# Assignment8 – Reasoning for Propositional Logic

**Problem 8.1 (Calculi Comparison)**

*Prove* (or disprove) the validity of the following *formulae* in i) *Natural Deduction* ii) *Tableau* and iii) *Resolution*:

1. $P \wedge Q \Rightarrow (P \vee Q)$
2. $(A \vee B) \wedge (A \Rightarrow C) \wedge (B \Rightarrow C) \Rightarrow C$
3. $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$
4. Can you identify any advantages or disadvantage of the *calculi*, and in which situations?

**Problem 8.2 (Equivalence of CSP and SAT)**

We consider

- *CSPs* $\langle V, D, C\rangle$ with *finite domains* as before
- *Boolean satisfiability problems* $\langle V, A\rangle$ where $V$ is a set of propositional variables and $A$ is a propositional formula over $V$.

We will show that these problem classes are equivalent by reducing their instances to each other.

1. Given a **SAT** *instance* $P = \langle V, A\rangle$, define a CSP instance $P' = \langle V', D', C'\rangle$ and two *bijections*:
   - $f$ mapping satisfying assignments of $P$ to solutions of $P'$,
   - and $f'$ the inverse of $f$.

   We already know that *constraint networks* are equivalent to *higher-order CSPs*. Therefore, it is sufficient to give a *higher-order CSP*.
2. Given a CSP instance $\langle V, D, C\rangle$, define a **SAT** *instance* $(V', A')$ and *bijections* as above.

**Problem 8.3 (Satisfiability and Validity)**

Consider propositional logic with propositional variables $\{P, Q, R\}$. For each of the following statements, give a counter-example that refutes it:

1. The formula $(P \wedge Q \vee Q \wedge R) \Rightarrow (\neg P \vee \neg R)$ is satisfied by all assignments.
2. If a formula $F$ cannot be proved in the natural deduction calculus, then $\neg F$ is valid.
3. If, for two formulas $F$, $G$, all assignments satisfy $F \Rightarrow G$ and no assignment satisfies $F$, then no assignment satisfies $G$.

**Problem 8.4 (SAT Solver)**

We want to implement a basic SAT solver. To simplify the I/O, we assume the following:

- The propositional variables are numbered $P_1, P_2, ....$
- The input formula $A$ is already in CNF and represented as a list of lists of integers such that:
  - the main list $[C_1, ..., C_m]$ represents $A$ as a list of conjuncts, i.e., $A = C_1 \wedge ... \wedge C_m$.
  - each $C_i = [L_1, ..., L_{n_i}]$ represents one of the disjunctions, i.e., $C_i = L_1 \vee ... \vee L_{n_i}$
  - each integer $L$ represent a literal, with negative numbers representing negations.

For example, $[[1, 2, -3], [2, -5]]$ represents the formula $(P_1 \vee P_2 \vee \neg P_3) \wedge (P_2 \vee \neg P_5)$.

1. Implement a naive SAT solver, e.g., by checking all assignments.
   Run an experiment where you generate large formulas with many variables and test how well your implementation scales. Think about what kind of formulas make your code take the longest and use those, e.g., unsatisfiable formulas take longer if they require testing all assignments.
2. Implement optimizations and repeat your experiments.
   For example, you can implement DPLL or clause learning.