**INSTITUTO TECNOLÓGICO DE AERONÁUTICA - ITA**

DIVISÃO DE CIÊNCIA DA COMPUTAÇÃO

**CTC-11 – Lógica Matemática**



**Exame Final**

**Lista de PROLOG**

**ALUNOS**

Felipe Tuyama de Faria Barbosa ftuyama@gmail.com

João Vitor Marques joaovitormarques17@gmail.com

**PROFESSOR**

Paulo Marcelo Tasinaffo tasinaffo@ita.br

São José dos Campos, 01 de Abril de 2016

1. **Questão**

Considere uma base de conhecimento que descreva fragmentos do conhecimento humano, por exemplo, sistemas jurídicos, diagnóstico médico, normas ou regulamentos para obtenção do título de engenheiro pelo ITA, etc.

Para o exemplo escolhido estabeleça os sete passos da engenharia do conhecimento para criar um sistema de inferência em Lógica de Primeira Ordem (LPO).

1. **Questão**

Resolva o exercício 1 utilizando a linguagem Prolog. Utilize nesta resolução o aplicativoSWI-Prolog e faça testes e consultas nas bases de dados consideradas.

1. **Questão**

Exercício de Aquecimento: escolha cinco aplicativos Prolog de seu agrado – entre os exemplos apresentados em sala de aula – e rode-os no aplicativo *SWI-Prolog*. Façam consultas e vejam quais são as respostas obtidas pelo compilador.

1. **Questão**

Ordenar uma lista formada apenas por números. Exemplificando:

a. ?- sort([1,5,3,9,7],X).

b. X = [1,3,5,7,9]).

c. yes

d. ?- sort([5,1,7,3,7,9],[1,3,5,7,7,9]).

e. yes

**Solução:**

Consiste em uma aplicação recursiva do método tradicional Insertion Sort, utilizando Buffers e Listas para realizar a ordenação, conforme explicitado no código abaixo:

% São José dos Campos, 01 de Abril de 2016.

% Programador: Felipe Tuyama.

% % Questão 4.

%% Ordenando uma Lista em PROLOG

%% usando Insertion Sort

*isort*(*List*, *Sorted*) **:-** *inSort*(*List*, [], *Sorted*).

*inSort*([], *Buf*, *Buf*).

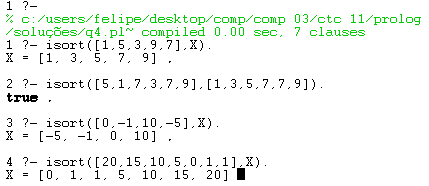
*inSort*([*H*|*T*], *Buf*, *Sorted*) **:-** *insert*(*H*, *Buf*, *NBuf*), *inSort*(*T*, *NBuf*, *Sorted*).

*insert*(*A*, [*B*|*T*], [*B*|*NBuf*]) **:-** *A* > *B*, *insert*(*A*, *T*, *NBuf*).

*insert*(*A*, [*B*|*T*], [*A*,*B*|*T*]) **:-** *A*=< *B*.

*insert*(*A*, [], [*A*]).

**Resultado:**

****

1. **Questão**

Encontrar o menor elemento de uma lista. Exemplificando:

f. ?- min([3,5,1,7,9],X).

g. X = 1

h. yes

i. ?- min([3,5,1,7,9],3).

j. no

k. ?- min([3,5,1,7,9],1).

l. yes

**Solução:**

Basta percorrer a lista recursivamente eliminando os seus elementos. Quando sobrar apenas um elemento, ele será o mínimo provisório. Por Backtracking, percorremos novamente a lista, sempre comparando o elemento atual com o mínimo provisório, até que ao final o mínimo provisório será o mínimo da lista inteira.

% São José dos Campos, 01 de Abril de 2016.

% Programador: Felipe Tuyama.

% % Questão 5.

