Universidade do Minho

SMT

Mestrado Integrado em Engenharia Informática

Verificação Formal

2ºSemestre 2020/21

Aluno:
Etienne Costa

Docente:
Maria João Frade

15 de março de 2021

Conteúdo

1 Resumo

O relatório apresentado diz respeito ao segundo trabalho proposto no âmbito da unidade curricular de verificação formal. O universo de discurso recaí sobre problemas de decisão, cuja instância são fórmulas lógicas com relação a combinações de teorias de fundo expressas na lógica clássica de primeira ordem com igualdade.

2 Matriz

Partindo do seguinte programa em C sobre inteiros :

```
1
2 for (i=1; i<=3; i++)
3     for(j=1; j<=3; j++)
4         M[i][j] = i+j
```

é apresentado um **programa em C**, constituido por uma sequência de atribuições, equivalemente ao programa anterior:

```
1  #include <stdio.h>
2  #define  N 4
3
4  int main(){
5
6  int M [N][N];
7  int i,j;
8  i=j=1;
9  // M[1][1]=2
10 M[i][j]=i+j;
11 //j=2;
12 j++;
13 // M[1][2]=3
14 M[i][j]=i+j;
15 //j=3
16 j++;
17 // M[1][3]=4
18 M[i][j]=i+j;
19 //i=2
20 i++;
21 //j=1
22 j=1;
23 // M[2][1]=3
24 M[i][j]=i+j;
25 //j=2
26 j++;
27 // M[2][2]=4
28 M[i][j]=i+j;
29 //j=3
30 j++;
31 // M[2][3]=5
32 M[i][j]=i+j;
33 //i=3
34 i++;
35 //j=1
36 j=1;
37 // M[3][1]=4
38 M[i][j]=i+j;
39 //j=2
40 j++;
41 // M[3][2]=5
42 M[i][j]=i+j;
43 //j=3
44 j++;
45 // M[3][3]=6
46 M[i][j]=i+j;
47
48 return 0;
49
50 }
```

Com base no programa anterior é feita uma codificação lógica num ficheiro em formato **SMT-LIBv2** :

No início do ficheiro encontra-se declarada a lógica utilizada bem como o conjunto de variáveis lógicas , pois o valor de uma variável lógica é imutável, existindo a necessidade de criar uma nova sempre que é feita uma atribuição.

```
(set-logic QF_AUFLIA)

; Declaração das funções constantes : i
(declare-const i0 Int)
(declare-const i1 Int)
(declare-const i2 Int)

; Declaração das funções constantes : j
(declare-const j0 Int)
(declare-const j1 Int)
(declare-const j2 Int)
(declare-const j3 Int)
(declare-const j4 Int)
(declare-const j5 Int)
(declare-const j6 Int)
(declare-const j7 Int)
(declare-const j8 Int)

; Declaração das funções constantes : M[][]
(declare-const M0 (Array Int (Array Int Int)))
(declare-const M1 (Array Int (Array Int Int)))
(declare-const M2 (Array Int (Array Int Int)))
(declare-const M3 (Array Int (Array Int Int)))
(declare-const M4 (Array Int (Array Int Int)))
(declare-const M5 (Array Int (Array Int Int)))
(declare-const M6 (Array Int (Array Int Int)))
(declare-const M7 (Array Int (Array Int Int)))
(declare-const M8 (Array Int (Array Int Int)))
(declare-const M9 (Array Int (Array Int Int)))

; i0=1
(assert (= i0 1))
; j0=1
(assert (= j0 1))
; M0[i0][j0] = i0+j0
(assert (= M1 (store M0 i0 (store (select M0 i0) j0 (+ i0 j0)) ) ))
; j1=j0+1
(assert (= j1 (+ j0 1)))
; M0[i0][j1] = i0+j1
(assert (= M2 (store M1 i0 (store (select M1 i0) j1 (+ i0 j1)) ) ))
; j2=j1+1
(assert (= j2 (+ j1 1)))
; M0[i0][j2] = i0+j2
(assert (= M3 (store M2 i0 (store (select M2 i0) j2 (+ i0 j2)) ) ))
; i1=i0+1
(assert (= i1 (+ i0 1)))
; j3=1
(assert (= j3 1))
; M0[i1][j3] = i1+j3
(assert (= M4 (store M3 i1 (store (select M3 i1) j3 (+ i1 j3)) ) ))
; j4=j3+1
(assert (= j4 (+ j3 1)))
; M0[i1][j4] = i1+j4
(assert (= M5 (store M4 i1 (store (select M4 i1) j4 (+ i1 j4)) ) ))
; j5=j4+1
(assert (= j5 (+ j4 1)))
; M0[i1][j5] = i1+j5
(assert (= M6 (store M5 i1 (store (select M5 i1) j5 (+ i1 j5)) ) ))
```

```
; i2=i1+1
(assert (= i2 (+ i1 1)))
; j6=1
(assert (= j6 1 ) )
; M0[i2][j6] = i2+j6
(assert (= M7 (store M6 i2 (store (select M6 i2) j6 (+ i2 j6 ) ) ) ))
; j7=j6+1
(assert (= j7 (+ j6 1)))
; M0[i2][j7] = i2+j7
(assert (= M8 (store M7 i2 (store (select M7 i2) j7 (+ i2 j7 ) ) ) ))
; j8=j7+1
(assert (= j8 (+ j7 1)))
; M0[i2][j8] = i2+j7
(assert (= M9 (store M8 i2 (store (select M8 i2) j8 (+ i2 j8 ) ) ) ))
```

