Applied Logic Weekly Assignment Report

Bohdan Tymofieienko B.S.
b.tymofieienko@student.fontys.nl
4132645

Mikolaj Hilgert M.H.
m.hilgert@student.fontys.nl
4158385

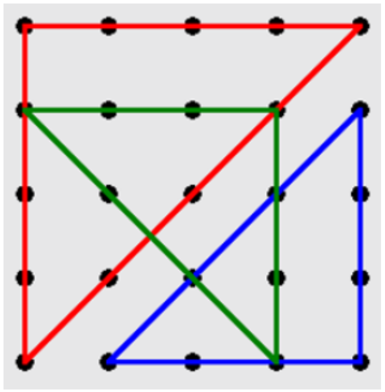
January 10, 2022

Contents

1 Week iv. 3 Triangles and Non-transitive Dice

1.1 Three Triangles

In this puzzle the target is to arrange three triangles on a 5×5 grid so that their vertices are also contained in a grid and all the cells are covered.



To make $Z3$ solve this puzzle we use some properties from geometry.

1. **Only one and exactly one triangle can be formed from three different points in space contained these points are vertices of the triangle.**

2. **If point in space is contained within triangle it lies either on opposite side, either on adjacent side or on the diagonal.**

Having 5×5 grid we would want $Z3$ to find such an arrangement of three sets of vertices *(corresponding to three triangles)* so that all the vertices are covered. Or in other words each cell is located either on opposite side, either on adjacent side or on the diagonal.

To make this possible we introduce some labels. Let all the vertices of triangle A, B, C be labeled as $1, 2, 3$ respectively. Let all the other cells be labeled as 0*(could be interpreted as cells if looking at the picture)*.

Translated in $Z3$ syntax we are getting next assertions.

First of all we have to assure that each entry in a grid is between 0 and 3. As was mentioned before 1,2,3 are triangle vertices and 0 is a dot on a grid.

```
 (forall ((x Int)(y Int))
     (=> (and (<= 1 x 5) (<= 1 y 5))
          (<= 0 (B x y) 3)
     )
)
```

Next, we specify that there exists three sets of three vertices, which define existence of three triangles on a grid. Function 'ExistsTriangle' declares existence of three distinct pairs of vertices. It assures that there exists three pairs of X and Y values *(6 values in total)* which are not the equal, so that each triangle has 3 vertices.

```
     (ExistsTriangle 1)
     (ExistsTriangle 2)
     (ExistsTriangle 3)
```

Since there **exactly** three sets of vertices we have to specify explicitly this explicitly. Reversing the statement there exists exactly 16 points on a grid which are **not** vertices. Function Total counts amount of zero per row.

```
 (=
         (+ (Total 1) (Total 2) (Total 3) (Total 4) (Total 5))
 16)
```

In the last assertion we assure that all the cells are covered. For all cells on the grid which are not vertices one of three possible cases apply: lies on adjacent side, on opposite side or on diagonal.

```
(forall ((x Int)(y Int))
     ;select only non-verticies elements
     (=> (and (<= 1 x 5) (<= 1 y 5) (= (B x y) 0))
          (or
               ;ROW-> are zeroes in between two verticies in a row
               ;Dot lies on opposite side
               (LiesOnAdjacentSide x y)
               ;Column -> are zeroes in between two verticies in a column
               ;Dot lies on adjacent side
               (LiesOnOppositeSide x y)
               ;Diagonal-> are zeroes in between two verticies in a diagonal
               ;Dot lies on diagonal
               (LiesOnHypothenus x y)
          )
     )
)
```

To assure that dot is lying on the side we may consider a following statement. In the same row, column or diagonal there exist two vertices of the same triangle so that the cell itself is between those vertices. In $Z3$ we assure this in the next way

```
(define-fun LiesOnAdjacentSide ((x Int)(y Int)) (Bool)
    (exists ((y1 Int)(y2 Int))
        (and
            (CheckCoords y y1 y2)
            ;same column
            (= (B x y1) (B x y2))
            ;only consider vertices
            (not (= (B x y1) (B x y2) 0))
        )
    )
)
```

The example above demonstrates adjacent side example. In a similar way it is done for opposite side and hypotenuse.

To assure that two vertices lies on the same diagonal we do a following verification. When both of the distinct are true, two cells are not located on the same diagonal.

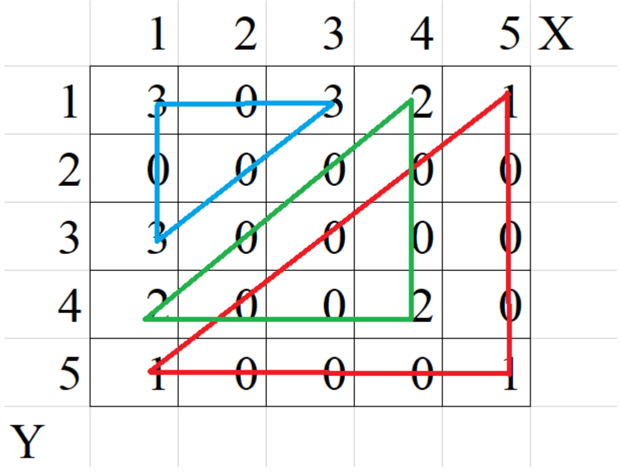
```
(define-fun SameDiag ((x1 Int) (y1 Int) (x2 Int) (y2 Int)) (Bool)
        (not (and
                (distinct (- y1 y2) (- x1 x2))
                (distinct (- y1 y2) (- x2 x1))
        ))
)
```

