado que **o número de conjuntos mapeados A é contável** podendo ser estabelecida uma relação entre cada número natural, de 0 a infinito.

Primeiro definimos a operação P(n) que retornará o enésimo número primo em ordem crescente. Agora definimos uma função injetora que mapeará todos os elementos dos conjuntos A_i nos números naturais.

> **$f(x) = P(i)^{(F_i(x))}$**, onde x é um valor dado de um conjunto A_i, i é um número dado ao conjunto A_i, P(i) é o enésimo número primo baseado no número dado ao conjunto A_i e F_i é a função injetora que mapeia cada elemento de um conjunto A_i nos naturais. Assim cada elemento de cada conjunto irá gerar um número natural diferente baseado nos números fatoráveis apenas pelo enésimo número primo.

Questão 2: (2,5 pontos) Vimos em sala a famosa codificação de Godel para pares de naturais, a saber a codificação **(x, y) → (2^x)*(3^y)**. Nesta questão, vamos introduzir uma nova codificação de pares de naturais chamada de codificação por entrelaçamento de dígitos. Sejam x e y naturais quaisquer. Suponha que d_k^x * d_{k-1}^x . . . d_1^x * d_0^x e d_k^y * d_{k-1}^y . . . d_1^y * d_0^y sejam as respectivas representações dos números x e y na base 10, onde d_i^x e d_i^y, para todo i ∈ {0, . . . , k}, são os (i + 1)-ésimos dígitos decimais de x e de y, respectivamente, da direita para esquerda (completando, se necessário, o menor dos números com 0's à esquerda para igualar as quantidades de dígitos). A codificação por entrelaçamento de dígitos do par (x, y) é definida como descrito abaixo:

(x, y) = (d_k^x * d_{k-1}^x * . . . * d_1^x * d_0^x, d_k^y * d_{k-1}^y * . . . * d_1^y * d_0^y) →

d_k^y * d_k^x * d_{k-1}^y * d_{k-1}^x * . . . * d_1^y * d_1^x * d_0^y * d_0^x

Seguem alguns exemplos:

- (8, 19) = (08, 19) → 1098
- (19, 8) = (19, 08) → 0189 = 189
- (25, 13) → 1235
- (0, 0 → 00 = 0
- (99, 9999) = (0099, 9999) → 90909999

Implemente para a máquina NORMA uma sub-rotina com a sintaxe

C := pair_entrelaça(A, B)

que receba de entrada as componentes do par (x, y), armazenadas respectivamente nos registradores A e B, e "retorne" o valor da codificação por entrelaçamento de dígitos do par (x, y) no registrador C preservando os valores iniciais contidos em A e B. Explique como seu código funciona e indique claramente nele os registradores auxiliares. A implementação deve ser feita em uma das estruturações vistas em aula (monolítica, iterativa ou recursiva; consulte os slides para mais detalhes). Para facilitar, você pode utilizar livremente qualquer uma das sub-rotinas que foram apresentadas em sala nos slides. Entretanto, quaisquer sub-rotinas usadas no seu código que não se encontram definidas nos slides deverão ser explicitamente implementadas.

Exemplo de teste para o código da resolução no simulador de norma (http://www.inf.ufrgs.br/~rma/simuladores/norma.html**):**

Dado o par (2,11) a resposta deverá ser 1012. Para dar a entrada na máquina é utilizada a codificação vista em aula. Então para (2,11) entre com (2^2*(2*11 + 1) – 1) que é igual a 91.

A saída em Y deverá ser 1012. Evite casos de teste que gerem uma entrada muito grande, como (8, 19) que irá gerar 9983.

Seguem link de vídeos que fiz com o código rodando nas resoluções de teste passadas utilizando "força bruta" para a entrada dos pares.

