



Relatório do 2º Trabalho Prático de Inteligência Artificial

Henrique Raposo nº33101

José Serra nº 33289

18 Março, 2019

1 Introdução

Neste trabalho tratamos dois conhecidos problemas sendo eles:

1. Quadrado mágico(3x3 e 4x4);
2. Sudoku;

Como enunciado, tratámos o quadrado mágico como um problema de CSP e representámos ambos como problemas de satisfação de restrições em Prolog, utilizando também os algoritmos de backtracking e forward checking, dados nas aulas teóricas da disciplina, de maneira a responder às várias alíneas enunciadas.

Nota: O relatório do trabalho apenas refere o código do Quadrado mágico 3x3, no entanto também estará disponível nos ficheiros do trabalho o tratamento do problema para o Quadrado 4x4, que decidimos não incluir no relatório pois grande parte da estrutura do código seria semelhante ao Quadrado 3x3.

2 Quadrado Mágico

2.1 Alinea a): Estados, Variáveis, Restrições

O código seguinte representa os estados, as variáveis(nome,domínio e valor) e as respectivas restrições para o Quadrado mágico3x3.

```
dominio([1,2,3,4,5,6,7,8,9]).
```

```
estado_inicial( e([v(n(1,1),D,_Num),  
                  v(n(1,2),D,_Num),  
                  v(n(1,3),D,_Num),  
                  v(n(2,1),D,_Num),  
                  v(n(2,2),D,_Num),  
                  v(n(2,3),D,_Num),  
                  v(n(3,1),D,_Num),  
                  v(n(3,2),D,_Num),  
                  v(n(3,3),D,_Num)],[]) ):- dominio(D).
```

```

%Restricoes:
%-todos os Algarismos de todas as linhas e colunas diferentes.
%-todas as somas de linhas e colunas iguais.
restricoes(e(LstNAfect,LstAfect)):-
    diferentes(LstAfect),
    somas_iguais(LstAfect).

diferentes([]).
diferentes([v( _,_,V )|LstAfect]):- member(v( _,_,V ),LstAfect),!,fail.
diferentes([_ |LstAfect]):- diferentes(LstAfect).

%Verifica a igualdade das somas
somas_iguais(L):- % linhas
    findall(V,member(v(n(1,_,_,V), L),L1), somatorio(L1),
    findall(V,member(v(n(2,_,_,V), L),L2), somatorio(L2),
    findall(V,member(v(n(3,_,_,V), L),L3), somatorio(L3),

    %colunas
    findall(V,member(v(n( _,1),_,_,V), L),B1), somatorio(B1),
    findall(V,member(v(n( _,2),_,_,V), L),B2), somatorio(B2),
    findall(V,member(v(n( _,3),_,_,V), L),B3), somatorio(B3),

    % diagonal Principal
    findall(V,member(v(n(1,1),_,_,V),L),D1),
    findall(V,member(v(n(2,2),_,_,V), L),D2),
    findall(V,member(v(n(3,3),_,_,V), L),D3),append(D1,D2,M),append(M,D3,X),somatorio(X),

    % diagonal Secundaria
    findall(V,member(v(n(3,1),_,_,V),L),D4),
    findall(V,member(v(n(2,2),_,_,V), L),D5),
    findall(V,member(v(n(1,3),_,_,V), L),D6),append(D4,D5,P),append(P,D6,J),somatorio(J).

somatorio( [Num1,Num2,Num3] ):-!,
    15 is Num1+Num2+Num3.
somatorio(_).

```

2.2 Alinea b) Algoritmo Backtracking e Sucessor

Para a resolução em backtracking, é necessária a afectação de valores do domínio às diferentes posições e a sua confirmação através das restrições definidas. Para esse fim serve o seguinte código:

```

b:- consult(qmagico),
    estado\_inicial(E0),
    back(E0,A),
    esc(A).

back(e([],A),A).
back(E,Sol):-
    sucessor(E,E1),
    restricoes(E1),
    back(E1,Sol).

%Predicado Sucessor
sucessor(e([v(N,D,_) |R],E),e(R,[v(N,D,V) |E])):- member(V,D).

