Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Programação em Lógica com Restrições

2008/09 (2º semestre) – 16 Junho de 2009

Exame de Época Normal	**Duração:** 2h30	***SEM CONSULTA***
Nome: _____		Nº _____

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Total
1	2	2	2	1	2	1	1	3	1	4	**20**

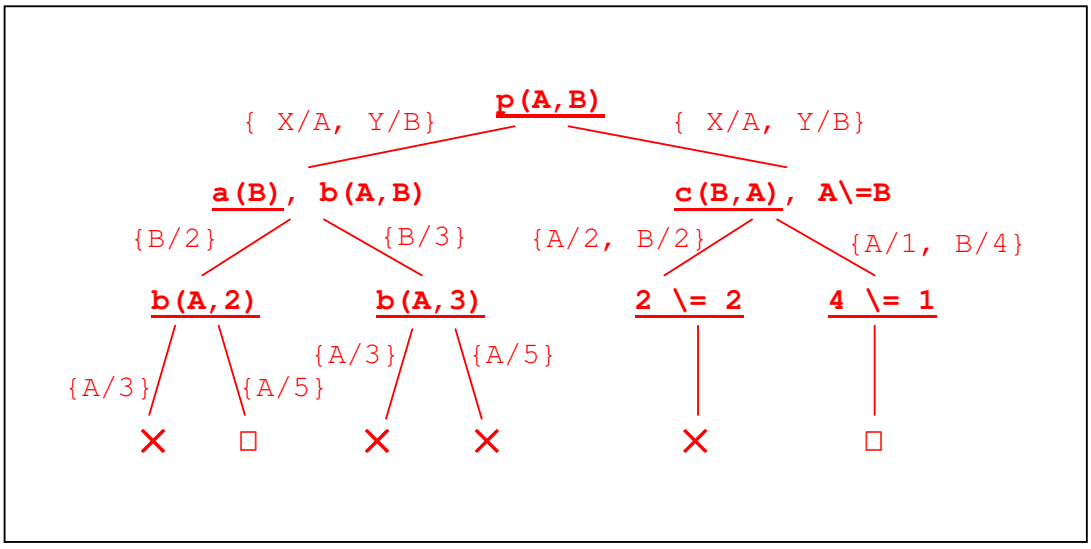
Q1 (1val)
Mostre que os predicados p/2 e r/2, não são equivalentes, nomeadamente designando valores termos X e Y para o qual um predicado seja verdadeiro (suceda) e o outro não.

```
p(X,Y) :- a(X), !, b(Y).
p(2,2).

r(X,Y) :- a(X), b(Y), !.
r(2,2).

a(1).  a(2).  a(4).
b(1).  b(3).  b(3).
```

Resposta: `X = 2  Y = 2`

Q2 (2val)
Desenhe a árvore de execução do predicado `p/2`, chamado com variáveis A e B, i.e. `p(A,B)`. Indique as substituições feitas nos nós intermédios da árvore, bem como as soluções obtidas.

```
p(X,Y) :- a(Y), b(X,Y).
p(X,Y) :- c(Y,X), X\= Y.

a(2), a(3).
b(3,1). b(5,2).
c(2,2). b(4,1).
```



Q3 (2 val)
Dada uma lista L cujos elementos são números inteiros, defina o predicado **inversoes(+L,-N)** que associa essa lista ao número N de inversões que a lista contem. Uma inversão é definida como uma sequência de 3 números em que o número do meio é maior ou menor que os outros dois. Por exemplo, para L = [1,3,2,4,6,5,7,8], a chamada inversoes(L,N) deve retornar N = 4.

```
inversoes([],0).
inversoes([_],0).
inversoes([_,_],0).
inversoes([A,B,C|T],N):-
   check([A,B,C],K),
   inversoes([B,C|T],M),
   N is M + K.

check([A,B,C],1):- A > B, B < C, !.
check([A,B,C],1):- A < B, B > C, !.
check([_,_,_],0).
```

