Logic for Computer Science

Project Report

Alexandru Dobre

November 30, 2025

1. Introduction

This report is about solving the knight's tour problem using a SAT solver.

We define s as the step the knight is in. For instance, when the knight is in the initial position (i_0, j_0), $s = 0$. We define the boolean variable $x_{s,i,j}$ which is True iff the knight is in cell (i, j) at step s. The indices range as follows: steps $s \in [0, M \times N - 1]$, rows $i \in [0, M - 1]$, and columns $j \in [0, N - 1]$.

2. First Question

We encode the problem using the following constraints:

a) At step $s = 0$, the knight must be at the given position (i_0, j_0).

$$x_{0,i_0,j_0}$$

b) For every step s, the knight must be in exactly one cell. We split this into two parts:
- At least one cell:

$$\bigwedge_{s} \left(\bigvee_{i,j} x_{s,i,j} \right)$$

- At most one cell: For every pair of distinct cells, the knight cannot be in both.

$$\bigwedge_{s} \left(\bigwedge_{(i,j) \neq (i',j')} \left(\neg x_{s,i,j} \vee \neg x_{s,i',j'} \right) \right)$$

c) The knight can only move according to chess rules. We split this into two parts:
- For every step $s < M \times N - 1$, if the knight is at (i, j), it must move to a valid cell at step $s + 1$.

$$\bigwedge_{s=0}^{M \times N - 1} \bigwedge_{i,j} \left(\neg x_{s,i,j} \vee \bigvee_{\text{Valid}(i',j')} x_{s+1,i',j'} \right)$$

- For every step $s > 0$, the knight must have been in a valid cell at step $s - 1$.

$$\bigwedge_{s=1}^{M \times N} \bigwedge_{i,j} \left(\neg x_{s,i,j} \vee \bigvee_{\text{Valid}(i',j')} x_{s-1,i',j'} \right)$$

d) Every cell (i, j) must be visited at exactly one time step.
- At least once:

$$\bigwedge_{i,j}\left(\bigvee_{s} x_{s,i,j}\right)$$

- At most once: For any cell, it cannot be visited at two different steps s and s'.

$$\bigwedge_{i,j}\left(\bigwedge_{s\neq s'}\left(\neg x_{s,i,j} \vee \neg x_{s',i,j}\right)\right)$$

3. Second Question

4. Third Question

For each starting position (i_0, j_0), we execute `question1()`. For each valid solution found from (i_0, j_0), `nb_sol` is incremented and the solution is made insatisfiable by negating every (s, i, j) that validates it. Other solutions from the same (i_0, j_0) starting position are then searched with these constraints.

5. Fourth Question

The idea is to iterate through all starting positions (i, j) on the board and use `question1(3, 4, i, j)` for each starting position. For each found solution, the 3 symetries are created and stored in a set to exclude possible duplicates.

6. Fifth Question

We start at starting position (i_0, j_0) and firstly test if this starting position outputs a solution by executing `question1()`. If so, we test the uniqueness of the solution. If the solution is unique, we only return $(0, i_0, j_0)$. If there are two or more solutions,