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

Inteligência Artificial 2017-2018

1º Exame

2018/01/26, 02:00H

Nota: nos exercícios que se seguem, a lógica e as ferramentas computacionais utilizadas funcionam exatamente como explicado nas aulas; as respostas devem seguir os mesmos padrões.

I – Perguntas avulsas

Responde às seguintes perguntas sem justificar.

(1 Valor) 1a) Diz os nomes dos três componentes da arquitetura geral dos Sistemas Baseados em Conhecimento.

R: Base de Conhecimento, Motor de Inferência e Interface

(1 Valor) 1b) As redes neuronais são uma tecnologia simbólica ou não simbólica? **`[Uma resposta errada desconta 0.5 Valores]`**

R: Não simbólica

(0.5 Valores) 1c) Como se chama o conjunto de regras de produção que, num dado momento, têm a condição satisfeita.

R: Conjunto de conflito

II – Lógica de Predicados

Considera os seguintes predicados

TemCalcanharViscoso(x)	*x* tem o calcanhar viscoso
TemComichãoCerebral(x)	*x* tem comichão cerebral
Atendimento(x, a)	*a* é o tipo de atendimento para *x*

(2.5 Valores) 2 – Considera a seguinte base de conhecimento em forma clausal:

1. {¬TemCalcanharViscoso(x), Atendimento(x, LavagemPedonal)}
2. {¬TemComichãoCerebral(x), Atendimento(x, LavagemCerebral)}

Mostra, por refutação, que a seguinte proposição se deriva da base de conhecimento:

$\forall x\ [(\text{TemCalcanharViscoso}(x) \vee \text{TemComichãoCerebral}(x)) \Rightarrow \exists y\ \text{Atendimento}(x, y)]$

R:

Para provar por refutação, nega-se o objetivo e converte-se o resultado para forma clausal. Posteriormente, mostra-se que as cláusulas da base de conhecimento juntamente com as do objetivo negado formam um conjunto contraditório.

Conversão do objetivo negado para forma clausal:

$\neg\forall x\ [(\text{TemCalcanharViscoso}(x) \vee \text{TemComichãoCerebral}(x)) \Rightarrow \exists y\ \text{Atendimento}(x, y)]$

1. Substituir implicações por disjunções: $(A \Rightarrow B) \equiv (\neg A \vee B)$

   $\neg\forall x\ [\neg(\text{TemCalcanharViscoso}(x) \vee \text{TemComichãoCerebral}(x)) \vee \exists y\ \text{Atendimento}(x, y)]$

2. Mover negações para os literais
   $(\neg\forall x\ A) \equiv (\exists x\ \neg A)$

   $\exists x\ \neg[\neg(\text{TemCalcanharViscoso}(x) \vee \text{TemComichãoCerebral}(x)) \vee \exists y\ \text{Atendimento}(x, y)]$

   $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$
   $\exists x\ [\neg\neg(\text{TemCalcanharViscoso}(x) \vee \text{TemComichãoCerebral}(x)) \wedge \neg\exists y\ \text{Atendimento}(x, y)]$
   $\neg\neg A \equiv A$
   $\exists x\ [(\text{TemCalcanharViscoso}(x) \vee \text{TemComichãoCerebral}(x)) \wedge \neg\exists y\ \text{Atendimento}(x, y)]$
   $(\neg\exists x\ A) \equiv (\forall x\ \neg A)$

   $\exists x\ [(\text{TemCalcanharViscoso}(x) \vee \text{TemComichãoCerebral}(x)) \wedge \forall y\ \neg\text{Atendimento}(x, y)]$

3. Skolemização

   $(\text{TemCalcanharViscoso}(SK) \vee \text{TemComichãoCerebral}(SK)) \wedge \forall y\ \neg\text{Atendimento}(SK, y)$

4. Remover os quantificadores universais

   $(\text{TemCalcanharViscoso}(SK) \vee \text{TemComichãoCerebral}(SK)) \wedge \neg\text{Atendimento}(SK, y)$

5. Obter uma conjunção de disjunções de literais

   N/A porque já temos uma proposição nesse formato

6. Escrever as cláusulas

   $\{\text{TemCalcanharViscoso}(SK), \text{TemComichãoCerebral}(SK)\}$
   $\{\ \neg\text{Atendimento}(SK, y)\ \}$