Código da resolução:

```
//*********************************************************************
// Zera o registrador recebido por parâmetro
//
// Variáveis:
//  Zera_A: Armazena o valor de entrada e o valor de saída
//---------------------------------------------------------------------
operation zera(Zera_A){
  1: if zero Zera_A then goto 0 else goto 2
  2: do sub Zera_A 999999999999 goto 1
}
//*********************************************************************


//*********************************************************************
// Soma Soma_A com Soma_B, salvando o resultado em Soma_A
//
// Variáveis:
//  Soma_A: Armazena o valor de entrada e saída
//  Soma_B: Armazena o valor de entrada
//  Soma_C: Armazena o valor de C usado para manter o valor de Soma_B
//---------------------------------------------------------------------
operation soma(Soma_A,Soma_B){
  1: if zero Soma_B then goto 0 else goto 2
  2: if zero Soma_B then goto 6 else goto 3
  3: do dec  Soma_B goto 4
  4: do inc  Soma_A goto 5
  5: do inc  Soma_C goto 2
  6: if zero Soma_C then goto 0 else goto 7
  7: do dec  Soma_C goto 8
```

```
  8: do inc  Soma_B goto 6
}
//*********************************************************************


//*********************************************************************
// Soma Somad_A com Somad_B, salvando o resultado em Somad_A e
// destruindo Somad_B
//
// Variáveis:
//  Somad_A: Armazena o valor de entrada e saída
//  Somad_B: Armazena o valor de entrada
//---------------------------------------------------------------
operation soma_destrutiva(Somad_A,Somad_B){
  1: if zero Somad_B then goto 0 else goto 2
  2: do dec Somad_B goto 3
  3: do inc Somad_A goto 1
}
//*********************************************************************


//*********************************************************************
// Subtrai SubtraiS_A com SubtraiS_B, salvando o resultado em
// SubtraiS_A
//
// Variáveis:
//  SubtraiS_A: Armazena o valor de entrada e saída
//  SubtraiS_B: Armazena o valor de entrada, preserva este valor
//  SubtraiS_C: Armazena o valor de B para evitar destruição
//---------------------------------------------------------------e
operation subtracao_parcial_destrutiva(SubtraiS_A,SubtraiS_B){
```

```
 1: do zera(SubtraiS_C) goto 2
 2: do soma(SubtraiS_C,SubtraiS_B) goto 3
 3: if zero SubtraiS_A then goto 0 else goto 4
 4: if zero SubtraiS_C then goto 0 else goto 5
 5: do dec SubtraiS_C goto 6
 6: do dec SubtraiS_A goto 3
}
//*********************************************************************


//*********************************************************************
// Subtrai Subtrai_A com Subtrai_B, salvando o resultado em Subtrai_A
//
// Variáveis:
//  Subtrai_A: Armazena o valor de entrada
//  Subtrai_B: Armazena o valor de entrada
//  Subtrai_C: Armazena o valor de Subtrai_C usado para manter Subtrai_B
//  Subtrai_R: Armazena o valor de saída
//  de Subtrai_B
//---------------------------------------------------------------------e
operation subtrai(Subtrai_A,Subtrai_B,Subtrai_R){
 1: do zera(Subtrai_R) goto 2
 2: do zera(Subtrai_C) goto 3
 3: do soma(Subtrai_R,Subtrai_A) goto 4
 4: if zero Subtrai_R then goto 9 else goto 5
 5: if zero Subtrai_B then goto 9 else goto 6
 6: do dec  Subtrai_B goto 7
 7: do dec  Subtrai_R goto 8
 8: do inc  Subtrai_C goto 4
 9: if zero Subtrai_C then goto 0 else goto 10
```

```
 10: do dec  Subtrai_C goto 11
 11: do inc  Subtrai_B goto 9
}
//*********************************************************************


//*********************************************************************
