– Questões –

1. *[2.0 pontos, 20 minutos, Paradigmas de computação]* Os dois principais paradigmas de computação são o imperativo (linguagens procedurais e orientadas a objetos) e o declarativo (linguagens funcionas e lógicas). Na abordagem imperativa, dado um determinado problema, o foco do programa é o que o computador deve fazer para se atingir uma solução. Na declarativa, o foco do programa é em como o computador deveria fazer para resolver o problema. Nesta questão, vamos considerar a definição do máximo divisor comum, definido por:

> "O máximo divisor comum ou MDC (mdc) entre dois números inteiros é o maior número inteiro que é fator de tais números. Por exemplo, os divisores comuns de 12 e 18 são 1, 2, 3 e 6, logo $mdc(12, 18) = 6$. " [fonte: wikipedia em português]

> "O algoritmo mais conhecido para o cálculo do máximo divisor comum é o algoritmo euclidiano. Ele começa com dois números inteiros positivos, e forma um novo par que consite no menor número entre os dois e a diferença entre o maior e o menor número. O processo se repete até que os dois números do par sejam iguais. Este número igual é o máximo divisor comum do par de inteiros do começo. " [fonte: wikipedia em inglês, tradução livre]

Com o objetivo de constrastar os diferentes paradigmas de computação, escreva uma versão imperativa, uma funcional e uma versão em linguagem lógica de um programa que calcule o máximo divisor comum entre dois números inteiros positivos.

Não se preocupe muito com a sintaxe, mas sim em utilizar construções que possibilitem salientar as características e especificidades de cada paradigma. Com base nos programas implementados, discurse sobre as semelhanças e diferenças entre fundamentos, formalismos e capacidades de abstração de cada paradigma.

2. *[1.5 pontos, 10 minutos, Características de LPs]* Existem vários critérios que são utilizados na análise e projeto de linguagens de programação. Eles são úteis porque permitem diferenciar as linguagens com base em definições técnicas. Dentre estes vários critérios, escolha dois que estão em conflito entre si. Defina cada um deles e explique porque eles estão em conflito.

3. *[1.0 pontos, 10 minutos, Programação funcional]* O que é currying? Utilize o código em ML abaixo para fornecer um exemplo.

```
1 fun soma a b c = a + b + c;
```

Continua no verso ↩

4. *[2.0 pontos, 15 minutos, Cálculo Lambda]* Reduza ao máximo as seguintes expressões Lambda utilizando as operações de Redução-β e Redução-α do Cálculo Lambda. A resposta para cada expressão deve ter todos os passos de redução. Indique, em cada um desses passos de redução, qual das operações é aplicada e sobre qual variável esta operação está sendo feita.

1. $(\lambda z \cdot z)(\lambda y \cdot y\ y)(\lambda x \cdot x\ a)$
2. $(\lambda x \cdot (\lambda y \cdot (x\ y))\ y)\ z$
3. $(((\lambda x \cdot \lambda y \cdot (x\ y))(\lambda y \cdot y))\ w)$
4. $(\lambda z \cdot z * z)(\lambda i \cdot i - 2)\ 3$

5. *[1.0 pontos, 10 minutos, Elementos de primeira ordem]* Em uma linguagem de programação funcional, o que significa dizer que funções são elementos de primeira ordem? Que características um elemento de programação deve ter para ser considerado de primeira ordem?

6. *[1.5 pontos, 10 minutos, Processo de inferência lógico]* Considerando a base de conhecimento na linguagem lógica Prolog, listada abaixo, mostre todos passos do mecanismo de resolução do motor de inferência para a consulta

```
capital_estado (X, sc).
```

```
1 cidade(portoalegre).
2 cidade(floripa).
3 cidade(alegrete).
4 estado(rs).
5 estado(sc).
6 capital(floripa, sc).
7 capital(portoalegre, rs).
8 capital_estado(X,Y) :- estado(Y), cidade(X), capital(X, Y).
```

7. *[1.0 pontos, 10 minutos, Operador de corte]* Qual seria o resultado da consulta

```
capital_estado (X, rs).
```

se a linha **8** da base de conhecimento da questão anterior fosse a regra de inferência listada abaixo? Explique a razão do resultado detalhando os passos do mecanismo de resolução do Prolog, justificando assim a sua resposta.

```
1 capital_estado(X,Y) :- estado(Y), cidade(X), !, capital(X, Y).
```

Cartão: 00220454 Nome: Alister Machado dos Reis

1) Imperativo:

```
int mdc(int a, int b){
    if (a == b) return a;
    if (a > b){
        return mdc(a-b, b);
    } else return mdc(b-a, a);
}
```

0,5

Funcional:

```
mdc x x = x
mdc a b =
    if a > b then (mdc (a-b) b)
             else (mdc (b-a) a)
```

0,5

Lógico.

```
mdc(X, X, X) :- !.
mdc(X, Y, M) :-
    (X > Y, Z is X-Y, mdc(Z, Y, M));
    (Z is Y-X, mdc(Z, X, M)).
```

0,25

Y>X

No paradigma imperativo focamos mais em dizer como a máquina deve operar, dando-a ordens. Isto ocorre pois este paradigma foi influenciado pela Máquina de Turing, que baseia-se em mudanças de estado e operações sobre uma memória (fita) para resolver problemas. É, de certa forma, um nível baixo de abstração caso analisemos comparando-o aos outros paradigmas.

