TÉCNICO LISBOA

Lógica para Programação

Exame de 1ª Época

14 de Junho de 2019

15:00–17:00

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.	1.5	
4.	2.0	
5.	2.0	
6.	1.0	
7.	4.0	
8.	2.0	
9.	1.5	
10.	1.5	
11.	2.5	
Total	20.0	

1. (1.0) Para cada uma das seguintes afirmações, diga se é verdadeira (V) ou falsa (F). Cada resposta correcta vale 0.5 valores e *cada resposta errada desconta 0.2 valores*.

   (a) Um argumento é válido se e só se tanto as premissas como a conclusão forem verdadeiras.
   Resposta: ___
   Resposta:
   F

   (b) Se $\{\alpha_1, \ldots, \alpha_n\} \models \beta$, então o OBDD reduzido da *fbf* $\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta$ é a folha $\boxed{V}$.
   Resposta: ___
   Resposta:
   F

2. (1.0) Considere 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

   Indique a fórmula em Lógica de Primeira Ordem que melhor traduz as seguintes frases em Língua Natural:

   (a) *Existe uma série da qual todas os alunos gostam.*
   A: $\exists x[Serie(x) \rightarrow \forall y[Aluno(y) \wedge Gosta\_de(y, x)]]$
   B: $\exists x[Serie(x) \rightarrow \forall y[Aluno(y) \rightarrow Gosta\_de(y, x)]]$
   C: $\exists x[Serie(x) \wedge \forall y[Aluno(y) \rightarrow Gosta\_de(y, x)]]$
   D: $\exists x[Serie(x) \wedge \forall y[Aluno(y) \wedge Gosta\_de(y, x)]]$
   Resposta: ___
   Resposta:
   C

   (b) *Todas os alunos gostam de uma série.*
   A: $\forall x[Aluno(x) \rightarrow \exists y[Serie(y) \wedge Gosta\_de(x, y)]]$
   B: $\forall x[Aluno(x) \rightarrow \exists y[Serie(y) \rightarrow Gosta\_de(x, y)]]$
   C: $\forall x[Aluno(x) \wedge \exists y[Serie(y) \rightarrow Gosta\_de(x, y)]]$
   D: $\forall x[Aluno(x) \wedge \exists y[Serie(y) \wedge Gosta\_de(x, y)]]$
   Resposta: ___
   Resposta:
   A

3. (1.5) Considere o seguinte conjunto de *fbfs* (em que x, y e z são variáveis, b é uma constante, e f é uma função)

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

   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
$\{P(x, f(y), f(x)), P(z, f(b)), z\}$	$\{x, z\}$	$\{z/x\}$
$\{P(z, f(y), f(z)), P(z, f(b)), z\}$	$\{y, b\}$	$\{b/y\}$
$\{P(z, f(b), f(z)), P(z, f(b)), z\}$	$\{f(z), z)\}$	—

Unificador mais geral (se existir): não existe, porque o último conjunto de desacordo não contém uma variável e um termo que não mencione essa variável.

4. (2.0) Demonstre que

$$\{\} \vdash \exists x[P(x) \wedge \neg Q(x)] \rightarrow \neg\forall x[P(x) \rightarrow Q(x)]$$

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	$\exists x[P(x) \wedge \neg Q(x)]$	Hip
2	$x_0 \mid P(x_0) \wedge \neg Q(x_0)$	Hip
3	$P(x_0)$	E$\wedge$, 2
4	$\neg Q(x_0)$	E$\wedge$, 2
5	$\forall x[P(x) \rightarrow Q(x)]$	Hip
6	$\neg Q(x_0)$	Rei, 4
7	$P(x_0)$	Rei, 3
8	$P(x_0) \rightarrow Q(x_0)$	E$\forall$, 5
9	$Q(x_0)$	E$\rightarrow$, (7, 8)
10	$\neg\forall x[P(x) \rightarrow Q(x)]$	I$\neg$, (5, (6, 9))
11	$\neg\forall x[P(x) \rightarrow Q(x)]$	E$\exists$, (1, (2, 10))
12	$\exists x[P(x) \wedge \neg Q(x)] \rightarrow \neg\forall x[P(x) \rightarrow Q(x)]$	I$\rightarrow$, (1, 11)

5. (2.0) Demonstre o seguinte argumento

$$\{\exists x[P(x) \wedge \neg Q(x)]\} \vdash \neg\forall x[P(x) \rightarrow Q(x)])$$

usando resolução e fazendo uma prova por refutação.

