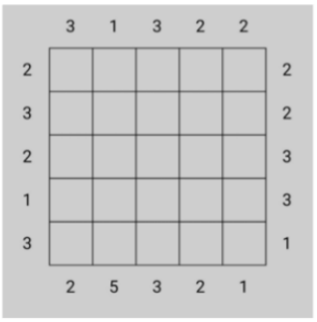
Z3 Assignments, week 3
Group 2

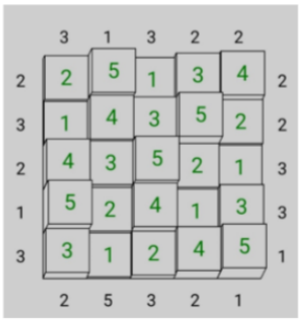
Skyscrapers

Problem description:

Wk 3 Assignment (I): towers aka. skyscraper

- https://brainbashers.com/skyscrapers.asp
- (Android) "Simon Tatham's Puzzles"
- ***Provide a 'general solution'***





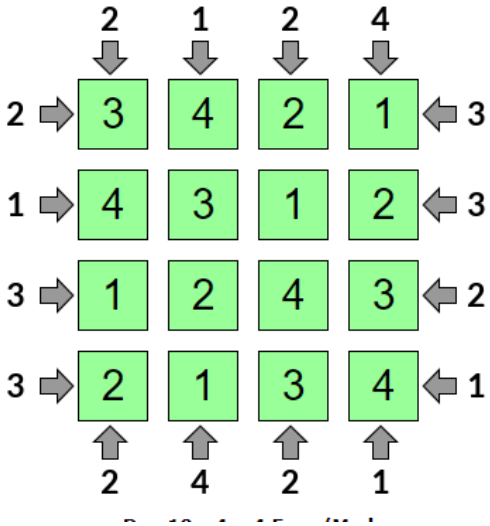
Note: In the .zip file, 2 different variants of the same solution, one taking only 3 clues and the other takes 16.

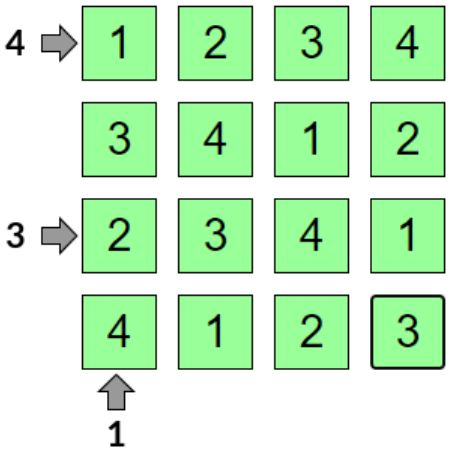
Solution explanation:

Solution

- *To find a general solution we have chosen the 4x4 Skyscrapers puzzles.*
- *First, we need to create the table model which in this case is* ***(declare-fun Cell (Int Int) Int)*** *which will be used to store the number corresponding to each cell.*
- *Then two functions, respectively* ***UniquePerRow*** *and* ***UniquePerCol*** *assure that each value should be distinct.*

- *Since this puzzle is using clues, we came up with the idea to let the computer know, based on row or column, how many skyscrapers it can see. Thus, we will have 4 functions to define the constraints for the skyscrapers.*
- *For **See1Skyscrapers**, we need to assign each variable the highest possible value, in this case being 4.*
- *For **See4Skyscrapers**, we need to order the values from the smallest to the biggest.*
- *For **See2Skyscrapers** and **See3Skyscrapers**, we need to check each possible combination of values we could make in order to see 2 or 3 skyscrapers. There are 3 different possibilities per function. We check every value at the given steps and if Z3 finds a pattern which matches one of our conditions, it then assigns the values accordingly.*
- *Finally, depending on the difficulty of the 4x4 puzzle, we can assign up to 16 clues for a puzzle. (As it can be seen in the attached screenshots)*





