An Overview of *Light Up* Agent

R. Benson, R. Verma, M. Voyonovich

December 2024

1 Automated Reasoner

The automated reasoning toolkit we used is called Z3-Solver [1]. Z3 accepts logical formulas in a variety of input languages, including SMT-LIB, a standard format for expressing satisfiability modulo theories (SMT) problems. Formulas can involve variables, logical operators (AND, OR, NOT), comparisons (equalities, inequalities), and more complex constraints such as bit vectors, arrays, and real numbers.

The Z3 solver works by trying to determine if a given logical formula is satisfiable (that is, whether there is an assignment of values to variables that makes the formula true) or unjustifiable (i.e., no such assignment exists). It does this in the following steps:

- Simplifies the input formula, removing redudant parts
- Takes in constraints (more commonly referred to as theories)
  - Linear Arithmetic
  - Bit-vectors
  - Arrays
  - Abstract Function
- The solver then applies a conflict-driven algorithm to solve the provided formula. CDCL algorithms explore different variable assignments, and when a conflict is detected– it learns from those conflicts to avoid repeating the same mistakes.

  If a conflict were to arise, Z3 uses an incremental SAT solver for solving any propositional logic components.

The main motivation to use Z3 was its extensive documentation and large user community.

2 Application

The Z3 solver is used to model the puzzle as a system of constraints. Each cell in the grid is associated with a boolean variable representing whether a light is placed in that cell. These boolean variables are defined as follows:

$$\text{light}_{i,j} \in \{\text{True}, \text{False}\}$$

Where:

- True means a light is placed at cell (i, j),
- False means no light is placed at that cell.

Z3 will try to find an assignment of these boolean variables that satisfies all the constraints of the puzzle.

3 Theorems

The following types of constraints are added to the Z3 solver:

3.1 Illumination Theorems

Each white cell (-1) must be illuminated by at least one light placed horizontally or vertically to it. This constraint is expressed as follows:

$$\forall (i, j) \text{ such that grid}[i][j] = -1, \quad \exists (l_i, l_j) \in \text{ cells of } (i, j), \quad \text{light}_{l_i, l_j} = \text{True}$$

The solver adds a disjunction of all horizontal or vertical positions that can light up each white cell:

$$\text{solver.add(Or(illumination conditions))}$$

Here illumination conditions are the Boolean variables of adjacent cells that contain lights.

3.2 Numbered Cell Theorems

For each numbered black cell (cells marked with a number between 0 and 4), the solver ensures that the number of adjacent cells containing lights equals the number specified in the grid. This is modeled using the Partial Boolean Equality (PbEq) function:

$$\text{solver.add(PbEq}(\{(\text{light}_{n_i, n_j}, 1) \text{ for each adjacent cell } (n_i, n_j)\}, \text{grid}[i][j]))$$

Here PbEq ensures that the number of adjacent lights matches the number in the grid.

3.3 Black Cell Constraints

Black cells (any cell that is not -1) cannot contain lights, so the solver adds the constraint:

$$\forall (i, j) \text{ such that grid}[i][j] \neq -1 \quad \text{or} \quad \text{grid[i][j]} \geq 0, \quad \text{light}_{i,j} = \text{False}$$

This prevents any lights from being placed in black cells.

3.4 No Conflicting Lights Theorems

The solver ensures that no two lights can "see" each other. This means that lights must not be placed in such a way that they illuminate each other in a straight line (horizontally or vertically). The constraint is the following:

$$\begin{aligned}
&\forall (i, j), (l_i, l_j) \text{ such that } (i, j) \neq (l_i, l_j), \\
&\text{if the lights at } (i, j) \text{ and } (l_i, l_j) \text{ can illuminate each other,} \\
&\neg(\text{light}_{i,j} \wedge \text{light}_{l_i, l_j})
\end{aligned}$$

This ensures that no two lights can see each other, which is a critical part of the puzzle's rules.

4 Solving the Puzzle

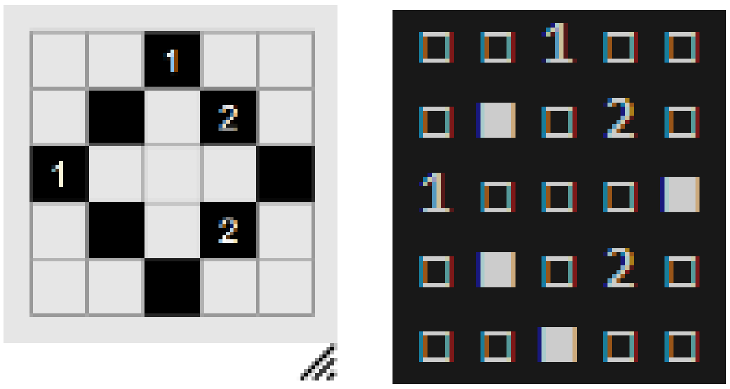
Once all the constraints have been added, the solver checks if a solution exists. The solver uses the `check()` function to determine if there is a satisfying assignment for all the boolean variables. If the solution exists, the solver returns a model that assigns values (True or False) to each $\text{light}_{i,j}$.

$$\text{if solver.check() == sat:}$$

If the solution is satisfiable, the model is extracted and used to build the solution grid, where cells with a light are marked as True and cells without a light are marked as False. This solution is then displayed visually by printing the grid.

5 Walkthrough

Consider a small 5x5 grid as an example:



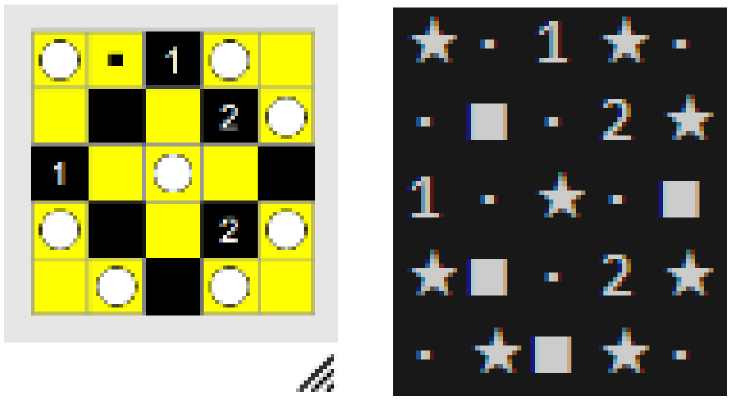
(a) Example grid on Tatham's [2] Website

(b) Example grid transcribed onto our website

The solver will:

- Ensure that the white cells (-1) are illuminated by at least one light.
- Ensure that the number of adjacent lights around numbered black cells corresponds to the number specified in those cells.
- Ensure that no lights are placed in black cells (any cell that is not -1).
- Ensure that lights do not "see" each other.

The solution to the puzzle would be a configuration of lights that satisfies all the constraints.



(a) First Image

(b) Second Image

Figure 2: Tatham's Solver [2] verifying Automated Reasoner

References

Documentation for Online Z3 Guide (tech. rep.). (2024). Microsoft Research.

Tatham, S. (2024). "*Light Up*" [Accessed: 2024-12-11].