Programação em Lógica

Início	Quarta, 20 de Novembro de 2019 às 17:17
Estado	Prova submetida
Data de submissão:	Quarta, 20 de Novembro de 2019 às 19:32
Tempo gasto	2 horas 14 minutos
Nota	**15,40** de um máximo de 20,00 (**77**%)

Informação

Destacar pergunta

Vários pianistas atuaram numa audição de piano, tocando obras de diversos compositores. A seguinte base de dados guarda os compositores cujas obras foram interpretadas por cada pianista (predicado ***tocou(Pianista, ListaCompositores)***) e a ordem pela qual cada pianista tocou (predicado ***ordem(N, Pianista)***).

```
tocou(ana, [bach, chopin]).
tocou(filipe, [lizst, beethoven]).
tocou(carlos, [chopin, mozart]).
…
ordem(1, filipe).
ordem(2, ana).
ordem(3, carlos).
…
```

Responda às perguntas 1 a 6 **SEM** utilizar predicados de obtenção de soluções múltiplas (findall, setof e bagof), e **SEM** usar qualquer biblioteca do SICStus.

Pergunta 1

Respondida · Pontuou 1,000 de 1,000 · Destacar pergunta

Implemente o predicado ***antes(?Pianista1, ?Pianista2)*** que sucede se o pianista ***Pianista1*** tocou antes do pianista ***Pianista2***.

```
antes(Pianista1, Pianista2) :-
  ordem(P1, Pianista1),
  ordem(P2, Pianista2),
  P1 < P2.
```

Pergunta 2

Respondida · Pontuou 1,000 de 1,000 · Destacar pergunta

Implemente o predicado ***comecou_com(?Compositor)*** que unifica ***Compositor*** com o primeiro compositor interpretado na audição.

```
comecou_com(Compositor) :-
  ordem(Ordem, Pianista),
  comecou_com_aux(Pianista, Primeiro, Ordem, BestOrdem, [Pianista]), !,
  tocou(Primeiro, [Compositor | Resto]).

comecou_com_aux(CurPianista, FirstP, CurOrder, BestOrder, AuxList) :-
  ordem(NewOrder, NewPianista),
  (\+ member(NewPianista, AuxList)), !,
  check_order(CurPianista, NewPianista, BestPianista, CurOrder, NewOrder, NewFirstOrder),
  comecou_com_aux(BestPianista, FirstP, NewFirstOrder, BestOrder, [NewPianista | AuxList]).
```

Pergunta 3

Respondida · Pontuou 1,000 de 1,000 · Destacar pergunta

Implemente o predicado ***ausente(+Compositor)*** que sucede se nenhum pianista tocou peças do compositor ***Compositor***.

```
ausente(Compositor) :-
  ausente_aux(Compositor, []).

ausente_aux(Compositor, AuxListPianistas) :-
  tocou(Pianista, Compositores),
  (\+ member(Pianista, AuxListPianistas)), !,
  (\+ member(Compositor, Compositores)),
  ausente_aux(Compositor, [Pianista | AuxListPianistas]).

ausente_aux(_, _).
```

Pergunta 4

Respondida · Pontuou 0,500 de 1,000 · Destacar pergunta

Implemente o predicado ***um_pianista(?Compositor)*** que sucede se um e um só pianista tocou peças do compositor ***Compositor***.

```
um_pianista(Compositor) :-
  um_pianista_aux(Compositor, [], ListaPianistas),
  length(ListaPianistas, 1).

um_pianista_aux(Compositor, AuxListPianistas, ListaFinal) :-
  tocou(Pianista, Compositores),
  (\+ member(Pianista, AuxListPianistas)),
  (\+ member(Compositor, Compositores)), !,
  um_pianista_aux(Compositor, [Pianista | AuxListPianistas], ListaFinal).
```

Pergunta 5

Respondida · Pontuou 1,000 de 1,000 · Destacar pergunta

Implemente o predicado ***audicao/0*** que imprime no ecrã o programa da audição, em linhas da forma <***N: pianista - listaCompositores***>, onde **N** é o número de ordem (as linhas devem estar ordenadas por este número).

```
audicao :-
  audicao_aux(1).


audicao_aux(N) :-
  ordem(N, Pianista),
  tocou(Pianista, ListaCompositores), !,
  write(N), write(': '),
  write(Pianista), write(' - '),
  write(ListaCompositores), nl,
```

Pergunta 6

Respondida · Pontuou 0,600 de 1,000 · Destacar pergunta

Pretende-se saber o que tocou o pianista que atuou antes do Gustavo. Para tal, formulou-se a pergunta

```
?- ordem(N1, P), tocou(P, LC), N2 is N1+1, ordem(N2, gustavo).
```

Comente sobre o esforço de pesquisa desta pergunta e sugira uma formulação alternativa.