No paradigma funcional focamos em o que a máquina deve calcular e qual a definição disso como função. Isso acontece pois o paradigma é fundamentado em Cálculo Lambda, um sistema de computação baseado em definições (abstração) e aplicações de funções. Provê uma abstração boa, sendo que o programador não precisaria se preocupar com memória, etc.

Já no paradigma lógico dizemos o que é verdade para o motor de inferência. Além disso, definimos regras que permitem ao motor determinar verdades partindo de outras. É um paradigma também bem abstrato, que usa casamento de padrões, unificação e um motor de inferência com capacidade de backtracking para determinar a validade de uma proposição ou não, e assim gerar resultados.

1,75

Nome: Elder Machado dos Reis

Código: 2...

2) Ortogonalidade, que é a capacidade de combinar os tipos básicos (ou não) de LP entre si, pode ter efeito na simplicidade da linguagem, tanto positivamente quanto negativamente.

O problema em tornar a linguagem ortogonal demais é definir os resultados que esta ortogonalidade gerará, a tipagem de expressões também deverá ser diferenciado.

Por exemplo, Javascript é muito ortogonal, porém apresenta alguns resultados inesperados devido a isso. Todos os exemplos abaixo foram testados em JS.

[ ] + [ ] = ""   [ ] + { } = { }   { } + [ ] = 0   { } + { } = NaN

Os dois resultados destacados são o que mais indicam o quão ruim pode ser tornar uma linguagem ortogonal demais. A soma não é mais comutativa para estes casos.

Com tudo dito acima, compromete-se tanto a Simplicidade quanto a Confiabilidade da LP.

1,5

[ ] + [ ] =

Os dois resultados destacados são o que mais indicam o quão uma pode ser tornar uma linguagem ortogonal demais, tornando não é mais acumulativo para estes casos.

Com tudo dito acima, comprometendo-se tanto a simplicidade quanto a confiabilidade da LP.

(1,5)

3) Currying é o processo de transformar uma função com mais de um argumento em várias de um argumento só, um formato mais próximo ao Cálculo Lambda. Isso permite a aplicação parcial de funções (momento mágico da programação funcional).

(1,0)

fun soma a b c = a+b+c

↳ (fn a => fn b => fn c => a+b+c) ($\lambda a.\lambda b.\lambda c.a+b+c$)

Podemos a partir disso, aplicar parcialmente a função e obter o seguinte

val soma2-com-1 = soma 1

↳ $(\lambda a.\lambda b.\lambda c.a+b+c)\,1 \rightarrow_{\beta} (\lambda b.\lambda c.1+b+c)$

soma2-com-1 é uma função de dois parâmetros que soma os dois e soma um ao resultado.

4) 1) $(\lambda z . z)(\lambda y . y\, y)(\lambda x . x a) \to_\beta (\lambda y . y\, y)(\lambda x . x a)$
$\to_\beta (\lambda x . x a)(\lambda x . x a) \to_\beta (\lambda x . x a)\, a \to_\beta a a$ ✓

2) $(\lambda x . (\lambda y . (x\, y))\, y)\, z \to_\beta (\lambda y . (z\, y))\, y \to_\beta z\, y$ Faltou uma redução-α ✗

3) $(((\lambda x\, \lambda y . (x\, y))(\lambda y . y))\, w) \to_\alpha (((\lambda x . \lambda k (x\, k))(\lambda y . y))\, w)$

$\to_\beta (\lambda k . ((\lambda y . y)\, k)\, w) \to_\beta (\lambda k . k)\, w \to_\beta w$ ✓

4) $(\lambda z . z * z)(\lambda i . i - 2)\, 3$ (não faz sentido associar a esq.)
$\to_\beta (\lambda z . z * z)(3-2) \to_\beta (3-2)*(3-2)$ ✓

(1,75)

Supondo que dá pra fazer matemática... $1 * 1 = 1$

5) Significa dizer que são os elementos de base da linguagem. São casos que operam sobre os tipos básicos da linguagem. É importante notar, também, que funções também são elementos de alta ordem em linguagens funcionais, ou seja, funções podem receber funções como parâmetro e combiná-las ou aplicá-las de uma forma específica, como nas formas funcionais de composição, construção e aplica-se a todos

(1,0)

6) capital_estado (X, sc) ->
(Y=sc) estado (sc) ? ✓ % deu certo 1º item, vai p/ segundo
cidade (X)
cidade (portoalegre) ? ✓ => X=portoalegre
capital (portoalegre, sc) ? ✗ % backtrack
cidade (floripa) ? ✓ => X=floripa
capital (floripa, sc) ? ✓ -> (X= floripa) % primeiro resultado
cidade (alegrete) ? ✓ => X=alegrete
capital (alegrete, sc) ? ✗

Fim da busca. Resultados: X= floripa

(1,5)