```

2.3 Alinea c) Algoritmo Forward Checking

O forward checking, tem o objetivo de limitar o domínio das variáveis à medida que as variáveis vão sendo afetadas. Ou seja quando a variável é afetada, o domínio das outras variáveis tem os valores que se encontravam no domínio com exceção do valor colocado na variável afectada. O código seguinte diz respeito ao algoritmo forward checking.

```
f:-consult(qmagico),
    estado_inicial(E0),
    back1(E0,A),
    write(A),nl,nl,esc(A).

back1(e([],A),A).
back1(E,Sol):- sucessor(E,E1),
    restricoes(E1),
    forwardC(E1,E2),
    back(E2,Sol).

%ForwardChecking
forwardC(e(NAfect,[v(N,D,V)|Afect]),e(NAfectS,[v(N,D,V)|Afect])):-
    atualizaDom(V, NAfect, NAfectS).

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

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

2.4 Alinea d) e) Alguns Exemplos práticos

Quadrado Vazio: (Algoritmo Backtracking):

```
?- b.
Warning: /home/serra/Desktop/la
Singleton-marked variable
Warning: /home/serra/Desktop/la
Singleton variables: [L]
2 . 7 . 6
9 . 5 . 1
4 . 3 . 8
true
```

Posição (1,1) do Quadrado preenchida com o número 8:

```
?- b.
Warning: /home/serra/Desktop/ta2/
Singleton-marked variable
Warning: /home/serra/Desktop/ta2/
Singleton variables: [Lst]
8 . 1 . 6
3 . 5 . 7
4 . 9 . 2
true
```

Posição (3,1) do Quadrado preenchida com o número 2:

```
?- b.
Warning: /home/serra
Singleton-ma
Warning: /home/serra
Singleton va
4 . 3 . 8
9 . 5 . 1
2 . 7 . 6
true
```

Posição (1,1) do Quadrado preenchida com o número 1:

```
?- b.
Warning: /home/...
Single
Warning: /home/...
Single
false.
```

3 Sudoku

3.1 Alinea a): Estados, Variáveis, Restrições

O código seguinte representa os estados, as variáveis(nome,dominio e valor) e as respectivas restrições para o Sudoku representado no enunciado.

```
1 4 7 | 2 3 8 | 5 6 9
2 5 8 | 1 6 9 | 3 4 7
3 6 9 | 4 5 7 | 1 2 8
- - - - -
4 7 1 | 3 8 2 | 6 9 5
5 8 2 | 6 9 1 | 4 7 3
6 9 3 | 5 7 4 | 2 8 1
- - - - -
7 1 4 | 8 2 3 | 9 5 6
8 2 5 | 9 1 6 | 7 3 4
9 3 6 | 7 4 5 | 8 1 2
true
```

```
dominio([1,2,3,4,5,6,7,8,9]).
```

```
estado_inicial(E):-
    gerar_estado(E1),
    preencher_posicoes(E1, E).
```

```
gerar_estado(E):-
    functor(E, e, 2), arg(1, E, T), arg(2, E, []),
    tamanho_tabuleiro(S),
    gerar_tab(T2, S),
    flatten(T2, T).
```

