EI **ETI** **IGE**

Inteligência Artificial 2015-2016

Teste de SBC

2015/11/17

Nota: Nos exercícios que se seguem, a programação em lógica, e as regras de produção estão expressas e funcionam tal como nos sistemas usados e explicados nas aulas. As respostas devem seguir os mesmos padrões.

I – Lógica de Predicados

(4 Valores) 1 – Considere a seguinte base de conhecimentos em lógica de predicados de primeira ordem.

i) $\forall x\ [(\text{Assassino}(x) \vee \text{Corrupto}(x)) \Rightarrow \text{Criminoso}(x)]$

ii) $\exists x\ \text{Assassino}(x)$

Mostre, sem passar para forma clausal, que da base de conhecimentos se pode derivar $\exists x\ \text{Criminoso}(x)$.

R:

i) $\forall x\ [(\text{Assassino}(x) \vee \text{Corrupto}(x)) \Rightarrow \text{Criminoso}(x)]$	Δ
ii) $\exists x\ \text{Assassino}(x)$	Δ
iii) $\text{Assassino}(SK)$	ii) EI
iv) $(\text{Assassino}(SK) \vee \text{Corrupto}(SK)) \Rightarrow \text{Criminoso}(SK)$	i) UI
v) $\text{Assassino}(SK) \vee \text{Corrupto}(SK)$	iii) OI
vi) $\text{Criminoso}(SK)$	iv), v) MP
vii) $\exists x\ \text{Criminoso}(x)$	vi) EG

(4 Valores) 2 - Considera a seguinte base de conhecimentos, já em forma clausal

i.	$\{ \neg\text{Palhaço}(x), \text{Artista}(x, \text{Circo}) \}$	Δ
ii.	$\{ \neg\text{Trapezista}(x), \text{Artista}(x, \text{Circo}) \}$	Δ
ii.	$\{ \text{Palhaço}(SK) \}$	Δ

Mostra por refutação, usando a resolução, que $\exists x\ \text{Artista}(x, \text{Circo})$ se pode derivar da base de conhecimentos.

R:

A prova por refutação, usando resolução, envolve dois passos:

- Negar o objetivo e converter para forma clausal

- Juntar as cláusulas correspondentes à negação do objetivo à base de conhecimentos e mostrar que o resultado é contraditório.

Conversão do objetivo negado para forma clausal

Objetivo negado: $\neg\exists x$ Artista(x, Circo)

Conversão

1) $(A \Rightarrow B) \equiv (\neg A \vee B)$. N/A porque não há implicações

   $\neg\exists x$ Artista(x, Circo)

2) Mover as implicações para os literais

   $\forall x \neg$Artista(x, Circo)

3) Skolemização. N/A porque não há $\exists$s

4) Remoção dos $\forall$s

   $\neg$Artista(x, Circo)

5) Converter numa conjunção de disjunções. N/A porque já está nesse formato

   $\neg$Artista(x, Circo)

6) Escrever em forma clausal

   {$\neg$Artista(x, Circo)}

Derivar a contradição

i.	{ $\neg$Palhaço(x), Artista(x, Circo) }	Δ
ii.	{ $\neg$Trapezista(x), Artista(x, Circo) }	Δ
iii.	{ Palhaço(SK) }	Δ
iv.	{$\neg$Artista(x, Circo) }	$\neg$Obj
v.	{ Artista(SK, Circo) }	1, 3
vii	{}	4, 5

II – Representação computacional de conhecimento

(4 Valores) 3 – Considera a seguinte base de conhecimento, expressa informalmente, de um sistema que determina os preços de ingresso dos clientes num espetáculo multimédia interativo.

X é jovem se X tem uma idade menor do que 13 anos

X é idoso se tem uma idade maior que 64 anos

O preço dos bilhetes de clientes jovens é de 4€

O preço dos bilhetes de clientes idosos é de 5€

O preço dos bilhetes de clientes que não sejam jovens nem idosos é de 7.5€

Considerando os predicados *client/1*, *jovem/1*, *idoso/1*, *preço/2* e *idade/2*, descritos na tabela, e ainda os predicados *>/2* e *</2*, representa o conhecimento apresentado através de cláusulas Prolog.

client(X)	X é cliente
young(X)	X é jovem
old(X)	X é idoso
price(X, P)	O preço do bilhete de X é P
age(X, I)	A idade de X é I

R:

```
young(X):- age(X, A), A < 13.
old(X):- age(X, A), A > 64.

price(X, 4):- client(X), young(X).
price(X, 5):- client(X), old(X).
price(X, 7.5):- client(X), \+young(X), \+old(X).
```

