Z3 Assignments, week 4
Group 2

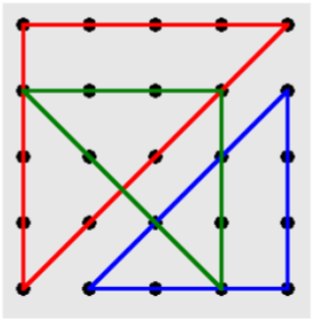
Triangles

Problem description:

Wk 4 Assignment (I)

In the picture below you see 3 triangles that cover each point of a 5x5 grid, in such a way that all vertices of all triangles are themselves points of that grid (not just points with integer coefficients).

Use Z3 to find another set of triangles subject to the same conditions.

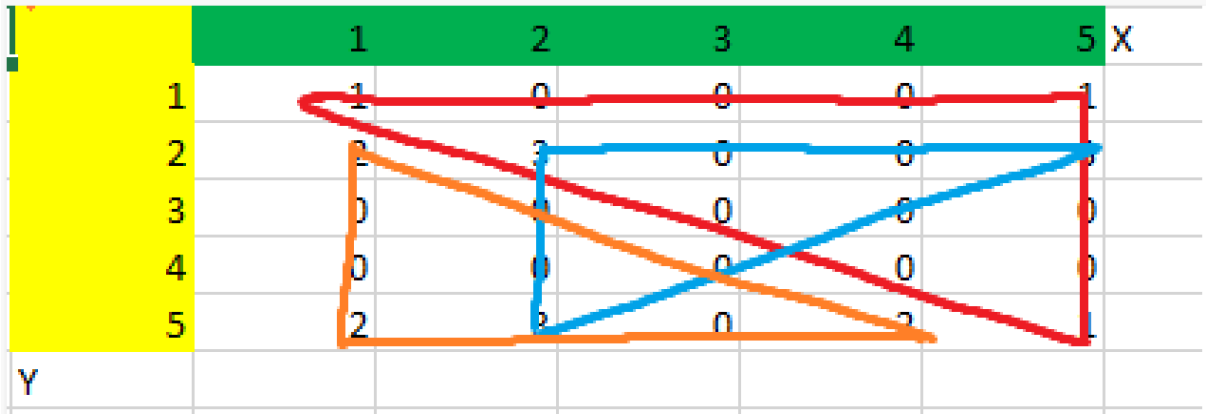


Solution explanation:

Solution:

- *In order to solve this puzzle, we create a matrix where the value of each cell will represent the vertex for the triangle.*
- *Our grid contains 4 distinct values: 0 - meaning that the point is not a triangle point/vertice. 1 up to 3 - signifying the 3 different triangles.*
- *In order to instantiate the bounds, we first initialise a 5x5 grid that should have values from 0 up to 3.*
- *An observation we made is that for a 5x5 grid, in order to have 3 distinct pairs of 3 points (each pair forming a triangle), there should be exactly 16 0's (or non-vertices/angle points) that appear on the grid and 9 points that represent the 3 triangles.*

- *Furthermore, in a function, we specify that there should be exactly 3 pairs of x and y values (representing a triangle) that exist within the grid. This function is called 3 times, assigning the values from 1 to 3 in order to create 3 distinct triangles.*
- *We then let Z3 choose from the following:*
    1) *Either the X1 and X2 values are the same and Y1 and Y2 are distinct. This is done in order to form a line/edge of a triangle. Another specification is that X1 and X3 are distinct/X2 and X3 are distinct. Z3 will have to choose between either (Y1 and Y3) or (Y2 and Y3) being equal.*
    2) *We have noticed that, in order to cover each point in the grid, triangles should have the length equal to 4 or 3. Therefore, we tell Z3 to choose the length for the triangles keeping in mind the other constraints., by using a formula which we discovered while examining the triangles.*
    3) *Another constraint to make sure that each point is covered is that the 0's between each 2 vertices should be equal.*
    Note: The same logic goes for different Y1 and Y2.
    4) *Finally, the points are created and are given a value later when the function is called 3 times in the assertion section.*



