DI-FCT/UNL **17 de Junho de 2008**

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.

true, fail	F	1+1*2 =:= 4-1	V
true ; fail	V	[a,b,c|D] = [a,b|_]	V
true = Fail	V	var(f(X))	F
X+Y = Z-X	F	member(X,[1,2,3,X]), X=2, !	V
f(u8)=Z	V	a \== a(X)	V

1 b) Prencha correctamente os quadrados em branco do predicado **soma_inverte(+Lista, -Soma, -ListaInvertida)** abaixo, de modo a que, dada uma **Lista** de números, devolva a sua **Soma**, bem como a sua inversão (em **ListaInvertida**):

```
soma_inverte(Lista, Soma, ListaInvertida):-
    soma_inverte(Lista, 0,Soma, [],ListaInvertida).

soma_inverte( [] , S, S , R , R).

soma_inverte([H|T], Si,So, Ai, Ao ):-
    S1 is Si+H ,
    soma_inverte( T , S1,So, [H|Ai] , Ao).
```

1 c) Para o seguinte programa

```
p(1). p(2). p(3).
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=1`
`?- r(R,3).`	`no`
`?- r(10,R).`	`no`
`?- r(20,R).`	`R=2`
`?- s(R).`	`R=20`
`?- 3=X, s(X), p(X).`	`no`

1 d) No programa abaixo, qual deverá ser a precedência ***Prec***, e a associatividade ***Assoc***, na definição do operador '&', de modo a que tenha o comportamento desejado no tratamento de dados de bilhetes de identidade (BI)?

```
:- op(Prec, Assoc, &).

'Ana Silva' & 'F' & 5904391.
'Rui Ferreira' & 'M' & 8788342.

num_bi(Num):- _ & Num.
bi_recente(Nome, Num):- Nome & _ & Num, Num >= 8000000.
```

Prec = (e.g.) **600** (<1000) **Assoc =** **yfx**

Nota: Os operadores pré-definidos neste programa são ':-', ',', e '>=', com precedências 1200, 1000, e 700, respectivamente.

N º :	

Grupo 2 (3 valores)

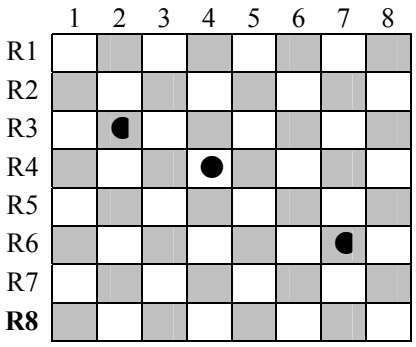
Consistência de Restrições.

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::[1,3,4,6,7,9], Y::[0..3,6,8..12], Z :: 0..12, X#=<Y, Y#<Z, Z-1#=<X`.

Valor	0	1	2	3	7	8	9	10	11	12
em dom(Y)?	NÃO	SIM	SIM	SIM	NÃO	SIM	SIM	NÃO	NÃO	NÃO

2 b) Aplicando Consistência de Nó, indique se os seguintes valores pertencem (**SIM**) ou **NÃO** ao domínio de **R8**, no problema das 8 rainhas, após toda a propagação gerada pela colocação de 3 rainhas conforme a figura abaixo (R3=2, R4=4, R6=7).



Valor	1	2	3	6	8
em dom(R8)?	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::1..4, B::1..4, C::[0,1,3,4,5], sujeito a *Restr(A,B)*, *Restr(B,A)*, e *Restr(A,C)*, onde *Restr(X,Y)* é a restrição indicada pela tabela dos pares possíveis abaixo.

Restr(X,Y)

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

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

Grupo 3 (10 valores)

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

3 a) **par(+Grupos, -Par)**, que dada uma lista de listas (em **Grupos**), devolve um par **X-Y**, onde **X** é o 2º elemento duma dessas listas, e **Y** é o 2º elemento doutra dessas listas. Por retrocesso deverão ser gerados todos esses possíveis pares.

E.g.
```
?- par([[a,b,c],[x,y,z],[1,2,3,4]], P).
P = b-y ;
P = b-2; .... P = 2-y
```

(Um tal predicado até pode ser útil para gerar possíveis jogos de 'play-off', tendo as classificações de vários grupos, onde os primeiros classificados já estão apurados.)

```
par(L, A-B):- member([_,A|_], L), member([_,B|_], L), A\=B.
```

3 b) **pares(+Grupos, -Pares)**, que, assumindo o predicado da alínea anterior já definido, devolve a lista de todos os pares (de 2os elementos de Grupos).

