## Tutorials for Decision Procedures
### Exercise sheet 2

### Exercise 1: DPLL

Execute the DPLL-algorithm (or CDCL, if you like) from the lecture on the following
clause set. Write down what you do, in particular give the resulting clause set after every
step. Assume that the algorithm starts with setting the variable $P_1$.

$$\{\{\overline{P_1}, \overline{P_7}\}, \{P_1, P_7\}, \{P_1, \overline{P_3}\}, \{\overline{P_1}, P_3, P_9\}, \{\overline{P_2}, \overline{P_4}, P_9\}, \{\overline{P_2}, \overline{P_5}\}, \{P_2, P_5\},$$
$$\{P_3, P_6, P_8\}, \{\overline{P_3}, \overline{P_9}\}, \{\overline{P_3}, P_6\}, \{P_3, \overline{P_6}, P_7\}, \{\overline{P_3}, \overline{P_8}\}, \{\overline{P_3}, P_8\}, \{P_4, \overline{P_5}\},$$
$$\{\overline{P_4}, \overline{P_6}, \overline{P_9}\}, \{P_4, \overline{P_7}\}, \{\overline{P_4}, P_7\}, \{P_4, P_9\}, \{P_6, \overline{P_7}\}, \{P_6, P_9\}\}$$

### Exercise 2: SMT-LIBv2

SMT-LIBv2 is a standard for describing logical formulae in many first-order theories
which is read by several modern SMT-solvers. On the lecture-website there is a com-
mented example script encoding a boolean formula.

Use the example script to learn the most basic SMT-LIBv2 commands and write your
own script which describes the knights and knaves problem that we saw in the lecture on
propositional logic.

Use an SMT-LIBv2 compliant SMT-solver (e. g. Z3 or SMTInterpol which are linked at
the lecture's website) to check the satisfiability of the problem and, in case of a positive
answer, retrieve a fulfilling valuation.

(We can use a SMT-Solver instead of a SAT-Solver because propositional logic is a subset
of any relevant fragment of first-order logic.)

### Exercise 3: Sudoku Generator

A sudoku is a $n^2 \times n^2$ matrix whose elements elements are labelled
by numbers from 1 to $n^2$. The figure on the right shows an example
for $n = 3$. Every row and every column contains each number. Ad-
ditionally every $n \times n$-submatrix (there are $n^2$ of this) contains each
number.

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

- Encode the sudoku-rules into boolean contraints. Tipp: Introduce $n^2$ boolean vari-
ables per field (that means you need $n^6$ variables overall). For instance boolean
variable `v_x_y_i` should have the semantics "the field at position (x,y) is filled out
with number i".

- Write a program (in a not-too-exotic language of your choice) which takes an integer $n$ as argument and generates a SMT-LIBv2 -file with all necessary constraints for a Sudoku of size $n$.

- Use a SMT-LIBv2 compliant SMT-solver to generate Sudokus.

- Experiment with different numbers for $n$ and measure the run-time of your program and of the solver.

(Please submit your program source code and a summary of your experiments.)