Source code:

```
;Instantiating the 5x5 grid, where Int1 is x and Int2 is Y
;It stores 4 different values - 0, meaning the point is not a triangle
vertice
; 1 - 3: 3 different values, appearing three times in pairs of two
; They represent each of the triangle's 3 vertices
(declare-fun T (Int Int) Int)


;Function for creating a triangle, where the value passed is from 1 to 3
(define-fun CreateTriangle ((value Int)) Bool

      ;There exists 3 vertices (3 pairs of x and y)
    (exists ((x1 Int) (x2 Int) (x3 Int) (y1 Int) (y2 Int) (y3 Int))
            (and
                        (<= 1 x1 5) (<= 1 y1 5) ;These values should be
in the grid's bounds (between 1 and 5)
                        (<= 1 x2 5) (<= 1 y2 5)
                        (<= 1 x3 5) (<= 1 y3 5)

            ;We let Z3 choose between two ways to build a unique
triangle
            ;based on the specified conditions
                  (or
                  ;1) IF the y1 and y2 values are the same, then
                  ; x1 and x2 should be distinct. This is done to form a
line
                (and
                    (= y1 y2)
                    (not (= x1 x2))

                    (not (= y3 y1))
                    (not (= y3 y2))
                        ;To make a triangle, the y3, y2 and y1 values
should be distinct respectively
                        ;We let Z3 choose which X values should be equal
in order to make a right-angled triangle
                    (or
                        (= x3 x1)
                        (= x3 x2)
                    )
```

```
                        ;We let Z3 determine the length of the triangles
                        ; This is done by checking the difference of the
specified points below
                    (or
                        (and
                            (= 3 (- y3 y1))
                            (= 3 (- x1 x2))
                            (= 3 (- y3 y2))
                        )
                        (and
                            (= 4 (- y3 y1))
                            (= 4 (- x1 x2))
                            (= 4 (- y3 y2))
                        )
                    )
                        ;The zero's between each vertice should be equal
                    (and
                        (= (- y3 y1) (- x1 x2))
                        (= (- y3 y2) (- x1 x2))
                    )

                )
                        ;The same goes for the other variant, but now the
x1 and x2 are the same
                        ;and y1 and y2 are different
                        (and
                    (= x1 x2)
                    (not (= y1 y2))
                    (not (= x3 x1))
                    (not (= x3 x2))
                    (or
                        (= y3 y1)
                        (= y3 y2)
                    )

                    (or
                        (and
                            (= 3 (- x3 x1))
                            (= 3 (- y1 y2))
                            (= 3 (- x3 x2))
                        )
                        (and
                            (= 4 (- x3 x1))
                            (= 4 (- y1 y2))
                            (= 4 (- x3 x2))
                        )
```

```
                    )
                    (and
                        (= (- x3 x1) (- y1 y2))
                        (= (- x3 x2) (- y1 y2))
                    )
                )
                  )
                  ;We assign the values for the 3 points
            (= (T x1 y1) value)
            (= (T x2 y2) value)
            (= (T x3 y3) value)
       )
    )
)


;Function for applying the 5x5 grid's max values - from 0 to 3
(define-fun MaxValuesForGrid ((x Int)(y Int)) Bool
    (and
        (<= 0 (T x y) 3)
    )
)

;Function for counting and asserting how many 0's (non-vertices)
;should appear in the grid
;In the case of this assignment, 16 of the points should be 0's and 9
should be vertices
(define-fun Total ((x Int)) Int
        (+
           (ite (= (T x 1) 0) 1 0)
           (ite (= (T x 2) 0) 1 0)
           (ite (= (T x 3) 0) 1 0)
           (ite (= (T x 4) 0) 1 0)
           (ite (= (T x 5) 0) 1 0)
        )
)


(assert (and
    ;The values for the 5x5 grid are from 1 to 5
    (forall ((x Int) (y Int))
        (implies
            (<= 1 x 5)
            (<= 1 y 5)
            (MaxValuesForGrid x y)
```

```
        )
    )
      ;We create 3 triangles
      (CreateTriangle 1)
      (CreateTriangle 2)
      (CreateTriangle 3)

      ;Assigning that the sum of all 0's (non-vertices) should be equal
to 16
    (= 16 (+ (Total 1) (Total 2) (Total 3) (Total 4) (Total 5)))

))




(check-sat)
(get-value (
    (T 1 1)
    (T 1 2)
    (T 1 3)
    (T 1 4)
    (T 1 5)

    (T 2 1)
    (T 2 2)
    (T 2 3)
    (T 2 4)
    (T 2 5)

    (T 3 1)
    (T 3 2)
    (T 3 3)
    (T 3 4)
    (T 3 5)

    (T 4 1)
    (T 4 2)
    (T 4 3)
    (T 4 4)
    (T 4 5)

    (T 5 1)
    (T 5 2)
    (T 5 3)
```

```
    (T 5 4)
    (T 5 5)
))
```

Intransitive Dice:

Problem Description:

Wk 4 Assignments (II)

Let Z3 find a set of 3 *intransitive dice*.

See: https://en.wikipedia.org/wiki/Intransitive_dice

What are intransitive dice?

"A set of dice is intransitive (or nontransitive) if it contains three dice, A, B, and C, with the property that A rolls higher than B more than half the time, and B rolls higher than C more than half the time, but it is not true that A rolls higher than C more than half the time. In other words, a set of dice is intransitive if the binary relation – X rolls a higher number than Y more than half the time – on its elements is not transitive. More simply, A normally beats B, B normally beats C, but A does not normally beat C."

The solution we found is the following:

Die 1 : 9,2,2,9,2,2
Die 2: 1,4,8,6,1,6
Die 3: 7,5,3,5,3,3

The way this was confirmed to be intransitive is by using the below shown library and by manually checking with probability.