// Dado dois registradores, compara e retorna se o primeiro é menor
// Variáveis:
//  Menor_A: Armazena o valor de entrada
//  Menor_B: Armazena o valor de entrada
//  Menor_C: Armazena o resultado da subtração
//---------------------------------------------------------------
test eh_menor(Menor_A,Menor_B){
  1: do subtrai(Menor_A,Menor_B,Menor_C) goto 2
  2: if zero Menor_C then goto 3 else goto false
  3: do subtrai(Menor_B,Menor_A,Menor_C) goto 4
  4: if zero Menor_C then goto false else goto true
}
//*********************************************************************


//*********************************************************************
// Descrição: Retorna o maior entre dois números
//
// Variáveis:
//  MAX_A: Entrada A
//  MAX_B: Entrada B
//  MAX_R: Saída, resultado
//---------------------------------------------------------------
operation max(MAX_A, MAX_B, MAX_R){
```

```
  1: do zera(MAX_R) goto 2
  2: if eh_menor(MAX_A,MAX_B) then goto 3 else goto 4
  3: do soma(MAX_R, MAX_B) goto 0
  4: do soma(MAX_R, MAX_A) goto 0
}
//*****************************************************************


//*****************************************************************
// Descrição: Dado duas entradas, subtrai sucessivamente até antes de
// zero e retorna o número de subtrações
//
// Variáveis:
//  SUBS_A: Entrada, destruida
//  SUBS_B: Entrada, número a se subtrair de A até SUBS_A < SUBS_B
//  SUBS_C: Saída, contagem de subtrações
//-----------------------------------------------------------------
operation sub_sucessiva(SUBS_A,SUBS_B,SUBS_C){
  1: if zero SUBS_B then goto 0 else goto 2
  2: if zero SUBS_A then goto 0 else goto 3
  3: do zera(SUBS_C) goto 4
  4: if eh_menor(SUBS_A,SUBS_B) then goto 0 else goto 5
  5: do subtracao_parcial_destrutiva(SUBS_A,SUBS_B) goto 6
  6: do inc SUBS_C goto 4
}
//*****************************************************************


//*****************************************************************
// Divide um número por dois e retorna o quociente e o resto
//
```

```
// Variaveis:
//  DP2_A: Armazena o valor de entrada a ser dividido
//  DP2_Div: Armazena o dividendo de entrada A para cálculos
//  DP2_Qo: Armazena o quociente resultante da divisão
//  DP2_Re: Armazena o resto resultante da divisão
//---------------------------------------------------------------
operation divide_por_dois(DP2_A,DP2_Qo,DP2_Re){
 //Zera valores iniciais
 1: do zera(DP2_Div) goto 2
 2: do zera(DP2_Qo) goto 3
 3: do zera(DP2_Re) goto 4
 //Salva A em DP2_Div
 4: do soma(DP2_Div,DP2_A) goto 5
 //Caso DP2_Div== 0 => Fim
 5: if zero DP2_Div then goto 0 else goto 6
 6: do dec DP2_Div goto 7
 //Caso DP2_Div== 1 => DP2_Re
 7: if zero DP2_Div then goto 8 else goto 9
 //Salva o DP2_Re
 8: do inc DP2_Re goto 0
 //Soma DP2_Qo e continua subtração sucessiva
 9: do dec DP2_Div goto 10
 10: do inc DP2_Qo goto 5
}
//*****************************************************************


//*****************************************************************
// Realiza a divisão sucessiva de um número por 2 até alcançar um resultado
// ímpar
```

```
//
// Variáveis:
//  DP2AI_A: Armazena o valor de entrada a ser dividido
//  DP2AI_TA: Armazena o número que esta sendo dividido temporariamente
//  DP2AI_Re: Armazena o resto da divisão
//  DP2AI_Cd: Armazena o contador de divisão por 2 com resultado par
//  DP2AI_Ri: Armazena o último resto e por fim o resto ímpar
//  DP2AI_Qo: Armazena o resultado de cada divisão
//-------------------------------------------------------------------
operation dividir_por_2_ate_impar(DP2AI_A,DP2AI_Qo,DP2AI_Ri,DP2AI_Cd){
