

# Relatório do 2º Trabalho Prático

## **Inteligência Artificial**

Universidade de Évora  
Engenharia Informática 2020/2021



Docente: Irene Rodrigues  
Discentes: Leonardo Catarro, 43025  
Diogo Solipa, 43071

# Desenvolvimento

## Exercício 1

a) Representação de estados, estado inicial, operador sucessor e restrições

- $e(v(c(x), D, A)) \rightarrow c(x)$  – representa a cadeira  $x$   
D – domínio, ou seja, as pessoas  
A – inicialmente é ignorada, mas no fim guarda a pessoa que está sentada na cadeira  $c(x)$

```
/*Estado Inicial*/
estado_inicial(e([
    v(c(1),D,_),
    v(c(2),D,_),
    v(c(3),D,_),
    v(c(4),D,_),
    v(c(5),D,_),
    v(c(6),D,_),
    v(c(7),D,_),
    v(c(8),D,_)], [])):- pessoas(D).

/*Pessoas*/
pessoas(['Maria', 'Manuel', 'Madalena', 'Joaquim', 'Ana', 'Julio', 'Matilde', 'Gabriel']).
```

Estado inicial

```
/*Sucessor*/
sucessor(e([v(N,D,V) | R],E),e(R,[v(N,D,V) | E])):- member(V,D).
```

Operador sucessor

```
/*Restricoes*/
restrict(I, X, Y, J) :- restricoes(L), member(esq(X, Y), L), \+ (I is J+1; (I=1, J=8)).
restrict(I, X, Y, J) :- restricoes(L), member(dir(X, Y), L), \+ (I is J-1; (I=8, J=1)).
restrict(I, X, Y, J) :- restricoes(L), member(lado(X, Y), L), \+ ((I is J+1; (I=1, J=8));(I is J-1;(I=8, J=1)) ).
restrict(I, X, _, _) :- restricoes(L), member(cabeceira(X),L), \+ (I=1; I=5).
restrict(I, X, Y, J) :- restricoes(L), member(frente(X, Y), L), \+ (I is J-4;(I=4, J=6);(I=2, J=8)).
restrict(I, X, Y, J) :- I \= J, X=Y.

restricoes([esq('Manuel','Maria'), lado('Joaquim', 'Matilde'), frente('Joaquim','Maria'), cabeceira('Gabriel')]).

/*Ve Restricoes*/
ve_restricoes(e( Nafec,Afect)):-
    \+ (member(v(c(I), Di,Vi), Afect),
        member(v(c(J), Dj,Vj),Afect),
        restrict(I,Vi,Vj,J)).
```

Restrições

**b) Resolução do problema com o algoritmo backtracking**

```
/*Pesquisa Backtracking*/  
back(e([],A),A).  
back(E,Sol):-  
    sucessor(E,E1),  
    inc,  
    ve_restricoes(E1),  
    back(E1,Sol).
```

```
pesquisa_back:-  
    nos(0),  
    estado_inicial(E0),  
    back(E0,A),  
    sort(A, L),  
    esc(L),  
    nos(N),  
    write('Nós: '),  
    write(N).
```

**Nota:** da execução do problema com algoritmo backtracking resultaram diversos resultados. De seguida vamos apresentar o resultado mais eficiente(em termos de nós visitados)

**Primeiro Output:**

```
v(c(1),[Maria,Manuel,Madalena,Joaquim,Ana,Julio,Matilde,Gabriel],Manuel)  
v(c(2),[Maria,Manuel,Madalena,Joaquim,Ana,Julio,Matilde,Gabriel],Madalena)  
v(c(3),[Maria,Manuel,Madalena,Joaquim,Ana,Julio,Matilde,Gabriel],Matilde)  
v(c(4),[Maria,Manuel,Madalena,Joaquim,Ana,Julio,Matilde,Gabriel],Joaquim)  
v(c(5),[Maria,Manuel,Madalena,Joaquim,Ana,Julio,Matilde,Gabriel],Gabriel)  
v(c(6),[Maria,Manuel,Madalena,Joaquim,Ana,Julio,Matilde,Gabriel],Ana)  
v(c(7),[Maria,Manuel,Madalena,Joaquim,Ana,Julio,Matilde,Gabriel],Julio)  
v(c(8),[Maria,Manuel,Madalena,Joaquim,Ana,Julio,Matilde,Gabriel],Maria)  
Nós: 2492
```

