TÉCNICO LISBOA

Lógica para Programação

Exame de 2ª Época

2 de Julho de 2019

9:00–11:00

Nome: _____ Número: _____

- Esta prova, individual e sem consulta, tem **11** páginas com **11** perguntas. A cotação de cada pergunta está assinalada entre parêntesis.
- Escreva o seu número em todas as folhas da prova. O tamanho das respostas deve ser limitado ao espaço fornecido para cada questão. O corpo docente reserva-se o direito de não considerar a parte das respostas que excedam o espaço indicado.
- Pode responder utilizando lápis.
- Em cima da mesa devem apenas estar o enunciado, caneta ou lápis e borracha e cartão de aluno. Não é permitida a utilização de folhas de rascunho, telemóveis, calculadoras, etc.
- Boa sorte!

Pergunta	Cotação	Nota
1.	1.0	
2.	1.0	
3.	2.0	
4.	1.5	
5.	2.0	
6.	1.5	
7.	3.0	
8.	2.0	
9.	2.0	
10.	1.5	
11.	2.5	
Total	20.0	

1. (1.0) Para cada uma das seguintes questões, escolha a única alternativa correcta. Cada resposta correcta vale 0.5 valores e *cada resposta errada desconta 0.2 valores*.

   (a) Uma cláusula é
   A. uma disjunção de *fbfs* atómicas.
   B. uma disjunção de literais.
   C. uma conjunção de literais.
   Resposta: ___
   Resposta:
   B

   (b) Uma regra de procura
   A. recebe um literal e um programa e devolve uma cláusula definida.
   B. recebe um objectivo e devolve uma regra.
   C. recebe um objectivo e devolve um dos seus sub-objectivos.
   Resposta: ___
   Resposta:
   A

2. (0.5 + 0.5) Considere a constante $O\_Incrivel\_mundo\_de\_Gumball$ e os seguintes predicados:

   $Aluno(x)$: se x é um aluno
   $Serie(x)$: se x é uma série
   $Gosta\_de(x, y)$: se x gosta de y

   (a) Indique a fórmula em Lógica de Primeira Ordem que melhor traduz a frase em Língua Natural *Os alunos não gostam de nenhuma série.*:
   A: $\forall x, y[Aluno(x) \rightarrow (Serie(y) \wedge \neg Gosta\_de(x, y))]$
   B: $\forall x, y[(Aluno(x) \wedge \neg Gosta\_de(x, y)) \rightarrow Serie(y)]$
   C: $\forall x, y[\neg Gosta\_de(x, y) \rightarrow (Aluno(x) \wedge Serie(y))]$
   D: $\forall x, y[(Aluno(x) \wedge Serie(y)) \rightarrow \neg Gosta\_de(x, y)]$
   Resposta: ___
   Resposta:
   D

   (b) Indique a fórmula em Lógica de Primeira Ordem que **NÃO** traduz a frase em Língua Natural *Nem todos os alunos gostam de "O Incrível Mundo de Gumball".*
   A: $\exists x[Aluno(x) \wedge \neg Gosta\_de(x, O\_Incrivel\_mundo\_de\_Gumball)]$
   B: $\neg\forall x[Aluno(x) \rightarrow Gosta\_de(x, O\_Incrivel\_mundo\_de\_Gumball)]$
   C: $\neg\forall x[Aluno(x) \wedge Gosta\_de(x, O\_Incrivel\_mundo\_de\_Gumball)]$
   D: $\neg\forall x[\neg Aluno(x) \vee Gosta\_de(x, O\_Incrivel\_mundo\_de\_Gumball)]$
   Resposta: ___
   Resposta:
   C

3. (2.0) Demonstre o seguinte teorema

$$\{\} \vdash (\forall x[(P(x) \rightarrow R(x))] \wedge \exists y[P(y)]) \rightarrow \exists z[R(z)]$$

usando o sistema dedutivo da Lógica de Primeira Ordem (apenas pode usar as regras de premissa, hipótese, repetição, reiteração, e as regras de introdução e eliminação de cada um dos símbolos lógicos).

