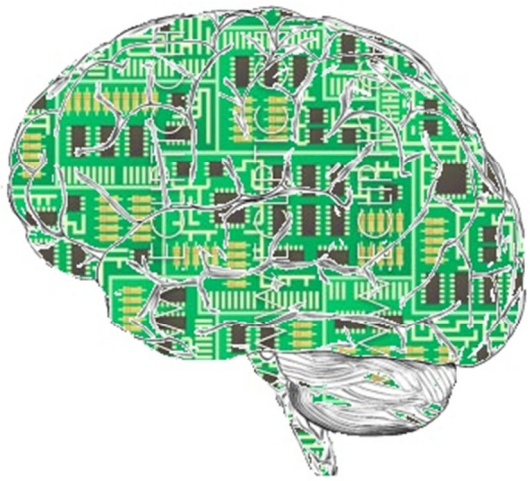
Curso: Engenharia Informática

Disciplina: Inteligência Artificial

Professora: Irene Rodrigues



2º Trabalho Prático

Trabalho Elaborado Por:

Luís Polha Nº 20464

Marlene Oliveira Nº 25999

Ano Lectivo 2011/2012

1.

a.

Cada estado do problema consistirá na representação do quadrado mágico naquele momento. O estado inicial irá conter a representação do quadrado mágico antes que sejam efectuadas quaisquer alterações. Cada estado irá conter as listas de variáveis afectadas e não afectadas. Na representação destas mesmas variáveis constará o seu nome, o seu domínio e valor no estado em questão.

O nome de cada variável será semelhante aos seguintes: n(X) e op(Y), em que X e Y correspondem, na nossa notação, ao número atribuído àquela casa do quadrado mágico, sendo que n e op indicam se a variável é um número ou um operador.

O domínio das variáveis que irão conter os números será o conjunto dos números inteiros de 1 a 9, ou seja, [1,2,3,4,5,6,7,8,9]. O valor atribuído a cada uma destas variáveis terá de pertencer ao domínio da mesma.

O domínio das variáveis que irão conter os operadores será o conjunto dos operadores aritméticos, neste caso: [+,-,*,/]. O valor atribuído a cada uma destas variáveis terá de pertencer ao domínio da mesma.

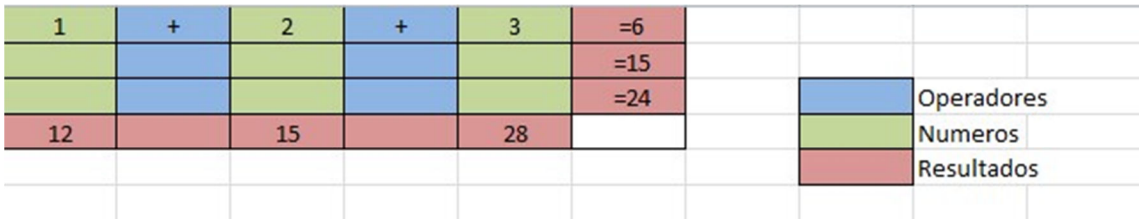


Figura 1 – Representação do Quadrado Mágico do enunciado.

Utilizando um esquema semelhante ao da Figura 1:

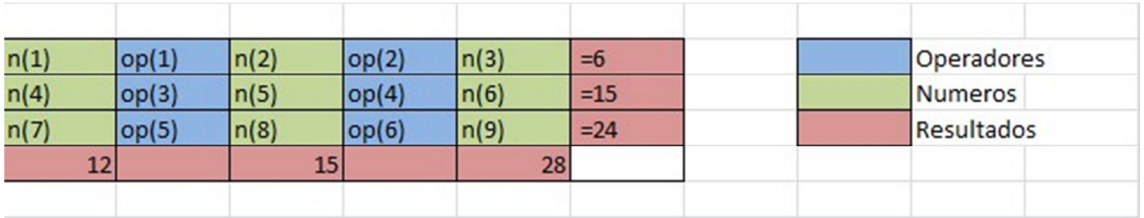


Figura 2 – Esquema do Quadrado Mágico do enunciado de acordo com a nossa notação.

