

---

# Logic Set Theory

By: Bas Smit

Class: AP-S4

Version: 1

Date: 17/04/2024

---



# Table of content

<b>Table of content</b> .....	<b>2</b>
<b>Coin Denominations</b> .....	<b>3</b>
Approach.....	3
Input.....	4
Output.....	4
<b>Skyscrapers</b> .....	<b>5</b>
Approach.....	6
Input.....	7
Output.....	8
<b>Frogs</b> .....	<b>9</b>
Approach.....	9
Input.....	9
Output.....	10
<b>Slide puzzle (8 tiles)</b> .....	<b>12</b>
Approach.....	12
Input.....	13
Output.....	14

## Coin Denominations


A country mints 3 denominations of coins, in whole numbers of centos. It requires 3 of these coins to make 20 centos, or 23 centos, or 29 centos. What are the denominations of coins?

### Approach

As For all problems I started out with thinking about what the output was gonna look like (tip from herman last year). Here I came to the conclusion that for the output I need 3 coins each with a different value.

Then I found That the problem only had 2 "rules". Firstly the coins cannot be of a negative value. Secondly 3 coins needed to be combined to make 20, 23 and 29 cents (duplicates are allowed).

## Input



```
1  (assert
2    (and
3      (coinCombinations 20)
4      (coinCombinations 23)
5      (coinCombinations 29)
6    )
7  )
```

## Output

```
PS C:\Users\bassm\OneDrive\Documenten\GitHub\Academic-prep\Z3\Exercises> .\z3.exe .\coinDenominations.smt
sat
(((coin 1) 7)
 (coin 2) 6)
 (coin 3) 11))
PS C:\Users\bassm\OneDrive\Documenten\GitHub\Academic-prep\Z3\Exercises>
```

## Skyscrapers

Find a 'general' way to solve puzzles of the following kind: For the precise rules, variants and other details, see: <https://brainbashers.com/skyscrapers.asp>. See also "Simon Tatham's Puzzles". Your solution should be able to solve at least 'hard' 4x4 and 5x5 puzzles

## Approach

For this problem I found a couple of rules each skyscraper puzzle must abide by.

1. The given values for a 5x5 grid can only be between 1 and 5 (inclusive)
2. Each row can not include duplicates
3. Each column can not include duplicates

I also needed to find a way to describe how N skyscrapers were visible in each row and column. I did this by stating that if you can see the next skyscraper it should be bigger than all previous ones. Apply this logic and define how many skyscrapers you can see on each side and that is how i came to my solution

## Input



```
1  (assert (= (seeTowersFromLeft 1) 2))
2  (assert (= (seeTowersFromLeft 2) 2))
3  ;(assert (= (seeTowersFromLeft 3) 3))
4  (assert (= (seeTowersFromLeft 4) 2))
5  ;(assert (= (seeTowersFromLeft 5) 2))
6
7  ;(assert (= (seeTowersFromRight 1) 3))
8  (assert (= (seeTowersFromRight 2) 3))
9  ;(assert (= (seeTowersFromRight 3) 4))
10 ;(assert (= (seeTowersFromRight 4) 3))
11 ;(assert (= (seeTowersFromRight 5) 2))
12
13 ;(assert (= (seeTowersFromTop 1) 5))
14 ;(assert (= (seeTowersFromTop 2) 4))
15 ;(assert (= (seeTowersFromTop 3) 5))
16 (assert (= (seeTowersFromTop 4) 3))
17 ;(assert (= (seeTowersFromTop 5) 5))
18
19 ;(assert (= (seeTowersFromBottom 1) 2))
20 ;(assert (= (seeTowersFromBottom 2) 5))
21 (assert (= (seeTowersFromBottom 3) 2))
22 (assert (= (seeTowersFromBottom 4) 1))
23 (assert (= (seeTowersFromBottom 5) 4))
```

