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

# Inteligência Artificial 2016/2017

## Exame 1 – 2017/01/10

Lê cuidadosamente as instruções desta prova feita em moldes não habituais.

O Exame compõe-se de quatro módulos: um de perguntas obrigatórias e um de perguntas facultativas da primeira parte da matéria (*SBC*); e um de perguntas obrigatórias e outro de perguntas facultativas relativos à segunda parte da matéria (*Prolog*). Podes escolher os módulos que necessitares e achares convenientes para passar ou para melhorar a nota. Para obter aprovação na cadeira é forçoso obter, pelo menos, 9.5 em cada um dos módulos obrigatórios das duas partes da matéria. O módulo facultativo de uma parte da matéria só conta se o módulo obrigatório da matéria correspondente tiver sido feito com sucesso (na data atual ou anteriormente).

Nas tuas respostas, podes usar (sem definir) qualquer recurso das ferramentas usadas, desde que esse recurso tenha sido dado nas aulas, exceto se o enunciado diga que não podes usar. Os seguintes são exemplos desses recursos que podes usar: *write/1*, *read/1*, *member/2*, *append/3*, e *select/3* do Prolog; e *writelist/1* do PSys.

# SBC: Grupo 1 – Perguntas obrigatórias (15 Minutos)

1 – Aplica o primeiro passo da conversão para forma clausal à proposição

$$\forall x[(\text{Amigo(Isabel, x)} \wedge \text{Amigo(Luis, x)}) \Rightarrow \text{Amigo(Lúcifer, x)}]$$

**R:**

$$\forall x\ [\neg(\text{Amigo(Isabel, x)} \wedge \text{Amigo(Luis, x)}) \vee \text{Amigo(Lúcifer, x)}]$$

2 – Representa a seguinte cláusula em *Prolog*, tendo o cuidado de fazer as alterações necessárias para respeitar as convenções de minúsculas e maiúsculas da linguagem (importante).

{¬Amigo(Isabel, x), ¬Amigo(Luis, x), Amigo(Lúcifer, x)}

**R:**

```
amigo(lucifer, X):- amigo(isabel, X), amigo(luis, X).
```

3 – Considera o predicado *trabalhador/3*, definido através de factos como os seguintes

```
trabalhador(dária, desenvolvimento, aptoide).
trabalhador(margarida, recursos_humanos, aptoide).
trabalhador(nuno, desenvolvimento, google).
```

Usando apenas o predicado *trabalhador/3*, define o predicado *trabalhador_empresa/2*, tal

trabalhador_empresa(Trab, Empr) significa que *Trab* é um trabalhador da empresa *Empr*. O seguinte seria um exemplo de interação com o predicado:

```
?- trabalhador_empresa(X, E).
X = daria       E = aptoide;
X = margarida   E = aptoide
```

**R:**

```
trabalhador_empresa(T, Org):-
     trabalhador(T, _, Org).
```

4 – Numa determinada unidade de investigação científica existe um sistema computacional para ajudar a gerir resultados de investigação. Numa das suas funcionalidades, o sistema envia mensagens de *email* à Direção, de cada vez que um membro da unidade obtém uma classificação na tese de mestrado superior a um determinado limite. Usando a ferramenta PSys, escreve a regra de produção que implementa esta funcionalidade. A regra deve ser feita de forma que a mensagem de *email* só é enviada uma vez por cada tese.