Additionally, we were tasked to find a set vertices different from the one given in original puzzle answer. To do that we will assert one cell manually. In this way $Z3$ finds a model different from the classical one.

```
(= (B 3 1) 3)
```

1.1.1 Conclusion

The result can be interpreted as next. As you can see three triangles cover the whole grid and set of triangles is different from the one given in puzzle description.



	1	2	3	4	5	X
1	3	0	3	2	1	
2	0	0	0	0	0	
3	3	0	0	0	0	
4	2	0	0	2	0	
5	1	0	0	0	1	
Y						

1.2 Non-transitive Dice

A set of dice is intransitive (or non-transitive) 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. This means that there is no dice that beats all. More like that, there exists another die that beats it.

The task is to have *Z3* find a set 3 dice so that they are non-transitive. This again can be done using a table which will be defined by: A die and face number that returns the value on the die's face. In **Z3** this is represented with the following;

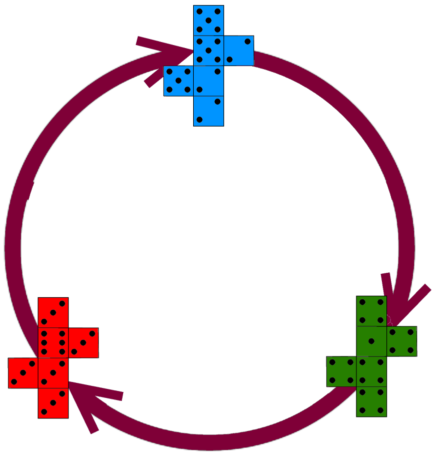
```
;dice—num, face—num —> value
(declare—fun B (Int Int) Int)
```

Now given the fact that we have the board and the disks defined, we can set the first assertions. So the first thing to be defined are the bounds of the problem, this being setting out what the dice consists of and how many there are.

```
(forall ((dice Int)(die Int))
    ; Three dice . Six faces each
    (=> (and (<= 1 dice 3) (<= 1 die 6))
        ;Values of die from 1 to 6.
        (<= 1 (B dice die) 9)
    )
)
```

So what was defined is that as per the problem specification we will be finding 3 non-transitive dice, thus we say that forall dice, they are to be between 1 and 3. Then we specify that there are 6 faces per die. Finally, we say that the value of the faces on a die may be between 1 and 9.

Next, we must add the non-transitive aspect of the problem, namely the fact that for every die, there exists another die that beats it. This means that with 3 non-transitive dice. Die 3 has a dice that beats it, Say die 2, according tho this specification that means die 1 would beat it more than half the time, and continuing this specification that means dice 1 also has a die that beats it, in this case it cannot be 2 as die 1 beats die 2. So consequently die 3 should beat die 1.



To specify this behaviour in *Z3*. We say that for all dice [1,3], there exists a 'beating die'. which is also between [1,3] so what we tell *Z3* is that it must find a set of 3 dice so that from that set of dice each one is beat by another dice more than half the time. So over 50% of the time.

```
(forall ((die Int))
    ;for die in bound
    (=> (<= 1 dice 3)
        ;there exists a die that beats another die, within the same bounds
        (exists ((beatingDie Int))
            (and
            (<= 1 beatingDie 3)
```

With the quantifier and bounds set, we may now say how the probability aspect will be enforced. So, since the dice play may be shown in tabular form, with dice with 6 faces, there are 36 possible play outcomes with two dice. To have one die beat another, it means that it must beat it **more than** 18 times. As that would mean it wins < 50% of the time. To do this in *Z3*, we can define a function that checks whether a face value on a beating die beats the other faces on the 'beaten' die. Then we can call it for each face of the beating dice to reduce repeated code.

```
;function that plays all possible outcomes of one die with the the dice
...defined above
    (define-fun FacePlayOutcome ((face Int)(die Int)) (Int)
        (+
            (ite (> face (B die 1)) 1 0)
            (ite (> face (B die 2)) 1 0)
            (ite (> face (B die 3)) 1 0)
            (ite (> face (B die 4)) 1 0)
            (ite (> face (B die 5)) 1 0)
            (ite (> face (B die 6)) 1 0)
        )
    )


...found in forall
            ;probability condition -> so that the beating dice beats the dice
            ;more than half the time (36 possible play variations,
            ;so more than 18 (half))
            (>
                (+
                    (FacePlayOutcome (B beatingDie 1) die)
                    (FacePlayOutcome (B beatingDie 2) die)
                    (FacePlayOutcome (B beatingDie 3) die)
                    (FacePlayOutcome (B beatingDie 4) die)
                    (FacePlayOutcome (B beatingDie 5) die)
                    (FacePlayOutcome (B beatingDie 6) die)
                )
            18)
            )
        )))
```