%% Determinar o mínimo de uma lista

%

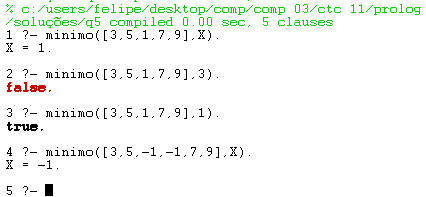
*minimo*(*List*, *Min*) **:-** *min*(*List*, *Min*).

*min*([*Min*],*Min*) **:-** !.

*min*([*X*|*T*], *X*) **:-** *min*(*T*, *Min*), *X* =< *Min*, !.

*min*([\_|*T*], *Min*) **:-** *min*(*T*, *Min*).

**Resultados:**

****

1. **Questão**

Contar o número de vezes que um certo elemento aparece em uma dada lista. Exemplificando:

m. ?- count([1,3,5,7,3,9],3,X).

n. X = 2

o. yes

p. ?- count([a,b,a,a,b,a,a,c,a,b],a,X).

q. X = 6

r. yes

s. ?- count([1,3,6,7,9],5,0).

t. Yes

**Solução:**

Esse algoritmo é muito semelhante ao de determinar o tamanho da lista. Porém em vez de incrementar sempre o tamanho da lista na recursão Backtracking, um contador é incrementado sempre que o Elemento contado é detectado.

% São José dos Campos, 01 de Abril de 2016.

% Programador: Felipe Tuyama.

% % Questão 6.

%% Contando o número de vezes que um

%% dado elemento aparece em uma lista

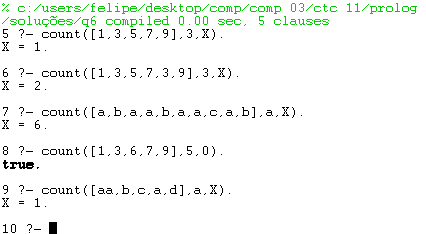
*count*(*List*, *Elem*, *Count*) **:-** *counter*(*List*, *Elem*, *Count*).

*counter*([], \_, 0).

*counter*([*Elem*|*T*], *Elem*, *CountP*) **:-** *counter*(*T*, *Elem*, *Count*), *CountP* *is* *Count* + 1, !.

*counter*([\_|*T*], *Elem*, *Count*) **:-** *counter*(*T*, *Elem*, *Count*).

**Resultado:**

****

1. **Questão**

O problema do macaco e a da banana. Há um macaco ao lado da porta em uma sala. No meio da sala uma banana está pendurada no teto. O macaco está zangado e quer obter a banana, mas não pode alcançá-la do chão. Ao lado da janela da sala há uma caixa que o macaco pode utilizar.

O macaco pode realizar as seguintes ações: (1) caminhar sobre o chão, (2) escalar a caixa, (3) empurrar a caixa ao redor da sala e (4) agarrar a banana se ele estiver estendido sobre a caixa diretamente sobre a banana. Pode o macaco obter a banana? Responda as seguintes perguntas com relação a esse problema:

a) o problema resolvido já se encontra listado em uma das transparências do professor desta

|  |  |  |
| --- | --- | --- |
| disciplina. Sendo assim digite-o no editor do Prolog, executando-o | com | a pergunta: |
| “ ? podeobter(estado(sobreporta,sobrechão, najanela, nãotem)).”. Nesse | caso, | verifique a |

resposta dada pelo Prolog.

b) O problema resolvido no item (a) possui as “ações” ordenadas desta forma: agarrar,escalar, empurrar e andar. De acordo com essa ordem, as *semânticas procedurais*do Prolog, o macaco preferirá *agarrar*em vez de escalar, escalar em vez de empurrar, etc. Entretanto, o que aconteceria se a ordem das cláusulas fosse diferente desta? Suponha então, colocar a relação andar em primeiro lugar. Nesta nova ordem refaça a pergunta “ ? podeobter(estado(sobreporta,sobrechão, najanela, nãotem)).” e verifique se Prolog entre num *looping infinito.*

c) Se for constatado que o Prolog realmente entra num *looping infinito*dê uma explicação plausível porque esse problema ocorre.