Resposta:

1	$\forall x[P(x) \to R(x)] \land \exists y[P(y)]$	Hip
2	$\forall x[P(x) \to R(x)]$	E$\land$, 1
3	$\exists y[P(y)]$	E$\land$, 1
4	$x_0 \mid P(x_0)$	Hip
5	$\forall x[P(x) \to R(x)]$	Rei, 2
6	$P(x_0) \to R(x_0)$	E$\forall$, 5
7	$R(x_0)$	E$\to$, (4, 6)
8	$\exists z[R(z)]$	I$\exists$, 7
9	$\exists z[R(z)]$	E$\exists$, (3, (4, 8))
10	$(\forall x[P(x) \to R(x)] \land \exists y[P(y)]) \to \exists z[R(z)]$	I$\to$, (1, 9)

4. (1.5) Considere o seguinte conjunto de *fbfs* (em que x, y e z são variáveis, a e c são constantes e f é uma função)

$$\{P(x, f(a), z), P(y, f(y), c)\}$$

Preencha as linhas necessárias da seguinte tabela, de forma a seguir o algoritmo de unificação para determinar se as *fbfs* são unificáveis. Em caso afirmativo, indique o unificador mais geral; caso contrário, indique que as *fbfs* não são unificáveis.

Conjunto de fbfs	**Conjunto de desacordo**	**Substituição**

Unificador mais geral (se existir):

Resposta:

Conjunto de fbfs	**Conjunto de desacordo**	**Substituição**
$\{P(x, f(a), z), P(y, f(y), c)\}$	$\{x, y\}$	$\{y/x\}$
$\{P(y, f(a), z), P(y, f(y), c)\}$	$\{y, a\}$	$\{a/y\}$
$\{P(a, f(a), z), P(a, f(a), c)\}$	$\{z, c\}$	$\{c/z\}$
$\{P(a, f(a), c)\}$		

Unificador mais geral (se existir):

$\{y/x\} \circ \{a/y\} \circ \{c/z\} = \{a/x, a/y, c/z\}$

5. (2.0) Demonstre o seguinte teorema

$$\{\forall x[(P(x) \rightarrow R(x))] \wedge \exists y[P(y)]\} \vdash \exists z[R(z)]$$

usando resolução.

Resposta:

Para provar o teorema teremos de fazer uma prova por refutação:

- *Forma clausal das premissas e da negação da conclusão:*
  $\{\{\neg P(x), R(x)\}, \{P(a)\}, \{\neg R(z)\}\}$ (em que a é uma constante de Skolem)

- *Prova:*

1	$\{\neg P(x), R(x)\}$	Prem
2	$\{P(a)\}$	Prem
3	$\{\neg R(z)\}$	Prem
4	$\{R(a)\}$	Res, (1,2), $\{a/x\}$
5	$\{\}$	Res, (3,4), $\{a/z\}$

6. (1.5) Considere a conceptualização (D, F, R) em que:

$D = \{\diamond, \square, \odot\}$
$F = \{\}$
$R = \{\ldots\}$.

Considere a interpretação I: $\{a, b, c, P, S\} \mapsto D \cup F \cup R$, tal que:

$I(a) = \diamond$
$I(b) = \square$
$I(c) = \odot$

Preencha a tabela abaixo, de forma a que a interpretação I seja um modelo do conjunto de *fbfs*

$$\Delta = \{\neg P(c), \neg P(a), \neg P(b), \forall x, y[S(x, y) \rightarrow P(x)]\}.$$

$I(P)$	
$I(S)$	

Resposta:

$I(P)$	$\{\}$
$I(S)$	$\{\}$

7. (3.0) Considere a *fbf* $(P \rightarrow Q) \wedge (Q \rightarrow R)$, e a ordem $P \prec Q \prec R$.

