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

Inteligência Artificial 2013-2014

Exame. Duração: 2:00H

2014/01/27

Todos os componentes computacionais do teste são baseados nas ferramentas dadas nas aulas. As respostas devem seguir as convenções usadas nessas ferramentas e considerar o seu funcionamento.

Sempre que não for dito nada em contrário, não devem ser feitas validações explícitas dos dados processados pelos programas.

Sempre que não for dito nada em contrário, podem usar-se todos os predicados e procedimentos existentes na linguagem Prolog.

As perguntas dos capítulos I e IV devem ser respondidas numa folha; as restantes perguntas (capítulos II e III) devem ser respondias noutra folha de avaliação.

I – Lógica de Predicados

Neste grupo, considera os predicados *Pessoa/1*, *Palavra/1*, *Frase/1*, *AplicavelAPlenosPulmões/1* e *PalavraFrase/2*:

Pessoa(x): x é uma pessoa	AplicavelAPlenosPulmões(x): x é uma palavra aplicável a plenos pulmões
Palavra(x): x é uma palavra	Feia(x, p): Para x, a palavra p é feia, por exemplo Feia(Luís, Procrastinar)
Frase(x): x é uma frase	
PalavraFrase(p, f): p é uma palavra da frase f	

(2.5 Valores) 1 – Usando lógica de predicados de primeira ordem, representa o conhecimento *"Todas as frases têm pelo menos uma palavra"*

R:

$\forall f\ [Frase(f) \Rightarrow \exists p\ (Palavra(p) \wedge PalavraFrase(p, f))]$

(2.5 Valores) 2 – Considera a seguinte base de conhecimento em lógica de predicados sobre frases e palavras.

i. $\exists p\ \exists x\ (Palavra(p) \wedge Pessoa(x) \wedge Feia(x, p))$
ii. $\forall p\ \forall x\ [(Palavra(p) \wedge Pessoa(x) \wedge Feia(x, p)) \Rightarrow AplicavelAPlenosPulmões(p)]$

Usando refutação, prova que $\exists p$ AplicavelAPlenosPulmões(p) pode ser derivado da base de conhecimento. Na tua resposta, podes assumir que a conversão da proposição (ii) para forma clausal dá origem a $\{\neg$Palavra(p), $\neg$Pessoa(x), $\neg$Feia(x, p), AplicavelAPlenosPulmões(p)$\}$, e que a conversão de $\neg\exists x\ P(x)$ dá origem a $\{\neg P(x)\}$.

R:

A resolução envolve 4 fases, não obrigatoriamente por esta ordem:

1 – Negar o objetivo

2 – Converter o objetivo negado em forma clausal

3 – Converter a base de conhecomento em forma clausal

4 – Mostrar que o conjunto de clausulas que vindas de 2 e 3 é um conjunto contraditório.

Objetivo: $\exists p$ AplicavelAPlenosPulmões(p)

Objetivo negado: $\neg\exists p$ AplicavelAPlenosPulmões(p)

Conversão do objetivo negado: o objetivo negado tem o formato $\neg\exists x$ P(x) cuja conversão é dada no enunciado: {$\neg$P(x) }. No caso específico da proposição $\neg\exists p$ AplicavelAPlenosPulmões(p), a conversão dá origem a {$\neg$AplicavelAPlenosPulmões(p) }

Conversão de $\exists p\ \exists x$ (Palavra(p) $\wedge$ Pessoa(x) $\wedge$ Feia(x, p)) em forma clausal:

a) $(A \Rightarrow B) \equiv (\neg A \vee B)$. N/A porque não há implicações
b) Mover implicações para os literais. N/A porque não há negações
c) Skolemização. Existem 2 variáveis existenciais, nenhuma das quais é quantificada dentro do âmbito de um quantificador universal. Nesse caso, cada uma é substituída por uma nova constante
   Palavra(SK1) $\wedge$ Pessoa(SK2) $\wedge$ Feia(SK2, SK1)
d) Remover os $\forall$s. N/A porque não existem $\forall$s
e) Converter a proposição numa conjunção de disjunções de literais. Não se aplica porque a proposição já está nesse formato.
f) Escrever as cláusulas
   { Palavra(SK1) }
   { Pessoa(SK2) }
   { Feia(SK2, SK1) }

Escrever $\forall p\ \forall x$ [(Palavra(p) $\wedge$ Pessoa(x) $\wedge$ Feia(x, p)) $\Rightarrow$ AplicavelAPlenosPulmões(p)] em forma clausal. O enunciado fornece a conversão:

{$\neg$Palavra(p), $\neg$Pessoa(x), $\neg$Feia(x, p), AplicavelAPlenosPulmões(p)}