Source code:

```
(declare-fun Cell (Int Int) Int)

;All cells in one row 'row' are distinct
(define-fun UniquePerRow ((row Int)) Bool
    (and (distinct (Cell row 1)
                   (Cell row 2)
```

```
            (Cell row 3)
            (Cell row 4)
    ))

)

; All cells in one column 'col' are distinct
(define-fun UniquePerCol ((col Int)) Bool
    (distinct (Cell 1 col)
            (Cell 2 col)
            (Cell 3 col)
            (Cell 4 col)
    )
)

; Since we can see only 1 skyscrapper, the value of each variable should
be highest value (in a 4x4 it should be 4), depending on the cell
(define-fun See1Skyscrapers ((v1 Int) (v2 Int) (v3 Int) (v4 Int)) Bool
    (or
        (and(= v1 4))
        (and(= v2 4))
        (and(= v3 4))
        (and(= v4 4))
    )
)

; We check each possible combination of values to make sure we can only
see 2 skyscrapers
(define-fun See2Skyscrapers ((v1 Int) (v2 Int) (v3 Int) (v4 Int)) Bool
    (or
         (and (< v1 v2) (> v2 v3) (> v2 v4))
         (and (> v1 v2) (< v1 v3) (> v3 v4))
         (and (> v1 v2) (> v1 v3) (< v1 v4)))
)

; We check each possible combination of values to make sure we can only
see 3 skyscrapers
(define-fun See3Skyscrapers ((v1 Int) (v2 Int) (v3 Int) (v4 Int)) Bool
    (or (and (< v1 v2) (< v2 v3) (> v3 v4))
        (and (< v1 v2) (> v2 v3) (< v2 v4))
        (and (< v1 v3) (< v2 v1) (< v3 v4)))
)

; We need to make sure that the variables are sorted from the the
smallest to the biggest
(define-fun See4Skyscrapers ((v1 Int) (v2 Int) (v3 Int) (v4 Int)) Bool
```

```
    (or
       (and (< v1 v2) (< v2 v3) (< v3 v4))
    )
)

; assert part
(assert (and
(forall ((c Int) (r Int))
    (implies
        (and
            (<= 1 c 4)
            (<= 1 r 4)
        )
        (and
        (<= 1 (Cell r c) 4)
        (UniquePerRow r )
        (UniquePerCol c )
        )
    )
)

; Define clues
(see4Skyscrapers (Cell 1 1) (Cell 1 2) (Cell 1 3) (Cell 1 4))
(see3Skyscrapers (Cell 3 1) (Cell 3 2) (Cell 3 3) (Cell 3 4))
(see1Skyscrapers (Cell 4 1) (Cell 3 1) (Cell 2 1) (Cell 1 1))




))
(check-sat)
(echo "The cell values are: ")
(get-value (
    (Cell 1 1)
    (Cell 1 2)
    (Cell 1 3)
    (Cell 1 4)

    (Cell 2 1)
    (Cell 2 2)
    (Cell 2 3)
    (Cell 2 4)

    (Cell 3 1)
    (Cell 3 2)
    (Cell 3 3)
```

```
    (Cell 3 4)

    (Cell 4 1)
    (Cell 4 2)
    (Cell 4 3)
    (Cell 4 4)

))
```

---

3 Water Jugs Puzzle:

Problem Description:

Wk 3 Assignments (II)

- We have 3 jugs of water, capacity: 8, 5, 3 liters resp.
- Initially 8 liters of water in the first jug, other two empty
- Wanted end-situation: 4 liters in the first jug, 4 liters in the second jug, 3rd jug empty
- The jugs are irregularly shaped and unmarked; no water can be spilled; each step pouring water from one jug to another stops when either the source jug is empty or the destination jug is full, whichever happens first.