Para além dos predicados da ferramenta PSys (e.g., >, >=, =), só podes usar a ação e os factos que se descrevem seguidamente.

| | |
|---|---|
| informar_direçao(X, N, T) | O sistema envia uma mensagem de email à Direção, dizendo que o investigador X obteve a nota N na sua tese de mestrado com o título T. Além disso cria o facto de controlo msg_enviada(X). |

| | |
|---|---|
| tese_feita(X, N, T) | X obteve a nota N na sua tese de mestrado, intitulada T. |
| msg_enviada(X) | Resulta da ação informar_direçao(X, N, T). Significa que o sistema enviou uma mensagem de *email* à Direção, informando sobre a tese de mestrado do investigador X. |
| min(Nota) | Valor da classificação da tese de mestrado, a partir do qual deve ser enviada uma mensagem à direção |

**R:**

```
if   (tese_feita(X, Nota, T) and min(Min) and Nota > Min and
      \+ msg_enviada(X))
then informar_direcao(X, Nota, T).
```

# SBC: Grupo 2 – Perguntas facultativas (20 Minutos)

Responde apenas a uma das perguntas que se apresentam seguidamente.

1 – Numa empresa, há um sistema computacional de gestão dos resultados obtidos pelos vários trabalhadores da empresa. Uma das funcionalidades do sistema consiste em atribuir um bónus salarial aos vendedores que tenham vendido todos os produtos que lhes tenham sido atribuídos para venda. Para esses vendedores, o valor do bónus é calculado em função do número de produtos vendidos. Os vendedores a que não tenham sido atribuídos produtos para venda, ou os que não tenham vendido todos os produtos que lhes tenham sido atribuídos para venda, não têm bónus (i.e., bónus 0).

Usando a ferramenta *PSys*, escreve o conjunto de regras de produção necessárias para as funcionalidades que se descrevem seguidamente, as quais são uma parte substancial da gestão de bónus salariais:

1. Criar o facto bónus(X, 0) para cada vendedor X que não tenha recebido produtos para venda
2. Criar o facto bónus(X, 0) para cada vendedor X que não tenha vendido todos os produtos que lhe tenham sido atribuídos para venda

*[O enunciado impõe que implementes as duas seguintes funcionalidades em separado]*

3. Criar o objetivo produtos_contabilizados(X) para cada vendedor X a quem tenha sido atribuído pelo menos um produto para venda, e que tenha vendido todos os produtos que lhes tenham sido atribuídos
4. Para cada trabalhador X, para o qual exista o objetivo produtos_contabilizados(X), contar os produtos vendidos, criar o facto bónus(X, N), em que N é o número de produtos vendidos, e remover o objetivo produtos_contabilizados(X).

É fundamental assegurar que o sistema não entre em ciclos indefinidos. Por exemplo, o sistema não deve voltar a criar o objetivo produtos_contabilizados(X) para o trabalhador *X* depois de o ter removido.

Na tua resolução podes recorrer aos recursos da ferramenta PSys (e.g., *assert/0*), e ainda às ações e predicados que se descrevem seguidamente.

**Predicados**

| **Predefinidos** | |
|---|---|
| vendedor(V) | V é um vendedor |
| produto(V, P) | P é um produto atribuído ao vendedor V para venda |
| produto_vendido(V, P) | O vendedor V vendeu o produto P |
| nprodutos_vendidos(V, N) | O vendedor V vendeu N produtos. |
| **Factos de controlo** | |
| objetivo(G) | O sistema tem o objetivo de atingir um estado em que o objetivo G tenha sido satisfeito, por exemplo objetivo(produtos_contabilizados(V)). |
| bonus(V, N) | O vendedor V tem um bónus salarial que será calculado com base na venda de N produtos. O facto bónus(V, 0) significa que o vendedor não terá direito a bónus salarial. |

**Ações**

| | |
|---|---|
| remover_objetivo(G) | O sistema remove o primeiro facto que emparelha com objetivo(G). |
| criar_objetivo(G) | O sistema cria o facto objetivo(G). |