Resposta:

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

- *Prova:*

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

6. (1.0) Considere a conceptualização (D, F, R) em que:

   $D = \{\diamond, \square\}$
   $F = \{\}$
   $R = \{\{\}, \{(\square)\}\}$.

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

   $I(a) = \diamond$
   $I(b) = \square$
   $I(P) = \{\}$
   $I(S) = \{(\square)\}$

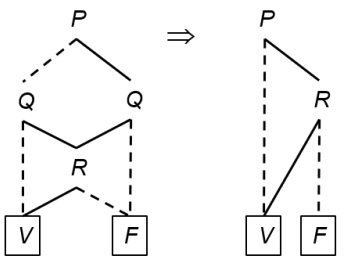
   Para cada uma das *fbfs* na tabela abaixo, indique o seu valor segundo a interpretação I.

α	$I(\alpha)$
$P(a) \rightarrow S(a)$	
$P(b) \vee S(b)$	
$S(b) \wedge \neg P(a)$	
$S(b) \rightarrow P(b)$	

   Resposta:

α	$I(\alpha)$
$P(a) \rightarrow S(a)$	V
$P(b) \vee S(b)$	V
$S(b) \wedge \neg P(a)$	V
$S(b) \rightarrow P(b)$	F

7. (4.0) Considere a *fbf* representada pela aplicação do seguinte operador lógico entre os dois OBDDs:



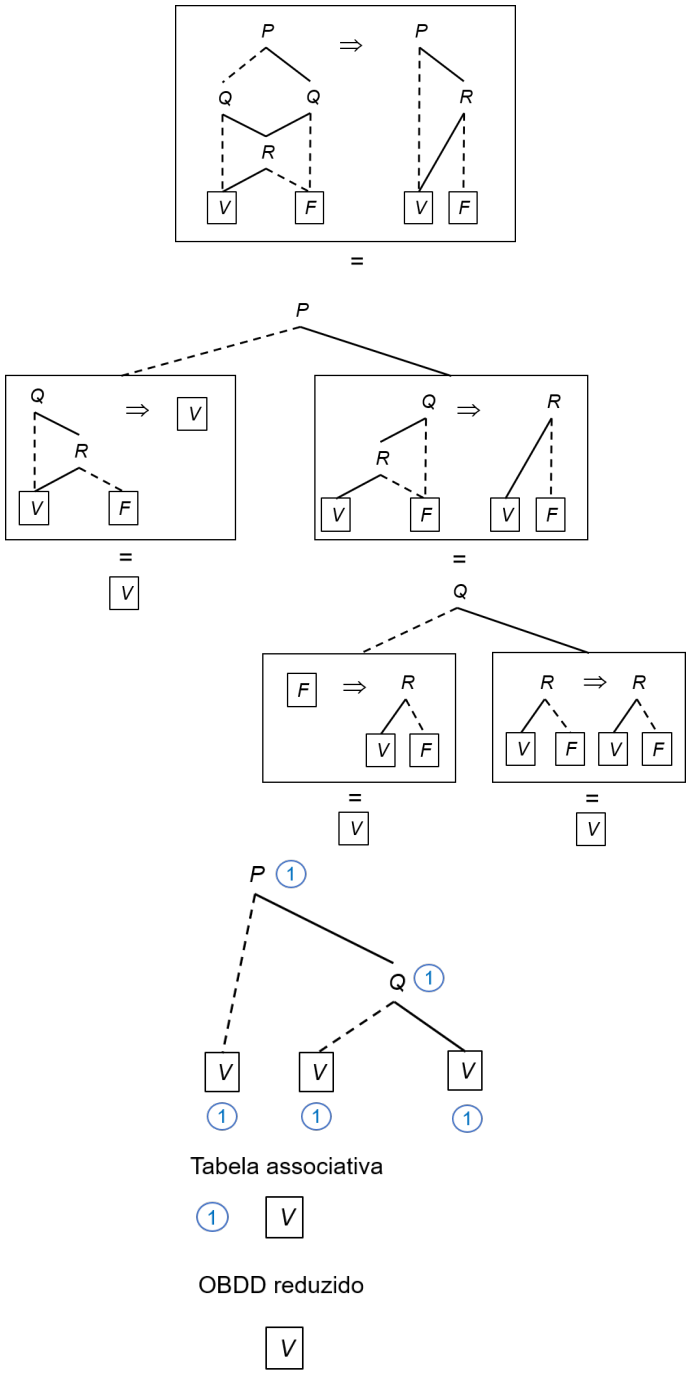
(a) (0.5) Os OBDDs são compatíveis? Justifique a sua resposta.