Use Z3 to find a solution (and present the answer in human readable form)

Note: In the provided zip files, we have included 2 different variants of the same solution, one taking 8 stages to solve and the other - 10. This value can be tweaked if needed. (the solver is adaptable)

Solution explanation:

Solution:

1) *To properly tackle this problem, we decided it would be best to create a graph based on the 3 different jugs and the amount of steps (stages) it would take for it to reach the desired values. The graph can be seen in the image below.*
2) *We first initialise all of the cells (different jugs at different stages) with* ***(declare-fun Jug (Int Int) Int)*** *, where the first integer value is the jug number (1 being max capacity 8, 2 being 5 and 3 being 3), and the second integer being the step it is at.*
3) *The* ***(PossibleStagesToSolve)*** *is declared with the idea of changing the desired steps just by switching 1 value, instead of having to go through all of the functions.*
4) ***(WaterContentPerStage)*** *is a constraint that makes sure at each row (stage) the water content between the 3 jugs remains constant (8 litres in this case).*
5) ***(MaxValueForJug)*** *is a function that constrains the maximum capacity each jug can have (In this case, jgNr1 =max8, jgNr2 = max5 and jgNr3 = max3)*

- *ORDERING:*

1) *We first assign the values (litres of water) each jug initially has. After that, we assign the* ***(PossibleStagesToSolve)*** *to the amount of steps that we want the computer to take to get to the desired result. (In the provided zip files , we have chosen 8 stages and 10 stages, but we will use the 8 stages as an example to work on, as it is more known.)*
2) *We then apply the max litres each jug can have*

3) *We then have a (not (exists ….) clause, that takes in 2 different stages. We use this in order to make sure that the Z3 does not take the same step twice (redundant/repeating steps)*
4) *As we move further, we have another (exists) condition, where we specify that at some stage between 1 and* ***(PossibleStagesToSolve)****, there is a stage that has the desired end values (jg1 = 4, jg2 = 4, jg3 = 0)*
5) *We then make sure that for all of the stages, the water content combined between the three jugs remains constant (8 litres in this case).*
6) *Finally, we have a big (forall) clause in which we set the following conditions/constraints:*

- *We make sure that only two of the jugs can change their contents at any given stage, but the 3rd's value (litres) remains the same. The way we do this is by having 3 different possible routes that Z3 can pick from:* ***(JUG1 and JUG2 change, JUG3 remains)*** *or* ***(JUG1 and JUG3 change, JUG 2 remains)*** *or* ***(JUG2 and JUG3 change, JUG1 remains).*** *This is done by specifying that (for example) Jug3 at* ***(+ 1 stage/the next stage)*** *is equal to jug3 at the* ***previous stage.***
- *We then specify that the sum of values (for example) between Jug 1 and jug 2 at the* ***following step*** *are equal to the sum of Jug 1 and Jug 2 at the* ***previous step.***

GRAPH

Step	Jug 1	Jug 2	Jug 3
1	8	0	0
2	3	5	0
3	3	2	3
4	6	2	0
5	6	0	2
6	1	5	2
7	1	4	3
8	4	4	0