Tendo por base a codificação lógica feita , usou-se um SMT Solver (*z3*) para estabelecer no final da execução do programa, a validade das propriedades que se seguem :

Propriedade 1 : *Se* $i = j$ *então* $M[i][j] \neq 3$

```
(push)
(echo "----------------------------------------------------------------")
(declare-const i Int)
(declare-const j Int)
(assert (not( => (= i j)  (not (= (select (select M9 i) j) 3) ) ) ) )
(check-sat)
(echo "----------------------------------------------------------------")
(pop)
```

Returns SAT : *Pois pode estar a tentar aceder um endereço de memória inválido, i.e, posições que não estão definidas no Array, visto que o seu tamanho inicial não está declarado .*
Ex: $i = 4$ e $j = 4$ ... $M[4][4] \neq 3$

Propriedade 2 : *Para quaisquer* i *e* j *entre* 1 *e* 3, $M[i][j] = M[j][i]$

```
(push)
(echo "----------------------------------------------------------------")
(declare-const i Int)
(declare-const j Int)
(assert  (and (<= 1 i ) (<= i 3 ) ) )
(assert  (and (<= 1 j ) (<= j 3 ) ) )
(assert (not (= (select (select M9 i) j)  (select (select M9 j) i) )))
(check-sat)
(echo "----------------------------------------------------------------")
(pop)
```

Returns UNSAT : *Podemos afirmar que a afirmação é verdadeira .*

Propriedade 3 : *Para quaisquer* i *e* j *entre* 1 *e* 3, *se* $i < j$ *então* $M[i][j] < 6$

```
(push)
(echo "----------------------------------------------------------------")
(declare-const i Int)
(declare-const j Int)
(assert  (and (<= 1 i ) (<= i 3 ) ) )
(assert  (and (<= 1 j ) (<= j 3 ) ) )
(assert (not (=> (< i j) (< (select (select M9 i) j) 6))))
(check-sat)
(echo "----------------------------------------------------------------")
(pop)
```

Returns UNSAT : *Podemos afirmar que a afirmação é verdadeira .*

Propriedade 4 : *Para quaisquer* i, a *e* b *entre* 1 *e* 3 *se* $a > b$ *então* $M[i][a] > M[i][b]$

```
(push)
(echo "----------------------------------------------------------------")
(declare-const i Int)
(declare-const a Int)
(declare-const b Int)
(assert  (and (<= 1 i ) (<= i 3 ) ) )
(assert  (and (<= 1 a ) (<= a 3 ) ) )
(assert  (and (<= 1 b ) (<= b 3 ) ) )
(assert (not (=> (> a b) (> (select (select M9 i) a)  (select (select
    M9 i) b) ))))
(check-sat)
(echo "----------------------------------------------------------------")
(pop)
```