Esta pergunta exige um esforço de pesquisa desnecessário. A pergunta extrai um pianista qualquer, de qualquer ordem, e depois é que verifica se este veio imediamente antes do Gustavo. Caso não seja, vai ocorrer backtracking de modo a extrair outro pianista, fazendo a mesma comparação, e assim sucessivamente, até se encontrar o pianista correto. Isto vai originar uma árvore de pesquisa muito grande, desnecessariamente. Uma formulação alternativa seria extrair a ordem do Gustavo primeiro e, já sabendo qual é suposto ser a ordem do pianista que queremos ir buscar, extrair a informação sobre os compositores que este tocou.

Informação

Destacar pergunta

Suponha agora que se generaliza a base de dados anterior para conter uma audição em que intervêm músicos que tocam diferentes instrumentos. Para além dos indicados atrás, teremos agora também o predicado ***instrumento(Musico, Instrumento)***:

```
instrumento(ana, piano).
instrumento(filipe, piano).
instrumento(carlos, piano).
instrumento(luis, clarinete).
instrumento(maria, violino).
instrumento(afonso, clarinete).
instrumento(eva, acordeao).
…
```

Nas próximas questões já pode utilizar predicados de obtenção de soluções múltiplas (findall, setof e bagof), e bibliotecas do SICStus (com exceção da biblioteca clpfd).

Pergunta 7

Respondida Pontuou 1,000 de 1,000 Destacar pergunta

Implemente o predicado ***musicos_para_compositor(+Compositor, -ListaMusicos)*** que devolve em ***ListaMusicos*** os músicos que tocam obras do compositor ***Compositor***.

```
musicos_para_compositor(Compositor, ListaMusicos) :-
  findall(Musico, (tocou(Musico, Compositores), member(Compositor, Compositores)), ListaMusicos).
```

Pergunta 8

Respondida Pontuou 0,100 de 1,000 Destacar pergunta

Implemente o predicado ***composParaInstrumentos(+Compositor, -ListaDeInstrumentos)*** que unifica o segundo argumento com uma lista contendo os instrumentos (sem repetições) para os quais o compositor ***Compositor*** compôs músicas (ie. os instrumentos usados por quem interpretou peças desse compositor).

```
composParaInstrumentos(Compositor, ListaDeInstrumentos) :-
  musicos_para_compositor(Compositor, ListaMusicos),
  instrumentos_aux(ListaMusicos, [], UListaDeInstrumentos),
  sort(UListaDeInstrumentos, ListaDeInstrumentos).

instrumentos_aux([Musico | ListaMusicos], AuxLista, ListaDeInstrumentos) :-
  findall(Inst, instrumento(Musico, Inst), InstsMusico),
  append(AuxLista, InstsMusico, NewAuxLista),
  instrumentos_aux(ListaMusicos, NewAuxLista, ListaDeInstrumentos).
```

Pergunta 9

Respondida Pontuou 2,000 de 2,000 Destacar pergunta

Implemente o predicado ***musicos_por_instrumento(-ListaMusicosPorInstrumento)*** que devolve no seu argumento os músicos que tocam cada um dos instrumentos existentes, da forma *[Instr1-ListaMusicosInstr1, Instr2-ListaMusicosInstr2, ...]*. Por exemplo:

```
[ acordeão-[eva], clarinete-[luis,afonso], piano-[ana,filipe,carlos], violino-[maria] ]
```

```
musicos_por_instrumento(ListaMusicosPorInstrumento) :-
  musicos_aux([], ListaMusicosPorInstrumento).

musicos_aux(AuxListaInstrumentos, [Instrumento-ListaMusicos | ListaMusicosPorInstrumento]) :-
  instrumento(_, Instrumento),
  (\+ member(Instrumento, AuxListaInstrumentos)), !,
  instrumento_musicos(Instrumento, ListaMusicos),
  musicos_aux([Instrumento | AuxListaInstrumentos], ListaMusicosPorInstrumento).

musicos_aux(_, []).
```

Informação

Destacar pergunta

Nas questões de escolha múltipla deve escolher apenas uma opção. Uma resposta errada tem um desconto equivalente a 10% da cotação da alínea.