(4 Valores) 4 – Numa prova de uma dada competição, os participantes têm de passar num conjunto de pontos de controlo, conforme se exemplifica nos seguintes factos.

```
control_point(cp1).
control_point(cp2).
control_point(cp3).
```

Cada concorrente controlado (i.e., que passa por um ponto de controlo) é registado com um facto *participant_control/2*, por exemplo

```
participant_control(ana, cp1).
```

Escreve o predicado *valid_run/1*, tal que valid_run(X) significa que X passou em todos os pontos de controlo especificados. O predicado tem de funcionar para provas com um número qualquer de pontos de controlo.

R:

Para o predicado funcionar mesmo bem, é necessário o predicado *participant/1* com os participantes, um por cada facto. Nesse caso, a resolução seria a seguinte:

```
valid_run(X):-
      participant(X),
      \+( (control_point(CP), \+ participant_control(X, CP)) ).
```

Quem apresentou esta resolução tem a cotação completa. No entanto, como o enunciado, por esquecimento, não referiu o predicado *participante/1*, não descontamos nada a quem não o usou na resolução:

```
valid_run(X):-
      \+( (control_point(CP), \+ participant_control(X, CP)) ).
```

Isto também funciona mas apenas para testar.

(4 Valores) 5 – Sistema de regras de produção

Um *robot* tem de manter o nível de água do reservatório o mais próximo possível de um

determinado valor – nível desejado –, tendo para isso um balde (o balde tem uma dada capacidade).

Para manter o nível de água próximo do valor definido, o *robot* compara o nível atual da água com o nível desejado, e usa o balde para repor o nível do reservatório o mais próximo possível do nível mínimo desejado.

O *robot* tem também de encher o balde, mesmo que não o vá usar de imediato.

As duas tabelas que se seguem descrevem as ações e os predicados que o robot pode usar.

Ação	**Efeito no mundo**
fill_bucket/0	O balde fica cheio de água
dump_bucket/0	O balde fica vazio O nível de água do reservatório passa a ser igual ao seu nível anterior mais a capacidade do balde.

Predicado	**Descrição**
bucket_state/1	bucket_state(State): O estado do balde é o especificado (empty/full)
bucket_capacity/1	bucket_capacity(C): *C* é a capacidade do balde, em litros
reservoir_level/1	reservoir_level(Level): *Level* é o nível atual do reservatório, em litros
desired_level/1	desired_level(Level): *Level* é o nível desejado do reservatório, em litros

Além dos recursos descritos nas duas tabelas, o robot dispõe de todos os recursos da linguagem Prolog, por exemplo os operadores relacionais >, >=, =<, o operador *is/2*, e as funções aritméticas.

O programa de interface com o *robot*, *fill_reservoir/1* tem a seguinte definição:

```
fill_reservoir(Level):-
      assert(desired_level(Level)),
      assert(reservoir_level(100)),
      assert(bucket_state(full)),
      assert(bucket_capacity(25)),
      psys.
```

Escreve as regras que levam o *robot* a encher o balde sempre que ele está vazio, e a despejar o balde no reservatório sempre que o seu nível é menor que o nível desejado e a capacidade do balde é menor do que ou igual à diferença entre o nível desejado e o nível atual do reservatório.

R:

```
% Regra para manter os baldes cheios
if (bucket_state(empty)) then fill_bucket.

% Regra para controlar o nível do reservatório
if   (reservoir_level(Level) and desired_level(DLevel) and
      DLevel > Level and LevelDiff is DLevel-Level and
      bucket_capacity(C) and bucket_state(full) and C =< LevelDiff)
then  dump_bucket.
```

Vários alunos fizeram a segunda regra melhor que a correção dos profs:

```
% Regra para controlar o nível do reservatório
if   (reservoir_level(Level) and desired_level(DLevel) and
      bucket_capacity(C) and bucket_state(full) and
      C =< DLevel - Level)
then  dump_bucket.
```

Ficamos satisfeitos com isso.

Era assumido que as ações fill_bucket/0 e dump_bucket/0 fariam a gestão dos factos dos predicados bucket_state/1 e reservoir_level/1. No entanto, como o enunciado não era claro a este respeito, não descontamos nada a quem especificou os asserts e retracts corretos, além das ações adequadas. Claro está que, quem não especificou as ações adequadas ou quem especificou asserts e retracts errados sofreu o desconto correspondente.