```
% -----
% para preencher o tabuleiro basta adicionar uma variavel à lista L
% o exemplo seguinte preenche a posicao (1, 1) to tabuleiro com 2
% ex: L = [v(P(1,1), dominio, 2)]
preencher_posicoes(E, NE):-
    L = [v(p(1,2),dominio,1),v(p(1,6),dominio,8),v(p(1,8),dominio,7),v(p(1,9),dominio,3),
        v(p(2,4),dominio,5),v(p(2,6),dominio,9),
        v(p(3,1),dominio,7),v(p(3,7),dominio,9),v(p(3,9),dominio,4),
        v(p(4,6),dominio,4),
        v(p(5,5),dominio,3),v(p(5,6),dominio,5),v(p(5,8),dominio,1),v(p(5,9),dominio,8),
        v(p(6,1),dominio,8),v(p(6,4),dominio,9),
        v(p(7,4),dominio,7),
        v(p(8,1),dominio,2),v(p(8,2),dominio,6),v(p(8,4),dominio,4),v(p(8,8),dominio,3),
        v(p(9,3),dominio,5),v(p(9,6),dominio,3)],
    E = e(NAfect, []),
    NE = e(NAfect2, Afect),
    preencher(NAfect, L, NAfect2, Afect).
```

```

%Restrições
%-todos os Algarismos de todas as linhas, colunas e quadrantes diferentes.
%ve_restricoes(e(Nafec,Afect)):-
ve_restricoes(E):-
    ver_linhas(E),
    ver_colunas(E),
    ver_quadrantes(E).

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]).

ver_quadrantes(e(_, Afect)):-
    %ve o primeiro quadrante
    ver_quadrante(Afect, 1, 1, 3, Q1),
    todos_diff(Q1),
    %ve o segundo quadrante
    ver_quadrante(Afect, 1, 4, 6, Q2),
    todos_diff(Q2),
    %ve o terceiro quadrante
    ver_quadrante(Afect, 1, 7, 9, Q3),
    todos_diff(Q3),
    %ve o quarto quadrante
    ver_quadrante(Afect, 4, 1, 3, Q4),
    todos_diff(Q4),
    %ve o quinto quadrante
    ver_quadrante(Afect, 4, 4, 6, Q5),
    todos_diff(Q5),
    %ve o sexto quadrante
    ver_quadrante(Afect, 4, 7, 9, Q6),
    todos_diff(Q6),
    %ve o sétimo quadrante
    ver_quadrante(Afect, 7, 1, 3, Q7),
    todos_diff(Q7),
    %ve o oitavo quadrante
    ver_quadrante(Afect, 7, 4, 6, Q8),
    todos_diff(Q8),
    %ve o nono quadrante
    ver_quadrante(Afect, 7, 7, 9, Q9),
    todos_diff(Q9).

```

3.2 Alinea b) Algoritmo Backtracking e Sucessor

Para a resolução em backtracking, é necessária a afectação de valores do domínio às diferentes posições e a sua confirmação através das restrições definidas. Para esse fim serve o seguinte código:

```

p:- estado_inicial(E0), back(E0,A), esc(A).

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

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

```

3.3 Alinea c) Algoritmo Forward Checking

O forward checking, tem o objetivo de limitar o domínio das variáveis à medida que as variáveis vão sendo afetadas. Ou seja quando a variável é afetada, o domínio das outras variáveis tem os valores que se encontravam no domínio com exceção do valor colocado na variável afectada.

O código seguinte diz respeito ao algoritmo forward checking.

```
f:- estado_inicial(E0), back1(E0,A), esc(A).

back1(e([],A),A).
back1(E,Sol):- sucessor(E,E1),
               ve_restricoes(E1),
               forward_Checking(E1,E2),
               back(E2,Sol).

%percorre as nao afectadas e remove o valor do dominio
forward_Checking(E,NE):-
    cut_linha(E,NE1),
    cut_coluna(NE1,NE).

cut_linha(e(Nafect,[v(p(X,Y), D, V)|R]), NE):-
    member(v(p(X,_),_,V1),R),
    percorre_linha(V1,X,e(Nafect,[v(p(X,Y), D, V)|R]), NE).

cut_coluna(e(Nafect,[v(p(X,Y), D, V)|R]), NE):-
    member(v(p(_,Y),_,V1),R),
    percorre_coluna(V1,Y,e(Nafect,[v(p(X,Y), D, V)|R]), NE).
```