## Output

The output should be pretty clear but it shows the answer for a 5x5 grid with the sides being shown in the input (only look at the lines that are uncommented).

```
PS C:\Users\bassm\OneDrive\Documenten\GitHub\Academic-prep\Z3\Exercises> .\z3.exe .\Skyscraper.smt
sat
Getting values for coordinate (1,1) to (1,5)
(((coordinate 1 1) 4)
 (coordinate 1 2) 1)
• ((coordinate 1 3) 5)
  (coordinate 1 4) 3)
  (coordinate 1 5) 2))

Getting values for coordinate (2,1) to (2,5)
(((coordinate 2 1) 2)
 (coordinate 2 2) 5)
 (coordinate 2 3) 3)
 (coordinate 2 4) 1)
 (coordinate 2 5) 4))

Getting values for coordinate (3,1) to (3,5)
(((coordinate 3 1) 1)
 (coordinate 3 2) 4)
 (coordinate 3 3) 2)
 (coordinate 3 4) 5)
 (coordinate 3 5) 3))

Getting values for coordinate (4,1) to (4,5)
(((coordinate 4 1) 3)
 (coordinate 4 2) 2)
 (coordinate 4 3) 1)
 (coordinate 4 4) 4)
 (coordinate 4 5) 5))

Getting values for coordinate (5,1) to (5,5)
(((coordinate 5 1) 5)
 (coordinate 5 2) 3)
 (coordinate 5 3) 4)
 (coordinate 5 4) 2)
 (coordinate 5 5) 1))
```



## Frogs

All 6 frogs need to be moved to the other side. Every frog can either jump one place ahead or jump over one frog. Jumping backwards is not allowed. Model this with SMT, and let Z3 find a solution.

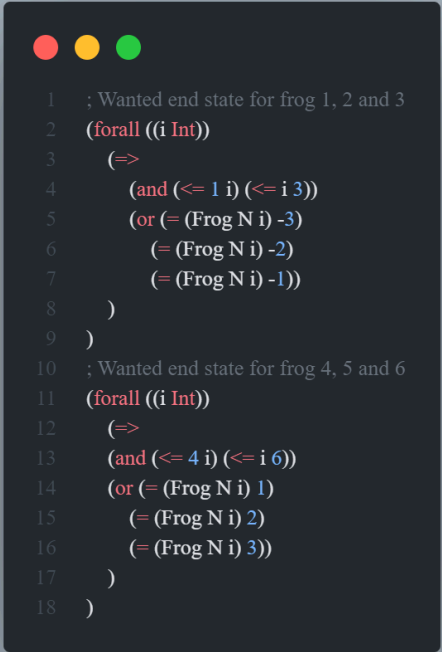
## Approach

Firstly as usual I looked for what my output was going to look like. For this as this was my first exercise with a Time component I looked at the water jugs exercise provided in the sharepoint. This gave me all the necessary information to complete this puzzle.

So then the puzzle “rules” needed to be defined

1. First at which time and position does each frog start
2. No 2 frogs can be on the same position
3. I choose a min value of -3 and a max of 3. Where each frog is positioned on -3, -2, -1, 1, 2, 3. This results in 0 being empty
4. Then I described all the positions each frog is allowed to be in.
5. Next I described that no frog can move backwards. I did this by comparing the previous position with the new position
6. 1 frog must move each N.
7. A frog can only move 0, 1 or 2 absolute positions for each N (this is so that frogs can move forward or backwards based on the way they are facing)

## Input



```
1 ; Wanted end state for frog 1, 2 and 3
2 (forall ((i Int))
3   (=>
4     (and (<= 1 i) (<= i 3))
5     (or (= (Frog N i) -3)
6         (= (Frog N i) -2)
7         (= (Frog N i) -1))
8   )
9 )
10 ; Wanted end state for frog 4, 5 and 6
11 (forall ((i Int))
12   (=>
13     (and (<= 4 i) (<= i 6))
14     (or (= (Frog N i) 1)
15         (= (Frog N i) 2)
16         (= (Frog N i) 3))
17   )
18 )
```