Pergunta 10

Respondida Pontuou 0,700 de 0,700 Destacar pergunta

Dados dois termos Prolog, o algoritmo de unificação:

Selecione uma opção de resposta:

- a. Devolve sempre o termo mais geral que os unifica no caso de serem unificáveis.
- b. [Não Responder]
- c. Falha se um deles for uma variável não instanciada.
- d. Nenhuma das restantes afirmações está correcta.
- e. Retorna sempre o termo unificador dos dois.
- f. Devolve sempre o termo mais específico que os unifica no caso de serem unificáveis.

Pergunta 11

Respondida Pontuou 0,700 de 0,700 Destacar pergunta

Considere um procedimento em Prolog com três cláusulas. Se houver um único cut (!) entre o segundo e o terceiro literal da segunda cláusula, o procedimento:

Selecione uma opção de resposta:

- a. Nenhuma das restantes afirmações está correcta.
- b. [Não Responder]
- c. Falha se houver retrocesso do segundo literal para o primeiro da segunda cláusula.
- d. A terceira cláusula nunca é usada.
- e. Falha se houver retrocesso do terceiro literal para o segundo da segunda cláusula.
- f. Falha se houver retrocesso do terceiro literal para o segundo da primeira cláusula.

Pergunta 12

Respondida Pontuou 0,800 de 0,800 Destacar pergunta

Usando um compilador de Prolog o **Procedimento1**:

```
p(X) :- X = 1, … outros_literais_1 …
p(X) :- X = 2, … outros_literais_2 …
```

É menos eficiente que o **Procedimento2**:

```
p(1) :- … outros_literais_1 …
p(2) :- … outros_literais_2 …
```

(assumindo que "outros_literais_N" são os mesmos nos 2 procedimentos) porque:

Selecione uma opção de resposta:

- a. Não permite o uso de indexação ao primeiro argumento.
- b. [Não Responder]
- c. Nenhuma das restantes afirmações está correcta.
- d. Quando a primeira cláusula é usada o X da segunda cláusula fica logo instanciado e as duas cláusulas comportam-se como se fossem a mesma.
- e. Na cabeça da segunda cláusula deveria ter Y em vez de X.

Pergunta 13

Respondida Pontuou 0,700 de 0,700 Destacar pergunta

Alterar a ordem dos objectivos/literais de uma resolvente :

Selecione uma opção de resposta:

- a. [Não Responder]
- b. Não altera a estrutura da árvore de procura, só a ordem de visita das subárvores.
- c. Nenhuma das restantes afirmações está correcta.
- d. A resolvente é sempre a mesma em todos os nós da árvore de procura e igual ao objectivo inicial.
- e. Pode alterar a estrutura da árvore de procura porque pode instanciar de modo diferente os objectivos subsequentes.
- f. A árvore de procura tem sempre a mesma profundidade.

Pergunta 14

Respondida Pontuou 0,800 de 0,800 Destacar pergunta

Considere que pretende escrever o conhecimento visto nos grupos anteriores da seguinte forma:

```
a ana tocou bach e chopin.
o filipe tocou lizst e beethoven.
o carlos toca piano.
a maria toca violino.
```

Qual a definição de operadores que permite escrever estes factos neste formato?

Selecione uma opção de resposta:

- a.
```
:-op(600, xfx, tocou).
:-op(500, xfy, e).

:-op(500, fy, a).
:-op(500, fy, o).
:-op(600, yfy, toca).
```
- b. Não é possível escrever factos neste formato em Prolog.
- c.
```
:-op(600, xfx, tocou).
:-op(500, xfy, e).

:-op(500, fx, a).
:-op(500, fx, o).
:-op(600, xfx, toca).
```
- d. [Não Responder]
- e.
```
:-op(600, xfx, tocou).
:-op(500, xfy, e).

:-op(500, xf, a).
:-op(500, xf, o).
:-op(600, xfx, toca).
```

Pergunta 15

Respondida Pontuou 0,500 de 0,800 Destacar pergunta

Na sequência da alínea anterior, o que faz o seguinte predicado? Qual o papel do *cut* presente no código? De que cor é? Justifique a sua resposta.

```
pred(X, Y, Z):-
    findall(W, o W toca X, Y), !,
    findall(V, a V toca X, Z).
```

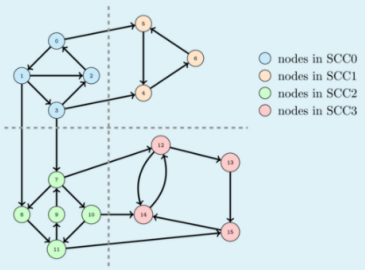
O predicado anterior recebe um instrumento em X, e retorna duas listas, sendo que a lista Y é a lista que contém todos os homens que tocam esse instrumento, e Z é a lista de todas as mulheres que tocam esse instrumento.