 //Zera valores iniciais
 1: do zera(DP2AI_TA) goto 2
 2: do zera(DP2AI_Re) goto 3
 3: do zera(DP2AI_Cd) goto 4
 //Salva A em DP2AI_TA
 4: do soma(DP2AI_TA, DP2AI_A) goto 5

 //Inicia looping zerando o último resto e inicializando com DP2AI_TA
 5: do zera(DP2AI_Ri) goto 6
 6: do soma(DP2AI_Ri, DP2AI_TA) goto 7

 //Realiza a divisão
 7: do divide_por_dois(DP2AI_TA,DP2AI_Qo,DP2AI_Re) goto 8

 //Caso o resto seja 0, então continua as divisões sucessivas com o resultado
 8: if zero DP2AI_Re then goto 9 else goto 0

 //Salva o resultado da divisão em DP2AI_TA para nova divisão
 9: do zera(DP2AI_TA) goto 10
```

```
 10: do soma(DP2AI_TA, DP2AI_Qo) goto 11

 //Incrementa o contador de divisões com resultado par
 11: do inc DP2AI_Cd goto 5
}
//*********************************************************************

//*********************************************************************
// Dada uma entrada, calcula o valor de Xo e Yo baseado na equação:
// X := (2^Xo) * (2*Yo + 1) - 1
//
// Variaveis:
//  CXOEYO_X: armazena o valor de entrada, X
//  CXOEYO_Xo: armazena o resultado de Xo
//  CXOEYO_Yo: armazena o resultado de Yo
//  CXOEYO_Ri: armazena o resto ímpar da operação de divisões sucessivas
//  CXOEYO_Re: armazena o resto da divisão da última equação
//---------------------------------------------------------------------
operation calcular_par_entrada(CXOEYO_X,CXOEYO_Xo,CXOEYO_Yo){
 //Realiza a subtração inicial (-1) da equação em X
 1:do inc CXOEYO_X goto 2

 //Divide X por 2 sucessivamente até
 //alcançar um resultado ímpar (resto = 1)
 //Ao encontrar este resultado, Xo irá ser o
 //número de vezes que se dividiu
 //x para chegar a este resultado.
 2:do dividir_por_2_ate_impar(CXOEYO_X,CXOEYO_Yo,CXOEYO_Ri,CXOEYO_Xo) goto 3
```

```
 //Decrementa o resultado ímpar para dividir por 2
 //Esta operação se refere ao cálculo:
 // 2*Yo + 1 = Resto ímpar
 3:do dec CXOEYO_Ri goto 4
 4:do divide_por_dois(CXOEYO_Ri,CXOEYO_Yo,CXOEYO_Re) goto 0
}
//*********************************************************************


//*********************************************************************
// Multiplica a entrada por 10 e salva o resultado na mesma
//
// Variaveis:
//  MLT_A: Entrada e Saída, multiplicando
//---------------------------------------------------------------------
operation multiplica10(MLT_A){
  //Zera o contador
  1: do zera(MLT_C) goto 2

  //Soma A ao Contador
  2: do soma(MLT_C,MLT_A) goto 3
  3: do zera(MLT_A) goto 4

  //Enquando C não for Zero soma 10 em A
  4: if zero MLT_C then goto 0 else goto 5
  5: do add MLT_A 10 goto 6
  6: do dec MLT_C goto 4
}
//*********************************************************************
```

```
//*********************************************************************
// Descrição: Retorna o valor de 10^DEZNA_X
//
// Variáveis:
//  DEZNA_X: Entrada
//  DEZNA_R: Saída, resultado
//  DEZNA_X_TEMP: Temporária, armazena o valor de DEZNA_X
//---------------------------------------------------------------------
operation dez_elevado_a_x(DEZNA_X,DEZNA_R){
 1: do zera(DEZNA_X_TEMP) goto 3
 3: do soma(DEZNA_X_TEMP, DEZNA_X) goto 4
 4: do zera(DEZNA_R) goto 5
 5: do inc DEZNA_R goto 6
 6: if zero DEZNA_X_TEMP then goto 0 else goto 7
 7: do dec DEZNA_X_TEMP goto 8
 8: do multiplica10(DEZNA_R) goto 6
}
//*********************************************************************


//*********************************************************************
// Descrição: Conta quantos dígitos um número possui
//
// Variáveis:
//  CONDIG_X: entrada
//  CONDIG_COUNT: saída
//  CONDIG_X_TEMP: variável para armazenar o valor de CONDIG_X
