Homework 2.

deadline: January 31

Suggested systems:
Problems 1,2: Prover9/Mace4 `https://www.cs.unm.edu/~mccune/prover9`
Problems 2,3,4: MiniZinc compiler & Gecode constraint solver
`http://www.minizinc.org/software.html`
The easiest option is to install the *Bundled binary package*, containing the compiler, an IDE, and several solvers including Gecode. Everything should be up and running out of the box.

Email solutions to `stanovsk@karlin.mff.cuni.cz`

Problem 1. (6)
The standard Łukasiewicz calculus for propositional logic is axiomatized by three axioms

$$a \to (b \to a),\ (a \to (b \to c)) \to ((a \to b) \to (a \to c)),\ (\neg a \to \neg b) \to (b \to a)$$

and uses modus ponens, i.e. $a,\ a \to b \vdash b$, as its inference rule.

(a) Use an automated theorem prover to show that the following formulas are provable: $a \to a,\ \neg a \to (a \to b),\ \neg\neg a \leftrightarrow a$,
(b) Use a model builder to show that the axioms are independent, that is, that any two of them do not imply the third one.

Hint: logical connectives will be function symbols, propositional formulas will be terms, one unary predicate will stand for provability. If you don't understand the problem, start here: `https://en.wikipedia.org/wiki/Hilbert_system` (or come to see me).
Submit all input and output files.

Problem 2. (6)
Find the smallest *non-medial latin quandle*, that is, an algebra $(A, *)$ with a single binary operation such that

- the multiplication table is a latin square (i.e., rows and columns are permutations),
- it satisfies the identity $x * (y * z) = (x * y) * (x * z)$,
- it *does not* satisfy the identity $(x * y) * (u * v) = (x * u) * (y * v)$.

Implement the problem as a CSP instance, and as a model building instance. Compare the running times.
This is one of my favourite benchmarks. The smallest model is not so small (> 10 elements) and it takes a while to find it (usually a few minutes).
Submit input files and models (e.g., the Mace4 output and the `.mzn` file) and tell me the running times.

Problem 3. (6)

A *magic square* is an $n \times n$ table filled with positive integers $1, 2, ..., n^2$ such that each cell contains a different integer and the sum of the integers in each row, each column, and both diagonals is equal. The sum is called the *magic constant*; n is the *order* of the magic square.

The input data consist of the order and a partially filled square. Write a MiniZinc model that will fill out the rest of the square (if it is possible) and compute the magic constant. Here is a sample input:

```
N = 3;
square =
[| 2, _, _
 | _, 5, _
 | _, 3, _
 |];
```

The output should be something like this:

```
The magic constant is 15.
The magic square:
2 7 6
9 5 1
4 3 8
```

You can use the following code to get the output in a nice format (`magic_constant` is the name of the variable used to compute the magic constant):

```
output
["The magic constant is \(magic_constant).\n"] ++
["The magic square:\n"] ++
[ show_int(floor(log10(int2float(N*N))+1), square[i,j]) ++
  if j = N then "\n" else " " endif | i, j in 1..N];
```

The input configurations are in a separate ZIP file. Submit models for each of the input files.

Problem 4. (6)

A very hungry hiker is buying provisions for a backpacking trip. He is choosing from a list of food items. The supply is unlimited. Each item has a certain amount of calories (in kcal) and a certain weight (in grams). He can only carry food up to a given weight limit. What is the maximum amount of calories he can take?

Here is a sample input.

```
LIMIT = 6500;
ITEMS = {apple, beer, bread, carrot, pea, steak, water};
WEIGHT = [88, 564, 892, 415, 8, 410, 500];
KCAL = [40, 385, 615, 290, 1, 245, 5];
```

The optimal solution is 1 beer, 2 loaves of bread, and 10 carrots which sums up to 4515 kcal. Note that `ITEMS` is of type `enum`, declared by ‘`enum ITEMS;`’.

The input configurations are in a separate ZIP file. Submit models for each of the input files.