1. **Questão**

Determine se os elementos de uma lista determinam um palíndromo. Um palíndromo é*uma palavra*ou *uma seqüência de caracteres*que podem ser lidos - da mesma forma - tanto da esquerda para à direita, como da direita para à esquerda (e.g., [x, a, m, a, x]).

**Solução:**

Para resolver esse problema utilizei duas sub-rotinas. A primeira, *inverter*, inverte a Palavra recursivamente em uma variável auxiliar Inversa. Basicamente, percorre-se a palavra recursivamente, concatenando cada letra encontrada ao final da variável Inversa.

Em seguida, *saoIguauis*, temos uma comparação recursiva de duas palavras. Oras, uma palavra é um palíndromo quando a palavra é igual a sua inversa. Temos assim o algoritmo dado pela sua própria definição.

% São José dos Campos, 01 de Abril de 2016.

% Programador: Felipe Tuyama.

% % Questão 8.

%% Determinando se uma palavra é palíndromo

%

*palindromo*(*Palavra*) **:-** *inverter*(*Palavra*, *Inversa*), *saoIguais*(*Palavra*, *Inversa*).

%% Subrotina auxiliar para inverter sequência de caracteres

*inverter*([],[]).

*inverter*([*X*|*T*],*L*) **:-** *inverter*(*T*,*B*), *conc1*(*B*,[*X*],*L*).

*conc1*([], *L*, *L*).

*conc1*([*X*|*L1*],*L2*,[*X*|*L3*]) **:-** *conc1*(*L1*,*L2*,*L3*).

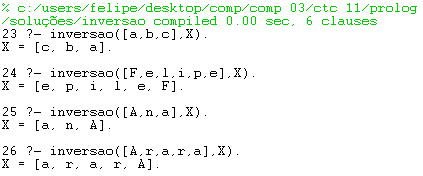
%% Subrotina que checa a igualdade de duas sequências de caracteres

*saoIguais*([], []).

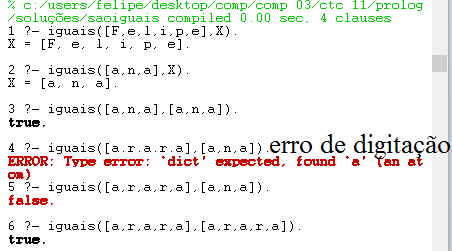
*saoIguais*([*X*|*T1*], [*X*|*T2*]) **:-** *saoIguais*(*T1*, *T2*).

**Resultado:**

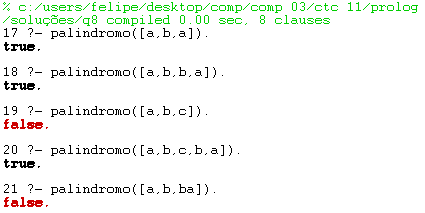
**Teste da Subrotina que inverte palavras.**

****

**Teste da Subrotina que verifica se duas palavras são iguais.**

****

**Teste da rotina que verifica se palavra é palíndromo.**

****

1. **Questão**

Determine se um dado número inteiro é primo. Exemplo: “ ? is \_ prime(7).”.

**Sugestão**: se o máximo divisor comum entre um “número dado” e o “próprio um” for o “próprio um”, então o número dado será primo.

1. **Questão**

Problema de *busca em largura*utilizando o *Prolog*. Seja o seguinte problema: duas fichas brancas e duas pretas estão dispostas conforme mostrado na figura abaixo. As fichas brancas estão separadas das pretas por um intervalo vazio. Deseja-se colocar as fichas pretas entre as brancas.

Duas operações são permitidas: *escorregar uma ficha para o intervalo vazio*ou *fazê-la saltar sobre a outra e aterrissar no intervalo*. A solução deste problema é extremamente simples, mas deve ser resolvida em *Prolog*. Que pergunta deve- se fazer ao Prolog para resolver esse problema?

**Solução:**

A pergunta ideal a ser feita para o Prolog é:

? – win ([b, b, i, p, p], Moves).