Exemplo da Representação de uma variável, que corresponderá a um número, em Prolog (pseudo-código), utilizando a nossa notação:

```
v(n(1),[1,2,3,4,5,6,7,8,9],V1)
```

Exemplo da Representação de uma variável, que corresponderá a um operador aritmético, em Prolog (pseudo-código), utilizando a nossa notação:

```
v(op(1),[+,-,*,/],Vop1)
```

Representação do estado inicial em Prolog:

```
estado_inicial(e([
        v(n(1),[1,2,3,4,5,6,7,8,9],_),
        v(op(1),['+','-','/','*'],_),
        v(n(2),[1,2,3,4,5,6,7,8,9],_),
        v(op(2),['+','-','/','*'],_),
        v(n(3),[1,2,3,4,5,6,7,8,9],_),
        v(n(4),[1,2,3,4,5,6,7,8,9],_),
        v(op(3),['+','-','/','*'],_),
        v(n(5),[1,2,3,4,5,6,7,8,9],_),
        v(op(4),['+','-','/','*'],_),
        v(n(6),[1,2,3,4,5,6,7,8,9],_),
        v(n(7),[1,2,3,4,5,6,7,8,9],_),
        v(op(5),['+','-','/','*'],_),
        v(n(8),[1,2,3,4,5,6,7,8,9],_),
        v(op(6),['+','-','/','*'],_),
        v(n(9),[1,2,3,4,5,6,7,8,9],_)
      ],[])).
```

Restrições do problema:

- Os números têm de ser todos diferentes;
- A operação aritmética da primeira linha tem de ser igual a 6;

- A operação aritmética da segunda linha tem de ser igual a 15;
- A operação aritmética da terceira linha tem de ser igual a 24;
- A soma dos números da primeira coluna tem de ser igual a 12;
- A soma dos números da segunda coluna tem de ser igual a 15;
- A soma dos números da terceira coluna tem de ser igual a 18.

Nota: Dado que não havia informação em contrário no enunciado, assume-se que os números das colunas são somados.

Para que fosse possível verificar que todos os números no quadrado mágico são diferentes, foi criado um predicado auxiliar que procura o valor de todas as variáveis do estado afectado que correspondem a casas onde se devem encontrar valores numéricos e guarda esses mesmos valores numa lista, sendo utilizado o predicado *all_different* do Prolog, que verifica se os elementos que constam numa lista são todos diferentes.

Para verificar as restrições das linhas, o nosso algoritmo instancia todos os elementos de uma linha do quadrado (números e operadores) e efectua a operação aritmética indicada, verificando se o resultado da mesma corresponde ao valor a que a operação nessa linha tem de tomar (na primeira linha este valor seria 6, na segunda linha este valor seria 15 e na terceira linha este valor seria 24).

As restrições das colunas são verificadas quando todas as linhas já foram preenchidas e verificadas. Os valores da soma dos números de cada uma das colunas têm de ser iguais a um valor estipulado para cada coluna (na primeira coluna seria 12, na segunda coluna seria 15 e na terceira coluna seria 18).

Código Prolog das restrições:

```
restricoes(e(_,A)):-
    rest(e(_,A)),
    varToVal(A,R),
    all_diff(R),
    !.
```

O predicado anterior aplica as restrições apresentadas anteriormente, recorrendo a alguns predicados auxiliares.

```
rest(e(_,A)):-
        length(A, Res),
        Res\=5,
        Res\=10,
        Res\=15.
```

O predicado auxiliar anterior garante que o algoritmo não termina prematuramente, ou seja, evita que o algoritmo termine antes que o quadrado tenha sido preenchido e as restrições de linhas, colunas e valores tenham sido verificadas.

```
rest(e(_,A)):-  ((length(A,5),
                    member(v(n(1),_,X),A),
                    member(v(op(1),_,Op1),A),
                    member(v(n(2),_,Y),A),
                    member(v(op(2),_,Op2),A),
                    member(v(n(3),_,Z),A),
                    alldiff(X,Y,Z),
                   (linha(X,Y,Z,Op1,Op2,6)))).
```

O predicado auxiliar anterior verifica que quaisquer cálculos efectuados na primeira linha terão como resultado final o valor 6. Esta restrição só é verificada quando existirem 5 variáveis instanciadas (a primeira linha já se encontra totalmente preenchida).

```
rest(e(_,A)):-
        ((length(A,10),
            member(v(n(4),_,X),A),
            member(v(op(3),_,Op1),A),
            member(v(n(5),_,Y),A),
            member(v(op(4),_,Op2),A),
            member(v(n(6),_,Z),A),
            alldiff(X,Y,Z),
           (linha(X,Y,Z,Op1,Op2,15)))).
```