   (a) (2.0) Obtenha o seu OBDD reduzido, por aplicação dos algoritmos *reduz* e *compacta* à respectiva árvore de decisão.
   Resposta:

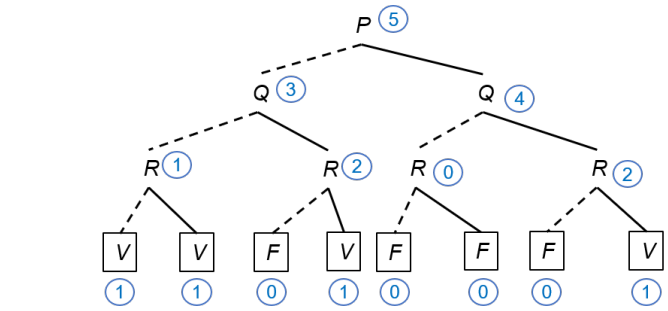
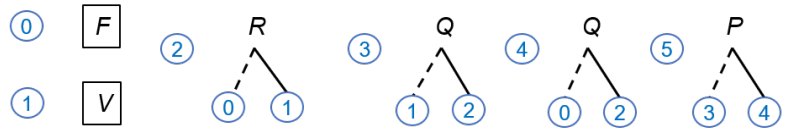
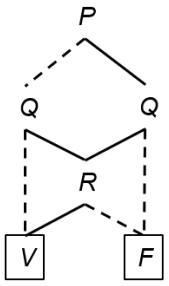


Tabela associativa





(b) (1.0) O resultado obtido na alínea anterior permite concluir que $\{(P \rightarrow Q) \wedge (Q \rightarrow R)\} \models R$? Justifique a sua resposta.

Resposta:

Não, R não é consequência semântica da fbf, pois é possível encontrar um modelo da fbf que não é modelo de R. Por exemplo, a interpretação I(P)=F,I(Q)=F,I(R)=F.

8. (2.0) Seja Δ = {{$\neg D$, B}, {$\neg C$, A}, {$\neg A$, D, C}, {$\neg C$, E}, {$\neg E$}}. Após aplicação do algoritmo DP a Δ, recorrendo a baldes, obtém-se:

b_C :	$\{\neg C, A\}, \{\neg A, D, C\}, \{\neg C, E\}$			
b_E :	$\{\neg E\}$	$\{\neg A, D, E\}$		
b_D :	$\{\neg D, B\}$		$\{\neg A, D\}$	
b_A :				$\{B, \neg A\}$
b_B :				

(a) (0.5) Supondo $I(E) = F$ e $I(C) = F$, qual dos seguintes casos **NÃO** conduz a uma testemunha de Δ:

A: $I(B) = V, I(A) = F, I(D) = V$
B: $I(B) = V, I(A) = F, I(D) = F$
C: $I(B) = V, I(A) = V, I(D) = V$
D: $I(B) = F, I(A) = F, I(D) = V$

Resposta: ___

Resposta:

D

(b) (0.5) Qual das seguintes frases é **FALSA**:

A: Numa testemunha de Δ verifica-se sempre $I(E) = F$.
B: Numa testemunha de Δ se $I(B) = V$ então $I(A) = F$.

C: Numa testemunha de Δ verifica-se sempre $I(C) = F$.
D: Numa testemunha de Δ ou $I(D) = V$ ou $I(A) = F$.

Resposta: ___
Resposta:
B

(c) (0.5) Seja $\Delta_1 = \Delta \cup \{A\} \cup \{\neg B\}$. Qual das seguintes afirmações é **VERDADEIRA**:
A: Δ_1 é uma tautologia.
B: Δ_1 é contraditória.
C: Δ_1 é simultaneamente falsificável e satisfazível.
D: Nenhuma das anteriores.

Resposta: ___
Resposta:
B

(d) (0.5) Tendo em conta a alínea c) qual das seguintes afirmações é **VERDADEIRA**:
A: $\Delta \cup \{A\} \models B$
B: $\Delta \models A \wedge B$
C: $\Delta \cup \{\neg B\} \models A$
D: $\Delta \cup \{B\} \models \neg A$

Resposta: ___
Resposta:
A

9. (1.5) Considere o seguinte programa em Prolog:
C_1: `a(2, 2).`
C_2: `a(X, Y) :- b(X, Y), c(Y, X).`
C_3: `b(22, 55).`
C_4: `b(33, 12).`
C_5: `b(44, 13).`
C_6: `c(22, 55).`
C_7: `c(12, 33).`
Supondo que vão sempre ser pedidas mais respostas enquanto tal for possível, qual a resposta do Prolog ao objectivo `?- a(X, Y)`...