Returns UNSAT : *Podemos afirmar que a afirmação é verdadeira .*

Propriedade 5 : *Para quaisquer* i *e* j *entre* 1 *e* 3, $M[i][j] + M[i+1][j+1] = M[i+1][j] + M[i][j+1]$

```
(push)
(echo "----------------------------------------------------------------")
(declare-const i Int)
(declare-const j Int)
(assert  (and (<= 1 i ) (<= i 3 ) ) )
(assert  (and (<= 1 j ) (<= j 3 ) ) )
(assert (not (= (+ (select (select M9 i) j)  (select(select M9 (+ i 1
    )) (+ j 1)))  (+ (select (select M9 (+ i 1) ) j) (select (select
    M9 i) (+ j 1 ))))))
(check-sat)
(echo "----------------------------------------------------------------")
(pop)
```

Returns SAT :
Pois pode estar a tentar aceder um endereço de memória inválido,i.e posições que não estão definidas no Array, visto que o seu tamanho inicial não está declarado .
Um exemplo simples seria para i *ou* j *igual a* 3.
Ex: $i = 3$ e $j = 1$ ... $M[3][1] + M[4][2] = M[4][1] + M[3][2]$

3 Survo

In a Survo puzzle, the task is to **fill a table with integers** so that each of **the integers appears only once and their row and column sums are equal to integers given on the bottom and the right side of the table. The smallest integer that can be used is 1 and the largest integer is equal to the number of cells in the table (which can vary)**.

3.1 Restrições e Modelação

Com base nesta breve introdução é possível especificar algumas restrições correspondentes às regras do puzzle bem com uma possível representação do modelo:

Modelação 1: Declaração do conjunto de variáveis.
Restrição 1: Restrições de valores .

```
x = {}
  for l in range(1,nrLinhas+1):
    x[l]={}
      for c in range(1,nrColunas+1):
        # Declaracao de variaveis.
        x[l][c]=Int('x'+str(l)+str(c))
        #Restricoes de valor.
        s.add(And (1 <= x[l][c], x[l][c] <=nrLinhas*
           nrColunas ))
```

Restrição 2: Todos os valores nas linhas são distintos.

```
for l in range(1,nrLinhas+1):
    # Restricoes da linha.
    s.add(Distinct ([x[l][c] for c in range(1,nrColunas
       +1)]))
```

Restrição 3: Todos os valores nas colunas são distintos.

```
for c in range(1,nrColunas+1):
        # Restricoes da coluna.
        s.add(Distinct ([x[l][c] for l in range(1,
           nrLinhas+1)]))
```

Restrição 4 :Soma das linhas igual a um valor X.

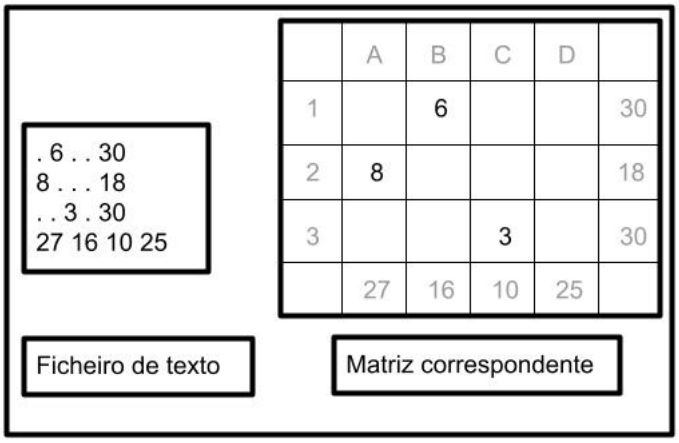
```
for l in range(1,nrLinhas+1):
    # Soma da linha igual a X
    s.add( Sum(([x[l][c] for c in range(1,nrColunas+1)]))
        == mapa[l-1][-1])
```

```
Restrição 5 :Soma das colunas igual a um valor Y.

for c in range(1,nrColunas+1):
    # Soma da coluna igual a Y
    s.add( Sum(([x[l][c] for l in range(1,nrLinhas+1)]))
       ==columnValue[c-1])
```