Examples

```
library(Rdice)

df <- data.frame(
  die1 = c(9,2,2,9,2,2),
  die2 = c(1,4,8,6,1,6),
  die3 = c(7,5,3,5,3,3)
)

is.nonTransitive(df)
```

Run

You should assume that any scripts or data that you put into this service are public.

Read more about your data's privacy and security here.

```
[1] TRUE
```

Solution explanation:

Solution:

- *In order to find 3 unique dice that satisfy intransitivity, we had a couple of observations to help instantiate the conditions. We noticed that for all of the cases provided in different examples, the dice's sides combined are equal between the three dice (or more).*

- *We decided to create a matrix in order to represent the values:*
  Rows representing Die 1, 2, 3
  Columns representing the respective sides

- *Another observation that we made is that the numbers that appear in a given die do not reappear in other dice. Having said this, there are some special cases where this might not apply, however it was out of the bounds of the assignment.*

- *For simplicity, we decided to also limit the values to be between 1 and 9 for all of the sides*

- *Lastly, we let Z3 have the option to choose between:*
  1) A > B, B > C and C > A
  2) A < B, B < C and C < A
- *All of this is done in 2 or's, each holding the conditions mentioned above respectively. This is done because if A > B, B > C, C could never be > A. Allowing Z3 to choose between all of the different conditions allows it to find solutions that satisfy the intransitivity more than half of the times.*

Source code:

```
(declare-fun Dice (Int Int) Int)
;A > B 5/9 times
;B > C 5/9 times
;C > A 5/9 times

(assert
    (and
    ;1) Values for each side should be from 1 to 9
    (forall ((side Int))
```

```
        (implies
                (and
                    (<= 1 side 6)
                )

                (and
                    (<= 1 (Dice 1 side) 9)
                    (<= 1 (Dice 2 side) 9)
                    (<= 1 (Dice 3 side) 9)
                )
        ))



    ;2) The values should be different per dice (if dice 1 has 4, then
dice 2 and 3 != 4);
    (forall ((side Int) (side2 Int) (side3 Int))
        (implies
                (and
                    (<= 1 side 6)
                    (<= 1 side2 6)
                    (<= 1 side3 6)
                )
                ;(distinct side side2 side3)
                (distinct (Dice 1 side) (Dice 2 side2) (Dice 3 side3))
         )
    )



        ;3) The sum of each of the dice's sides should be equal per dice
        (= (+ (Dice 1 1) (Dice 1 2) (Dice 1 3) (Dice 1 4) (Dice 1 5)
(Dice 1 6)) (+ (Dice 2 1) (Dice 2 2) (Dice 2 3) (Dice 2 4) (Dice 2 5)
(Dice 2 6)))
        (= (+ (Dice 2 1) (Dice 2 2) (Dice 2 3) (Dice 2 4) (Dice 2 5)
(Dice 2 6)) (+ (Dice 3 1) (Dice 3 2) (Dice 3 3) (Dice 3 4) (Dice 3 5)
(Dice 3 6)))
        (= (+ (Dice 1 1) (Dice 1 2) (Dice 1 3) (Dice 1 4) (Dice 1 5)
(Dice 1 6)) (+ (Dice 3 1) (Dice 3 2) (Dice 3 3) (Dice 3 4) (Dice 3 5)
(Dice 3 6)))

      ;4) There exists dice that follow the given conditions:
            ; if A>B, B>C, C !> A
            ; if A<B, B<C, C !< A
            ; We let Z3 choose the conditions in a way such that 3 dice
are formed where
```

```
            ; more than half of the times the above conditions should be
true
    (exists ((side Int) (side2 Int) (side3 Int))

    (and
                    (<= 1 side 6)
                    (<= 1 side2 6)
                    (<= 1 side3 6)

                (or
                    (> (Dice 1 side)  (Dice 2 side2))
                    (> (Dice 2 side2) (Dice 3 side3))
                    (> (Dice 3 side3) (Dice 1 side))
                )
                (or
                    (< (Dice 1 side)  (Dice 2 side2))
                    (< (Dice 2 side2) (Dice 3 side3))
                    (< (Dice 3 side3) (Dice 1 side))
                )
    )
    )


)
)

(check-sat)

(echo "Die 1")
(get-value (
    (Dice 1 1)
    (Dice 1 2)
    (Dice 1 3)
    (Dice 1 4)
    (Dice 1 5)
    (Dice 1 6)

))
(echo "Die 2")
(get-value (
    (Dice 2 1)
    (Dice 2 2)
    (Dice 2 3)
    (Dice 2 4)
    (Dice 2 5)
    (Dice 2 6)
```

```
))
(echo "Die 3")
(get-value (

    (Dice 3 1)
    (Dice 3 2)
    (Dice 3 3)
    (Dice 3 4)
    (Dice 3 5)
    (Dice 3 6)

))
```