# Propositional Satisfiability

This exercise sheet is inspired from exercises given in the lecture of Pascale Le Gall and Marc Aiguier at CentraleSupélec and the lecture of Markus Iser, Dominik Schreiber, and Tomas Balyo at KIT.

**Exercise 1** (Queens)**.**

The $n$-Queens problem consists of placing $n$ queens on an $n \times n$ chessboard such that no two queens threaten each other. This means that no two queens can be in the same row, column, or diagonal. The problem can be represented as a propositional satisfiability (SAT) problem.

1. Propose a CNF encoding of the $n$-Queens problem, i.e., a CNF formula is satisfiable if and only if there exists a solution to the $n$-Queens problem.

Here are some hints to help you with the encoding.

**Variables** Use a propositional variable $Q_{i,j}$ which is true if the queen is placed in row $i$ and column $j$. For the $n$-Queens problem, there should be $n^2$ variables.

**Constraints** For each row, ensure that exactly one queen is placed. This can be done by ensuring that at least one queen is placed in each row and at most one queen is placed in each row (you can use the encoding of the at-most-one constraint). You need to do the same for each column, and for each diagonal.

**DIMACS Encoding** You need to output the CNF formula in DIMACS format. The first line should contain the number of variables and the number of clauses. The second line should contain the clauses, where each clause is represented as a list of integers (the variable numbers) terminated by 0. Each clause should be on a separate line. Each variable needs to be represented by an integer, for example, $Q_{i,j}$ can be represented by the integer $(i-1) \times n + j$.

2. Write a program that takes as input the size of the chessboard $n$ and outputs the CNF formula in DIMACS format. The program should be able to handle any size of the chessboard.
3. Integrate a SAT solver to solve the $n$-Queens problem. The program should output the solution in a human-readable format, i.e., the positions of the queens on the chessboard.

**Exercise 2** (Sudoku)**.**

The Sudoku puzzle consists of a $9 \times 9$ grid divided into $3 \times 3$ subgrids. The goal is to fill the grid with numbers from 1 to 9 such that each number appears exactly once in each row, column, and subgrid.

1. Propose a CNF encoding of the Sudoku problem, i.e., a CNF formula is satisfiable if and only if there exists a solution to the Sudoku problem.

Here are some hints to help you with the encoding.

**Variables** Use a propositional variable $S_{i,j,k}$ which is true if the number $k$ is placed in row $i$ and column $j$. For the Sudoku problem, there should be $9^3$ variables.

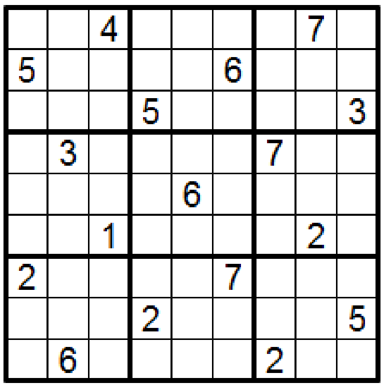

Figure 1: A Sudoku.

**Constraints** For each row, ensure that exactly one number is placed. This can be done by ensuring that at least one number is placed in each row and at most one number is placed in each row (you can use the encoding of the at-most-one constraint). You need to do the same for each column and for each subgrid.

**DIMACS Encoding** You need to output the CNF formula in DIMACS format. The first line should contain the number of variables and the number of clauses. The second line should contain the clauses, where each clause is represented as a list of integers (the variable numbers) terminated by 0. Each clause should be on a separate line. Each variable needs to be represented by an integer, for example, $S_{i,j,k}$ can be represented by the integer $(i-1)\times 9^2 + (j-1)\times 9 + k$.
Here is an example of a sudoku puzzle is given in Figure 1. The numbers in the cells are the initial values of the Sudoku puzzle. The sudoku in the image can be written as follows:

```
0 0 4 0 0 0 0 7 0
5 0 0 0 0 6 0 0 0
0 0 0 5 0 0 0 0 3
0 3 0 0 0 0 7 0 0
0 0 0 0 6 0 0 0 0
0 1 0 0 0 0 0 2 0
2 0 0 0 0 7 0 0 0
0 0 0 2 0 0 0 0 5
0 6 0 0 0 0 2 0 0
```

2. Write a program that takes as input the Sudoku puzzle (with some cells already filled) and outputs the CNF formula in DIMACS format. The program should be able to handle any valid Sudoku puzzle.
3. Integrate a SAT solver to solve the Sudoku puzzle. The program should output the solution in a human-readable format, i.e., the filled Sudoku grid.