Resposta:

Os OBDDs são compatíveis, pois ambos os OBDDs satisfazem a ordem $P \prec Q \prec R$

(b) (3.0) Considerando a relação de ordem $P \prec Q \prec R$, e usando o algoritmo *aplica*, obtenha o OBDD reduzido da *fbf*.

Resposta:

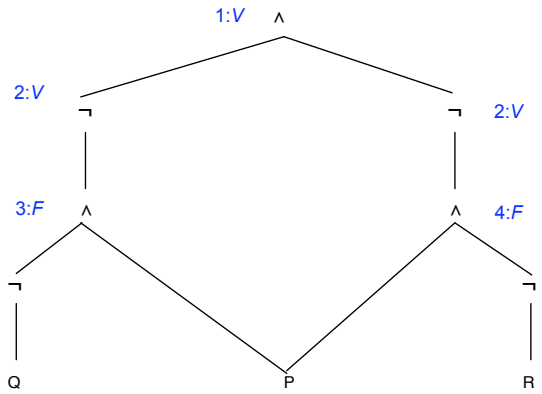


(c) (0.5) Quais as interpretações que satisfazem a *fbf* representada pelo OBDD obtido na alínea anterior?

Resposta:

Todas as interpretações satisfazem a fbf, pois a fbf é uma tautologia.

8. (2.0) Considere o seguinte DAG:



(a) (0.5) Diga a que fórmula corresponde este DAG.

Resposta:

$\neg(\neg Q \wedge P) \wedge \neg(P \wedge \neg R)$

(b) (0.5) Das seguintes frases indique a única **que é verdade**:

A: O teste de Q com marca temporária V leva à marca de P com V.
B: O teste de R com marca temporária V leva à marca de P com V.
C: O teste de P com marca temporária V leva à marca de Q com V.
D: O teste de P com marca temporária F leva à marca de R com V.

Resposta: ___

Resposta:

C

(c) (0.5) Das seguintes frases indique a única **que NÃO é** verdade:

A: Marcar P com a marca temporária V não permite marcar R
B: Marcar P com a marca temporária F não permite marcar R.
C: Marcar Q com a marca temporária V não permite marcar R.
D: Marcar Q com a marca temporária F não permite marcar R.

Resposta: ___

Resposta:

A

(d) (0.5) Aplicando o teste de nós, verifique se a fórmula é satisfazível. Em caso afirmativo, indique uma testemunha.

Resposta:

A fórmula é satisfazível. Testemunha: I(P) = I(Q) = I(R) = V.

9. (1.5) Considere o seguinte programa em Prolog:

C_1: `a(X, Y) :- b(X), c(X, Y).`
C_2: `a(2, 2).`

C_3: `b(22).`
C_4: `b(33).`
C_5: `b(44).`
C_6: `c(22, 21).`

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 = 22, Y = 21; X = 2, Y = 2`

(b) ... considerando que C_1 é agora: `a(X, Y) :- b(X), !, c(X, Y).`
Resposta:
`X = 22, Y = 21`

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

10. (1.5) Sem recorrer a predicados pré-definidos no Prolog, implemente:

(a) (0.75) o predicado `menor/2` tal que `menor(L, M)` significa que `M` é o menor inteiro da lista de inteiros `L`.
Resposta:

```
menor([H | T], Menor) :- menor(T, Menor, H).
menor([], Aux, Aux).
menor([H | T], Menor, Aux) :- Aux > H, !, menor(T, Menor, H).
menor([H | T], Menor, Aux) :- Aux =< H, menor(T, Menor, Aux).
```

