

Programação em Lógica com Restrições

Exame sem consulta - Duração: 3 horas

N.º :		Nome:	
-------	--	-------	--

Grupo 1 (4 valores)

1 a) Para cada um dos seguintes golos, indique se sucede (com V, de Verdadeiro) ou se falha (com F, de Falso).

Nota: Nesta alínea cada resposta certa vale 0,1 valores, e cada resposta errada vale -0,1 valores (negativo, portanto). Se não sabe a resposta, não responda.

!, fail

nonvar(p(X))

X+Y == Y+X

[A,2,B|_] = [_ ,A|[3|C]]

func(2) = 4

f(4,X)+1 =.. [+ ,A,X]

X+Y*2 = 4+B

!, member(X,[1,2,3,X]), X=2

2 \= X

G=member(X,[1,Y,X]), Y=2, call(G), !, X:=1+1

1 b) Preencha correctamente os quadrados em branco abaixo, relativos ao predicado **produto(+ListaDif, -Produto)**, que dada uma lista de diferença, **ListaDif** (na forma habitual **ListaAberta-Variável**), contendo números, devolve o valor do seu produto em **Produto**.

Exemplo:

```
?- produto([2,-1,3,1|T]-T, Produto).
Produto = -6
```

```
produto(LD, Produto):- prod(LD, 1,Produto).
```

```
prod( L-_, P,  ) :-  , !.
```

```
prod(  , Pi, P):-
```

```
 ,
```

```
prod(  , P1,  ).
```

1 c) Para o seguinte programa

```
p(3). p(2). p(1).
r(X,Y):- p(X), X>1, !, Y is X*10.
r(3,3):- !.
r(X,Y):- r(Y,X).
s(Z):- !, findall(X, r(_,X), Xs), member(Z,Xs).
s(3).
```

indique a (primeira) solução dos seguintes golos (caso não tenha solução, escreva 'no').

Exemplo: ?- p(X). X=3

?- r(R,X). R=3, X=30

?- r(10,R). no

?- r(20,R). no

?- s(R). R=30

?- 3=X, s(X), p(X). no

1 d) No programa abaixo, quais deverão ser as associatividades (*Assoc1* e *Assoc2*) e as precedências (*Prec1* e *Prec2*) na definição dos operadores ':' e '--', de modo a que tenha o comportamento desejado na verificação (*eh_arv/1*) que uma árvore binária está representada num termo na forma *nil* (constante denotando árvore vazia) ou *Nó:Esq—Dir* (onde *Esq* e *Dir* são também árvores na mesma forma), e na verificação (*membro/2*) dum membro numa tal árvore?

```
:- op(Prec1, Assoc1, :).
:- op(Prec2, Assoc2, --).
```

```
eh_arv(nil).
eh_arv(X:E--D):- eh_arv(E), eh_arv(D).
```

```
membro(X, X:_).
membro(X, _:Esq--_):- membro(X,Esq).
membro(X, _:--Dir):- membro(X,Dir).
```

:	Assoc1 =	xfx	Prec1 =	(e.g.) 700 (<1000)
--	Assoc2 =	xfx	Prec2 =	(e.g.) 600 (<Prec1)

Nota: Os operadores pré-definidos neste programa são ':-', e ':', com precedências 1200 e 1000, respectivamente.

N.º:

Grupo 2 (3 valores)

Consistência de Restrições.

Nota: Neste grupo, cada resposta (SIM/NÃO) errada também tem cotação negativa. Se não sabe a resposta, não responda.

- 2 a)** Aplicando Consistência de Intervalo, indique se os seguintes valores pertencem (**SIM**) ou **NÃO** ao domínio de **Y** após a propagação aplicada ao golo $X \in \{-3, -2, 0, 1, 2\}$, $Y \in \{-4, -3, -2, 0\} \setminus (3..5)$, $Z \in -9..9$, $X \# < Y$, $Z \# = 2 * X$, $3 * Y \# \geq 2 * Z$.

Valor	-4	-3	-2	-1	0	3	4	5
em dom(Y)?	NÃO	NÃO	SIM	NÃO	SIM	SIM	SIM	SIM

- 2 b)** Aplicando Consistência de Nó, indique se os seguintes valores pertencem (**SIM**) ou **NÃO** ao domínio de **R1**, no problema das 8 rainhas, após toda a propagação gerada pela colocação de 3 rainhas conforme a figura abaixo (R3=7, R5=4, R6=2).

	1	2	3	4	5	6	7	8
R1								
R2								
R3							●	
R4								
R5				●				
R6		●						
R7								
R8								

Valor	1	2	3	6	8
em dom(R1)?	NÃO	NÃO	SIM	NÃO	NÃO

- 2 c)** Aplicando Consistência de Arco, indique se os seguintes valores pertencem (**SIM**) ou **NÃO** ao domínio de **A**, após a propagação num problema sobre as variáveis $A::0..4$, $B::0..3$, $C::[0,1,3,4]$, sujeito a $B \# = C - 1$, $Restr(A, B)$, $Restr(B, A)$, $Restr(A, C)$, e $Restr(C, C)$, onde $Restr(X, Y)$ é a restrição indicada pela tabela dos pares possíveis abaixo.

