This document is a description of the two Sudoku solvers built for this project. The first part details the Z3 theorem prover based solution, whilst the second details the implementation of a chronological backtracking based algorithm.

For a detailed description about how to run the tools, please read the `README.md` file.

The input

The solvers take as input a file containing:

```
n
XXX  XXX  XXX
XXX  XXX  XXX
XXX  XXX  XXX

XXX  XXX  XXX
XXX  XXX  XXX
XXX  XXX  XXX

XXX  XXX  XXX
XXX  XXX  XXX
XXX  XXX  XXX
```

- Where `n` is the size of the grid (`n`×`n`). `n` can be 4, 9, 16, or 25
- Whitespace and blank lines are ignored
- `X` can be `_` or `-` or `.` for a square without value given initially, or a hexadecimal digit (`123456789ABCDEF...P`) up to `n`. It can be uppercase or lowercase

1 Z3 based solver

Our goal here is to use the Z3 API in Python to solve a given Sudoku problem.

1.1 Data structures

- **Arrays**: used to initially store the Sudoku grid read as a file given in the first argument of the program. Also used to describe the solution as a vector of Z3 variables, as follows
- After parsing the input

For this purpose, we create a solver object and add to it the following clauses:

- Distinct values in each line
- Distinct values in each column
- sol values must be natural numbers contained within the interval [1,n]
- Specify initial values of the sudoku problem as defined in the grid representing the problem
- Ensure uniqueness of values in each box

Then, we check the satisfiablity of the resulting formula and print the solution.

2 Backtracking based search engine

The following is the algorithm.

```
Input: Sudoku problem
Result: Sudoku solution
Fill in the trivial cells in the grid of size n
pos := ∅;
pre := ∅;
i := 0;
j := 0;
return FIND_SOLUTION(grid, n, i, j, pos, pre)
```

```
1: function FIND_SOLUTION(grid, n, i, j, pos, pre)
       if grid[i][j] == 0 then
       if cell encountered for the first time then
           pre.add(i,j) get a valid digit for this cell
           if no digit is possible then
               tofo
           end
       end
       end
2: end function
```

Algorithm 1: Backtracking algorithm based Sudoku solver

Comparison

TODO: express the downsides and upsides of my implementation. Have some graph to sythesize some benchmarking testing.