Mostrar que a reunião das cláusulas da base de conhecimento com as do objetivo negado constitui um conjunto contraditório:

1.	$\{\neg\text{TemCalcanharViscoso}(x), \text{Atendimento}(x, \text{LavagemPedonal})\}$	Δ
2.	$\{\neg\text{TemComichãoCerebral}(x), \text{Atendimento}(x, \text{LavagemCerebral})\}$	Δ
3.	$\{\text{TemCalcanharViscoso}(SK), \text{TemComichãoCerebral}(SK)\}$	$\neg$Obj
4.	$\{\ \neg\text{Atendimento}(SK, y)\ \}$	$\neg$Obj
5.	$\{\neg\text{TemCalcanharViscoso}(SK)\}$	1, 4
6.	$\{\neg\text{TemComichãoCerebral}(SK)\}$	2, 4
7.	$\{\text{TemComichãoCerebral}(SK)\}$	3, 5
8.	$\{\}$	6, 7

(2.5 Valores) 3 – Usando apenas os predicados descritos na tabela, representa, na linguagem da Lógica de Predicados de Primeira Ordem, o seguinte conhecimento apresentado informalmente:

> *Seja qual for o x, se não existe pelo menos um tipo de atendimento para esse x, então x não tem nem comichão cerebral e não tem o calcanhar viscoso.*

R: Existem várias representações superficialmente alternativas mas que, na realidade, são

idênticas. Aqui estão duas.

ALT 1

$\forall x\ [(\neg\exists y\ \text{Atendimento}(x, y)) \Rightarrow (\neg\text{TemCalcanharViscoso}(x) \wedge \neg\text{TemComichãoCerebral}(x))]$

ALT 2

$\forall x\ [(\text{TemCalcanharViscoso}(x) \vee \text{TemComichãoCerebral}(x)) \Rightarrow \exists y\ \text{Atendimento}(x, y)]$

III – Regras de Produção

Considera um sistema computacional no âmbito de decisões de investimentos na Bolsa. As decisões de investimento dependem da tendência de subida ou de descida dos vários produtos disponíveis. Para o efeito, além de diversa outra informação, admite que existem factos do predicado *produto/4* que descrevem a tendência mais recente de variação de cada produto.

produto(Id, ValorInicial, ValorAtual, Tempo): o produto com a identificação *Id* manifesta uma tendência para subir / descer cuja duração é *DuraçãoTendência* (em número de dias). O valor do produto no início da tendência era *ValorInicial* e o valor final da tendência é *ValorAtual*. Se *ValorAtual* > *ValorInicial*, trata-se de uma tendência de subida; se *ValorAtual* < *ValorInicial*, trata-se de uma tendência de descida. Os factos do predicado *produto/4* existem no sistema de informação da bolsa.

Exemplos:

`produto(google, 1042.68, 1068.52, 3).` A Google manifesta uma tendência de 3 dias para subir. No primeiro dia da tendência, cada ação valia 1042.68. Ao fim desses três dias, cada ação vale 1068.52.

`produto(ctt, 3.7, 3.2, 4).` Os CTT manifestam uma tendência de descida de 4 dias. No início, as ações valiam 3.7. Atualmente, valem 3.2.

`produto(msft, 84.56, 82.67, 5).` Tendência de descida da Microsoft.

`produto(amzn, 1109.15, 1133.35, 4).` Tendência de subida da Amazon.

(2.5 Valores) 4 – Escreve uma regra que imprime, no ecrã do computador, os produtos com uma tendência de subida ou de descida de N ou mais dias. Os produtos não têm de ser listados por nenhuma ordem especial. **O número de dias N é passado ao sistema através do seu programa de interface *listar_tendencias/1*, o qual cria o facto *ndias/1* correspondente, chama o PSys, e volta a remover o facto *ndias/1* criado**. Exemplo:

```
?- listar_tendencias(4).
amzn: valor inicial: 1109.15 valor atual: 1133.35 duração: 4
ctt: valor inicial: 3.7 valor atual: 3.2 duração: 4
msft: valor inicial: 84.56 valor atual: 82.67 duração: 5
```

Na resposta, podes usar todos os recursos da tecnologia baseada em Prolog, por exemplo os predicados >, >=, <, =<, e *is*, e as ações *assert* e *rectract*. Além disso, apenas podes usar os predicados e ações descritos seguidamente.