Q4 (2 val)
Considere uma lista "dicionário", **Dic**, que contem pares $[c_i\text{-}d_i]$, associando a cada letra minuscula $\mathbf{c_i}$ uma outra letra minúscula $\mathbf{d_i}$. Implemente um predicado **`codifica(S,Dic,T)`** em que T representa a string obtida por transformação da string S através do dicionário Dic. Por exemplo, para **`S = 'abcd'`** e **`Dic = [a-w,b-x,c-y,d-z]`** deverá ser **`T = 'wxyz'`**. **Nota:** Pode utilizar o predicado auxiliar **`conv(?S,?L)`** que relaciona uma string S com a lista L dos seus caracteres (por exemplo, **`conv('abcd',[a,b,c,d])`**, mas a sua não implementação desconta 1 **`valor`**.

```
codifica(S,Dic,T):-
    conv(S,Ls),
    cod(Ls,Dic,Lt),
    conv(T,Lt).

conv(S,L):- var(L), name(S,A), chars(A,L).
conv(S,L):- var(S), chars(A,L), name(S,A).

chars([],[]).
chars([H|T],[S|Cs]) :- name(S,[H]), chars(T,Cs).

cod([],_,[]).
cod([C|T], Dic, [D|R]):-
   member(C-D,Dic) ,
   cod(T,Dic,R).
```

Q5 (1 val)
Defina o predicado **`mbv(V,L)`** que sucede sse a variável V (o 1º argumento) é membro da lista L (o 2º argumento) cujos membros são variáveis.

```
mbv(X,[H|_]):- X == H.
mbv(X,[_|T]):- mbv(X,T).
```

Q6 (2 val)

Defina o predicado **`vars(+T,-L)`** que dado um termo T retorna a lista L que contem todas as variáveis de T, sem repetições, mas por qualquer ordem. Considere os seguintes exemplos:.

S	2	p(X,Y)	[p(X,3), f(Y,Z),X]	p(p(X),f(X))	p(Y,X,Y)
T	[]	[X,Y]	[X,Y,Z]	[X]	[Y,X]

Considere se achar útil o predicado pré-definido **`=..`**, bem como o predicado **`mbv/2`** definido na questão anterior.

```
vars(T,L):- vars(T,[],L).

vars(X,L,L):-     var(X), mbv(X,L).
vars(X,L,[X|L]):- var(X), \+ mbv(X,L).
vars([],L,L).
vars([H|T],L,Vs) :- vars(H,L,X), vars(T,X,M).
vars(P,L,Vs) :- P =..[_|R], vars(R,L,Vs).
```

Q7 (1 val)

Considere um predicado **`p(?X)`** que define os termos que têm uma propriedade p, isto é, sucede sse o termo **`X`** tem a propriedade p. Pretende-se definir um predicado **`separa(L,D1,D2)`** , que obtenha duas listas de diferença, D1 e D2, contendo os termos de L que, respectivamente, têm ou não têm a propriedade p.

Por exemplo, o predicado **`separa([1,a,3,d,2,b],[1,3,2|X]-X, [a,d,b|Z]-Z])`** deve ser verdadeiro de acordo com a definição, assumindo que o predicado p(X) significa "X é um inteiro".

```
separa([],X-X,Z-Z).
separa([H|T],[H|W]-X,Y-Z):-     p(H), separa(T,W-X,Y-Z).
separa([H|T],W-X,[H|Y]-Z):- \+ p(H), separa(T,W-X,Y-Z).
```

Q8 (1 val)

Pretende-se, a partir do predicado **`separa/3`** da questão anterior, definir o predicado **`ord_p(+L,-S)`**, que permite obter a lista L, ordenada de forma a que os seus elementos que têm a propriedade p apareçam antes dos que não têm essa propriedade, mantendo-se as posições relativas dentro destes subsconjuntos de termos. Em particular, para o exemplo da questão anterior deverá ser **`ord_p([1,a,3,d,2,b],[1,3,2,a,d,b]).`**

```
ord_p(L,S):- separa(L,S-X,X-[]).
```

Q9 (3 val)

Pretende-se verificar se numa string S, composta por caracteres alfanuméricos (maiúsculas, minúsculas e digitos), existem sequências de K dígitos. Para esse efeito, especifique uma gramática que aceite atrings desse tipo, e apenas desse tipo, quando chamada com o predicado aceita/4 definido como

```
aceita(As,K,Ds) :- name(As,L), teste(K, X, L, Z), name(Ds,X).
```

