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

Inteligência Artificial 2013-2014

Exame. Duração: 2:00H

2014/01/08

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 – Generalidades

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

(1 Valor) a) Que tipo de encadeamento se usa num sistema de regras de produção? [Não desconta]

R: Encadeamento para a frente.

(1 Valor) b) Em lógica de predicados de primeira ordem, para indicar que uma proposição é válida para todos os elementos do universo de discurso, usa-se o ∃ ou o ∀? [desconta 0.5]

R: ∀

(0.5 Valores) c) Na expressão x = Idade(João), Idade é um predicado ou uma função? [desconta 0.25]

R: Função

II – Lógica de Predicados

Considera os predicados *Pessoa/*1, *Palavra/1*, *Frase/1* e *Feia/2*:

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

(2.5 Valores) 2 – Usando lógica de predicados de primeira ordem, representa o conhecimento *"Existe pelo menos uma palavra que, na opinião de pelo menos uma pessoa, é feia"*

R:

∃p ∃x [Palavra(p) ∧ Pessoa(x) ∧ Feia(x, p)]

(2.5 Valores) 3 – Considera a seguinte base de conhecimento em lógica de predicados.

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

Sem recorrer à forma clausal, mostra que $\exists p\ Palavra(p)$ se pode derivar da base de conhecimentos.

R:

Objetivo: $\exists p\ Palavra(p)$

Derivação:

i.	$\forall f\ [Frase(f) \Rightarrow \exists p\ (Palavra(p) \wedge PalavraFrase(p, f))]$	Δ
ii.	$\exists f\ Frase(f)$	Δ
iii.	$Frase(SK1)$	ii EI
iv.	$Frase(SK1) \Rightarrow \exists p\ (Palavra(p) \wedge PalavraFrase(p, SK1))$	i UI
v.	$\exists p\ (Palavra(p) \wedge PalavraFrase(p, SK1))$	iii, iv MP
vi.	$Palavra(SK2) \wedge PalavraFrase(SK2, SK1)$	v EI
vii.	$Palavra(SK2)$	vi AE
viii.	$\exists p\ Palavra(p)$	vii EG

III – Representação computacional de conhecimento

(2.5 Valores) 4 – Imagina um sistema de regas condição-conclusão para determinar se uma dada frase diz respeito a um dado assunto. Escreve as regras e/ou os factos que correspondem ao conhecimento que se segue.

> *Uma frase diz respeito a um dado assunto se a lista de palavras da frase cobrir semanticamente a lista de palavras-chave do assunto. A lista de palavras L1 cobre semanticamente a lista de palavras L2 se pelo menos uma das palavras de L2 ou um seu sinónimo for uma das palavras de L1.*

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

sentence(Sentence)	*Sentence* é uma frase
subject(Subject)	*Subject* é um assunto
is_about(Sentence, Subject)	*Sentence* é uma frase que diz respeito ao assunto *Subject*
words_of_sentence(Sentence, Words)	*Words* é a lista de palavras da frase *Sentence*
keywords_of_subject(Subject, Keywords)	*Keywords* é a lista de palavras-chave do assunto *Subject*
semantically_covers(Words, Keywords)	A lista de palavras *Words* cobre semanticamente a lista de palavras-chave *Keywords*
synonym(Word, Synonym)	*Synonym* é um sinónimo de *Word*

O sistema deve poder funcionar como no seguinte exemplo:

```
?- solve(is_about(Sentence, Subject)).

Sentence = s1     Subject = subject3;

Sentence = s2     Subject = subject1;

……
```

R:

```
if   (sentence(Sent) and words_of_sentence(Sent, Words) and
      subject(Subj) and keywords_of_subject(Subj, Keywords) and
      semantically_covers(Words, Keywords))
then is_about(Sent, Subj).

if   (member(W, Keywords) and member(W, Words))
then semantically_covers(Words, Keywords).

if   (member(W, Keywords) and not(member(W, Words)) and
      synonym(W, S) and member(S, Words))
then semantically_covers(Words, Keywords).
```

(2.5 Valores) 5 – Tens um sistema que te ajuda a encontrar e obter filmes desejados, na internet. O sistema procura o filme, usando a ação `search_movie` (ver a tabela). A ação produzirá um facto com a informação do resultado da procura, *search_result/2* (ver tabela). Os resultados poderão ser os seguintes:

- O átomo `movie_not_found`, se o filme não tiver sido encontrado;
- A estrutura `download_address(URL)` com o URL que pode ser usado para descarregar o filme, se o filme tiver sido encontrado e puder ser descarregado;
- A estrutura `selling_site(URL)` especificando o sítio onde o filme pode ser comprado

Se o filme não for encontrado, o sistema informa o utilizador. Se tiver sido encontrado um URL para descarregar o filme, o sistema descarrega-o, guarda-o no disco, e informa o utilizador. Se o filme tiver de ser comprado, o sistema informa o utilizador, abre a página de compra do filme e mostra-a ao utilizador no seu *browser*. Será o utilizador a decidir se pretende ou não comprar o filme, o que terá de fazer sem usar o sistema.

Escreve um conjunto de regras de produção para controlar o sistema acabado de descrever. No fim da sua operação, não devem persistir em memória os factos temporários criados durante o seu funcionamento (e.g., *search_result/2* e *desired_movie/1*).