(a) ... considerando o programa anterior.
Resposta:
```
X = Y, Y = 2; X = 33, Y = 12; false.
```

(b) ... considerando que C_1 é agora: `a(X, Y) :- b(X, Y), c(X, Y), !.`
Resposta:
```
X = 22, Y = 55.
```

(c) ... considerando que C_1 é agora: `a(X, Y) :- not(b(X, Y)), c(X, Y).`
Resposta:
```
X = 33, Y = 12 ; false.
```

10. (1.5) Implemente em Prolog:

(a) (1.0) O predicado `seguidosDe/3` tal que `seguidosDe(X, L1, L2)` significa que `L2` é a lista de elementos de L1 que aparecem imediatamente a seguir a `X`. Por exemplo:
```
?- seguidosDe(1, [1, 2, 7, 8, 1, 'ola', 1, 7, 1, b], L).
L = [2, ola, 7, b]
```
Resposta:

```
seguidosDe(_, [], []) :- !.
seguidosDe(_, [_], []) :- !.
seguidosDe(X, [X, Y | R], [Y | R1]) :- seguidosDe(X, [Y | R], R1).
seguidosDe(X, [H | R], R1) :- X \= H, seguidosDe(X, R, R1).
```

(b) (0.5) O predicado `contaSeguidos/3` em que `contaSeguidos(X, L, N)` significa que N é o número de elementos que aparecem imediatamente a seguir a `X` na lista `L`. Sugestão: use o predicado definido anteriormente.
Por exemplo:

```
?- contaSeguidos(2, [1, 2, 7, 2, 1, 5, 2, a, 1, 1], N).
N = 3
```

Resposta:

```
contaSeguidos(X, L, N) :- seguidosDe(X, L, L1), length(L1, N).
```

11. As questões que se seguem dizem respeito ao contexto do projecto. Na implementação dos predicados pode, ou não, usar os meta-predicados sobre listas. Fica ao seu critério.

(a) (1.0) Implemente o predicado `conta_vars_0_1/2`, tal que `conta_vars_0_1(Fila, [N_Vars, N_Zeros, N_Uns])` significa que `N_Vars`, `N_Zeros` e `N_Uns` são o número de variáveis, de zeros e de uns da fila `Fila` de um puzzle binário, respectivamente. Se `Fila` tiver algum elemento que não seja nem uma variável, nem zero, nem um, o predicado deve devolver `false`. Por exemplo,

```
?- Fila = [0, 1, _, 1, 1, 0, 0, 1], conta_vars_0_1(Fila, Res).
Fila = [0, 1, _G529, 1, 1, 0, 0, 1],
Res = [1, 3, 4].

?- Fila = [5, 1, _, 1, 1, 0, 0, 1], conta_vars_0_1(Fila, Res).
false.
```

Resposta:
Sem usar meta-predicados:

```
conta_vars_0_1(Fila, Res) :-
    conta_vars_0_1(Fila, Res, [0, 0, 0]).

conta_vars_0_1([], Res, Res).

conta_vars_0_1([P | R], Res, Temp) :-
    (var(P), !, incrementa_contador(Temp, 1, Novo_Temp)
                        ;
     P == 0, !, incrementa_contador(Temp, 2, Novo_Temp)
                    ;
     P == 1, !, incrementa_contador(Temp, 3, Novo_Temp)),
    conta_vars_0_1(R, Res, Novo_Temp).


incrementa_contador(Contadores, Pos, Res) :-
    Pos_1 is Pos - 1,
    length(Antes, Pos_1),
    append([Antes, [Cont], Depois], Contadores),
    !,
    Novo_Cont is Cont + 1,
    append([Antes, [Novo_Cont], Depois], Res).
```