**c) Resolução do problema com o algoritmo forward checking**

```
/*Pesquisa ForwardChecking*/  
forward(e([],A),A).  
forward(E,Sol):-  
    sucessor(E,E1),  
    inc,  
    ve_restricoes(E1),  
    forCheck(E1,E2),  
    forward(E2, Sol).
```

```
pesquisa_forward:-  
    nos(0),  
    estado_inicial(E0),  
    forward(E0,A),  
    sort(A, L),  
    esc(L),  
    nos(N),  
    write('Nós: '),  
    write(N).
```

```
forCheck(e(Lni,[v(N,D,V)|Li]), e(Lnii, [v(N,D,V)|Li])) :- corta(V,Lni,Lnii).  
corta(_,[],[]).  
corta(V, [v(N,D,_)|Li], [v(N,D1,_)|Lii):- delete(D,V,D1), corta(V,Li,Lii).
```

**Nota:** da execução do problema com algoritmo forward checking resultaram diversos resultados. De seguida vamos apresentar o resultado mais eficiente(em termos de nós visitados)

**Primeiro Output:**

```
v(c(1), [Maria, Manuel, Madalena, Joaquim, Ana, Julio, Matilde, Gabriel], Manuel)
v(c(2), [Maria, Madalena, Joaquim, Ana, Julio, Matilde, Gabriel], Madalena)
v(c(3), [Maria, Joaquim, Ana, Julio, Matilde, Gabriel], Matilde)
v(c(4), [Maria, Joaquim, Ana, Julio, Gabriel], Joaquim)
v(c(5), [Maria, Ana, Julio, Gabriel], Gabriel)
v(c(6), [Maria, Ana, Julio], Ana)
v(c(7), [Maria, Julio], Julio)
v(c(8), [Maria], Maria)
Nós: 887
```

**d) Alterações para melhorar complexidade temporal e espacial**

Para esta alínea procedemos à alteração das restrições e à ordem pela qual estas são avaliadas. Com tais alterações reparamos algumas mudanças em termos de nós visitados e tempos de execução.

Com algumas alterações o tempo e/ou espaço eram superiores, com outras diminui-a

**e) No relatório apresente os resultados para 4 exemplos diferentes:**

Para este foram apenas apresentados resultados para uma mesa de 8 pessoas. Para as restantes mesas o procedimento é semelhante, tendo em conta o maior ou menor numero de lugares/cadeiras. Por exemplo, com 12 pessoas, temos o mesmo numero de pessoas às cabeceiras(2), porém teremos mais pessoas à esquerda e direita. Para o exemplo de 4 pessoas, o mais simples, temos apenas uma pessoa a cada cabeceira, uma à esquerda e uma à direita.

**Exercício 2**

**a)**

- $e(v(p(x,y), D, A)) \rightarrow c(x)$  – representa a cadeira x

D – lista, com os algarismos de 1-9 para serem colocados nas posições do tabuleiro. Domínio.