Mostrar que o conjunto de todas as cláusulas é contraditório

1.	{ Palavra(SK1) }	Δ
2.	{ Pessoa(SK2) }	Δ
3.	{ Feia(SK2, SK1) }	Δ
4.	{$\neg$Palavra(p), $\neg$Pessoa(x), $\neg$Feia(x, p), AplicavelAPlenosPulmões(p) }	Δ
5.	{$\neg$AplicavelAPlenosPulmões(p) }	$\neg$Objetivo
6.	{$\neg$Palavra(p), $\neg$Pessoa(x), $\neg$Feia(x, p)}	4, 5
7.	{$\neg$Pessoa(x), $\neg$Feia(x, SK1)}	1, 6
8.	{$\neg$Feia(SK2, SK1)}	2,7
9.	{}	3, 8

II – Representação computacional de conhecimento

(2.5 Valores) 3 – Imagina um sistema de regas condição-conclusão para determinar se uma dada frase revela uma dada emoção. Escreve as regras e/ou os factos que correspondem ao conhecimento que se segue.

> *Uma frase revela uma dada emoção se a lista de palavras da frase contiver uma das palavras associadas a essa emoção. Uma frase revela uma dada emoção se descrever um episódio da classe C e a classe de episódios C está associada a essa emoção. A emoção medo está associada às palavras medo, pavor, terror, e receio.*

Para representar o conhecimento, usa os predicados descritos na tabela.

sentence(Sentence)	*Sentence* é uma frase
emotion(Emotion)	*Emotion* é uma emoção
reveals_emotion(Sentence, Emotion)	*Sentence* é uma frase que revela a emoção *Emotion*
words_of_sentence(Sentence, Words)	*Words* é a lista de palavras da frase *Sentence*
emotion_words(Emotion, Words)	*Words* é a lista de palavras associadas à emoção *Emotion*
describes_episodic_class(Sentence, Class)	A frase *Sentence* descreve um episódio da class *Class*
emotional_episodic_class(Emotion, Class)	A classe de episódios *Class* está associada à emoção *Emotion*

O sistema deve poder funcionar como no seguinte exemplo:

```
?- solve(reveals_emotion(Sentence, Emotion)).
Sentence = s1     Emotion = medo;
Sentence = s2     Emotion = júbilo;
……
```

R:

```
if   (sentence(Sent) and words_of_sentence(Sent, SWords) and
      emotion(Emotion) and emotion_words(Emotion, EWords) and
      member(W, EWords) and member(W, SWords))
then reveals_emotion(Sent, Emotion).

if   (sentence(Sent) and describes_episodic_class(Sent, Class) and
      emotional_episodic_class(Emotion, Class))
then reveals_emotion(Sent, Emotion).

fact(emotion_words(medo, [medo, pavor, terror, receio])).
```

(2.5 Valores) 4 – Tens um sistema que te ajuda a encontrar e obter livros desejados, pelo melhor preço. O sistema procura o livro, nos sítios web das lojas *online* de venda de livros, usando a ação `search_book` (ver a tabela). A ação produzirá um ou mais factos com a informação do resultado da procura, *search_result/2* (ver tabela). Os resultados poderão ser os seguintes:

- O átomo `book_not_found`, se o livro não tiver sido encontrado;
- A estrutura `selling_site(URL, Cost)` especificando um sítio onde o livro pode ser comprado e o respetivo preço. Em resultado da procura, vários destes resultados podem ser criados se houver mais do que uma loja de venda *online* em que o livro pode ser comprado.

Depois da procura, se o livro puder ser comprado numa loja *online*, o sistema escolhe a loja em que o preço é mais barato, informa o utilizador, abre o sítio web correspondente ao melhor preço, no *browser* usual, e

termina a sua operação. Se o livro não tiver sido encontrado, o sistema informa o utilizador e termina a sua operação. Em qualquer caso, os factos temporários criados durante o seu funcionamento não devem persistir na memória depois do sistema acabar a sua operação.

A seguinte tabela descreve as ações que o sistema pode executar

search_book(Book)	Procura o livro *Book* nos sítios web de lojas *online* de venda de livros e cria um ou mais factos *search_result/2*, os quais poderão ser
	`search_result(Book, book_not_found)`
	`search_result(Book, selling_site(URL, Cost))`

open_site(URL)	Abre o sítio especificado no URL, usando o *browser* usual
remove_all_search_results(Book)	Remove todos os factos *search_results/2* relativos ao livro *Book*.
retract(Clause)	Remove a primeira cláusula que emparelha com *Clause*
writelist(List)	Escreve os elementos da lista *List*, todas na mesma linha
Nl	Produz uma mudança de linha no output.

find_books/1 é o procedimento de interface entre o utilizador e o sistema baseado em conhecimento:

```
find_books(Books):-
      clearMemory,
      assert_desired_books(Books),
      psys.

% Cria um facto desired_book/1, por cada livro na lista.
assert_desired_books([B|Books]):-
      assert(desired_book(B)),
      assert_desired_books(Books).
assert_desired_books([]).
```

Escreve as regras de produção que controlam o sistema descrito. Além das ações já descritas e dos predicados *desired_book/1* e *search_result/2*, as regras do sistema podem recorrer a qualquer predicado da linguagem Prolog, por exemplo os operadores de comparação numérica.

