Paradigmas da Programação I
LECom (2º ano)

Exame de 2ª Época

Data: 15 de Fevereiro de 2005
Hora: 14:30

Dispõe de 2:30 horas para realizar este exame
Leia as questões com toda a atenção e responda com calma e clareza em folha convencional

Questão 1: Representação de Conhecimento

Suponha que lhe pedem para desenvolver o sistema (completo, com HW e SW) para gestão da famoso Rally Barcelona-Dakar, fazendo o controlo em cada etapa e a respectiva afixação de resultados em painéis electrónicos.
Usando a abordagem seguida nas aulas para modelar sistemas de informação em lógica, recorrendo à linguagem de programação Prolog para poder interrogar de seguida o seu universo de discurso, construa uma Base de Conhecimento (BC) que descreva o sistema de gestão pedido.
Para isso guarde informação individual sobre: os pilotos e os co-pilotos (dados pessoais); as viaturas (modelo, marca, categoria, etc.); os patrocinadores; as etapas (número de ordem, local, distancia desde a anterior, etc.). Registe depois as relações entre essas várias entidades de modo a poder caracterizar cada equipa e gerir os resultados de cada uma.

Após identificar o tipo de cláusulas (factos ou regras) que deve usar para modelar o sistema, indique algum tipo de perguntas que poderia ver respondidas com o seu programa lógico, mostrando em particular: como sabia se um piloto chegou ao fim ou se desistiu; como poderia apurar o tempo total de uma equipa; como obteria informação sobre os patrocinadores de uma equipa; e as marcas apoiadas por um patrocinador.

Questão 2: Bases de Conhecimento, Árvores de Prova e de Procura

A forma mais eficaz de reconhecer comandos, lendo a sequência de caracteres escritos pelo utilizador de um sistema de controlo, é através do uso de um *autómato finito determinista (AD)*. Abaixo mostra-se parte da função de transição de estados desse AD (`delta`) e o reconhecedor.

```
delta(1, m, 2).   delta(2, o, 3).   delta(3, v, 4).   delta(4, e, -1).
                  delta(2, u, 5).   delta(5, d, 6).   delta(6, a, -2).
delta(1, c, 7).   delta(7, o, 8).   delta(8, l, 9).   delta(9, a, -3).
.....................................................
delta(1, _, 0).   delta(2, _, 0).   delta(3, _, 0). .....

invalido(Q) :- delta(Q, C, Q1), delta(Q, C, Q2), Q1 \== Q2.

corresponde(-1, cmd1).   corresponde(-2, cmd2).
corresponde(-3, cmd3).   corresponde(-4, cmd4).
ultimoFinal(-4).
```

```
reconhece( Chars, Cmd )  :- rec( 1, Chars, Cmd ).
rec( 0, _, nil ) :- write('ERRO !'), !.
rec( N, _, Cmd ) :- N < 0, !, corresponde(N, Cmd).
rec( N, [C|Rc], Cmd ) :- delta(N, C, N1), rec(N1, Rc, Cmd).
```

Responda, então, às alíneas seguintes:

a) Observe com atenção o programa acima e interpreta o predicado `reconhece/2`.

b) Usando uma *Árvore de Prova*, prove que o interpretador Prolog daria a seguinte resposta

```
?- delta(1, C, Q).
   C = m
   Q = 2
   yes
```

c) Usando uma *Árvore de Procura*, mostre que o interpretador de Prolog falha ao tentar provar a seguinte questão

```
invalido(2).
```

Explique o que significa essa falha.

d) Usando uma *Árvore de Procura*, mostre a resposta calculada pelo interpretador de Prolog à seguinte questão

```
reconhece([c,o,l,a], CMD).
```

Para evitar cair numa monótona repetição, apresente apenas os primeiros e o último níveis da árvore.

e) Defina um predicado `hacaminho(Qorigem,Qdestino)` que verifica se é possível partir do estado `Qorigem` atingir o estado `Qdestino` usando a função de transição `delta/3`.

f) Recorrendo ao predicado standard do Prolog `assertz/1`, que adiciona dinamicamente cláusulas a uma BC, escreva um predicado `acrescenta/1` que recebe como argumento o nome de um novo comando e junta à BC mais uma correspondência (isto é, um facto `corresponde/2`) caso esse comando ainda não exista; o novo comando ficará associado ao próximo número negativo em relação ao último usado (indicado pelo facto `ultimoFinal/1`). O predicado `ultimoFinal/1` deve ser actualizado de acordo.
Assim, por exemplo, a execução de `acrescenta(cmd5)` juntaria à BC o facto `corresponde(-5, cmd5)` e actualizava `ultimoFinal(-5)`.

Questão 3: Manuseamento de Listas

Sobre operações com Listas em Prolog, responda às alíneas seguintes:

a) Supondo que pode usar um predicado previamente definido `strlen/2` que devolve no 2ºargumento o comprimento da palavra que lhe é passada no 1º argumento, implemente um predicado `filtro/2` que, dada uma lista de palavras, retorne outra lista só com as palavras de comprimento entre 3 e 6 caracteres. Exemplo:

```
?- filtro([este, exemplo, da, uma, ideia, precisa, do, que, se, pretende], L).
   L = [este, uma, ideia, que]
```

b) Pretende-se que implemente um predicado `maximo/2` que, dada uma lista de números, devolva o valor máximo da lista (devolve a marca `nil` caso a lista esteja vazia). Exemplo:

```
?- maximo([1,32,3], L).
   L = 32
```

c) Pretende-se que implemente um predicado `valida/1` que sucede (devolve `yes`) se e só se o dicionário probabilístico DP passado como argumento estiver bem formado. Para isso é necessário que as probabilidades, associadas às possíveis traduções de cada palavra na língua original, somem 100%. O DP é uma lista de pares em que o 1º elemento é uma palavra e o 2º elemento é uma lista de possíveis traduções; para cada tradução é fornecido outro par: a palavra na língua destino a e respectiva probabilidade). Caso encontre alguma falha, deve escrever no écran a palavra original onde encontra o problema. Exemplos:

```
?- soma([ (agora,[(now,100)]),
          (gato,[(cat,95),(error,5)]),
          (bonito,[(beautiful,40),(nice,30),(pretty,30)]) ]).
   yes
?- soma([ (casa,[(home,100),(house,100)]),
          (livro,[(book,95)]),
          (bom,[(good,80),(nice,20)]) ]).
   casa
   no
```

d) Pretende-se que implemente um predicado `diff2/3` que, dados dois conjuntos (representados como listas), devolva o conjunto formado por todos os elementos do 2º que não pertencem ao 1º. Exemplo:

```
?- soma( [1,2,3,4,5,6,7,8], [1,3,5,7,9], L ).
   L = [9]
```