A – inicialmente é ignorada, mas no fim guarda o algarismo colocado na posição  $p(x,y)$ .

```
estado_inicial(e([v(p(1, 1), [1,2,3,4,5,6,7,8,9], _), v(p(1, 3), [1,2,3,4,5,6,7,8,9], _), v(p(1, 4), [1,
v(p(2, 1), [1,2,3,4,5,6,7,8,9], _), v(p(2, 2), [1,2,3,4,5,6,7,8,9], _), v(p(2, 3), [1,
v(p(3, 2), [1,2,3,4,5,6,7,8,9], _), v(p(3, 3), [1,2,3,4,5,6,7,8,9], _), v(p(3, 4), [1,
v(p(4, 1), [1,2,3,4,5,6,7,8,9], _), v(p(4, 2), [1,2,3,4,5,6,7,8,9], _), v(p(4, 3), [1,
v(p(5, 1), [1,2,3,4,5,6,7,8,9], _), v(p(5, 2), [1,2,3,4,5,6,7,8,9], _), v(p(5, 3), [1,
v(p(6, 2), [1,2,3,4,5,6,7,8,9], _), v(p(6, 3), [1,2,3,4,5,6,7,8,9], _), v(p(6, 5), [1,
v(p(7, 1), [1,2,3,4,5,6,7,8,9], _), v(p(7, 2), [1,2,3,4,5,6,7,8,9], _), v(p(7, 3), [1,
v(p(8, 3), [1,2,3,4,5,6,7,8,9], _), v(p(8, 4), [1,2,3,4,5,6,7,8,9], _), v(p(8, 6), [1,
v(p(9, 1), [1,2,3,4,5,6,7,8,9], _), v(p(9, 2), [1,2,3,4,5,6,7,8,9], _), v(p(9, 4), [1,
[v(p(1, 2), [1,2,3,4,5,6,7,8,9], 1), v(p(1, 6), [1,2,3,4,5,6,7,8,9], 8), v(p(1, 8), [1,
v(p(2, 4), [1,2,3,4,5,6,7,8,9], 5), v(p(2, 6), [1,2,3,4,5,6,7,8,9], 9),
v(p(3, 1), [1,2,3,4,5,6,7,8,9], 7), v(p(3, 7), [1,2,3,4,5,6,7,8,9], 9), v(p(3, 9), [1,
v(p(4, 6), [1,2,3,4,5,6,7,8,9], 4),
v(p(5, 5), [1,2,3,4,5,6,7,8,9], 3), v(p(5, 6), [1,2,3,4,5,6,7,8,9], 5), v(p(5, 8), [1,
v(p(6, 1), [1,2,3,4,5,6,7,8,9], 8), v(p(6, 4), [1,2,3,4,5,6,7,8,9], 9),
v(p(7, 4), [1,2,3,4,5,6,7,8,9], 7),
v(p(8, 1), [1,2,3,4,5,6,7,8,9], 2), v(p(8, 2), [1,2,3,4,5,6,7,8,9], 6), v(p(8, 5), [1,
v(p(9, 3), [1,2,3,4,5,6,7,8,9], 5), v(p(9, 6), [1,2,3,4,5,6,7,8,9], 3]))).
```

Estado inicial(imagem não completa)

Nota: Derivado ao extenso estado inicial, pois contam todas as posições do tabuleiro(onde algumas já lhe foram atribuídos algarismos), obtemos por apenas mostrar uma pequena parte de forma a se entender a sua estrutura

	1				8		7	3
			5		9			
7						9		4
					4			
				3	5		1	8
8			9					
			7					
2	6			4				3
		5			3			

Tabuleiro resultado do estado inicial

```
% função sucessor
sucessor(e([v(N,D,V)|R],E),e(R,[v(N,D,V)|E])):-
    member(V,D).
```

Operador sucessor

```
%Restricoes
ve_restricoes(E):-
    ver_linhas(E),
    ver_colunas(E),
    ver_quadrantes(E).
```

Vê restrições

```
% verifica se os elementos numa lista sao todos diferentes
todos_diff([]).
todos_diff([X|R]):-
    \+ member(X,R), todos_diff(R).
```

Função auxiliar à aplicação das restrições

```
ver_linhas(e(Nafect,[v(p(X,Y), D, V)|R])):-
    findall(V1,member(v(p(X,_),_,V1),R),L),
    todos_diff([V|L]).

ver_colunas(e(Nafect,[v(p(X,Y), D, V)|R])):-
    findall(V1,member(v(p(_,Y),_,V1),R),L),
    todos_diff([V|L]).
```

