Pips Solver

Nicholas Hernandez Kai Gowers

CSCI3392/MATH4312: Fall 2025

Abstract

This project presents a complete automated solver for Pips, a numerical logic puzzle in which dominoes must be placed onto an irregular grid so that each region of the board satisfies specific arithmetic constraints. We convert the visual puzzle layout into a formal constraint satisfaction model by extracting all valid cells, identifying adjacencies, encoding domino pieces, and translating region rules into logical constraints. Using the Z3 solver, we express domino placement, non-overlap conditions, and regional arithmetic requirements as Boolean and integer constraints. Z3 then computes a satisfying assignment that uniquely specifies where each domino must go and what value occupies each cell. The solver works for arbitrarily shaped boards and complex region structures, and includes visualization tools for rendering solutions on the original grid layout. This approach demonstrates how Pips can be solved exactly and efficiently using modern constraint-solving techniques.

1 ”Pips” Overview

NYT Pips is a puzzle where you place dominoes on a weirdly-shaped grid. Each square ends up with a number (a “pip”) from one side of a domino. Some groups of squares (regions) have constraints like:

- “sum of this region is 4”
- “all numbers in this region are equal”
- “sum of this region is less than 3”

The challenge is to place every domino exactly once, satisfying both the adjacency rules and regional constraints. Our solver turns this into a constraint satisfaction problem and solves it automatically using the Z3 SMT solver.

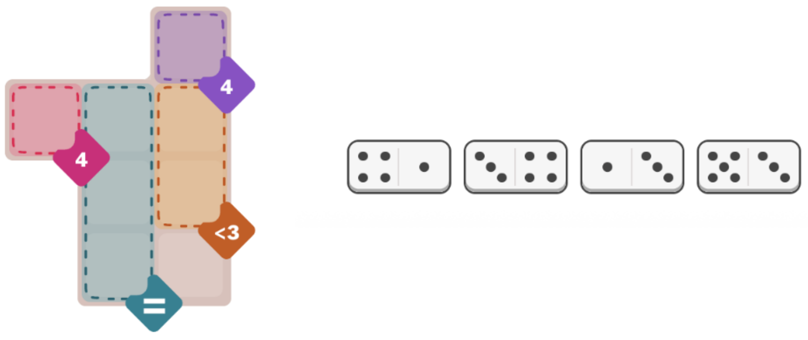

Figure 1: NYT Pips Easy Example

2 Board Representation

2.1 Board

The puzzle board is represented as a 2D array in which valid cells hold temporary placeholders (e.g. numbers such as 0,1,2,...), while empty spaces contain the token "-1".

```
board = [
    [-1, -1, 0],
    [1, 2, 3],
    [-1, 4, 5],
    [-1, 6, 7]
]
```

2.2 Edges

Two cells are adjacent if they are orthogonally next to each other (up, down, left, right). From this, we construct the edge list:

$$\text{edges} = \{(u, v) \mid u \text{ and } v \text{ are adjacent valid cells}\}.$$

```
edges = [(0, 3), (1, 2), (2, 3), (2, 4), (3, 5), (4, 5), (4, 6), (5, 7), (6, 7)]
```

3 Domino Encoding

Each domino is a pair (a, b) representing the pip values on its two halves. We index dominos as $d = 0, 1, \ldots, D - 1$. Each domino may appear on any edge and in either orientation.

$$\text{dominos} = [(a_0, b_0), (a_1, b_1), \ldots].$$

```
dominos = [(4,1), (3,4), (1,3), (5,3)]
```

4 Region Constraints

Regions are represented as tuples:

$$(\text{cells}, \ \text{op}, \ \text{target}).$$

The operator may be:

$$\text{sum\_eq}, \ \text{sum\_lt}, \ \text{sum\_gt}, \ \text{all\_eq}, \ \text{all\_diff}.$$

```
regions = [
    ([1], "sum_eq", 4),
    ([2,4,6], "all_eq", None),
    ([0], "sum_eq", 4),
    ([3,5], "sum_lt", 3)
]
```

5 Z3 Encoding

To solve the puzzle, we translate the board into a collection of logical variables and constraints evaluated by the Z3 solver. The encoding enforces three key ideas:

(a) every domino must be placed exactly once

(b) every cell must be covered by exactly one domino-side

(c) the number placed in each cell must agree with the pip value of the domino that covers it

5.1 Placement Variables

For each domino d, each edge e of the cell-adjacency graph, and each orientation $o \in \{0, 1\}$ (normal or reversed), we create a Boolean variable

$$\text{place}_{d,e,o}.$$

It is true exactly when domino d is placed on edge e with orientation o.

5.2 Cell Value Variables

For each cell c of the board we introduce an integer variable

$$v_c \in \{0, \dots, 6\},$$

representing the pip value ultimately assigned to that cell.

5.3 Domino Placement Constraints

Each domino must be used exactly once. Since each possible placement corresponds to a Boolean variable, we enforce

$$\sum_{e,o} \text{place}_{d,e,o} = 1 \qquad \text{for every domino } d.$$

This ensures that Z3 chooses one, and only one, placement-orientation pair for each domino.

5.4 Cell Coverage Constraints

For each cell c, we precompute the set of all possible placements that could cover it. Exactly one of these placements must be chosen:

$$\sum_{(d,e,o) \text{ covers } c} \text{place}_{d,e,o} = 1.$$

This prevents overlap and guarantees that every cell is filled by exactly one domino-side.

5.5 Cell Value Consistency

If a placement (d, e, o) would assign pip value x to a cell c, then whenever that placement is selected, the value of the cell must match x. This is expressed with implications:

$$\text{place}_{d,e,o} \;\Rightarrow\; v_c = x.$$

Because exactly one placement covering c is true, this pins down v_c to the pip value contributed by the chosen domino.

5.6 Region Constraints

Each region R on the board specifies a numerical rule on the cell values within it. These rules translate naturally into arithmetic constraints over the v_c. Examples include:

$$\sum_{c \in R} v_c = T, \qquad v_{c_1} = v_{c_2} \text{ for all } c_1, c_2 \in R,$$

or other constraints. These formulas capture the semantics of the puzzle's region labels.

6 Solving

We ask Z3 for a model:

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

and extract the orientation and position of each domino.

```
placements = [(0, 0, 0), (1, 1, 1), (2, 5, 1), (3, 8, 1)]

# (d, e, o) --> domino d is placed on edge e with orientation o
```

7 Visualization

We render the final board on a 2D grid.

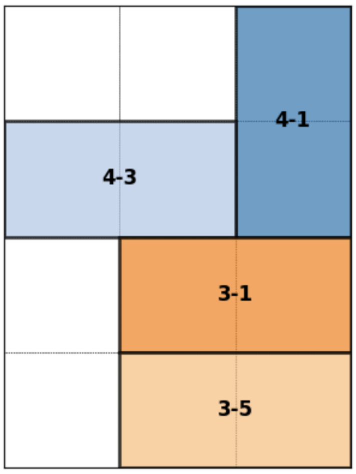

Figure 2: NYT Pips Easy Example Solution

8 Conclusion

By translating the puzzle into a constraint satisfaction model and leveraging Z3's reasoning capabilities, we obtain a general-purpose solver that works for arbitrary Pips boards. This demonstrates how modern SAT solvers can be applied effectively to logic puzzles involving spatial and numerical reasoning.