O predicado auxiliar anterior verifica que quaisquer cálculos efectuados na segunda linha terão como resultado final o valor 15. Esta restrição só é testada quando existem 10 variáveis instanciadas (a primeira e a segunda linhas já se encontram totalmente preenchidas).

```
rest(e(_,A)):-
        ( (length(A,15),
            member(v(n(1),_,X1),A),
            member(v(n(2),_,X2),A),
            member(v(n(3),_,X3),A),
            member(v(n(4),_,X4),A),
            member(v(n(5),_,X5),A),
            member(v(n(6),_,X6),A),
            member(v(n(7),_,X),A),
            member(v(op(5),_,Op1),A),
            member(v(n(8),_,Y),A),
            member(v(op(6),_,Op2),A),
            member(v(n(9),_,Z),A),
            !,
            alldiff(X,Y,Z),
            linha(X,Y,Z,Op1,Op2,24),
            coluna(X1, X4, X,12),
            coluna(X2, X5, Y,15),
            coluna(X3, X6, Z,18))).
```

O predicado auxiliar anterior verifica anterior verifica que quaisquer cálculos efectuados na segunda linha terão como resultado final o valor 24. Esta restrição só é testada quando existem 15 variáveis instanciadas (todas as linhas já se encontram totalmente preenchidas).

No predicado auxiliar são também verificadas as restrições das colunas. Tal verificação só é efectuada quando o quadrado mágico já se encontra totalmente preenchido.

Nota: Por uma questão de simplificação, os restantes predicados auxiliares não se encontram no relatório. Porém, tais predicados podem ser consultados no ficheiro que contém o código Prolog do trabalho.

Operador Sucessor utilizado:

```
sucessor(e([v(Nome,Dominio,Valor)]|Restantes],E), e(Restantes,[v(Nome,Dominio, Valor)|E]):-
        member(Valor,Dominio).
```

Este operador retorna o estado que sucede ao estado actual.

b.

A Pesquisa Backtracking é um algoritmo de pesquisa não informada para problemas de satisfação de restrições. Esta pesquisa comporta-se como uma pesquisa em profundidade com afectação de uma única variável.

O código Prolog da Pesquisa Backtracking utilizada foi o seguinte:

```
backtracking(e([],Solucao),Solucao).
backtracking(EstAct,Solucao):-
        sucessor(EstAct,EstSeg),
        restricoes(EstSeg),
        backtracking(EstSeg,Solucao).
```

c.

Forward Checking é uma extensão da Pesquisa Backtracking que permite fazer a verificação para a frente (look ahead).

Código Prolog das alterações efectuadas à Pesquisa Backtracking de modo a que esta possa fazer Forward Checking:

```
fc_aux2([],_,[]).
fc_aux2([X|Xs], V, [Y|Ys]):-
        ((member(v(op(_),_,_),[X]),
          Y = X);
        member(v(n(I),D,_),[X]),
        remove(V,D,NewD),
    Y=v(n(I), NewD,_)),
        fc_aux2(Xs, V, Ys).
```

Os predicados anteriores verificam se uma variável de uma lista corresponde a um valor ou a um operador e, caso este seja um valo, removem o mesmo do domínio da variável. Caso a variável corresponda a um operador, não são efectuadas alterações à lista de variáveis fornecida como argumento. Caso contrário, o valor dado como argumento é removido do domínio da variável correspondente e a lista retornada corresponderá à lista das variáveis, que já possuirá todas as alterações efectuadas aos domínios dessas mesmas variáveis.

```
fc_aux1(e(NAf,[Af|Res]),e(NAf,[Af|Res])):-
        member(v(op(_),_,_),[Af]).


fc_aux1(e(NAf,[Af|Res]),e(NAfFC,[Af|Res])):-
        member(v(n(_),_,V),[Af]),
        fc_aux2(NAf, V, NAfF).
```

O predicado anterior recebe como argumento um estado e efectua as alterações necessárias aos domínios das variáveis da lista de variáveis não afectadas. Estes predicados retornam o estado que já irá conter as alterações efectuadas à lista de variáveis não afectadas.

d.

Para melhorar a capacidade espacial, poderíamos alterar o algoritmo de modo a que este escolhesse variáveis com menos valores no domínio. Porém, como neste problema o Forward Checking já tem como resultado a solução mais eficiente, tal não foi efectuado.