Usando o meta-predicado `include`:

```
conta_vars_0_1(Fila, [N_Vars, N_Zeros, N_Uns]) :-
    include(var, Fila, Vars),
    length(Vars, N_Vars),
    include(==(0), Fila, Zeros),
    length(Zeros, N_Zeros),
    include(==(1), Fila, Uns),
    length(Uns, N_Uns),
    Soma is N_Vars + N_Zeros + N_Uns,
    length(Fila, Soma).
```

(b) (0.5) Usando o predicado definido na alínea anterior, implemente o predicado `verifica_fila/1`, tal que `verifica_fila(Fila)`, em que `Fila` é uma lista, significa que `Fila` é uma lista tal que:

- o número de elementos é par;
- o número de zeros e o número de uns são iguais a metade do número de elementos.

Por exemplo,

```
?- Fila = [0, 1, 0, 1, 1, 0, 0, 1], verifica_fila(Fila).
Fila = [0, 1, 0, 1, 1, 0, 0, 1].

?- Fila = [0, 1, 0, 1, 1, 0, 0, _], verifica_fila(Fila).
false.

?- Fila = [0, 1, 0], verifica_fila(Fila).
false.
```

Resposta:

```
verifica_fila(Fila) :-
    length(Fila, Total),
    Total mod 2 =:= 0,
    Metade is Total // 2,
    conta_vars_0_1(Fila, [_, Metade, Metade]).
```

(c) (0.5) Usando o predicado definido na alínea anterior, implemente o predicado `verifica_filas/1`, tal que `verifica_filas(Filas)`, em que `Filas` é uma lista de filas, significa que todas as filas satisfazem o predicado definido na alínea anterior. Por exemplo,

```
?- Fila1 = [0, 1, 0, 1, 1, 0, 0, 1],
   Fila2 = [1, 1, 0, 1, 1, 0, 0, 1],
   verifica_filas([Fila1, Fila2]).
false.

?- Fila1 = [0, 1, 0, 1, 1, 0, 0, 1],
   Fila2 = [1, 1, 0, 0, 1, 0, 0, 1],
   verifica_filas([Fila1, Fila2]).
Fila1 = [0, 1, 0, 1, 1, 0, 0, 1],
Fila2 = [1, 1, 0, 0, 1, 0, 0, 1].
```

Resposta:
Sem usar meta-predicados:

```
verifica_filas([]).

verifica_filas([F | R_F]) :-
    verifica_fila(F),
    verifica_filas(R_F).
```

Usando o meta-predicado `maplist`:

```
verifica_filas2(Filas) :-
    maplist(verifica_fila, Filas).
```

(d) (0.5) Usando o predicado definido na alínea anterior, implemente o predicado `verifica_puzzle/1`, tal que `verifica_puzzle(Puz)`, em que `Puz` é um puzzle binário, significa que todas as filas (linhas e colunas) de `Puz` satisfazem o predicado `verifica_fila/1`. Sugestão: Use o predicado `transpose/2`, tal que `transpose(Mat, Mat_T)` significa que `Mat_T` é a transposta da matriz `Mat`. Por exemplo, sendo `Puz` o puzzle

```
[[0,0,1,1],
 [1,0,1,0],
 [0,1,0,1],
 [1,1,0,0]]
```

teríamos

```
 ?- ..., verifica_puzzle(Puz).
Puz = [[0, 0, 1, 1], [1, 0, 1, 0], [0, 1, 0, 1], [1, 1, 0, 0]].
```

e sendo `Puz` o puzzle

```
[[0,0,1,1],
 [1,0,1,0],
 [0,1,0,1],
 [0,1,1,0]]
```

teríamos

```
 ?- ..., verifica_puzzle(Puz).
false.
```

Resposta:

```
verifica_puzzle(Puz) :-
    verifica_filas(Puz),
    transpose(Puz, Puz_T),
    verifica_filas(Puz_T).
```