//---------------------------------------------------------------------
operation contador_de_digitos(CONDIG_X,CONDIG_COUNT){
 1:do zera(CONDIG_COUNT) goto 2
```

```
 2:if zero CONDIG_X then goto 0 else goto 3
 3:do inc CONDIG_COUNT goto 4
 4:do zera(CONDIG_TEMP) goto 5
 5:do add CONDIG_TEMP 9 goto 6
 6:do zera(CONDIG_X_TEMP) goto 7
 7:do soma(CONDIG_X_TEMP,CONDIG_X) goto 8
 8:do subtracao_parcial_destrutiva(CONDIG_X_TEMP,CONDIG_TEMP) goto 9
 9:if zero CONDIG_X_TEMP then goto 0 else goto 10
10:do inc CONDIG_COUNT goto 11
11:do multiplica10(CONDIG_TEMP) goto 8
}
//*****************************************************************


//*****************************************************************
// Descrição: Pega o valor do digito mais significativo dado a sua
// posicao. Tolera tentativas de pegar digito maior que o msd,
// retornando ZERO.
// Variáveis:
//  PEGENE_X: Entrada, número para pegar o digito
//  PEGENE_DC: Entrada, posição do dígito msd
//  PEGENE_A: Saída, valor do dígito msd
//  PEGENE_P10: Temporária, potencia de 10 utilizada no cálculo
//-----------------------------------------------------------------
operation pegar_enesimo_digito_se_msd(PEGENE_X,PEGENE_DC,PEGENE_A){
 1:do zera(PEGENE_A) goto 2
 2:if zero PEGENE_X then goto 0 else goto 3
 3:if zero PEGENE_DC then goto 0 else goto 4
 4:do zera(PEGENE_P10) goto 5
 5:do dec PEGENE_DC goto 6
```

```
 6:do dez_elevado_a_x(PEGENE_DC,PEGENE_P10) goto 7
 7:do sub_sucessiva(PEGENE_X,PEGENE_P10,PEGENE_A) goto 8
 8:do inc PEGENE_DC goto 0
}
//*****************************************************************

main{
  //Zera as variáveis da main
  1: do zera(Xo) goto 2
  2: do zera(Yo) goto 3
  3: do zera(C_Xo) goto 4
  4: do zera(C_Yo) goto 5
  5: do zera(Contador) goto 6
  6: do zera(Contador_Digito) goto 7
  7: do zera(Contador_Digito_Temp) goto 8
  8: do zera(A) goto 9
  9: do zera(B) goto 20

 //Calcula a entrada baseado na codificação
 // -> 2^y*(2*x + 1) - 1
 20: do calcular_par_entrada(X,A,B) goto 21
 21: do soma(Xo,A) goto 22
 22: do soma(Yo,B) goto 30

 //Busca o número dígitos do maior valor, (12,9) = 2
 30: do contador_de_digitos(Xo, C_Xo) goto 31
 31: do contador_de_digitos(Yo, C_Yo) goto 32
 32: do max(C_Xo,C_Yo,Contador) goto 33
```

```
//Verifica se há um número de dígitos > 0
33: if zero Contador then goto 0 else goto 34

//Cria um contador para os dígitos do resultado final
//Exemplo: A = 12 e B = 08, o resultado final terá 4 dígitos
34: do soma(Contador_Digito,Contador) goto 35
35: do soma(Contador_Digito,Contador) goto 36

//Subtrai 1 do resultado devido a lógica do sistema, para pegar o
//quarto digito é necessário usar o número 3, inicia por 0.
36: do dec Contador_Digito goto 37

//Pega o enésimo digito de Y(B)
37: do pegar_enesimo_digito_se_msd(Yo,Contador,Digito) goto 38
//Verifica se não é zero, caso for zero pula está etapa
38: if zero Digito then goto 44 else goto 39
//Multiplica o enésimo digito por sua posição no valor final
//Ex: se o quarto digito for 1, multiplica até obter o valor 1000
39: do soma(Contador_Digito_Temp, Contador_Digito) goto 40
40: if zero Contador_Digito_Temp then goto 43 else goto 41
41: do multiplica10(Digito) goto 42
//Decrementa os contadores e transfere o valor do Digito para o Y final
42: do dec Contador_Digito_Temp goto 40
43: do soma_destrutiva(Y,Digito) goto 44
44: do dec Contador_Digito goto 45
//Realiza o mesmo processo para X(A)
45: do pegar_enesimo_digito_se_msd(Xo,Contador,Digito) goto 46
46: if zero Digito then goto 52 else goto 47
47: do soma(Contador_Digito_Temp, Contador_Digito) goto 48
```