Para melhorar a capacidade temporal, poderíamos alterar o algoritmo de modo a que este escolhesse a variável com mais restrições ou a variável com menos restrições. Porém, como neste problema o Forward Checking já tem como resultado a solução mais eficiente, tal não foi efectuado.

e. Resultados para quatro exemplos diferentes:

Resultado 1:

1+2+3

4+5+6

7+8+9

Solução retornada pelo algoritmo:

[v(n(1),[1,2,3,4,5,6,7,8,9],1),

v(op(1),[+,-,/,*],+),

v(n(2),[1,2,3,4,5,6,7,8,9],2),

v(op(2),[+,-,/,*],+),

v(n(3),[1,2,3,4,5,6,7,8,9],3),

v(n(4),[1,2,3,4,5,6,7,8,9],4),

v(op(3),[+,-,/,*],+),

v(n(5),[1,2,3,4,5,6,7,8,9],5),

v(op(4),[+,-,/,*],+),

v(n(6),[1,2,3,4,5,6,7,8,9],6),

v(n(7),[1,2,3,4,5,6,7,8,9],7),

v(op(5),[+,-,/,*],+),

v(n(8),[1,2,3,4,5,6,7,8,9],8),

v(op(6),[+,-,/,*],+),

v(n(9),[1,2,3,4,5,6,7,8,9],9)]

Resultado 2:

1-4+9

3+5+7

8*6/2

Solução retornada pelo algoritmo:

[v(n(1),[1,2,3,4,5,6,7,8,9],1),

v(op(1),[+,-,/,*],-),

v(n(2),[1,2,3,4,5,6,7,8,9],4),

v(op(2),[+,-,/,*],+),

v(n(3),[1,2,3,4,5,6,7,8,9],9),

v(n(4),[1,2,3,4,5,6,7,8,9],3),

v(op(3),[+,-,/,*],+),

v(n(5),[1,2,3,4,5,6,7,8,9],5),

v(op(4),[+,-,/,*],+),

v(n(6),[1,2,3,4,5,6,7,8,9],7),

v(n(7),[1,2,3,4,5,6,7,8,9],8),

v(op(5),[+,-,/,*],*),

v(n(8),[1,2,3,4,5,6,7,8,9],6),

v(op(6),[+,-,/,*],/),

v(n(9),[1,2,3,4,5,6,7,8,9],2)]

Resultado 3:

1*2*3

4+5+6

7+8+9

Resultado retornado pelo algoritmo:

[v(n(1),[1,2,3,4,5,6,7,8,9],1),

v(op(1),[+,-,/,*],*),

v(n(2),[1,2,3,4,5,6,7,8,9],2),

v(op(2),[+,-,/,*],*),

v(n(3),[1,2,3,4,5,6,7,8,9],3),

v(n(4),[1,2,3,4,5,6,7,8,9],4),

v(op(3),[+,-,/,*],+),

v(n(5),[1,2,3,4,5,6,7,8,9],5),

v(op(4),[+,-,/,*],+),

v(n(6),[1,2,3,4,5,6,7,8,9],6),

v(n(7),[1,2,3,4,5,6,7,8,9],7),

v(op(5),[+,-,/,*],+),

v(n(8),[1,2,3,4,5,6,7,8,9],8),

v(op(6),[+,-,/,*],+),

v(n(9),[1,2,3,4,5,6,7,8,9],9)]

Resultado 4:

3+9-6

2+5+8

7-1*4

Resultado retornado pelo algoritmo:

[v(n(1),[1,2,3,4,5,6,7,8,9],3),

v(op(1),[+,-,/,*],+),

v(n(2),[1,2,3,4,5,6,7,8,9],9),

v(op(2),[+,-,/,*],-),

v(n(3),[1,2,3,4,5,6,7,8,9],6),

v(n(4),[1,2,3,4,5,6,7,8,9],2),

v(op(3),[+,-,/,*],+),

v(n(5),[1,2,3,4,5,6,7,8,9],5),

v(op(4),[+,-,/,*],+),

v(n(6),[1,2,3,4,5,6,7,8,9],8),

v(n(7),[1,2,3,4,5,6,7,8,9],7),

v(op(5),[+,-,/,*],-),

v(n(8),[1,2,3,4,5,6,7,8,9],1),

v(op(6),[+,-,/,*],*),

v(n(9),[1,2,3,4,5,6,7,8,9],4)]