Restrições das linhas e colunas

```
ver_quadrantes(e(_,Afect)) :-
    ver_quad(Afect, 1, 1, 3, Q1),
    todos_diff(Q1),
    ver_quad(Afect, 1, 4, 6, Q2),
    todos_diff(Q2),
    ver_quad(Afect, 1, 7, 9, Q3),
    todos_diff(Q3),
    ver_quad(Afect, 4, 1, 3, Q4),
    todos_diff(Q4),
    ver_quad(Afect, 4, 4, 6, Q5),
    todos_diff(Q5),
    ver_quad(Afect, 4, 7, 9, Q6),
    todos_diff(Q6),
    ver_quad(Afect, 7, 1, 3, Q7),
    todos_diff(Q7),
    ver_quad(Afect, 7, 4, 6, Q8),
    todos_diff(Q8),
    ver_quad(Afect, 7, 7, 9, Q9),
    todos_diff(Q9).
```

```
ver_quad(L, X, Y, Y2, Q) :-
    Y = Y2, X1 is X+2,
    add_quad(L, X, Y, X1, Q).
```

```
ver_quad(L, X, Y, Y2, Q) :-
    Y < Y2, Y1 is Y+1,
    X1 is X+2,
    add_quad(L, X, Y, X1, L1),
    append(L1, L2, Q),
    ver_quad(L, X, Y1, Y2, L2).
```

Restrições dos quadrantes

Restrições

Nota: No sudoku, existem restrições, por linha, coluna e quadrante. Em cada um deste não pode haver nenhum algarismo repetido

b) Resolva o problema com o algoritmo de backtracking

```
% função BackTracking
back(e([],A),A).
back(E,Sol):-
    sucessor(E,E1),
    ve_restricoes(E1),
    back(E1,Sol).
```

```
sudoku_back:-
    estado_inicial(E0),
    back(E0,A),
    esc(A).
```

Output:

```
| ?- sudoku_back.
5 . 1 . 9 . 4 . 2 . 8 . 6 . 7 . 3
6 . 3 . 4 . 5 . 7 . 9 . 1 . 8 . 2
7 . 2 . 8 . 3 . 1 . 6 . 9 . 5 . 4
3 . 5 . 2 . 1 . 8 . 4 . 7 . 9 . 6
9 . 7 . 6 . 2 . 3 . 5 . 4 . 1 . 8
8 . 4 . 1 . 9 . 6 . 7 . 3 . 2 . 5
4 . 9 . 3 . 7 . 5 . 2 . 8 . 6 . 1
2 . 6 . 7 . 8 . 4 . 1 . 5 . 3 . 9
1 . 8 . 5 . 6 . 9 . 3 . 2 . 4 . 7
```

c) Resolva o problema com o algoritmo de forward checking

```
% função forward checking
forward(e([],A),A).
forward(E,Sol):-
    sucessor(E,E1),
    ve_restricoes(E1),
    forCheck(E1, E2),
    forward(E2, Sol).

forCheck(e(Lni,[v(N,D,V)|Li]), e(Lnii,[v(N,D,V)|Li])) :- corta(V,Lni,Lnii).

corta(_,[],[]).
corta(V,[v(N,D,_)|NAfect],[v(N,DS,_)|NAfectS]):-
    delete(D,V,DS),
    corta(V, NAfect, NAfectS).
```

Não foi possível aplicar o algoritmo de forward checking no sudoku, derivado a um bug na função de forward checking. O output foi sempre “no”, mesmo após diversas tentativas de correção do bug!

#### Output:

```
| ?- sudoku_forward.  
recutabledPath- setting to  
(20 ms) no
```

## Conclusão

Com este trabalho conseguimos por em prática todo o conhecimento lecionado nas aulas teóricas e práticas da disciplina relativamente a problemas de satisfação de restrições.

Embora tenhamos tido dificuldades na resolução do 2º exercício, visto que não percebemos bem como lidar com a caixa e o agente como 1 só, e como resolver o output “no” obtido, no geral consideramos que conseguimos aplicar o conhecimento adquirido.

A experiência de desenvolver o Sudoku em Prolog, foi bastante desafiante, mas também gratificante e interessante.