(b) (0.75) o predicado `menor/3` em que `menor(N, L, M)` significa que `M` é igual ao menor inteiro da lista de inteiros `L` caso este seja menor que `N`, sendo `M` igual a `N` caso contrário. Por exemplo:

```
?- menor(2, [3, 4, 6, 10], M).
M = 2.
?- menor(6, [3, 4, 6, 10], M).
M = 3.
```

Resposta:

```
menorQue(N, L, Menor) :- menor(L, M),  M < N, !, Menor is M.
menorQue(N, _, 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) (0.5) Implemente o predicado `conta_vars_fila/2`, tal que `conta_vars_fila(Fila, Num_vars)`, em que `Fila` é uma fila de um puzzle binário, significa que `Num_vars` é o número de variáveis de `Fila`. Por exemplo,

```
?- conta_vars_fila([_, _, 0, 0, _, 1], Num_vars).
Num_vars = 3.
```

Sugestão: utilize o predicado pré-definido `var`.

Resposta:

Sem usar meta-predicados:

```
conta_vars_fila([], 0).

conta_vars_fila([P | R], Num_vars) :-
    var(P),
    !,
    conta_vars_fila(R, Num_vars_R),
    Num_vars is Num_vars_R + 1.

conta_vars_fila([_ | R], Num_vars) :-
    conta_vars_fila(R, Num_vars).
```

Usando o meta-predicado `include`:

```
conta_vars_fila(Fila, Num_vars) :-
    include(var, Fila, Vars_Fila),
    length(Vars_Fila, Num_vars).
```

(b) (0.5) Implemente o predicado `conta_0_1_fila/3`, tal que `conta_0_1_fila(ZU, Fila, N)`, em que `ZU` é `0` ou `1`, e `Fila` é uma fila de um puzzle binário, significa que `N` é o número de ocorrências de `ZU` em `Fila`. Por exemplo,

```
?- conta_0_1_fila(1, [_, _, 0, 0, _, 1], N).
N = 1 .
?- conta_0_1_fila(0, [_, _, 0, 0, _, 1], N).
N = 2 .
```

Resposta:

Sem usar meta-predicados:

```
conta_0_1_fila(_, [], 0).

conta_0_1_fila(ZU, [P | R], N) :-
    P == ZU,
    !,
    conta_0_1_fila(ZU, R, N_R),
    N is N_R + 1.

conta_0_1_fila(ZU, [_ | R], N) :-
```

Usando o meta-predicado `include`:

```
conta_0_1_fila(ZU, Fila, N) :-
    include(==(ZU), Fila, ZUs_Fila),
    length(ZUs_Fila, N).
```

(c) (1.5) Usando os predicados definidos nas alíneas anteriores, implemente o predicado `conta_vars_uns_Puzzle/3`, tal que `conta_vars_uns_Puzzle(Puz, Vars, Uns)` significa que `Vars` e `Uns` são as listas contendo o número de variáveis e de uns, respectivamente, das linhas do puzzle `Puz`. Por exemplo, sendo `Puz` o puzzle

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

teríamos

```
?- ..., conta_vars_uns_Puzzle(Puz, Vars, Uns).
Puz = ...,
Vars = [2, 1, 2, 2],
Uns = [1, 2, 0, 1] .
```

Resposta:
Sem usar meta-predicados:

```
conta_vars_uns_Puzzle([], [], []).

conta_vars_uns_Puzzle([Lin | R], Vars, Uns) :-
    conta_vars_fila(Lin, Vars_Lin),
    conta_0_1_fila(1, Lin, Uns_Lin),
    conta_vars_uns_Puzzle(R, Vars_R, Uns_R),
    Vars = [Vars_Lin | Vars_R],
    Uns = [Uns_Lin | Uns_R].
```

Usando o meta-predicado `maplist`:

```
conta_vars_uns_Puzzle2(Puz, Vars, Uns) :-
    maplist(conta_vars_fila, Puz, Vars),
    maplist(conta_0_1_fila(1), Puz, Uns).
```

RASCUNHO

RASCUNHO