**R:**

```
% Se o vendedor tiver vendido todos os produtos
% Se não existir o objetivo de calcular o bónus
% Se ainda não houver o facto bónus/2
% Então criar o objetivo de contar o número de produtos vendidos
if   (vendedor(V) and produto(V, _) and
      \+( (produto(V, P) and \+ produto_vendido(V, P)) ) and
     \+ objetivo(produtos_contabilizados(V)) and \+ bonus(V, _))
then criar_objetivo(produtos_contabilizados(V)).

% Se o vendedor não tiver vendido todos os produtos
% Se o facto bónus/2 ainda não estiver calculado
% Então criar o facto com bónus(V, 0)
if   (vendedor(V) and produto(V, P) and
      \+ produto_vendido(V, P) and \+ bonus(V, _))
then assert(bonus(V, 0)).
```

```
% Se não tiverem sido atribuídos produtos para venda ao vendedor,
% Então criar o facto com bónus(V, 0)
if   (vendedor(V) and \+produto(V, _) and \+ bonus(V, _))
then assert(bonus(V, 0)).

% Contabilizar produtos vendidos para os vendedores com direito a bonus
if   (vendedor(V) and objetivo(produtos_contabilizados(V)) and
      \+ bonus(V, _) and nprodutos_vendidos(V, N))
then (assert(bonus(V, N)), remover_objetivo(produtos_contabilizados(V))).
```

2 – Resolve os seguintes exercícios de lógica:

*(3 Valores)* a) Recorrendo apenas aos predicados do domínio descritos de seguida, representa o seguinte conhecimento em Lógica de Predicados de Primeira Ordem:

> *Se x é um vendedor e x vendeu todos os produtos que lhe foram atribuídos para vender, então x é um super vendedor.*

| | |
|---|---|
| Vendedor(v) | v é um vendedor |
| SuperVendedor(v) | v é um super vendedor |
| Produto(v, p) | p é um produto atribuído a v para ser vendido |
| ProdutoVendido(v, p) | p foi vendido por v |

*(5 Valores)* Partindo de

1. $\forall v[\text{Vendedor}(v) \Rightarrow \exists p\ (\text{Produto}(v, p) \wedge \text{ProdutoVendido}(v, p))]$
2. $\forall p\ \neg\text{ProdutoVendido}(\text{Álvaro}, p)$

mostra que o Álvaro não é vendedor, recorrendo aos dois passos que se seguem:

(i) Mostra por refutação que, da proposição $\forall p\ \neg\text{ProdutoVendido}(\text{Álvaro}, p)$, se deriva o objetivo $\neg\exists p\ (\text{Produto}(\text{Álvaro}, p) \wedge \text{ProdutoVendido}(\text{Álvaro}, p))$

(ii) Depois, usando a BC original, $\neg\exists p\ (\text{Produto}(\text{Álvaro}, p) \wedge \text{ProdutoVendido}(\text{Álvaro}, p))$, derivada em (i), e as regras de inferência da lógica de predicados de primeira ordem, no seu formato habitual, mostra que o Álvaro não é um vendedor.

Para a tarefa (i), assume os seguintes resultados:

| **Formato habitual** | **Forma clausal** |
|---|---|
| $\forall x\ \neg P(x)$ | $\{\neg P(x)\}$ |
| $\neg\neg\exists x\ (P1(x) \wedge P2(x))$ | $\{P1(SK)\}\ \{P2(SK)\}$ |

**R:**

a) $\forall v[(\text{Vendedor}(v) \wedge \forall x(\text{Produto}(v, x) \Rightarrow \text{ProdutoVendido}(v, x))) \Rightarrow \text{SuperVendedor}(v)]$

Ou

$\forall v[(\text{Vendedor}(v) \wedge \neg\exists x(\text{Produto}(v, x) \wedge \neg\text{ProdutoVendido}(v, x))) \Rightarrow \text{SuperVendedor}(v)]$

b) O Álvaro não é um vendedor $\neg\text{Vendedor}(\text{Álvaro})$

*(3 Valores)* Mostrar por refutação que, da proposição ∀p ¬ProdutoVendido(Álvaro, p), se deriva ¬∃p (Produto(Álvaro, p) ∧ ProdutoVendido(Álvaro, p))

*(2 Valores)* Converter a única proposição da base de conhecimento para forma clausal:

De acordo com a tabela de conversões do enunciado, a conversão de ∀p ¬ProdutoVendido(Álvaro, p) resulta na cláusula

{¬ProdutoVendido(Álvaro, p)}

Objetivo: ¬∃p (Produto(Álvaro, p) ∧ ProdutoVendido(Álvaro, p))

Objetivo negado: ¬¬∃p (Produto(Álvaro, p) ∧ ProdutoVendido(Álvaro, p))

Converter o objetivo negado para forma clausal:

De acordo com a tabela de conversões do enunciado, a conversão de ¬¬∃p (Produto(Álvaro, p) ∧ ProdutoVendido(Álvaro, p)) resulta nas cláusulas

{Produto(Álvaro, SK)}

{ProdutoVendido(Álvaro, SK)}

*(1 Valor)* Refutação:

| | | |
|---|---|---|
| 1. | {¬ProdutoVendido(Álvaro, p)} | Δ |
| 2. | {Produto(Álvaro, SK)} | ¬Obj |
| 3. | {ProdutoVendido(Álvaro, SK)} | ¬Obj |
| 4. | {} | 1, 3 |

*(2 Valores)* Mostrar que se deriva ¬Vendedor(Álvaro)

| | | |
|---|---|---|
| 1. | ∀v[Vendedor(v) ⇒∃p (Produto(v, p) ∧ ProdutoVendido(v, p))] | Δ |
| 2. | ∀p ¬ProdutoVendido(Álvaro, p) | Δ |
| 3. | ¬∃p (Produto(Álvaro, p) ∧ ProdutoVendido(Álvaro, p)) | 2 por refutação (i) |
| 4. | Vendedor(Álvaro) ⇒∃p (Produto(Álvaro, p) ∧ ProdutoVendido(Álvaro, p)) | 1 UI |
| 5. | ¬Vendedor(Álvaro) | 4, 3 MT |

# Prolog: Grupo 1 – Perguntas obrigatórias (15 Minutos)

1 – Sabendo que

N! = N × (N-1)!, para N > 0
0! = 1

define recursivamente o predicado *fatorial/2*. Esta tem de ser uma definição totalmente declarativa. Não faças validações.

**R:**

```
fatorial(N, F):-
      N > 0,
      N1 is N - 1,
      fatorial(N1, F1),
      F is N * F1.
fatorial(0, 1).
```

2 – Define recursivamente o predicado *soma/2*, tal que soma(L, S) significa que S é a soma dos elementos de uma lista. Esta tem de ser uma definição totalmente declarativa. Não faças validações.

**R:**

```
soma([X|L], S):-
      soma(L, S1),
      S is X + S1.
soma([], 0).
```

3 – Admite que o predicado *excecional/1* é usado para manter uma base de dados de pessoas excecionais, por exemplo:

```
excecional(gandhi).
excecional(mandela).
excecional(marie_curie).
```

Escreve uma interação, na linha de comando do interpretador de Prolog, que produz a lista de todas as pessoas excecionais, por exemplo a lista `[gandhi, mandela, marie_curie]`.

**R:**

```
?- findall(X, excecional(X), L).

L = [gandhi, mandela, marie_curie]
```

4 – O seguinte programa, quando completo, soma os valores numéricos armazenados em factos do predicado *num/1*. No final, o programa devolve a soma dos números lidos.

```
somar_numeros(_) :-
      assert(soma(0)),
```

```
somar_numeros(Soma):-
      retract(soma(Soma)).
```

```
atualizar_soma(X):-
      retract(soma(S)),
      S1 is S + X,
      assert(soma(S1)),
      !.
```

Completa a primeira cláusula de *somar_numeros/1*. A resolução tem de recorrer obrigatoriamente a um mecanismo de repetição por falha.

**R:**

```
somar_numeros(_) :-
      assert(soma(0)),
      num(N),
      atualizar_soma(N),
      fail.
```

# Prolog: Grupo 2 – Perguntas facultativas (20 Minutos)

Responde apenas a uma das perguntas que se apresentam seguidamente.