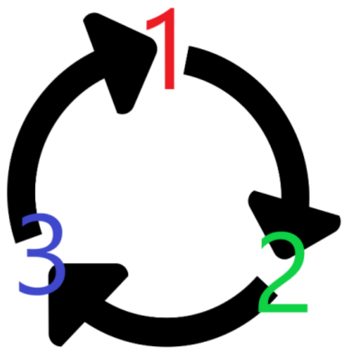
1.2.1 Conclusion

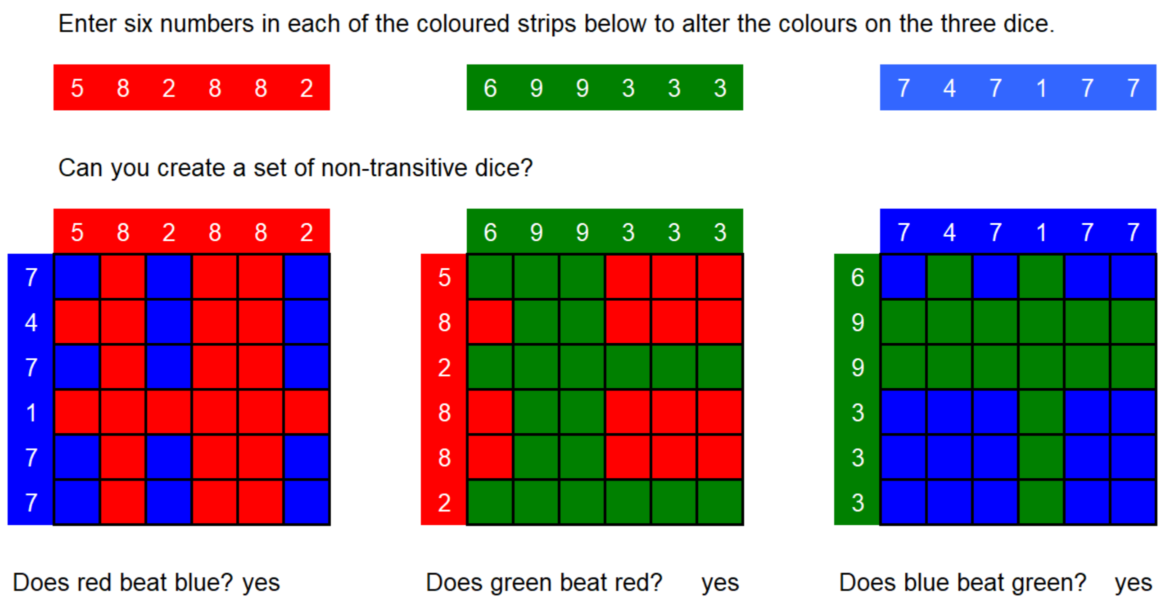
This is the raw output given by *Z3*:

	Face 1	Face 2	Face 3	Face 4	Face 5	Face 6
Die 1	5	8	2	8	8	2
Die 2	6	9	9	3	3	3
Die 3	7	4	7	1	7	7

These are all the conditions needed for this problem, so to verify the output, we manually check the output and we can see that **Die 1 beats die 2, 2 beats die 3, but die 3 beats die 1** for more than half the time.

NOTE: This is shown visually in the following diagram, Say Die 1 is *Red*, Die 2 is *Green* and Die 3 is *Blue*.





Enter six numbers in each of the coloured strips below to alter the colours on the three dice.

5	8	2	8	8	2	6	9	9	3	3	3	7	4	7	1	7	7

Can you create a set of non-transitive dice?

	5	8	2	8	8	2
7	🟦	🟥	🟦	🟥	🟥	🟦
4	🟥	🟥	🟦	🟥	🟥	🟦
7	🟦	🟥	🟦	🟥	🟥	🟦
1	🟥	🟥	🟥	🟥	🟥	🟥
7	🟦	🟥	🟦	🟥	🟥	🟦
7	🟦	🟥	🟦	🟥	🟥	🟦

Does red beat blue? yes

	6	9	9	3	3	3
5	🟩	🟩	🟩	🟥	🟥	🟥
8	🟥	🟩	🟩	🟥	🟥	🟥
2	🟩	🟩	🟩	🟩	🟩	🟩
8	🟥	🟩	🟩	🟥	🟥	🟥
8	🟥	🟩	🟩	🟥	🟥	🟥
2	🟩	🟩	🟩	🟩	🟩	🟩

Does green beat red? yes

	7	4	7	1	7	7
6	🟦	🟩	🟦	🟩	🟦	🟦
9	🟩	🟩	🟩	🟩	🟩	🟩
9	🟩	🟩	🟩	🟩	🟩	🟩
3	🟦	🟦	🟦	🟩	🟦	🟦
3	🟦	🟦	🟦	🟩	🟦	🟦
3	🟦	🟦	🟦	🟩	🟦	🟦

Does blue beat green? yes