INSTITUTO SUPERIOR TÉCNICO

Licenciatura Engenharia Informática e de Computadores

Lógica para Programação

Solução do Segundo Teste

19 de Junho de 2010

11:00–12:30

Nome: _____ Número: _____

1. Escolha a *única* resposta *correcta* para as seguintes questões. Cada resposta certa vale 1 valor e *cada resposta errada desconta 0.4 valores*.

   (a) **(1.0)** Seja $s_1 = \{a/x, f(x)/y, y/z\}$ e $s_2 = \{b/x, z/y, g(x)/z, b/w\}$. Considerando que x, y, z e w são variáveis, o valor de $s_1 \circ s_2$ é dado por:

   A. $\{a/x, f(b)/y\}$
   B. $\{a/x, f(b)/y, b/w\}$
   C. $\{b/x, g(a)/z, b/w\}$
   D. $\{a/x, f(x)/y, y/z, b/x, z/y, g(x)/z, b/w\}$

   Resposta:

   B.

   (b) **(1.0)** Uma cláusula de Horn

   A. contém no máximo um literal negativo
   B. contém no máximo um literal positivo
   C. não contém nenhum literal negativo
   D. não contém nenhum literal positivo

   Resposta:

   B.

   (c) **(1.0)** Uma função de selecção

   A. é uma regra para escolher um literal numa cláusula objectivo como candidato à aplicação do princípio da resolução
   B. é uma regra para escolher um literal numa cláusula determinada como candidato à aplicação do princípio da resolução
   C. é uma função do conjunto de literais e do conjunto das cláusulas determinadas para o conjunto dos programas
   D. é uma função do conjunto de literais e do conjunto dos programas para o conjunto das cláusulas determinadas

   Resposta:

   A.

   (d) **(1.0)** Considere as seguintes definições de operadores em PROLOG:

```
:- op(1000, fx, op1).
:- op(800, xfy, op2).
:- op(800, xfy, op3).
```

e a expressão `op1 a op2 b op2 c op3 d`. Qual o resultado da leitura correcta desta expressão:

A. `op1(a) op2 (b op2 (c op3 d))`
B. `((op1(a) op2 b) op2 c) op3 d`
C. `op1(((a op2 b) op2 c) op3 d)`
D. `op1(a op2 (b op2 (c op3 d)))`

Resposta:

D.

2. Considere a conceptualização $C = (D, F, R)$ e a interpretação, I, definidas do seguinte modo:

$D = \{0, 1\}$
$F = \{\{(0,1),(1,0)\},\{(0,0),(1,1)\}\}$
$R = \{\{(0)\},\{(1)\}\}$

$I(zero) = 0$
$I(um) = 1$
$I(identidade) = \{(0,0),(1,1)\}$
$I(outro) = \{(0,1),(1,0)\}$
$I(menorquedois) = \{(0),(1)\}$
$I(impar) = \{(1)\}$

(a) **(1.0)** Prove, justificando, que a seguinte *fbf* não é satisfeita pela interpretação I:

$$menorquedois(outro(zero)) \wedge \neg menorquedois(identidade(zero))$$

Resposta:

A fórmula dada é satisfeita por I para C se e só se tanto $menorquedois(outro(zero)$ como $\neg menorquedois(identidade(zero))$ o forem. Ora $\neg menorquedois(identidade(zero))$ não é satisfeita por I para C, dado que $menorquedois(identidade(zero))$ o é, tal como se prova de seguida:
$(I(identidade)(I(zero))) = (I(identidade)(0)) = (0)$
$(0) \in I(menorquedois)$

(b) **(1.0)** Encontre uma fórmula que seja satisfeita pela interpretação I. Justifique.

Resposta:

Tal como provado na alínea anterior $menorquedois(identidade(zero))$ é satisfeita por esta interpretação para esta conceptualização.