1 – Escreve um procedimento que imprime, no monitor do computador, todas as salas de um cinema com lotação esgotada. No final, o procedimento devolve o número de salas imprimidas. A definição deste procedimento tem que se basear obrigatoriamente num mecanismo de repetição por falha.

Para tal, define o predicado *lotação_esgotada/1* que devolve, por *backtracking*, as salas com todos os lugares vendidos. Na definição do predicado *lotação_esgotada/1*, não podes usar nem asserts nem retracts.

Não faças validações.

As salas do cinema estão descritas através de factos dos seguintes predicados

| | |
|---|---|
| sala/1 | sala(X) significa que X é uma sala, por exemplo sala(1). |
| lugar/2 | lugar(Sala, Lugar) significa que Lugar é um lugar da sala Sala, por exemplo lugar(1, j10). |
| lugar_vendido/2 | lugar_vendido(Sala, Lugar) significa que o lugar Lugar da sala Sala foi vendido. |

**R:**

```
lotacão_esgotada(Sala):-
      sala(Sala),
      \+( (lugar(Sala, Lugar), \+lugar_vendido(Sala, Lugar)) ).

salas_esgotadas(_):-
      assert(contagem(0)),
      lotacão_esgotada(S),
      atualiza_contagem,
      write((S:esgotada)), nl,
      fail.
salas_esgotadas(N):-
      retract(contagem(N)).

atualiza_contagem :-
      retract(contagem(N)),
      N1 is N + 1,
      assert(contagem(N1)),
      !.
```

2 – Escreve o predicado *matching_pages/2*, tal que matching_page(Words, Pages) significa que *Pages* é o conjunto de páginas do documento considerado, cada uma das quais contém todas as palavras da lista *Words*.

*matching_pages/2* necessita de uma tabela (chamada *índice remissivo*) que, para cada palavra existente no documento, lista o conjunto das páginas que a contêm, por exemplo

```
word_pages(prolog, [1, 2, 5, 7, 10, 11]).
word_pages('IA', [1, 2, 3, 5, 6, 15, 16]).
……
```

Com base nesta tabela (que não tens de definir), podemos compreender a seguinte interação:

```
?- matching_pages([prolog, 'IA'], Pages).
Pages = [1, 2, 5]
```

Este resultado é a interseção do conjunto das páginas que contêm a palavra ‘IA’ com o conjunto de páginas que contêm a palavra prolog. matching_pages(Words, Pages) começa por determinar a lista dos conjuntos de páginas que contêm cada uma das palavras especificadas em *Words*. Seguidamente, calcula o resultado final através da interseção dos conjuntos dessa lista. Para fazer a interseção de uma lista de conjuntos, faz-se a interseção do primeiro conjunto da lista com o conjunto que resulta da interseção do resto da lista de conjuntos.

Para fazer a interseção de dois conjuntos, usa o predicado *set_intersection/3* sem o definir. Exemplo de interação com *set_intersection/3*:

```
?- set_intersection([a, b, c, d], [a, c, e, f], Set).
Set = [a, c]
```

Toda a resolução tem de ser totalmente declarativa. Também não podes usar nenhum dos predicados da família *findall*. Não faças validações.

**R:**

```
matching_pages(Words, Pages):-
      set_of_sets_of_word_matching_pages(Words, Sets),
      intersection_of_set_of_sets(Sets, Pages).

set_of_sets_of_word_matching_pages([W|Words], [Pages|Sets]):-
      word_pages(W, Pages),
      set_of_sets_of_word_matching_pages(Words, Sets).
set_of_sets_of_word_matching_pages([], []).

intersection_of_set_of_sets([S1|Sets], Set):-
      intersection_of_set_of_sets(Sets, S2),
      set_intersection(S1, S2, Set).
intersection_of_set_of_sets([Set], Set).
```