O cut é verde, pois este não influencia os resultados do predicado, mas serve para "podar" a árvore de pesquisa, eliminando computações inúteis que não iriam originar em mais soluções, quando se faz backtracking (por exemplo, o utilizador inserir ponto e vírgula para obter uma nova solução).

Informação

Destacar pergunta

Um grafo ou subgrafo é **fortemente conexo** se a partir de qualquer um dos seus vértices se conseguir alcançar qualquer um dos outros vértices. Na imagem abaixo o grafo não é fortemente conexo (por exemplo não se pode ir de 5 para 3) mas tem 4 componentes (subgrafos) que são fortemente conexos (cada componente assinalada a cor diferente na figura).



Assuma que um grafo é codificado em Prolog guardando apenas as suas arestas no formato ***edge(NoOrigem, NoDestino)***. Por exemplo o grafo da imagem seria representado pelos seguintes factos:

```
edge(0,1).    edge(0,5).    edge(1,2).    edge(1,3).
edge(1,8).    edge(2,0).    edge(3,2).    edge(3,4).
edge(3,7).    edge(4,6).    edge(5,4).    edge(6,5).
edge(7,8).    edge(7,10).   edge(7,12).   edge(8,11).
```

```
edge(9,7).          edge(10,11).          edge(10,14).          edge(11,9).
edge(11,15).        edge(12,13).          edge(12,14).          edge(13,15).
edge(14,12).        edge(15,14).
```

Pergunta 16

Respondida Pontuou 1,000 de 1,000 Destacar pergunta

Implemente o predicado ***caminho(NoOrigem, NoDestino)*** que sucede se existir um caminho entre o nó ***NoOrigem*** e o nó ***NoDestino***.

```
caminho(NoInicio, NoDestino) :-
  caminho_aux(NoInicio, NoDestino, [NoInicio]).

caminho_aux(NoAtual, NoDestino, Visitados) :-
  edge(NoAtual, NoDestino).

caminho_aux(NoAtual, NoDestino, Visitados) :-
  edge(NoAtual, NoProx),
  (\+ member(NoProx, Visitados)),
  NoProx \= NoDestino, !,
```

Pergunta 17

Respondida Pontuou 2,000 de 2,000 Destacar pergunta

Implemente o predicado ***fortementeConexo(+ListaDeVertices)*** que sucede se o conjunto de vértices recebido como argumento (que pertence ao grafo guardado na base de dados interna do Prolog) constituir um subgrafo fortemente conexo e falha no caso contrário. Por exemplo, para o grafo completo da figura seria:

```
| ?- fortementeConexo([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]).
no
```

Exemplo para a componente SCC0 assinalada na figura:

```
| ?- fortementeConexo([0,1,2,3]).
yes
```

```
fortementeConexo(ListaDeVertices) :-
  fortementeConexo_aux(ListaDeVertices, []).

fortementeConexo_aux(ListaDeVertices, VerticesVerificados) :-
  select(Vertice, ListaDeVertices, RestoVertices),
  (\+ member(Vertice, VerticesVerificados)), !,
  verificaCaminhosParaTodos(Vertice, RestoVertices),
  fortementeConexo_aux(ListaDeVertices, [Vertice | VerticesVerificados]).

fortementeConexo_aux(_, _).
```

Pergunta 18

Respondida Pontuou 0,000 de 2,500 Destacar pergunta

Implemente em Prolog o predicado ***componentes/1*** que calcula as componentes fortemente conexas maximais (não é possível acrescentar um nó/ramo sem que a componente deixe de ser fortemente conexa) do grafo em memória.
Para o exemplo da figura seria:

```
| ?- componentes(Componentes).
Componentes = [[0,1,2,3],[4,5,6],[7,8,9,10,11],[12,13,14,15]]
yes
```

```
componentes(Componentes) :-
  findall(N, (edge(N, _), edge(_, N)), List),
  sort(List, AllVertices),
  componentes_aux(AllVertices, [], AuxComponentes),
  reverse(AuxComponentes, Componentes).

componentes_aux([Vertice | VerticesQueFaltam], ComponentesAux, Componentes) :-
  findall(OutroVertice, (caminho(Vertice, OutroVertice), ListaVertices),
  retira_vertices_nao_conexos(Vertice, ListaVertices, NovaListaVertices),
  NovoComponente = [Vertice | NovaListaVertices],
```