3. **(2.0)** No contexto do projecto, defina o predicado `accao_possivel/2`. A expressão `accao_possivel(A,S)` afirma que é possível executar a acção `A` no estado `S`. Defina todos os predicados necessários, à excepção dos predicados `pre_cond/2` e `verif/2`, que podem utilizar sem os definir.

Resposta:

```
accao_possivel(A, S) :- pre_cond(A, P_C), verif_todas(P_C, S).

verif_todas([], _).
verif_todas([P | R], S) :- verif(P, S), verif_todas(R, S).
```

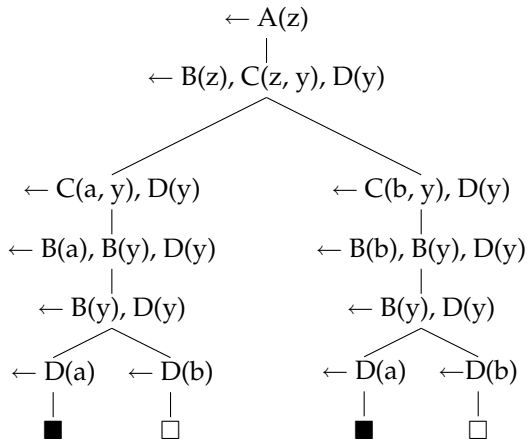
4. Considere o programa definido através do seguinte conjunto de cláusulas de Horn:

- $A(x) \leftarrow B(x), C(x,y), D(y)$
- $B(a)$

- $B(b)$
- $C(x, y) \leftarrow B(x), B(y)$
- $D(b)$
- $D(c)$

(a) **(1.0)** Usando a resolução SLD e uma função de selecção que escolhe o primeiro literal do objectivo para unificar (podendo usar a regra de procura que entender), indique explicitamente numa árvore SLD todas as soluções para o objectivo $\leftarrow A(z)$.

Resposta:



As respostas seriam: $\{a/z\}$ e $\{b/z\}$.

(b) **(1.0)** Sem fazer novos cálculos indique qual seria a resposta do programa ao objectivo $\leftarrow A(a)$? Justifique.

Resposta:

A resposta calculada seria a substituição vazia.

5. Considere o predicado `heroi/1`, em que a expressão `heroi(Z)` afirma que `Z` tem poderes e não é mau. Considere ainda o seguinte programa em PROLOG:

```
temPoderes(flash_gordon).
temPoderes(ming).
mau(ming).
```

(a) **(1.0)** Estenda o programa anterior, de modo a respeitar a definição do predicado `heroi/1` e de modo a que a resposta ao objectivo `heroi(Z)` seja `Z = flash_gordon`.

Resposta:

```
heroi(Z) :- tempoderes(Z), \+(mau(Z)).
```

(b) **(1.0)** Altere a extensão feita na alínea anterior, de modo a que a resposta do programa ao objectivo `heroi(Z)` seja `No`.

Resposta:

```
heroi(Z) :- \+(mau(Z)), tempoderes(Z).
```

6. **(1.0)** Defina o predicado `soma_pos_neg/3`. A expressão `soma_pos_neg(L, S_pos, S_neg)` afirma que `S_pos` e `S_neg` são as somas dos elementos positivos e negativos, respectivamente, da lista `L`. Por exemplo,

```
?- soma_pos_neg([1, 2, -3, 4, 0, -5], S_pos, S_neg).
S_pos = 7,
S_neg = -8
```

Resposta:

```
soma_pos_neg([P|R],S_pos,S_neg) :-
                  P >= 0, !,
                  soma_pos_neg(R,SR_pos,S_neg),
                  S_pos is SR_pos + P.

soma_pos_neg([P|R],S_pos,S_neg) :-
                  P < 0, !,
                  soma_pos_neg(R,S_pos,SR_neg),
                  S_neg is SR_neg + P.

soma_pos_neg([],0,0).
```

7. Considere o programa

```
rem_rep([], []).
rem_rep([P|R], L) :- membro(P,R), rem_rep(R, L).
rem_rep([P|R], [P|L]) :- rem_rep(R,L).
```

   (a) **(0.5)** Indique todas as respostas dadas pelo PROLOG para o objectivo
   `?- rem_rep([a, a], L).`
   Resposta:
   `L = [a]` e `L = [a,a]`.