R:

```
if   (desired_book(B) and \+ search_result(B, _))
then search_book(B).

if   search_result(B, book_not_found)
then (retract(desired_book(B)),
      writelist([B, ' not found']), nl,
      retract(search_result(B, _))).

if   (search_result(B, selling_site(URL, Cost)) and
      \+( (search_result(B, selling_site(_, C)) and C < Cost )))
then (retract(desired_book(B)),
      writelist(['Using your browser, buy ', B, ' at the site']),
      nl,
      open_site(URL),
      remove_all_search_results(B)).
```

III – Prolog

(2.5 Valores) 5 – Usando o *read/1*, escreve o procedimento *read_or_fail/1* que lê um termo do teclado ou do ficheiro atualmente aberto para leitura. No final do ficheiro, *read_or_fail/1* falha.

R:

```
read_or_fail(Term):-
      read(Term),
      Term \= end_of_file.
```

(2.5 Valores) 6 – Assume que tens os predicado *despesa_familia/2* que determina a despesa efetuada, com prendas de natal, com as pessoas de uma dada família. Escreve o procedimento *despesa_natal/1* que imprime no ecrã o nome da cada família e a despesa efetuada, em prendas de natal, com as pessoas dessa família. Além disso, *despesa_natal/1* devolve o montante total despendido com prendas de natal (para todas as famílias). Os nomes das famílias são mantidos em factos do predicado *família/1*, por exemplo `família(pereira).` *despesa_natal/1* tem de se certificar que o seu argumento é uma variável. Nesta pergunta não podes usar os predicados *findall/3*, *bagof/3*, e *setof/3*.

Exemplos

```
?- despesa_familia(pereira, X).
X = 25
```

```
?- despesa_natal(X).
Família pereira: 25
Família martins: 18
X = 43
```

Se te der jeito, assume que existe o predicado *writelist/1* que imprime, na mesma linha, todos os elementos de uma lista.

R:

```
despesa_natal(Total):-
      var(Total),
      assert(despesa(0)),
      família(Familia),
      despesa_familia(Familia, MontanteFamilia),
      writelist(['Familia ', Familia, ': ', MontanteFamilia]), nl,
      atualiza_despesa(MontanteFamilia),
      fail.
despesa_natal(Total):-
      retract(despesa(Total)).

atualiza_despesa(X):-
      retract(despesa(D)),
      D1 is D + X,
      assert(despesa(D1)),
      !.
```

(2.5 Valores) 7 – Admite que tens os predicados *família/1* e *prenda/4*, exemplificados a seguir.

```
família(pereira).
família(martins).
```

```
prenda(ana, livro, 10, pereira).
prenda(rui, dvd, 15, pereira).
prenda(sara, colar, 18, martins).
```

Escreve o predicado *prendas_familias/1*, de modo que prendas_familias(Lista) significa que Lista é a lista de prendas, organizadas por famílias, por exemplo

```
?- prendas_familias(L).
```

```
L = [pereira:[livro,dvd], martins:[colar]]
?-
```

Obrigatoriamente, o predicado *prendas_familias/1*começa por usar o *findall/3* para obter a lista de todas as famílias. Essa lista é passada a um predicado auxiliar recursivo que percorre a lista de famílias e, para cada uma delas, produz a lista de prendas recebidas por pessoas dessa família. A lista de prendas de uma família pode ser obtida usando o *findall/3*.

R:

```
prendas_familias(Lista):-
      findall(Familia, família(Familia), ListaFamilias),
      prendas_familias(ListaFamilias, Lista).

prendas_familias([F|Familias], [F:Prendas|RestoPrendas]):-
      findall(P, prenda(_, P, _, F), Prendas),
      prendas_familias(Familias, RestoPrendas).
prendas_familias([], []).
```

IV – Generalidades

8 – Responde às seguintes perguntas. Respostas erradas descontam conforme indicado.

(1 Valor) a) Em Prolog, o *repeat* pode ser usado para criar mecanismos de repetição por recursividade ou por falha? [desconta 0.5]

R: Por falha.

(1 Valor) b) No BRules, que tipo de encadeamento é usado? [Não desconta]

R: Para trás

(0.5 Valores) c) Que estratégia de resolução de conflitos é usada no PSys: escolhe uma regra arbitrariamente, escolhe a regra mais específica, ou escolhe a regra que depende de factos mais recentes? [desconta 0.25]

R: Escolhe uma regra arbitrariamente