## Output

```
PS C:\Users\bassim\OneDrive\Documenten\GitHub\Academic-prep\Z3\Exercises> .\z3.exe .\frogs.smt
sat
((Frog 0 1) (- 3))
((Frog 0 2) (- 2))
((Frog 0 3) (- 1))
((Frog 0 4) 1)
((Frog 0 5) 2)
((Frog 0 6) 3))

((Frog 1 1) (- 3))
((Frog 1 2) (- 2))
((Frog 1 3) 0)
((Frog 1 4) 1)
((Frog 1 5) 2)
((Frog 1 6) 3))

((Frog 2 1) (- 3))
((Frog 2 2) (- 2))
((Frog 2 3) 0)
((Frog 2 4) (- 1))
((Frog 2 5) 2)
((Frog 2 6) 3))

((Frog 15 1) 1)
((Frog 15 2) 2)
((Frog 15 3) 3)
((Frog 15 4) (- 3))
((Frog 15 5) (- 2))
((Frog 15 6) (- 1))
```

## Slide puzzle (8 tiles)

Let Z3 solve the following variation of the famous 15 puzzle (where you slide tiles to the empty space on the grid). The initial situation is on the left, the wanted end situation on the right: (See [https://en.wikipedia.org/wiki/15\\_puzzle](https://en.wikipedia.org/wiki/15_puzzle), and <http://www.mathematischebasteleien.de/15puzzle.htm> ('The Eight Puzzle'))

### Approach

Again I started with looking at how the output was going to be. Here I found a close resemblance to the Frogs puzzle only the input and output were different with a couple of differing rules.

1. Define start and end position
2. No 2 numbers can be on the same position
3. Given how I numbered each square from 1 to 9 to match the wanted solution (At time=N number 4 must be at position 4) a number can only move left or right, up or down. This results in them being able to move 0, 1 or 3 absolute position (moving from 1 to 4 is going down)
4. The position minimum is 1 and the max is 9 (inclusive). This is so the numbers can not go out of the box
5. 1 Number must change position each N
6. Lastly I described how each number can move based on what position they previously were

## Input



```
1  (assert (and
2    ; Begin positions of numbers
3    (= (Number 0 8) 1)
4    (= (Number 0 7) 2)
5    (= (Number 0 6) 3)
6    (= (Number 0 5) 4)
7    (= (Number 0 4) 5)
8    (= (Number 0 3) 6)
9    (= (Number 0 2) 7)
10   (= (Number 0 1) 8)
11
12   ; Wanted end positions of numbers (N=16 found by 1 2 4)
13   (= (Number N 1) 1)
14   (= (Number N 2) 2)
15   (= (Number N 3) 3)
16   (= (Number N 4) 4)
17   (= (Number N 5) 5)
18   (= (Number N 6) 6)
19   (= (Number N 7) 7)
20   (= (Number N 8) 8)
```

## Output

This output is only for finding Number 4 and 5's end position. I think my code is not optimized as the biggest N I have found was 16 which took more than 15 minutes of calculating.

The output does follow all the rules of the slide puzzle.

```
PS C:\Users\bassm\OneDrive\Documenten\GitHub\Academic-prep\Z3\Exercises> .\z3.exe .\SlidePuzzle8.smt
sat
((N 11))
(( (Number 0 1) 8)
  ( (Number 0 2) 7)
  ( (Number 0 3) 6)
  ( (Number 0 4) 5)
  ( (Number 0 5) 4)
  ( (Number 0 6) 3)
  ( (Number 0 7) 2)
  ( (Number 0 8) 1))

(( (Number 1 1) 9)
  ( (Number 1 2) 7)
  ( (Number 1 3) 6)
  ( (Number 1 4) 5)
  ( (Number 1 5) 4)
  ( (Number 1 6) 3)
  ( (Number 1 7) 2)
  ( (Number 1 8) 1))

(( (Number N 1) 9)
  ( (Number N 2) 8)
  ( (Number N 3) 6)
  ( (Number N 4) 4)
  ( (Number N 5) 5)
  ( (Number N 6) 3)
  ( (Number N 7) 1)
  ( (Number N 8) 7))
```