Restr(X, Y)

X	0	0	1	2	2	3
Y	0	1	2	2	3	3

Valor	0	1	2	3	4
em dom(A)?	NÃO	NÃO	SIM	NÃO	NÃO

Grupo 3 (10 valores)

Implemente em Prolog os seguintes predicados, procurando sempre que possível uma solução simples, eficiente, e legível:

3 a) **sublista(+Lista, -Sublista)**, que devolve uma *Sublista* dada uma *Lista*. Use apenas uma cláusula, recorrendo ao predicado de sistema *append(?Lista1, ?Lista2, ?ListaConcatenacao)*. (Nota: esta não será a melhor codificação, pois poderá até devolver sublistas repetidas, nomeadamente a lista vazia.)

E.g. `?- sublista([a,b,c,d], S).`
`S=[] ; S=[a] ; S=[b] ; S=[c] ; S=[d] ; S=[a,b] ; S=[b,c] ; ...`

```
sublista(L, Sub):- append(Prefix,_,L), append(_,Sub,Prefix).
```

3 b) Aproveitando o predicado anterior (**sublista/2** já definido), implemente **nums(+Lista, -Numeros)**, que dada uma **Lista** de termos, devolve a lista **Numeros** com todas as sublistas não vazias de *Lista* que contêm apenas números. Use também apenas uma cláusula, recorrendo ao *findall/3*. (Nota: também aqui, embora simples, esta não será a codificação mais eficiente.)

E.g. `?- nums([a,1,2,f(4),[8],5,5,-2.3], L).`
`L = [[1],[2],[1,2],[5],[5],[5,5],[-2.3],[5,-2.3],[5,5,-2.3]]`

```
nums(L, Nums):- findall(Ns,
    (sublista(L,Ns), Ns\=[], \+ (member(X,Ns), \+ number(X))), Nums).
```

Nº:	
-----	--

3 c) Implemente agora **nums_ord(+Listas, -Ordenadas)**, que dada uma lista, **Listas**, de listas de números (como a devolvida pelo predicado da alínea anterior) devolve apenas aquelas onde os números estão ordenados crescentemente com saltos de 1.

E.g. `?- nums_ord([[2,-1],[3,4],[1,2,2],[8],[1.5,2.5,3.5],[5,7]], Ords).`
`Ords = [[3,4],[8],[1.5,2.5,3.5]]`

```
nums_ord([Ns|L], [Ns|Os]):- ord(Ns), !, nums_ord(L, Os).
nums_ord([_L], Os):- nums_ord(L, Os).
nums_ord([], []).
```

```
ord([_]).
ord([A,B|T]):- B == A+1, ord([B|T]).
```

3 d) **seqs(+Listas, -Termos)**, que dada uma lista de listas de números, devolve a lista com as mesmas sequências, pela mesma ordem, mas agrupadas em termos com o functor **seq**.

E.g. `?- seqs([[1,2],[8],[5,5,-2.3]], L).`
`L = [seq(1,2), seq(8), seq(5,5,-2.3)]`

```
seqs([], []).
seqs([Ns|L], [Seq|Ss]):- Seq =.. [seq|Ns], seqs(L, Ss).
```

3 e) **somas(+Arvore, -Somas)**, que dada uma árvore binária de números não vazia (uma árvore tem a forma *nil*, para árvore vazia, ou *arv(Nó,Esq,Dir)*, onde *Nó* é a raiz e *Esq* e *Dir* são também árvores), devolve a lista com as duas somas de elementos correspondendo aos seus dois ramos, esquerdo e direito (ambos incluindo a raiz).

E.g.

```
?- somas(arv(10, arv(2,nil,nil), arv(-1,arv(5,nil,nil),arv(4,nil,nil))), L).
L = [12,18]      %10+2 e 10-1+5+4
```

```
somas(arv(No,E,D), [SE,SD]):- soma(E, No,SE), soma(D, No,SD).
soma(nil, S,S).
soma(arv(No,E,D), Si,So):- S1 is Si+No, soma(E, S1,S2), soma(D, S2,So).
```

3 f) **elementos(+Lista, +Elem1,+Elem2)**, que verifica se uma dada lista contém apenas os elementos **Elem1** e **Elem2**, e que a sua distribuição é equilibrada, i.e. ou estão em igual número ou há apenas mais uma ocorrência dum elemento do que do outro (no caso de listas com um número ímpar de elementos). Se a **Lista** contiver variáveis, estas devem ser instanciadas não deterministicamente com um desses elementos. Exemplos:

```
?- elementos([2,1,1,2,2],1,2). yes      ?- elementos([a,1,b],a,b). no
?- elementos([y,A,x,B],x,y).
A=x, B=y ; A=y, B=x ; no
```

```
elementos(Lista, A,B):- elems(Lista, A,B, 0,0).

elems([], _,_, NA,NB):- abs(NA-NB)=<1.
elems([A|T], A,B, NA,NB):- NA1 is NA+1, elems(T, A,B, NA1,NB).
elems([B|T], A,B, NA,NB):- NB1 is NB+1, elems(T, A,B, NA,NB1).
```

Nº:	
-----	--

3 g) Usando o predicado da alínea anterior (*elementos/3*) como já definido, implemente **galo(-Galo)**, que devolve um tabuleiro do jogo do galo (3 linhas, na forma de lista de 3 listas de 3 elementos) já preenchido com as jogadas *x* e *o*, numa posição de empate após um jogo completo válido.

