Algorithm Engineering **Summer 2026**

Dr. Phillip Keldenich
Dr. Dominik Krupke

Submission due: 02.07.2026 16:45

Homework 5

Work in teams of two. Homework is due every two weeks.

Submission format: Host your solution in a private Git repository on one of the following platforms and add Rouven as a collaborator: IBR GitLab (`kniep`), TU GitLab (`r.kniep`), GitHub (`rkniep`), or Codeberg (`rkniep`). If you want to use a different platform, please ask first.

Before the deadline, send the link to your submission commit by e-mail to `kniep@ibr.cs.tu-bs.de`.

Exercise 1 (SAT Modeling: n Queens): **(5 + 10 + 5 points)**

The n queens problem is a well-known classic puzzle: position n queens on an otherwise empty $n \times n$ chess board such that no queen can immediately capture another queen. Recall that queens in chess can move arbitrarily far horizontally, vertically and diagonally, so in the n queens problem, no two queens can be placed in the same row, column or diagonal.

a) Model the problem as SAT formula in CNF. Hint: you do not need any advanced cardinality constraints.

b) Implement your model using PySAT, i.e., fill out the implementation skeleton in the `exercise01` subdirectory. There is a suite of test cases you can run via `pytest` to check your implementation. The skeleton implementation defines two script entrypoints (`queens-sat`, `queens-cpsat`) that both take n as argument as well as an optional `--time-limit`. What is the largest n you can solve with your implementation within 5 minutes of running time?

c) Compare this model to a CP-SAT model in ortools CP-SAT, making use of the constraints CP-SAT supports; in particular, at-most-one and exactly-one constraints allow a more compact CP-SAT model compared to the SAT formulation in CNF. Which one can solve larger n?

Exercise 2 (SAT Preprocessing Rules): **(10 + 10 + 10 points)**

This exercise is theoretical (rather than a practical implementation task): we want to explore a few of the most impactful SAT pre- and inprocessing techniques and argue why they are correct.

a) It is often stated in the SAT literature that bounded variable elimination (BVE) is the single most impactful SAT pre- and inprocessing technique. A single variable x can be eliminated by resolving all clauses of the form $x \vee A$ with all clauses of the form $\overline{x} \vee B$. Tautological resolvents (i.e., resulting clauses containing $y \vee \overline{y}$ for some y) are dropped. Afterwards, all original clauses containing x or $\overline{x}$ are dropped

and the variable x is removed. While in principle, every variable can be removed in this manner, in practice we may roughly square the size of the formula for each eliminated variable, which is prohibitively expensive; therefore, in practice, such eliminations are usually only done if the number of clauses goes down (or only goes up by a small constant g) in a single elimination step.

Show that this approach is correct: show that the formula φ' resulting from eliminating a variable x from the original formula φ is satisfiable if and only if φ is satisfiable.

b) Vivification describes the process of strengthening or deleting clauses in the following way. We take the literals $\ell_1 \vee \cdots \vee \ell_k$ of a clause C. Starting from a trail at level 0 (which contains only fixed literals and no decisions), we then propagate the consequences of setting the ℓ_j to false one by one in some arbitrary order (ignoring clause C during propagation). In other words, we first decide $\overline{\ell_1}$ and propagate, then add a second decision for $\overline{\ell_2}$ and propagate, and so on, until either of the following happens: (1) deciding and propagating $\overline{\ell_1} \wedge \cdots \wedge \overline{\ell_j}$ propagates one of the later literals $\ell_i \in C$, $k \geq i > j$ to true, (2) deciding and propagating $\overline{\ell_1} \wedge \cdots \wedge \overline{\ell_j}$ propagates one of the later literals $\ell_i \in C$, $k \geq i > j$ to false, (3) deciding and propagating $\overline{\ell_1} \wedge \cdots \wedge \overline{\ell_j}$ causes a conflict, (4) we have introduced all k decisions.

Show that, in each of the cases (1)–(3), the formula can be simplified by either strengthening or removing the clause C.

c) Blocked Clause Elimination (BCE) is a simplification method that deletes clauses that are *blocked on a literal* ℓ. It can be helpful to speed up propagation and to enable BVE to eliminate more variables. A clause $C = \ell \vee a_1 \vee \cdots \vee a_k$ is blocked on a literal ℓ if, for each clause D containing $\overline{\ell}$, the resolution $C \diamond_\ell D$ is a tautology, i.e., $C \diamond_\ell D$ contains $y \vee \overline{y}$ for some variable y.

Show that BCE is correct, i.e., that a formula φ is satisfiable iff $\varphi \setminus \{C\}$ is satisfiable for a blocked clause C.

Note: like with BVE, the resulting formula is only equisatisfiable, not equivalent; it thus may be necessary to change the satisfying assignment for $\varphi \setminus \{C\}$ to obtain a satisfying assignment for φ.