1 The Pigeon Hole Principle

For every natural number \( n \geq 2 \), the Pigeon Hole Principle (PHP) states: “If \( n \) pigeons sit in \((n - 1)\) holes, then some hole contains more than one pigeon.” We want to formalize PHP in propositional logic (PL). There are different ways of doing this, but perhaps the most natural is:

- Use propositional atom \( P_{i,j} \) to indicate that pigeon \( i \) is in hole \( j \), where \( 1 \leq i \leq n \) and \( 1 \leq j < n \).

With this formal representation of “pigeon \( i \) is in hole \( j \)” we can formalize PHP with the following PL formula \( \varphi \):

\[
\varphi \equiv \bigwedge_{1 \leq i \leq n} \left( \bigvee_{1 \leq j < n} P_{i,j} \right) \rightarrow \bigvee_{1 \leq i < k \leq n} \left( \bigvee_{1 \leq j < n} (P_{i,j} \land P_{k,j}) \right)
\]

where \( \bigwedge \) and \( \bigvee \) are shorthand notation to write long sequences of conjunctions and disjunctions, respectively. In particular, for the case \( n = 3 \), we get:

\[
\varphi = (P_{1,1} \lor P_{1,2}) \land (P_{2,1} \lor P_{2,2}) \land (P_{3,1} \lor P_{3,2}) \rightarrow \\
(P_{1,1} \land P_{2,1}) \lor (P_{1,2} \land P_{2,2}) \lor (P_{1,1} \land P_{3,1}) \lor (P_{1,2} \land P_{3,2}) \lor (P_{2,1} \land P_{3,1}) \lor (P_{2,2} \land P_{3,2})
\]

**Exercise 1** Let \( f \) be a total function from \( \{1, \ldots, n\} \) to \( \{1, \ldots, n - 1\} \), and use the same propositional atoms \( P_{i,j} \) used in the formalization of PHP, to define PL formulas \( \varphi_1 \) and \( \varphi_2 \) such that:

1. \( \varphi_1 \) is satisfiable iff \( f \) is a total (possibly multivalued) function on \( \{1, \ldots, n\} \),
2. \( \varphi_2 \) is satisfiable iff \( f \) is not one-to-one.

Another way of formalizing PHP in PL is to define the formula \( \varphi' \equiv \varphi_1 \rightarrow \varphi_2 \).

**Exercise 2** This is an implementation exercise. The WFF \( \varphi \) that formalizes PHP is not only satisfiable, but also valid (or a tautology), i.e., all valuations of the atoms \( P_{i,j} \) should satisfy \( \varphi \). Use Isabelle, or any automated proof-assistant of your choice, to establish that \( \varphi \) is valid. Do the implementation for at least two cases, \( n = 3 \) and \( n = 4 \). Do you notice any difference in the execution times? How would you handle the case \( n = 10 \)?