Verificação Formal: Questões SMT

Rodolfo Silva (A81716)

17 Março 2020

1 Exercício 1 - Matriz

1.1 Alínea 1:

Listing 1: Codigo C desenvolvido que permite emular o comportamento dos ciclos for.

```
int matrix[4][4];

matrix[1][1] = 2; matrix[1][2] = 3; matrix[1][3] = 4;
matrix[2][1] = 3; matrix[2][2] = 4; matrix[2][3] = 5;
matrix[3][1] = 4; matrix[3][2] = 5; matrix[3][3] = 6;
```

1.2 Alínea 2:

Listing 2: Regras em SMT2 que permitem simular o comportamento do código feito na alínea 1.

```
(declare-const m_0 (Array Int (Array Int Int)))
(declare-const m_1 (Array Int (Array Int Int)))

;matrix[1][1] = 2
(assert (= m_1 (store m_0 1 (store (select m_0 1) 1 2))))
;matrix[1][2] = 3
(assert (= m_1 (store m_0 1 (store (select m_0 1) 2 3))))
;matrix[1][3] = 4
(assert (= m_1 (store m_0 1 (store (select m_0 1) 3 4))))
;matrix[2][1] = 3
(assert (= m_1 (store m_0 2 (store (select m_0 2) 1 3))))
;matrix[2][2] = 4
(assert (= m_1 (store m_0 2 (store (select m_0 2) 2 4))))
;matrix[2][3] = 5
(assert (= m_1 (store m_0 2 (store (select m_0 2) 3 5))))
;matrix[3][1] = 4
(assert (= m_1 (store m_0 3 (store (select m_0 3) 1 4))))
;matrix[3][2] = 5
(assert (= m_1 (store m_0 3 (store (select m_0 3) 2 5))))
;matrix[3][3] = 6
(assert (= m_1 (store m_0 3 (store (select m_0 3) 3 6 ))))
```

1.3 Alínea 3:

Listing 3: Declarações iniciais.

```
(declare-const i Int)
(declare-const j Int)
(declare-const a Int)
(declare-const b Int)
```

1.3.1 Subalínea (a)

Listing 4: Se i $\doteq jent\tilde{a}oM\lceil i\rfloor\lceil j\rfloor \neq 3$.

```
(push)
(assert (= (select (select m_1 i) i) 3))
(check-sat)
(pop)
--------------------------------------------------
Resultado : SAT
```

Um possível contra exemplo será com a atríbuição de i = 4, com M$\lceil 4\rfloor\lceil 4\rfloor = 3$.

Isto é possível devido a duas razões, a primeira é que, a definição da matriz foi feita manualmente e não se encontra definido que M$\lceil i\rfloor\lceil j\rfloor = i + j$.

A segunda será que os valores de i e j não foram limitados para estarem entre 0 e 3. Estas duas razões juntas permitem que seja possível que o contra-exemplo fornecido seja verdade.

1.3.2 Subalínea (b)

Listing 5: Para quaisquer i e j entre 1 e 3 têm-se que M$\lceil i\rfloor\lceil j\rfloor \doteq M\lceil j\rfloor\lceil i\rfloor$.

```
(push)
(assert (and (> i 0) (< i 4)))
(assert (and (> j 0) (< j 4)))
(assert (not (= (select (select m_1 i) j) (select (select m_1 j) i) )))
(check-sat)
(pop)
--------------------------------------------------
Resultado : UNSAT
```

Logo a afirmação é válida.

1.3.3 Subalínea (c)

Listing 6: Para quaisquer i e j entre 1 e 3 se i menor que j então M$\lceil i\rfloor\lceil j\rfloor < 6$.

```
(push)
(assert (and (> i 0) (< i 4)))
(assert (and (> j 0) (< j 4)))
(assert (not (=> (< i j) (< (select (select m_1 i) j) 6))))
(check-sat)
(pop)
--------------------------------------------------
Resultado : UNSAT
```

Logo a afirmação é válida.

1.3.4 Subalínea (d)

Listing 7: Para quaisquer i/a/b entre 1 e 3 se a maior que b então M$\lceil i \rfloor \lceil a \rfloor > M \lceil i \rfloor \lceil b \rfloor$.

```
(push)
(assert (and (> i 0) (< i 4)))
(assert (and (> a 0) (< a 4)))
(assert (and (> b 0) (< b 4)))
(assert (not (=> (> a b) (> (select (select m_1 i) a) (select (select m_1 i) b)
                                                                                ))))
(check-sat)
(pop)
-----------------------------------------------
Resultado : UNSAT
```

Logo a afirmação é válida.