   (b) **(0.5)** Se a finalidade do predicado `rem_rep` for remover os elementos repetidos de uma lista, o programa acima pode considerar-se correcto? Porquê?
   Resposta:
   O programa não pode ser considerado correcto porque a segunda resposta está errada, não foram removidos os elementos repetidos.

   (c) **(1.0)** Corrija o programa, usando o operador de corte.
   Resposta:

```
rem_rep([P|R], L) :- membro(P,R), !, rem_rep(R, L).
rem_rep([P|R], [P|L]) :- rem_rep(R,L).
rem_rep([], []).
```

   (d) **(1.0)** Corrija o programa sem usar directamente o operador de corte, mas recorrendo à negação por falhanço.
   Resposta:

```
rem_rep([P|R], L) :- membro(P,R), rem_rep(R, L).
rem_rep([P|R], [P|L]) :- \+  membro(P,R), rem_rep(R,L).
rem_rep([], []).
```

   (e) **(1.0)** Qual das duas versões do programa, (c) ou (d), é mais eficiente? Porquê?
   Resposta:
   A versão que usa o corte é mais eficiente, pois, no caso em que o primeiro elemento da lista aparece repetido, não são experimentadas mais cláusulas; no caso da negação por falhanço é experimentada a segunda cláusula, que leva necessariamente a um ramo falhado.

8. Considere o tipo árvore binária de inteiros.

   (a) **(0.5)** Escolha uma representação em PROLOG para o tipo árvore binária de inteiros.
   Resposta:
   A árvore vazia é representada pelo átomo `vazia`. Uma árvore de raiz `R`, árvore esquerda `A_E` e árvore direita `A_D` é representada pela estrutura `arv(R, A_E, A_D)`.

   (b) **(1.5)** Com base na representação escolhida, implemente as seguintes operações básicas:
   - Construtores:
     - `arv_vazia(A)` significa que `A` é a árvore vazia.
     - `constroi_arv(R, A_E, A_D, A)` significa que `A` é uma árvore de raiz `R`, árvore esquerda `A_E` e árvore direita `A_D`.
   - Selectores:
     - `raiz(A,R)` significa que `R` é a raiz da árvore `A`.
     - `arv_esq(A,A_E)` significa que `A_E` é a árvore esquerda da árvore `A`.
     - `arv_dir(A,A_D)` significa que `A_D` é a árvore direita da árvore `A`.
   - Reconhecedor:
     - `is_arv(A)` significa que `A` é uma árvore.
   - Teste:
     - `arv_iguais(A1,A2)` significa que `A1` e `A2` são árvores iguais.

   Resposta:

```
arv_vazia(vazia).
cria_arv(R, AE, AD, arv(R,AE,AD)).

raiz(arv(R, _, _), R).
arv_esq(arv(_, AE, _), AE).
arv_dir(arv(_, _, AD), AD).

is_arv(vazia).
is_arv(arv(R, AE, AD)) :- integer(R), is_arv(AE), is_arv(AD).

arv_iguais(vazia, vazia).
arv_iguais(arv(R1, AE1, AD1),arv(R2, AE2, AD2)) :-
                        !,
                        R1 =:= R2,
                        arv_iguais(AE1, AE2),
                        arv_iguais(AD1, AD2).
```

   (c) **(1.0)** Usando as operações básicas do tipo árvore, defina o predicado `soma_arv/2`, com o seguinte significado: `soma_arv(A,S)` afirma que `S` é a soma de todas as raízes da árvore `A`.
   Resposta:

```
soma_arv(vazia, 0).
soma_arv(arv(R, AE, AD),S) :-
           !,
           soma_arv(AE, SE),
           soma_arv(AD, SD),
           S is R + SE + SD.
```