EI(diurno e PL) **ETI (diurno e PL)** **IGE (diurno e PL)**

Inteligência Artificial 2017-2018

Teste de Prolog. Duração: 1H30

2017/12/21

> Todas as perguntas seguem as convenções do *Prolog*, exatamente como apresentado e exercitado nas aulas. As respostas devem seguir os mesmos padrões. Nas tuas respostas **não terás de usar** nenhum dos predicados definidos em aulas práticas ou teóricas (e.g., *member/2*, *select/3*, *nXList/3*). Se decidires usá-los, terás de os definir.

Conceitos e representação

Este teste gira à volta do conceito de evento. Cada evento é representado por dois componentes: o número de ordem do evento e a sua descrição. Quanto menor for o número de ordem, mais antigo será o evento. A descrição é um termo Prolog.

Os eventos podem ser representados em factos do predicado *event/2* ou como elementos de listas de eventos. Cada evento numa lista é representado por uma estrutura *ev/2* também com dois argumentos.

```
% Exemplo de evento armazenado num facto
% No evento número 29, a Isabel foi catapultada para a fama
event(29, catapultado(isabel, fama)).

% Exemplo de uma lista com dois eventos
[ev(15, esmagado(xi29)), ev(45, dessintonizado(joao))]
```

I - Prolog Declarativo

Nesta secção não poderás os mecanismos de controlo e de Input/Output da linguagem Prolog (e.g., cut, *repeat/0*, *fail/0*, *assert/1*, *read/1*, *write/1*). Também não podes usar os predicados da família *findall*. Em algum ponto de cada uma das tuas respostas, terás de usar recursividade.

(4 Valores) 1 - Define o predicado *event_numbers/2* que recebe uma lista de eventos e produz a lista dos seus números de ordem. A lista devolvida não tem que ficar ordenada. Não faças validações.

Exemplo

```
?- event_numbers([ev(29, catapultado(isabel, fama)), ev(15, esmagado(xi29)),
ev(27, enterrado(luis, inferno))], L).

L = [29, 15, 27]
```

R:

```
event_numbers([ev(N, _)|L1], [N|L2]):-
     event_numbers(L1, L2).
event_numbers([], []).
```

(4 Valores) 2 - Define o predicado *valid_events_list/1* que verifica, em primeiro lugar, se recebe uma lista. Além disso, verifica se cada um dos eventos da lista, se os houver, é um evento válido. Imagina, para isso, que tens já implementado o predicado *is_event/1* que tem sucesso se o seu argumento for um evento válido.

Exemplo

```
?- valid_events_list([ev(29, catapultado(isabel, fama)), ev(15,
esmagado(xi29)), ev(27, enterrado(luis, inferno))]).
```

True

```
?- valid_events_list([]).
```

True

```
?- valid_events_list([1, ev(15, esmagado(xi29))]).
```

False

R:

```
valid_events_list(L):-
     is_list(L),
     valid_e_list(L).

valid_e_list([E|L]):-
     is_event(E),
     valid_e_list(L).
valid_e_list([]).
```

(4 Valores) 3 - Define o predicado *initial_event_number/2* que recebe uma lista de eventos possivelmente desordenada, e devolve o número de ordem do evento mais antigo. Não faças validações.

Exemplo:

```
?- initial_event_number([ev(29, catapultado(isabel, fama)), ev(15,
esmagado(xi29)), ev(27, enterrado(luis, inferno))], Number).

Number = 15
```

R: apresentamos duas resoluções alternativas.

Alt1

```
initial_event_number([ev(N, _)|L], INum):-
      initial_event_number(L, I),
      minimum(N, I, INum).
initial_event_number([ev(Num,_)],Num).

minimum(X, Y, X):-
      X =< Y.
minimum(X, Y, Y):-
      Y < X.
```

Alt2

```
initial_event_number([ev(Num,_)|L],Num):-
      initial_event_number(L,Num2),
      Num < Num2.
initial_event_number([ev(Num,_)|L],Num2):-
      initial_event_number(L,Num2),
      Num >= Num2.
initial_event_number([ev(Num,_)],Num).
```

II - Prolog Procedimental

Nesta secção podes usar os mecanismos de controlo e de Input/Output da linguagem Prolog (e.g., cut, *repeat/0*, *fail/0*, *assert/1*, *read/1*, *write/1*). Em algum ponto de cada uma das tuas respostas, terás de usar um mecanismo de repetição por falha.

(4 Valores) 4 - Imagina que tens um repositório de eventos representados como os seguintes:

```
event(27, enterrado(luis, inferno)).
event(15, esmagado(xi29)).
event(29, catapultado(isabel, fama)).
```

Além disso tens o predicado *proptext/2* (já implementado) que gera a descrição em língua natural de uma proposição, por exemplo:

```
?-proptext(event(27, enterrado(luis, inferno)), Text).
Text ='No evento 27, o Luis foi enterrado no inferno'
```

Escreve o procedimento *tell_events/2* que imprime no ecrã a descrição dos eventos ocorridos entre dois números de ordem inclusive, por exemplo.

```
?-tell_events(15, 28).
No evento 27, o Luis foi enterrado no inferno
No evento 15, o asqueroso xi29 foi esmagado
```

Não faças validações. A ordem pela qual surgem as descrições dos eventos não é relevante.

R:

```
tell_events(N1, N2):-
      event(N, D),
      N >= N1, N =< N2,
      proptext(event(N, D), Text),
      write(Text), nl,
      fail.
tell_events(_, _).
```

(4Valores) 5 - Escreve o procedimento *store_events/0* que regista (*assert/1*), em factos *event/2*, eventos especificados pelo utilizador através do teclado. Por cada evento, é pedida apenas a sua descrição computacional. O número de ordem é gerado sequencialmente pelo programa. Quando, em vez da descrição de um evento, o utilizador introduz a palavra *end*, o processo termina. Ao terminar, o programa deve manter os factos com os eventos, mas deve apagar toda a restante informação criada temporariamente em memória.

Exemplo

```
?- store_events.
Description (or end): esmagado(ix200).
Description (or end): aparafusado(rita, c606).
Description (or end): end.

?- event(N, D).
N = 1 D = esmagado(ix200);
N = 2 D = aparafusado(rita, c606);
False
```

Não faças validações.

Nota: **ADMITE PARA SIMPLIFICAR QUE, SEMPRE QUE SE INICIA A EXECUÇÃO DO PROGRAMA, NÃO EXISTEM EVENTOS JÁ ARMAZENADOS. ASSIM SENDO, O PRIMEIRO EVENTO DEVE SER O NÚMERO 1.**

R:

```
store_events:-
      assert(event_number(1)),
      repeat,
            read_event(D),
            process_event(D), !,
      retract(event_number(_)).

read_event(D):-
      write('Description (or end): '),
      read(D).
```

```
process_event(end):- !.
process_event(D):-
      retract(event_number(N))
      assert(event(N, D)),
      N1 is N + 1,
      assert(event_number(N1)),
      !,
      fail.
```