Solve MathemaGrids
Z3

José Pereira pg27748
Marta Azevedo pg27763
Tiago Brito pg

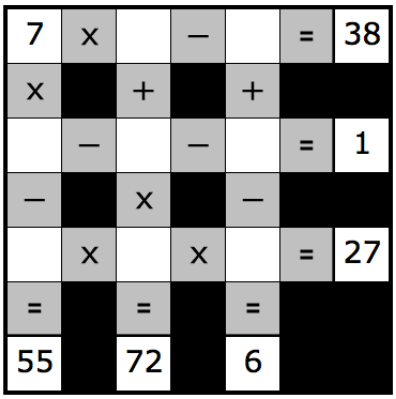
8 de Abril de 2015

1 MathemaGrids

O MathemaGrids é um puzzle onde o objectivo é preencher uma tabela *mxm* com inteiros entre 1 e m*m, de tal maneira que cada um desses números aparece apenas uma vez.

Para além disso, a posição dos números deve respeitar as operações que já estão presentes no tabuleiro.
Neste relatório, pretendemos descrever de forma clara a nossa implementação.

Para isso vamos usar como exemplo o seguinte tabuleiro:



Neste exemplo, já é dada uma *hint*, o 7 que aparece na primeira linha, primeira coluna.

2 Z3

Como SMT-SOLVER estamos a usar o Z3 Theorem Prover da Microsoft Research. Este, é usado em vários softwares de verificação formal.

Como linguagem de interface com o Z3 preferimos usar o PYTHON Z3PY devido à sua API fácil de usar e devido a ser uma linguagem simples e eficaz para o que queriamos fazer.

3 *solveMathemaGrids.py*

O *solveMathemaGrids.py* é invocado como um ficheiro python qualquer:

```
$ solveSurvo.py <ficheiro\_input>
```

por exemplo,

```
$ solveSurvo.py exemplo_1.txt
```

Caso consiga resolver o puzzle, é imprimido no ecra o output, ou seja, o puzzle completo com os as soluções. É necessário que o Z3 esteja instalado e conectado com o PYTHON.

3.1 Ficheiros input

Para representar um tabuleiro de MATHEMAGRIDS (o do exemplo) usamos o seguinte ficheiro de texto:

```
7*.-.=38
*,+,+
.-.-.=1
-,*,-
.*.*.=27
=,=,=,=
55,72,6
```

Os "." representam os espaços que têm que ser preenchidos com números e as "," representam os espaços que, apesar de existirem no tabuleiro, não podem ser inseridos com números.

Também estão representadas as operações sendo o ***** a multiplicação, o + a soma, **-** a subtração e / a divisão.

3.2 Ficheiros output

O MATHEMAGRIDS dá como output um ficheiro com a representação da matriz obtida e os valores que as linhas e as colunas que respeitam as operações dadas como imput.

O do exemplo :

——> **falta o output**

3.3 Codificação do MathemaGrids

Para codificar o puzzle,usamos as seguintes propriedades (descritas em `https://www.brainbashers.com/mathemagrids.asp`):

1. Usar todos os digitos de 1 a $m * m$ (no nosso exemplo, até 9);
2. Nenhum número pode ser repetido;
3. As operações são feitas da esquerda para a direita e de cima para baixo, sendo a ordem de prioridade normal da matemática ignorada;
4. Não podem existir divisões por 1 nem multiplicações por 1;
5. Em nenhum ponto, os resultados intermédios do cálculo são valores infeiores a zero.

As variaveis *x_membros* e *y_membros* representam a quantidade de espaços em que o jogador pode inserir números nas linhas e nas colunas respetivamente. As regras usadas são:

1. ```
   [ And(1 <= X[j][i], X[j][i] <= x_membros*y_membros)
    for i in range(x_membros) for j in range(y_membros) ]
   ```
2. ```
   [ Distinct([X[j][i] for i in range(x_membros)
   for j in range(y_membros)]) ]
   ```
3. Para garantir que as operações são feitas na ordem correta, escrevemos as equações horizontais e verticais que, neste exemplo são :
   ——> **faltam aqui as equações**
4. ```
   div_mult_por_1 = []
   for i in range(len(table)):
       for j in range(len(table[i])):
           if (table[i][j]=='/' or table[i][j]=='*' ):
               if (i%2==0):
                   div_mult_por_1.append(Not(X[i/2][(j+1)/2] == 1))
               else:
                   div_mult_por_1.append(Not(X[(i+1)/2][j/2] ==1))
   ```
5. Como a operação critica para isto acontecer é apenas a subtração:

   ```
   bigger_than_zero = []
   for i in range(len(table)):
       for j in range(len(table[i])):
           if (i%2==0):
               if(table[i][j]=='-'):
   ```
   ```

```
bigger_than_zero.append(Not(X[i/2][((j+1)/2)-1]
- X[i/2][(j+1)/2] < 0))
bigger_than_zero.append(Not(X[((i+1)/2-1)][j/2]
- X[((i+1)/2)][(j+1)/2] < 0))
```