De tal forma que informamos a posição inicial do tabuleiro e um parâmetro “Moves”, o qual será completado com todas as sequências de movimentos até uma possível solução para o problema. Basta digitar “;” para visualizar as múltiplas soluções do problema.

Assim como o problema do Macaco e a Banana, defini uma série de estados de tabuleiro em que ocorre a vitória do jogo. Em seguida, defini recursivamente que uma posição anterior que pode gerar a posição de vitória a partir de um movimento também é uma posição de vitória (adicionando esse movimento a uma lista).

Bastou em seguida declarar todos os movimentos válidos do jogo para o funcionamento do programa PROLOG. Talvez eu pudesse usar sublistas na definição dos movimentos para minimizar o código e o número de movimentos válidos (só tendo de declarar as regras universais). Porém, concluí que isso demandaria uma complexidade notável adicional de programação. Então optei pela maneira mais simples.

% São José dos Campos, 01 de Abril de 2016.

% Programador: Felipe Tuyama.

% % Questão 10.

%% Busca em Largura para PROLOG

%

*win*([*b*,*p*,*p*,*b*,*i*], [[*b*,*p*,*p*,*b*,*i*]]).

*win*([*b*,*p*,*p*,*i*,*b*], [[*b*,*p*,*p*,*i*,*b*]]).

*win*([*b*,*p*,*i*,*p*,*b*], [[*b*,*p*,*i*,*p*,*b*]]).

*win*([*b*,*i*,*p*,*p*,*b*], [[*b*,*i*,*p*,*p*,*b*]]).

*win*([*i*,*b*,*p*,*p*,*b*], [[*i*,*b*,*p*,*p*,*b*]]).

*win*(*Posic1*, [*Posic1*|*Moves*]) **:-** *mover*(*Posic1*, *Posic2*), *win*(*Posic2*, *Moves*).

% Movimentos das Pretas

*mover*([*X*,*Y*,*Z*,*i*,*p*], [*X*,*Y*,*Z*,*p*,*i*]). % Deslizar

*mover*([*X*,*Y*,*i*,*p*,*Z*], [*X*,*Y*,*p*,*i*,*Z*]). % Deslizar

*mover*([*X*,*i*,*p*,*Y*,*Z*], [*X*,*p*,*i*,*Y*,*Z*]). % Deslizar

*mover*([*i*,*p*,*X*,*Y*,*Z*], [*p*,*i*,*X*,*Y*,*Z*]). % Deslizar

*mover*([*X*,*Y*,*i*,*b*,*p*], [*X*,*Y*,*p*,*b*,*i*]). % Saltar

*mover*([*X*,*i*,*b*,*p*,*Y*], [*X*,*p*,*b*,*i*,*Y*]). % Saltar

*mover*([*i*,*b*,*p*,*X*,*Y*], [*p*,*b*,*i*,*X*,*Y*]). % Saltar

% Movimentos das Brancas

*mover*([*X*,*Y*,*Z*,*b*,*i*], [*X*,*Y*,*Z*,*i*,*b*]). % Deslizar

*mover*([*X*,*Y*,*b*,*i*,*Y*], [*X*,*Y*,*i*,*b*,*Y*]). % Deslizar

*mover*([*X*,*b*,*i*,*Y*,*Z*], [*X*,*i*,*b*,*Y*,*Z*]). % Deslizar

*mover*([*b*,*i*,*X*,*Y*,*Z*], [*i*,*b*,*X*,*Y*,*Z*]). % Deslizar

*mover*([*X*,*Y*,*b*,*p*,*i*], [*X*,*Y*,*i*,*p*,*b*]). % Saltar

*mover*([*X*,*b*,*p*,*i*,*Y*], [*X*,*i*,*p*,*b*,*Y*]). % Saltar

*mover*([*b*,*p*,*i*,*X*,*Y*], [*i*,*p*,*b*,*X*,*Y*]). % Saltar

**Resultado:**