Source code:

```
(declare-fun Jug (Int Int) Int)
;The possible stages that the puzzle can be solved
(declare-const PossibleStagesToSolve Int)

;The sum of each jug per each stage is equal to 8
(define-fun WaterContentPerStage ((stage Int)) Bool
    (and
        (= (+ (Jug 1 stage) (Jug 2 stage) (Jug 3 stage)) 8)
    )

)

;Defines the maximum capacity for each jug
(define-fun MaxValueForJug ((stage Int)) Bool
    (and
        (<= 0 (Jug 1 stage) 8)
        (<= 0 (Jug 2 stage) 5)
        (<= 0 (Jug 3 stage) 3)
```

```
    )
)

;Makes the jugs at each stage have values between 8 and 0
(define-fun ApplyMaxValue () Bool
    (and (MaxValueForJug 1) (MaxValueForJug 2)
         (MaxValueForJug 3) (MaxValueForJug 4)
         (MaxValueForJug 5) (MaxValueForJug 6)
         (MaxValueForJug 7) (MaxValueForJug 8)
    )
)

(assert
    (and
        ;1) Assigns the initial values of each jug and the possible stages
it takes to find an answer
        (= (Jug 1 1) 8)
        (= (Jug 2 1) 0)
        (= (Jug 3 1) 0)
        (= PossibleStagesToSolve 8)

        ;2) Makes sure that the values do not exceed the limit
        ApplyMaxValue

        ;3) No two stages are repeated
        (not
            (exists ((stage1 Int) (stage2 Int))
                (and
                    (<= 1 stage1 PossibleStagesToSolve)
                    (<= 1 stage2 PossibleStagesToSolve)
                    (distinct stage1 stage2)
                    (and
                        (= (Jug 1 stage1) (Jug 1 stage2))
                        (= (Jug 2 stage1) (Jug 2 stage2))
                        (= (Jug 3 stage1) (Jug 3 stage2))
                    )
                )
            )
        )

        ;4) There exists a stage between 1 and 8 that has the values 4 4 0
        (exists ((stage Int))
            (and
                (<= 1 stage PossibleStagesToSolve)
                (and
                (= (Jug 1 stage) 4)
                (= (Jug 2 stage) 4)
                (= (Jug 3 stage) 0)
```

```
                )

            )
        )

        ;5) For all stages, the water content between the jugs combined is 8
        (forall ((stage Int))
          (implies
            (and
                (<= 1 stage PossibleStagesToSolve)
            )
            (WaterContentPerStage stage)
          )
        )

 ;6) It makes sure only 2 values change per stage, 3rd remains the same and
 ; either 1 jug becomes full and the other one empty, or the other way
around
        (forall ((stage Int))
          (implies
            (and
                (<= 1 stage PossibleStagesToSolve)
            )
            (and
                (or
                    ;jg1 and j2 change, j3 remains
                    (and
                         (= (Jug 3 (+ stage 1)) (Jug 3 stage))
                         (= (+ (Jug 1 (+ stage 1)) (Jug 2 (+ stage 1))) (+
(Jug 1 stage) (Jug 2 stage)))
                            (or
                                (= (Jug 1 (+ stage 1)) 0)
                                (= (Jug 2 (+ stage 1)) 5)
                                (= (Jug 2 (+ stage 1)) 0)
                                (= (Jug 1 (+ stage 1)) 8)
                            )
                    )
                    ;jg1 and jg3 change, j2 remains
                    (and
                         (= (Jug 2 (+ stage 1)) (Jug 2 stage))
                         (= (+ (Jug 1 (+ stage 1)) (Jug 3 (+ stage 1))) (+
(Jug 1 stage) (Jug 3 stage)))
                            (or
                                  (= (Jug 1 (+ stage 1)) 0)
                                  (= (Jug 3 (+ stage 1)) 3)
                                  (= (Jug 3 (+ stage 1)) 0)
                                  (= (Jug 1 (+ stage 1)) 8)
```

```
                        )
                    )
                    ;j2 and j3 change, j1 remains
                    (and
                        (= (Jug 1 (+ stage 1)) (Jug 1 stage))
                        (= (+ (Jug 2 (+ stage 1)) (Jug 3 (+ stage 1))) (+
(Jug 2 stage) (Jug 3 stage)))
                            (or
                                    (= (Jug 3 (+ stage 1)) 0)
                                    (= (Jug 2 (+ stage 1)) 5)
                                    (= (Jug 2 (+ stage 1)) 0)
                                    (= (Jug 3 (+ stage 1)) 3)

                            )
                    )
                )
            )
          )
        )

    )
)

(echo "The values for Jug 1, Jug 2 and Jug 3:")
(check-sat)
(get-value (
    (Jug 1 1)
    (Jug 2 1)
    (Jug 3 1)

    (Jug 1 2)
    (Jug 2 2)
    (Jug 3 2)

    (Jug 1 3)
    (Jug 2 3)
    (Jug 3 3)

    (Jug 1 4)
    (Jug 2 4)
    (Jug 3 4)

    (Jug 1 5)
    (Jug 2 5)
    (Jug 3 5)

    (Jug 1 6)
    (Jug 2 6)
```

```
    (Jug 3 6)

    (Jug 1 7)
    (Jug 2 7)
    (Jug 3 7)

    (Jug 1 8)
    (Jug 2 8)
    (Jug 3 8)

))
```