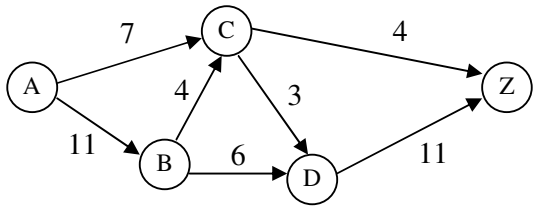
Por exemplo, para As = 'abc3457hd189g', teremos três respostas para a chamada do predicado **aceita(S,3,X), retornando Ds = '345', '457' e '189'.**

```
teste(K,X) -> t(K,K,[],X).
t(N,K,S,X) -> [C],{N>0, digit(C), M is N-1, cat(S,[C],T)}, t(M,K,T,X).
t(N,K,_,X) -> [C],{N>0, \+digit(C)}, t(K,K,[],X).
t(0,_,X,X) -> [].
t(0,K,[_|T],X) -> t(1,K,T,X).

digit(C):-
   name('0',[Zero]), name('9',[Nove]),
   Zero =< C, C =< Nove.

cat([],L,L).
cat([H|T],L,[H|R]) :- cat(T,L,R).
```

Q10 (1 val) Restrições Lineares

Pretende-se verificar qual o número máximo de Mbits que podem ser transmitidos por segundo do nó A para o nó Z através da rede indicada na figura. Cada arco tem uma etiqueta C_{ij} indicando a capacidade da ligação (o número máximo de Mbits que podem ser transmitidos por segundo).

Especifique o predicado max_trans(-K) que retorna o valor máximo M de Mbits que podem ser transmitidos por segundo entre A e Z.




```
trans(M):-
   { M = X1 + X2,
     X1 =< 7, X2 =< 11, X3 =< 4, X4 =< 6,
     X5 =< 3, X6 =< 4, X7 =< 11,
     X2 = X3 + X4,
     X1 + X3 = X5 + X6,
     X4 + X5 = X7
   },
   maximize(M).
```

Q11 (4 val) Domínios Finitos

O lema de Schur postula a existência de solução para o seguinte problema: dado um conjunto de bolas, numeradas de 1 a n, é possível colorir as bolas com 3 cores (por conveniência codificadas por inteiros 1,2 e 3) de forma a que:

- As bolas i e 2i não tenham a mesma cor
- Se as bolas i e j têm a mesma cor, a bola i+j tem uma cor diferente.

Especifique o predicado schur(+N,-L) que determina a lista L = [1-C_1, 2-C_2,..., N-C_N] para o número N dado, e em que as cores C_1, .., C_N satisfazem o lema de Schur.

Nota: Utilize se entender as restrições condicionais disponíveis no SICStus.

Sugestão: Utilize os predicados pairs(L,Ps) e triples(L,Ts) que, respectivamente, retornam uma lista com todos pares de variáveis (Ps = [C1-C2, C1-C3, CN-1-CN]) e uma lista com todos os triplos de variáveis (Ts = [C1-C2-C3, C1-C3-C4, ...]) que devem ser avaliadas. Caso não especifique estes predicados a questão apenas vale 3 valores.

```
schur(N,Vs):-
   vars(1,N,L,Vs),
   domain(Vs,1,3),
   cons_ij(L),
   cons_ijk(L),
   labeling([ff],Vs).

vars(I,N,[],[]):- I > N.
vars(I,N,[I-X|T],[X|R]):- I =< N,
   J #= I+1,
   vars(J,N,T,R).

cons_ij(L):- pairs(L,Ps), c_ij(Ps).
cons_ijk(L):- triples(L,Ts), c_ijk(Ts).

ts(I-Xi,[J-Xj|T],Qs,Rs):- K #= I+J, member(K-Xk,T), !,
    ts(I-Xi,T,[Xi-Xj-Xk|Qs],Rs).
ts(_,_,Rs,Rs).

c_ij([]).
c_ij([X-Y|T]) :- X #\= Y, c_ij(T).

c_ijk([]).
c_ijk([X-Y-Z|T]) :- (X #= Y) #=> (Z #\= X), c_ijk(T).


pairs(L, Ps):- pairs(L,[],Ps).
pairs([I-Xi|T],Qs,Ps):- J #= I*2, member(J-Xj,T), !,
   pairs(T,[Xi-Xj|Qs],Ps).
pairs(_,Ps,Ps).

triples(L,T):- triples(L,[],T).
triples([],Ts,Ts).
triples([I-Xi|T],Qs,Ts):-
   ts(I-Xi,T,[], Rs),
   cat(Qs,Rs,Ss),
   triples(T,Ss,Ts).
```