O seguinte é o programa que serve de interface com o utilizador:

```
find_movies(Movies):-
      clearMemory,
      assert_desired_movies(Movies),
      psys.

% Cria um facto desired_movie/1, por cada filme na lista.
assert_desired_movies([M|Movies]):-
      assert(desired_movie(M)),
      assert_desired_movies(Movies).
assert_desired_movies([]).
```

O sistema de produção poderá usar as ações que se descrevem seguidamente, as quais já estão implementadas.

search_movie(Movie)	Procura o filme Movie na internet e cria o facto *search_result/2*, o qual poderá ser
	`search_result(Movie, movie_not_found)`
	`search_result(Movie, selling_site(URL))`
	`search_result(Movie, download_address(URL))`

open_site(URL)	Abre o sítio especificado no URL, usando o *browser* por omissão
download(URL, File)	Descarrega o filme do URL especificado e guarda-o num ficheiro. Na variável File, devolve o *pathname* do ficheiro onde o filme foi guardado.
retract(Clause)	Remove a primeira cláusula que emparelha com *Clause*
writelist(List)	Escreve a lista de palavras no output, todas na mesma linha
nl	Causa uma mudança de linha

R:

```
if (desired_movie(M) and \+ search_result(M, _)) then search_movie(M).
```

```
if   search_result(M, movie_not_found)
then (retract(desired_movie(M)),
      writelist([M, ' not found']), nl,
      retract(search_result(M, _))).
```

```
if   search_result(M, selling_site(URL))
then (retract(desired_movie(M)),
      writelist(['Using your browser, buy ', M, ' at the site']),
      nl,
      open_site(URL),
      retract(search_result(M, _))).
```

```
if   search_result(M, download_address(URL))
then (retract(desired_movie(M)),
      download(URL, File),
      writelist(['I found and saved ', M, 'in ', File]), nl,
      retract(search_result(M, _))).
```

IV – Prolog

(2.5 Valores) 6 – Escreve o procedimento *read_term/2* para ler termos do teclado ou do ficheiro aberto para leitura, tal que read_term(Term, State) significa que *Term* é o termo lido e *State* é o estado do ficheiro. *State* terá o valor `ok`, se não tiver chegado o fim do ficheiro; e `eof`, no final do ficheiro. *read_term/2* tem de usar o procedimento *read_or_fail/1* que lê um termo ou falha no final do ficheiro. Não tens de implementar *read_or_fail /1*.

R:

```
read_term(Term, ok):-
      read_or_fail(Term),
      !.
read_term(_, eof).
```

(2.5 Valores) 7 – Sem usar os predicados *findall/3*, *bagof/3*, e *setof/3*, escreve o procedimento *despesa_familia/2* para calcular o valor monetário despendido, em prendas de natal, com as pessoas de uma dada família.

A informação sobre as prendas é mantida em factos do predicado *prenda/4*, e os nomes das famílias estão armazenados em factos do predicado *família/1*, como no seguinte exemplo.

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

despesa_familia/2 tem de verificar que o seu primeiro argumento é um átomo e representa uma família existente.

Exemplo:

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

R:

```
despesa_familia(Familia, _):-
      atom(Familia), família(Familia),
      assert(montante(0)),
      prenda(_, _, X, Familia),
      atualiza_montante(X),
      fail.
despesa_familia(_, Montante):-
      retract(montante(Montante)).

atualiza_montante(X):-
      retract(montante(M)),
      M1 is M + X,
      assert(montante(M1)),
      !.
```

(2.5 Valores) 8 – Escreve o predicado *sortudo/4*, tal que sortudo(Familia, Sortudo, Prenda, Preço) significa que *Sortudo* é a pessoa da família *Família* que recebeu a prenda mais cara dessa família. *Prenda* é a prenda recebida, e *Preço* é o seu preço. Para simplificar, admite que, na mesma família, não há duas pessoas que recebem prendas do mesmo preço.

Exemplo

```
?- sortudo(Familia, Sortudo, Prenda, Preco).
Familia = pereira  Sortudo = rui    Prenda = dvd      Preco = 15;
Familia = martins  Sortudo = sara   Prenda = colar    Preco = 18
```

Sugere-se que *sortudo/4* use o predicado *família/1* para verificar ou instanciar o argumento *Família*. Além disso, *sortudo/4* tem obrigatoriamente de usar o *findall/3* para determinar a lista de prendas recebidas pelas pessoas de cada família. Cada elemento dessa lista deve ser um triplo (Pessoa, Prenda, Preço). Essa lista será depois dada a um predicado recursivo que identifica o sortudo dessa família, a prenda que faz dele um sortudo, e o seu preço.

R:

```
sortudo(Familia, Sortudo, Prenda, Preco):-
      família(Familia),
      findall((X, P, Custo), prenda(X, P, Custo, Familia), Lista),
      sortudo_aux(Lista, Sortudo, Prenda, Preco).

sortudo_aux([(Sortudo, Prenda, Preco)], Sortudo, Prenda, Preco).
sortudo_aux([X1, X2 | Lista], S, P, Preco):-
      sortudo_aux([X2 | Lista], S2, P2, Preco2),
      prenda_mais_cara(X1, (S2, P2, Preco2), (S, P, Preco)).

prenda_mais_cara((S1, P1, Pre1), (_, _, Pre2), (S1, P1, Pre1)):-
      Pre1 > Pre2,
      !.
prenda_mais_cara(_, (S2, P2, Pre2), (S2, P2, Pre2)).
```