```
pares(L, Ps):- findall(P, par(L,P), Ps).
```

Nº:	

3 c) **descendente(+No, +Arvore, ?Descendente)**, que dada uma **Árvore** binária não ordenada de termos constantes sem repetições, e um Nó **No**, devolve **Descendente**, um nó terminal descendente de **No** em **Arvore**. (Uma árvore pode ser a constante nil, se vazia, ou um termo arv(X, Esq, Dir), onde X é a raiz, e Esq e Dir são também árvores)

```
descendente(No, arv(No,Esq,Dir), Desc):- !, terminal(arv(No,Esq,Dir), Desc).
descendente(No, arv(X,Esq,Dir), Desc):-  %No \= X,
        (descendente(No, Esq, Desc) ; descendente(No, Dir, Desc)).

terminal(arv(No,nil,nil), No):- !.
terminal(arv(_,Esq,_), No):- terminal(Esq, No).
terminal(arv(_,_,Dir), No):- terminal(Dir, No).
```

3 d) **subtermo(+Termo, -UmSubtermo)**, que dado um Termo, devolve um seu subtermo.

E.g. ?- subtermo(f(a,1+2,xy(0,g(u))), Sub).
Sub = f(a,1+2,xy(0,g(u))) ;
Sub = a ;
Sub = 1+2 ;
Sub = xy(0,g(u)) ;
Sub = 1 ;
…

```
%Resolução para um termo constante (i.e. sem variáveis), o que não foi
%referido no enunciado mas cuja assunção foi considerada correcta

subtermo(Termo, Termo).
subtermo(Termo, Sub):-
       Termo =.. [_|Args], member(Arg,Args), subtermo(Arg,Sub).
```

3 e) **ordena(+Lista, -ListaOrdenada)**, para ordenar uma Lista (sem repetições). Use o *setof/3* para o implementar numa só cláusula.

```
ordena(Lista, Ordenada):- setof(X, member(X,Lista), Ordenada).
```

3 f) **forma(+Termo, -Forma)**, que dado um Termo devolve outro termo, **Forma**, com o mesmo functor e a mesma aridade, mas em que todos os argumentos são variáveis (diferentes). Use apenas uma cláusula.

E.g.
```
?- forma(f(8,[],y), T).
T = f(A,B,C)
```

```
forma(Termo ,Forma):- functor(Termo, Functor, N), functor(Forma, Functor, N).
```

N º :	

3 g) **sufixo(+ListaDiferenca, +PrefixoDiferenca, -Sufixo)**, que dada uma lista de diferença **ListaDiferenca** (na forma L-V), e outra lista de diferença **PrefixoDiferenca**, correspondendo a um seu prefixo, devolve o remanescente **Sufixo**, sob a forma duma lista simples.

E.g. `?- sufixo([a,b,c,d|T]-T, [a,b|L]-L, Sufixo).`
`Sufixo = [c,d]`

```
sufixo(L-[], L-S, S).
```

3 h) **bool(+Bits, -Num)**, que dada uma lista de **Bits** (0/1), representando um número boleano, devolve o respectivo número, **Num**, na habitual representação decimal. Utilize DCGs (Gramáticas).

E.g.
```
?- bool([1,0,0,1,0], N).
N = 18 ;
no
```

```
bool(Bits, Num):- phrase(bool(0,Num), Bits).

bool(N,N) --> [].
bool(Ni,No) --> [B], {N1 is 2*Ni+B}, bool(N1,No).
```

N º :	

Grupo 4 (3 valores)

Um fabricante de brinquedos está a planear a produção de 2 brinquedos para a campanha de Natal. Para o brinquedo 1, o custo de início de produção é de 50 k€ gerando um lucro por brinquedo de 10 €. Para o brinquedo 2 os valores são respectivamente 80 k€ e 15 €. Embora o fabricante tenha duas fábricas, foi decidido usar apenas uma delas para evitar duplicar os custos de início de produção. O brinquedo 1 pode ser produzido a um ritmo de 50 por hora na fábrica A e de 40 por hora na fábrica B, enquanto que para o brinquedo 2 os valores respectivos são de 40 e 25 (os valores estão resumidos na tabela abaixo).
A fábrica A tem 500 horas de produção disponível enquanto que a fábrica B tem 700 horas.

	Custo inicial	Lucro unitário	Ritmo A	Ritmo B
Brinquedo 1	50000€	10€	50/h	40/h
Brinquedo 2	80000€	15€	40/h	25/h

Pretendendo-se maximizar o lucro obtido nesta campanha, resolva o problema com Restrições em ECLiPSe, implementando o(s) necessário(s) predicado(s). Indique o significado das variáveis de domínio que escolher.
Nota: não havendo restrições relacionando os brinquedos, pode-se assumir que só será fabricado um brinquedo (o que se revelar mais lucrativo)

```
:- lib(fd).

/*
 F = 0 -> Fábrica A
 F = 1 -> Fábrica B
 B = 0 -> Brinquedo 1
 B = 1 -> Brinquedo 2
*/
natal(F,B, Lucro):-
        [F,B] :: [0,1],
        Prod1 #= 50*500*(1-F) + 40*700*F,
        Prod2 #= 40*500*(1-F) + 25*700*F,
        Lucro #= (1-B)*(10*Prod1-50000) + B*(15*Prod2-80000),
        MLucro #= -Lucro,
        minimize(labeling([F,B], MLucro).
```