# Logic and Computability WS18, Assignment 8

January 18, 2018

For the following examples, model the problems as SMT instances and solve it using the SMT solver Z3 from Microsoft research. You can either solve the examples directly in SMT2LIB format, or you can use the Python API. **This time, you have to upload the solution for each example you solve in Stics!**

**SMT-2.0:** If you use SMT-2.0, you can use the online version at `http://rise4fun.com/Z3`. Note however, that some tasks are computationally quite intensive, and that the online version will get a timeout rather quickly. Therefore, it is recommended to install the necessary Z3 packages through your package manager e.g. `apt`.

**Python API:** If you are familiar with Python, we highly recommend to use the Python API. If you do, install the python3 bindings through `pip`. The name of the python bindings package is `z3-solver`. For these exercises, you will develop your solutions in small python3 scripts. No imports besides `z3` are allowed.

57. Equation systems are an essential part of linear algebra and can be represented in form of SMT. Solve the following linear equation systems using Z3. Model all variables as Z3 integers.

(a) **[1 Point]**

$$\begin{aligned} -18w - 7x + 3y + 2z &= -137 \\ 7w - 18x + 15y + 20z &= 410 \\ 17w + x - 20y - 5z &= 389 \\ -2w + 2x + 2y + 6z &= -50 \end{aligned} \tag{1}$$

(b) **[1 Point]**

$$\begin{aligned} -9w - 19x + 4y - 8z &= -152 \\ -15w + 10x + 20y + 14z &= -115 \\ 2w - 20x + 20y - 12z &= 86 \\ -15w + 18x + 11y - 10z &= 0 \end{aligned} \tag{2}$$

(c) **[2 Points]**

$$\begin{aligned} 3u - 16v + 6w + 5x + 11y - 10z &= 91 \\ -10u + 14v - 12w + 19x + 13y + 6z &= 379 \\ -4u - 16v - 4x - 16y - 16z &= -100 \\ -8u + 17v - 8w - 12x - 14y + 8z &= -29 \\ 5u - 7v - 12w + 13x - y - 5z &= 124 \\ 4u - 10v + 8w - 8x + 19y + 4z &= 252 \end{aligned} \tag{3}$$

**Hints for Python:**

- You have to distinguish between Z3 integer values and python integers. To get a Z3 integer value from a python integer use something like `num = z3.IntVal(5)`. To convert the other way around use `num.as_long()`.
- Store your Z3 integers inside python values. When you have a Z3 model, store it with `m = solver.model()`, and extract the assigned values using something like `x_val = m[x].as_long()`.

- You can just create Z3 variables with the names x, y, z and so on and simply copy the equation systems into Z3 expressions that you add to the solver. You can also put all these values into a matrix and generate the expressions using two for loops. Either solution is fine.

58. Alice, bob and carol are siblings. The product of their ages is 72 and the sum of their ages is 14. Alice was born before bob and bob was born before carol. Using Z3, do the following:

    (a) **[2 Points]** Find one solution to the riddle and determine a set of possible ages of the three children.

    (b) **[1 Point]** Find a second solution (at least one of the children must have a different age). Do so by introducing a new constraint that excludes the previous solution from being considered valid.

    (c) **[1 Point]** Try to find a third solution by introducing another constraint that excludes all prior solutions.

    **Hints:**

    - Use Z3 integers to model the ages of the three children.
    - All of the children were already born. You have to ensure that your solution will never contain negative numbers as ages.

59. **[4 Points]** Given is the following C function which does a simple addition and assumes some preconditions.

```
// This function safely adds two positive numbers.
// The expected result is non-negative
int add(int a, int b)
{
  int c = 0;
  if (a > 0 && b > 0)
    c = a + b;
  assert(c >= 0);
  return c;
}
```

Model this function. First, model each `int` as a Z3 integer and then check whether the assertion can be violated. After that, model each `int` as a Z3 bitvector of length 32, the same way it is represented inside a modern computer. Discuss your findings and explain any interesting results.

**Hints:**

- Since you want to violate the assertion at the end of the function, you have to add an expression into the solver, which states the opposite.
- Since some variables can have different values at different points in the function, remember to do single static assignments.
- Phyton: Do not use the python `and` keyword for the conjunction, and instead use `z3.And`. Similarly, use the `z3.If` for the purpose of branching in your symbolic program.
- Python: On 32 and 64 bit machines, `int` is a 32 bit type. Use `z3.BitVec("name", 32)` to get a bitvector of length 32, with the name "name".

60. A magic square is a $n \times n$ grid of numbers, where each row, column and diagonal sums up to the same *magic number*. An example is shown below:

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

The magic number in this case is 15, because all rows, columns, and both diagonals add up to this number.

(a) **[2 Points]** Formulate all the constraints needed to make a $4 \times 4$ magic square and generate it. What can you notice about the generated magic square?

(b) [**1 Point**] Add a symmetry breaking constraint to the magic square. A good example of such a constraint is the following: "All numbers must be different". Observe the impact on the returned solution.

**Hints:**

- You should model all fields of the magic square as a `z3.Int`. You can either generate the symbolic sums over the rows and columns using two for loops, or just add them manually.
- Since you want all rows, columns and diagonals to add up to the same magic number, you can add a variable for the magic number, and assert that all sums equal this variable.
- To add the constraint that all fields are different, you can use `z3.Distinct` and pass it an array of symbolic variables. This will add all the inequality constraints for you.