1.3.5 Subalínea (e)

Listing 8: Para quaisquer i e j entre 1 e 3 / M$\lceil i \rfloor \lceil j \rfloor + M \lceil i+1 \rfloor \lceil j+1 \rfloor \doteq M \lceil i+1 \rfloor \lceil j \rfloor + M \lceil i \rfloor \lceil j+1 \rfloor$.

```
(push)
(assert (and (> i 0) (< i 4)))
(assert (and (> j 0) (< j 4)))
(assert (= a (+ i 1)))
(assert (= b (+ i 1)))
(assert (not (= (+ (select (select m_1 i) j) (select (select m_1 a) b))
             (+ (select (select m_1 a) j) (select (select m_1 i) b)))))
(check-sat)
(pop)
-----------------------------------------------
Resultado : SAT
```

Nesta situação, apesar de se estar a limitar o valor de i e j, não se está a limitar o valor de i+1 e j+1. Isto pode dar origens a situações do gênero, i=3 j=3, M33 = 6, M44 = 0, M43 = 7, M34 = 7, onde o resultado final vai ser 6+0 = 7+7, o que não é verdade, logo pode servir como contra exemplo. Isto deve-se tambem ao facto de não estar presente no sistema a noção de que M[i][j] = i+j, como mencionado na subalínea (a), o que torna isto possível.

2 Exercício 2 - Puzzle Solver

O puzzle que se decidiu resolver foi o puzzle do Survo.

Para a resolução do puzzle foi necessário a criação de um progama *Python* que leia o ficheiro dado como *input* nos argumentos. Sendo que de seguida vai gerar o modelo a passar a um *SMTSolver*, o *solver* em questão usado foi o Z3 através da api Z3PY que existe para *Python*. Por fim, foi escrito num ficheiro, solucao.txt, a solução dada ao problema pelo *solver*.

Para ser possível a correta resolução do puzzle foi preciso definir uma lista de regras que vão ser descritas de seguida. Antes de se passar à explicação dessas regras, para a resolução das mesmas foram usadas as seguintes variavéis para ajuda:

- **Altura:** Corresponde à altura do tabuleiro que foi lido do ficheiro *input*.
- **Largura:** Corresponde à largura do tabuleiro que foi lido do ficheiro *input*.
- **X[altura][largura]:** Corresponde a uma matriz que vai conter todos os nomes das váriaveis a serem usadas, por exemplo, para a posição (1,1) do tabuleiro o seu valor na matriz vai ser X[1][1] = X_1_1.
- **tup[altura][largura]:** Corresponde a um *tupple* que contêm o estado inícial do tabuleiro de jogo fornecido no ficheiro de *input*. O valor 0(zero) foi usado para ilustrar a ausência de um valor para aquela célula.
- **somaLinhas[altura]:** Array com a soma de cada linha do puzzle fornecido.
- **somaColunas[largura]:** Array com a soma de cada coluna do puzzle fornecido.

Ainda mais, os exemplos de regras demonstradas em baixo vêm todas de um puzzle de teste usado com altura = 3 e largura = 4.

2.1 Cada célula pode conter valores entre {1...Altura*Largura}.

Listing 9: Código correspondente.

```
MxN = largura * altura

cells_c  = [ And(1 <= X[i][j], X[i][j] <= MxN)
             for i in range(altura) for j in range(largura) ]
```

O pedaço de código demonstrado em cima vai criar um *Array* com uma regra para cada variavél existente para o tabuleiro fornecido.

Regra exemplo:

```
And(x_1_1 >= 1, x_1_1 <= 12)
```

2.2 Cada numero só pode aparecer uma vez.

Listing 10: Código correspondente.

```
distinct_c  = [ Distinct([ X[i][j] for i in range(altura)
                                   for j in range(largura) ]) ]
```

Apenas uma regra vai ser gerada neste caso, que pode ser visualizada em baixo, esta permite garantir que não vão existir duas células com numeros repetidos.

```
Distinct(x_1_1,x_1_2,x_1_3,x_1_4,x_2_1,x_2_2,x_2_3,x_2_4,x_3_1,x_3_2,x_3_3,x_3_4)
```

2.3 Soma de todas as linhas.

Listing 11: Código correspondente.

```
rows_c    = [ And(sum(X[i])==somaLinhas[i]) for i in range(altura) ]
```

Regra exemplo:

```
And(0 + x_1_1 + x_1_2 + x_1_3 + x_1_4 == 30)
```

2.4 Soma de todas as colunas.

Listing 12: Código correspondente.

```
cols_c    = [ And(sum([ X[i][j] for i in range(altura) ])==somaColunas[j])
              for j in range(largura) ]
```

Regra exemplo:

```
And(0 + x_1_1 + x_2_1 + x_3_1 == 27)
```

2.5 Construção do tabuleiro.

Listing 13: Código correspondente.

```
instance = [ If(tup[i][j] == 0,True,X[i][j] == tup[i][j])
                  for i in range(altura) for j in range(largura) ]
```

Este pedaço de código final serve para traduzir o tabuleiro que veio no ficheiro para um tabuleiro que possa ser passado ao *solver*, usado para indicar quais as variáveis que já possuem um número atribuído e quais não possuem um número.

Regra exemplo:

```
If(True, True, x_1_1 == 0) -> Célula não possui número atribuído.
If(False, True, x_1_2 == 6) -> Célula possui número atribuído.
```

2.6 Resolução de um tabuleiro de teste por parte do programa.

	A	B	C	D	SOMA
1		6			30
2	8				18
3			3		30
SOMA	27	16	10	25	

Figura 1: Representação do estado inicial do tabuleiro.

	A	B	C	D	SOMA
1	12	6	2	10	30
2	8	1	5	4	18
3	7	9	3	11	30
SOMA	27	16	10	25	

Figura 2: Representação do estado final do tabuleiro depois de passar pelo programa.