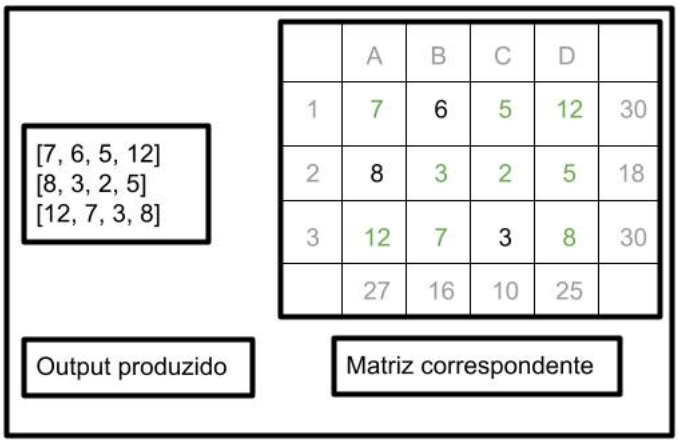
```
Modelação 2: Definição do estado inicial do puzzle.

for l in range(0,nrLinhas):
    for c in range(0,nrColunas):
        if(mapa[l][c]!='.'):
            s.add(x[l+1][c+1]==mapa[l][c])
```

3.2 Input - Output

De seguida é apresentada uma representação do ficheiro de texto (input) e a sua versão correspondente:



Tirando partido da API do **z3** , foi implementado um script em python capaz de devolver uma solução caso o modelo seja satisfazível. O output produzido corresponde na produção de um ficheiro de texto, contento uma representação da matriz .



	A	B	C	D	
1	7	6	5	12	30
2	8	3	2	5	18
3	12	7	3	8	30
	27	16	10	25	

Sendo que podemos constatar na solução obtida que todas as restrições são obedecidas.

- Todas as linhas e colunas possuem valores distintos.
- Todos os valores da matriz pertencem ao intervalo [1,12].
- Linha 1 : $7 + 6 + 5 + 12 = 30$
- Linha 2 : $8 + 3 + 2 + 5 = 18$
- Linha 3 : $12 + 7 + 3 + 8 = 30$
- Coluna A : $7 + 8 + 12 = 27$
- Coluna B : $6 + 3 + 7 = 16$
- Coluna C : $5 + 2 + 3 = 10$
- Coluna D : $12 + 5 + 8 = 25$

3.3 Resultados obtidos

3.3.1 Survo 2x7 - Puzzle 1

```
1
2 ====== INPUT ======
3 . . . . . . . 62
4 . . . . . . . 43
5 21 9 5 27 19 17 7
6 ==================
7
8 ====== OUTPUT ======
9 [12, 8, 3, 13, 11, 14, 1]
10 [9, 1, 2, 14, 8, 3, 6]
11 ====================
```

3.3.2 Survo 3x4 - Puzzle 2

```
1
2 ====== INPUT ======
3 . 6 . . 30
4 8 . . . 18
5 . . 3 . 30
6 27 16 10 25
7 ==================
8
9 ====== OUTPUT ======
10 [7, 6, 5, 12]
11 [8, 3, 2, 5]
12 [12, 7, 3, 8]
13 ====================
```

3.3.3 Survo 4x4 - Puzzle 3

```
1 ====== INPUT ======
2 . . 11 . 33
3 9 . . . 44
4 . . . 10 20
5 . 4 . . 39
6 24 14 43 55
7 ==================
8
9 ====== OUTPUT ======
10 [5, 1, 11, 16]
11 [9, 6, 15, 14]
12 [2, 3, 5, 10]
13 [8, 4, 12, 15]
14 ====================
```