**Exercise 3** (DPLL).

The DPLL algorithm is a backtracking algorithm for solving the SAT problem. It is based on the idea of unit propagation and pure literal elimination.

1. Implement the DPLL algorithm in a programming language of your choice. The implementation should take as input a CNF formula in DIMACS format and output whether the formula is satisfiable or not.

2. Test your implementation on various SAT instances, including the $n$-Queens problem and the Sudoku problem. Compare the performance of your implementation with existing SAT solvers.

**Exercise 4** (Pentomino)**.**

Develop and implement a program to encode the Pentomino Tiling puzzle in a programming language of your choice. The Pentomino Tiling puzzle involves arranging a set of pieces to completely fill a rectangular block of specified dimensions. There are 12 types of pieces: L, T, V, Z, N, F, X, W, P, I, Y, and U, which can form 18 unique shapes through rotation. Figure 2 illustrates the 12 types of pieces.

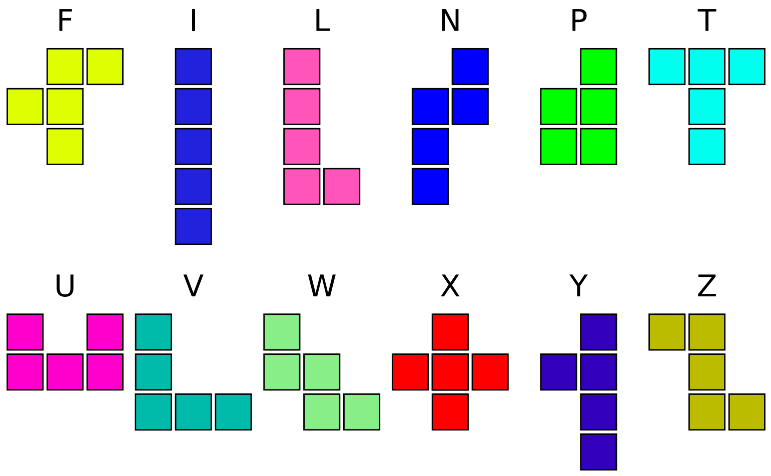

Figure 2: The 12 pentomino pieces.

The task is to create an encoder that accepts 14 inputs (block height, block width, and the counts of L, T, V, Z, N, F, X, W, P, I, Y, and U pieces) and outputs a CNF formula. This formula should be satisfiable if and only if the given Pentomino Tiling puzzle instance has a solution.

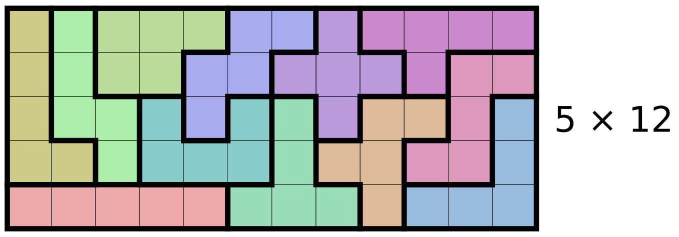

Figure 3: An example solution to the Pentomino Tiling puzzle.

Figure 3 provides an example solution to the Pentomino Tiling puzzle, where one of each of the 12 pieces is arranged into a 5 × 12 block. This corresponds to the 14 inputs: “5 12 1 1 1 1 1 1 1 1 1 1 1 1.”
An example of an unsolvable instance is “4 15 1 0 0 1 1 0 0 0 0 0 0 0 0 0,” where the block cannot be filled due to insufficient pieces.