Predicados

ndias(N): Facto de controlo. Significa que pretendemos conhecer tendências que se mantenham há pelo menos *N* dias. Facto criado e posteriormente removido pelo programa de interface listar_tendencias(N)

produto_listado(Id): Facto de controlo. Significa que o produto identificado por *Id* foi já listado no ecrã, na interação atual. Este facto deve ser criado explicitamente pela tua regra.

Ações

listar_produto(Id, ValorInicial, ValorAtual, Tempo): Esta ação, usada na pergunta 4, limita-se a apresentar, no ecrã, os valores das variáveis *ValorInicial*, *ValorAtual*, e *Tempo* correspondentes ao produto identificado por *Id*. Esta ação não cria factos de controlo.

R:

```
if    (ndias(N) and produto(Id, ValorInicial, ValorAtual, Tempo)
       and Tempo >= N and \+ produto_listado(Id))
then (assert(produto_listado(Id)),
       listar_produto(Id, ValorInicial, ValorAtual, Tempo)).
```

(2.5 Valores) 5 – Sabendo que o sistema dispõe dos factos *produto/4* relativos à data anterior e dos factos *valor_de_fecho/3* de cada ação, na data atual, escreve um conjunto de regras de produção para atualizar os factos *produto/4*, para todas as ações, na data atual.

Escreve apenas uma regra correspondente a cada uma das duas seguintes situações:

> *Se a tendência anterior é de subida e o valor de fecho é maior do que o valor final da tendência, então alterar o facto produto/4 de modo a aumentar o número de dias da tendência; o novo valor atual da tendência passa a ser o valor de fecho.*
>
> *Se a tendência anterior é de subida e o valor de fecho é menor do que o valor final da tendência, então alterar o facto do predicado produto/4 de modo que a duração da nova tendência (de descida) seja um dia, o novo valor inicial seja o valor final do facto original, e o novo valor atual seja o valor de fecho.*

Na resposta, podes usar todos os recursos da tecnologia baseada em Prolog, por exemplo os predicados >, >=, <, =<, e *is*, e as ações *assert* e *rectract*. Além disso, apenas podes usar os predicados descritos seguidamente.

valor_de_fecho(Id, Data, Valor): Na hora do fecho da bolsa da data especificada (*Data*), o produto identificado por *Id* tinha o valor *Valor*. Os factos do predicado *valor_de_fecho/3* existem no sistema de informação da bolsa.

tendencia_atualizada(Id, Data): Facto de controlo. Significa que o facto do predicado *produto/4*, relativo ao produto identificado por *Id*, foi atualizado com a informação do valor de fecho na data especificada (*Data*). Este facto deve ser criado explicitamente pelas tuas regras.

R:

```
if   (produto(Id, VInicial, VAtual, Tempo) and
       VAtual > VInicial and
       valor_de_fecho(Id, Data, Valor) and
       \+ tendencia_atualizada(Id, Data) and
       Valor > VAtual and
       Tempo1 is Tempo + 1)
then (retract(produto(Id, VInicial, VAtual, Tempo)),
       assert(produto(Id, VInicial, Valor, Tempo1)),
       assert(tendencia_atualizada(Id, Data))).

if   (produto(Id, VInicial, VAtual, Tempo) and
       VAtual > VInicial and
       valor_de_fecho(Id, Data, Valor) and
       \+ tendencia_atualizada(Id, Data) and
       Valor < VAtual)
then (retract(produto(Id, VInicial, VAtual, Tempo)),
       assert(produto(Id, VAtual, Valor, 1)),
       assert(tendencia_atualizada(Id, Data))).
```

IV – Prolog declarativo

Neste grupo não podes usar nem *cut*, nem *repeat*, nem *assert*, nem *retract*, nem nenhum outro mecanismo de controlo da linguagem Prolog.

Considera termos que representam tarefas. Cada termo é uma estrutura com o functor *tarefa* e dois argumentos: nome da tarefa e duração (em dias), por exemplo tarefa(preparar_aulas, 1).