E.g. ?- galo(G).

G = [[x,x,o],[o,o,x],[x,o,o]] ;

G = [[x,o,x],[x,x,o],[o,x,o]] ; ...

```
galo([[A,B,C],[D,E,F],[G,H,I]]):-
    elementos([A,B,C,D,E,F,G,H,I], x,o),
    \+ (member(L, [[A,B,C],[D,E,F],[G,H,I],[A,D,G],[B,E,H],[C,F,I],[A,E,I],[C,E,G]]),
        (L=[x,x,x] ; L=[o,o,o])).
```

3 h) ifs(+Lista, -Nifs, -Codigo), que dada uma **Lista** com instruções representando uma acção numa linguagem de computador, conforme a gramática (Codigo) abaixo, devolve o respectivo número **Nifs** de *if*'s utilizados, bem como o código, **Codigo**, num termo **A**, onde **A** é a acção apresentada na lista de código com *accao(A)*, conforme a linha 1 da gramática abaixo; ou **if(Cond,CodigoThen)** (linha 2, para um *if* simples, sem *else*); ou **if(Cond,CodigoThen,CodigoElse)** (linha 3: *if* com *else*). **Utilize DCGs (Gramáticas).**

- (1) Codigo $\rightarrow$ **accao(A)**
- (2) Codigo $\rightarrow$ **if cond(Cond) then** Codigo
- (3) Codigo $\rightarrow$ **if cond(Cond) then** Codigo **else** Codigo

Exemplos:

```
?- ifs([accao(xpto)], N, Cod) .
N=0, Cod=xpto ;
no
?- ifs([if,cond(c1),then,accao(a1),else,accao(a2)], N, Cod) .
N=1, Cod=if(c1,a1,a2) ;
no
?- ifs([if,cond(c1),then,if,cond(c2),then,accao(a1),else,accao(a2)],N,Cod) .
N=2, Cod=if(c1,if(c2,a1,a2)) ;
N=2, Cod=if(c1,if(c2,a1),a2) ;
no
?- ifs([if,cond(c1),then,accao(a1),else,if,cond(c2),then,accao(a2)],N,Cod) .
N=2, Cod=if(c1,a1,if(c2,a2)) ;
no
```

```
ifs(LCodigo, Nifs, Codigo):- phrase(accao(0,Nifs,Codigo), LCodigo).
```

```
accao(N,N, A) --> [accao(A)].
```

```
accao(Ni,No, Cod) -->
    [if,cond(Cond),then],
    {N1 is Ni+1},
    accao(N1,N2, CodThen),
    else(N2,No, Cond,CodThen,Cod).
```

```
else(Ni,No, Cond, CodThen, if(Cond,CodThen,CodElse)) --> [else], accao(Ni,No, CodElse).
```

```
else(N,N, Cond, CodThen, if(Cond,CodThen)) --> [].
```


Nº:	
-----	--

Grupo 4 (3 valores)

O problema dos quadrados latinos consiste em colorir um tabuleiro quadrado $N \times N$ com N cores, de maneira a que em cada linha e em cada coluna, todas as casas tenham cores diferentes. Resolva este problema com Restrições em SICStus Prolog, com o predicado **latin(+Board)** onde **Board** é já uma lista de listas, representando as linhas do tabuleiro com variáveis e, possivelmente, algumas casas já preenchidas (usaremos a codificação dos inteiros de 1 a N para representar as diferentes cores).

Exemplos:

```
?- latin([ [A,B], [C,D] ]).
A=1, B=2, C=2, D=1 ;
A=2, B=1, C=1, D=2 ;
no
?- latin([ [2,A,B], [C,1,2], [D,E,F] ]).
A=3, B=1, C=3, D=1, E=2, F=3 ;
no
```

```
:- use_module(library(clpfd)).
:- use_module(library(terms)).

latin(L):-
    length(L,N),
    dominios(L,N),
    colunas(L, Cols),
    diferentes(L),
    diferentes(Cols),
    term_variables(L, Vars),
    labeling([ff], Vars).

dominios([],_).
dominios([L/Ls], N):- domain(L, 1,N), dominios(Ls,N).

diferentes([]).
diferentes([L/Ls]):-all_distinct(L), diferentes(Ls).

colunas([], []).
colunas(L, [Col/Cs]):- cabecas(L, Col, L1), colunas(L1, Cs).

cabecas([[[H/T]Ls], [H/Hs], [T/Ts]]):- cabecas(Ls, Hs, Ts).
cabecas([], [], []).
```