48: if zero Contador_Digito_Temp then goto 51 else goto 49

49: do multiplica10(Digito) goto 50

50: do dec Contador_Digito_Temp goto 48

51: do soma_destrutiva(Y,Digito) goto 52

52: do dec Contador_Digito goto 53

//Decrementa o contador principal e verifica se ainda há mais dígitos para

//serem contados, se não finaliza

53: do dec Contador goto 54

54: if zero Contador then goto 0 else goto 37

}

Questão 3: (2,5 pontos) Para cada um dos termos lambda mostrados abaixo, faça o que é pedido nos itens (a)–(e) a seguir:

- (λx.xy) (x λy.yx) (λyz.zy)
- (λz.z (λy.yzx) y) (λxz.(λy.zxy) x)

(a) (0,25 ponto) Explicite os parênteses e os λ's ausentes no termo lambda em questão.

- (((λx.(xy)) (x λy.(yx))) (λyλz.(zy)))
- ((λz.((z (λy.((yz)x))) y)) (λxλz.((λy.((zx)y)) x)))

(b) (0,25 ponto) Identifique todas as ocorrências livres e todas as ocorrências ligadas das variáveis presentes no termo lambda em questão.

Ocorrências livres em vermelho e ligadas em verde

- (λx.xy) (x λy.yx) (λyz.zy)
- (λz.z (λy.yzx) y) (λxz.(λy.zxy) x)

(c) (0,5 ponto) Identifique todos os redexes presentes no termo lambda em questão.

- (λx.xy) (x λy.yx) (λyz.zy) => 1 Redex

  (λx.xy) (x (λy.yx)) = P

  (λyz.zy) = Q

- (λz.z (λy.yzx) y) (λxz.(λy.zxy) x) => 2 Redex

1:

(λz.z (λy.yzx) y) = P

(λxz.(λy.zxy) x) = Q

2:

(λy.zxy) = P

x = Q

(d) (0,75 ponto) O termo lambda em questão possui forma normal? Se sim, mostre uma sequência de β-reduções do referido termo lambda até sua forma normal e, em cada uma das reduções da referida sequência, explicite o redex que está sendo contraído. Se não, justifique.

- (λx.xy) (x λy.yx) (λyz.zy) => Possui!

  Dado (λx.xy) (x (λy.yx)) = P e (λyz.zy) = Q sua forma normal é:

  xy (λyz.zy)

- (λz.z (λy.yzx) y) (λxz.(λy.zxy) x) => Possui!

  Dado (λz.z (λy.yzx) y) = P e (λxz.(λy.zxy) x) = Q reduzindo temos:

  (λxz.(λy.zxy) x) (λy.yzx) y

  Dado que (λxz.(λy.zxy) x) (λy.yzx) = P e y = Q reduzindo temos:

  (λy.zxy) x y

  Dado que (λy.zxy) x = P e y = Q reduzindo temos:

  zxy y

  E chegamos a forma normal.

(e) (0,75 ponto) Usando exclusivamente a estratégia de avaliação aplicativa, mostre (se possível) uma sequência de três β-reduções partindo do termo lambda em questão e, em cada uma das reduções da referida sequência, explicite o redex que está sendo contraído.

- (λx.xy) (x λy.yx) (λyz.zy)

  Dado (λx.xy) = P e (x λy.yx) = Q temos:

  ((x λy.yx) y) (λyz.zy) => Forma normal.

- (λz.z (λy.yzx) y) (λxz.(λy.zxy) x)

  Dado (λy.zxy) = P e x = Q temos:

  (λz.z (λy.yzx) y) (λxz.(zxx))

  Dado (λz.z (λy.yzx) y) = P e (λxz.(zxx)) = Q temos:

(λz. (λxz.(zxx)) (λy.yzx) y)

Dado (λxz.(zxx)) = P e (λy.yzx) = Q temos:

zxx y => Forma normal.

Questão 4: (2,5 pontos) Escreva, sem usar nenhum combinador de ponto fixo, um termo lambda chamado fibonacci que compute a função $C_n \rightarrow C_{Fn}$. Em outras palavras, o termo lambda fibonacci, ao ser aplicado a um numeral de Church C_n, retorna o n-ésimo termo Fn da sequência de Fibonacci 0, 1, 1, 2, 3, 5, 8, ... Seguem alguns casos de teste:

```
let

true    = \a b. a ;

false   = \a b. b ;

if      = \c a b. c a b ;

succ    = \n p q. p (n p q) ;

add     = \m n. m succ n ;

isZero  = \n . n (\x.false) true ;

pair    = \m n b. b m n ;

fst     = \p. p true ;

snd     = \p. p false ;

shiftInc =  \p. pair (snd p) (succ (snd p)) ;

pred    = \n. fst (n shiftInc (pair 0 0)) ;

Y       = \f.(\x.f(x x))(\x.f(x x)) ;

fibonacci_rec = \M.\n.if (isZero n) 0 (if (isZero(pred n)) 1 (add (M (pred n)) (M (pred (pred n))))));

fibonacci = Y fibonacci_rec;


in

(fibonacci 5)
```

Infelizmente não consegui alcançar sozinho uma forma de realizar o Fibonacci sem ponto fixo. Tentei utilizando tripla e operações de add, mult e exp porém não obtive sucesso em nenhuma tentativa.