(2.5 Valores) 6 – Escreve o predicado *tempo_total/2* que recebe uma lista de tarefas e determina a soma das suas durações, por exemplo

```
?- tempo_total([tarefa(t1,1), tarefa(t2,10), tarefa(t3,4)], T).
T = 15
```

O predicado deve verificar que o primeiro argumento é uma lista de tarefas e que o segundo argumento é uma variável ou um número. O primeiro argumento de um termo *tarefa* é um átomo e o segundo é um número. Em alguma passagem da tua resposta, tens de usar recursividade

R:

```
tempo_total(L, T):-
     is_list(L), (var(T); number(T)),
     soma_dos_tempos(L, T).

soma_dos_tempos([tarefa(Nome, Tempo) | Tarefas], T):-
     atom(Nome), number(Tempo),
     soma_dos_tempos(Tarefas, T1),
     T is T1 + Tempo.
soma_dos_tempos([], 0).
```

(2.5 Valores) 7 – Escreve o predicado recursivo *tarefa_mais_demorada/2* que recebe uma lista de termos tarefa e determina o termo tarefa com maior duração, por exemplo

```
?- tarefa_mais_demorada([tarefa(t1,1), tarefa(t2,10),
tarefa(t3,4)], T).

T = tarefa(t2, 10)
```

Não faças validações dos dados recebidos pelo predicado. Para simplificar admite que as tarefas têm durações diferentes.

Sugestão não obrigatória: escreve um predicado que recebe dois termos tarefa e devolve aquele com maior duração.

R: Apresentam-se duas resoluções alternativas.

ALT 1

```
tarefa_mais_demorada([T|Tarefas], Mais):-
     tarefa_mais_demorada(Tarefas, Tmp),
     tmais(T, Tmp, Mais).
tarefa_mais_demorada([Tarefa], Tarefa).

tmais(tarefa(T1, D1), tarefa(T2, D2), tarefa(T1, D1)):-
     D1 > D2.
tmais(tarefa(T1, D1), tarefa(T2, D2), tarefa(T2, D2)):-
     D2 > D1.
```

ALT 2

```
tarefa_mais_demorada([tarefa(Tarefa,Tempo)|L],tarefa(Tarefa,Tempo)):-
     tarefa_mais_demorada(L, tarefa(Tarefa2,Tempo2)),
     Tempo2 < Tempo.

tarefa_mais_demorada([tarefa(Tarefa,Tempo)|L],tarefa(Tarefa2,Tempo2))
:-
     tarefa_mais_demorada(L, tarefa(Tarefa2,Tempo2)),
     Tempo2 > Tempo.

tarefa_mais_demorada([tarefa(Tarefa,Tempo)],tarefa(Tarefa,Tempo)).
```

V – Prolog procedimental

Neste grupo não podes usar recursividade; tens de usar obrigatoriamente o mecanismo adequado de repetição por falha. Respostas com a escolha errada do mecanismo de repetição terão classificação nula.

Considera termos que representam tarefas. Cada termo é uma estrutura com o functor *tarefa* e dois argumentos: nome da tarefa e duração (em dias), por exemplo tarefa(preparar_aulas, 1).

(2.5 Valores) 8 – Escreve o procedimento *armazenar_tarefas/0* para obter, do utilizador, dados sobre tarefas (nome e duração) e criar factos (*assert/1*) com as tarefas introduzidas. O

procedimento termina quando o nome da tarefa introduzida pelo utilizador for *fim*.

```
?- armazenar_tarefas.

Nome: t1.
Duração: 1.

Nome: t2.
Duração: 10.

Nome: fim.

?-
```

Admite que já existe o procedimento *ler_tarefa/2* que obtém, do utilizador, o nome e a duração de uma tarefa. *ler_tarefa/2* está preparado para não pedir a duração, quando a designação da tarefa for *fim*.

Não faças validações dos dados processados pelo procedimento.

R:

```
armazenar_tarefas:-
     repeat,
      ler_tarefa(Nome, Duracao),
      processar_tarefa(Nome, Duração), !.

processar_tarefa(fim, _):- !.
processar_tarefa(Nome, Duração):-
     assert(tarefa(Nome, Duração)),
     fail.
```