2048 is NP-Complete

Ahmed Abdelkader * Aditya Acharya † Philip Dasler ‡

Abstract

2048 is a single-player online puzzle game that went viral in March 2014. The game is played on a 4×4 board by sliding around and merging equal valued tiles to create tiles of higher value. The player wins by creating the *2048* valued tile, hence the name. We study the complexity of a slightly adapted version and prove that a number of natural decision problems turn out to be NP-Complete. We reduce from `3SAT` and implement our reduction as an online game.

1 Introduction

Each turn, the player picks a move from $\{\leftarrow, \rightarrow, \uparrow, \downarrow\}$ to slide all tiles on the board. Tiles slide as far as possible in the chosen direction until they hit either another tile or an edge of the board. When a sliding tile runs into a stationary one of equal value, they merge into a tile of double that value. Trailing tiles following a tile that just merged continue to slide uninterrupted and may merge among themselves as they come to rest one after the other. However, newly merged tiles cannot merge further in the same move. After each move, a *2* or *4* tile is generated in one of the empty cells. The player wins when a *2048* tile is created, hence the name of the game. Otherwise, the player loses when the board is full and no merges can be performed.

2048 combines features from two families of games: Candy Crush Saga [1] and PushPush [2]. A more detailed draft of this work appears in [3], where we also discuss the *attempt* in [4]. The interactive gadgets and *playable reduction* can be accessed at [5].

1.1 Adaptations and Problem Definition

We adapt the original *2048* as follows: (1) The input encodes the complete board configuration and no new tiles are generated. (2) The board is a rectangular grid of arbitrary size. In this paper, we are primarily concerned with the following decision problem:

Definition 1 (`2048-GAME`) *Given a configuration of tiles on an $m \times n$ board, is it possible to obtain a tile of value* 2048*? (More generally, 2^k with $k \geq 8$.)*

*University of Maryland, College Park, `akader@cs.umd.edu`
†University of Maryland, College Park, `acharya@cs.umd.edu`
‡University of Maryland, College Park, `daslerpc@cs.umd.edu`

[6] presented a proof of membership in NP, that applies to `2048-GAME`. The crucial piece is to bound the number of moves between two consecutive merges. Using a canonical orientation, all moves are interpreted as flips of the board, which send tiles along orbits of $O(mn)$ length. A pair of tiles that end up merging requires no more than $LCM(O(mn), O(mn)) = O(m^2n^2)$ moves.

In this paper, we prove NP-hardness by a reduction from 3SAT and obtain the main result.

Theorem 1 `2048-GAME` *is NP-Complete.*

2 Reduction from 3SAT

Given an instance of `3SAT` with n variables and m clauses, we produce an instance of `2048-GAME`. The board is filled using a 2-4 lattice to provide a rigid base for placing gadgets and planning their movements. We allow no merges using lattice tiles, which requires preserving their parity. This confines all merges to multiples of 2×2 blocks. We use *row* and *column* to denote a 2-row and a 2-column, respectively.

Displacers: These are the building blocks of all gadgets which allow us to communicate signals across the board. They come in two main forms: horizontal D and vertical D^T. Typically, a displacer starts in an *inactive* state where the middle 2×2 block, highlighted below, is shifted perpendicularly to the displacer's direction. An inactive displacer cannot merge, by any move sequence, before it is activated. The only way to activate it is to use another properly aligned displacer to engage its middle block. Collapsing tiles in a displacer shrinks it to a 2×2 block, which results in a *parity-preserving pull* in a row or a column.

$$D = \begin{bmatrix} 8 & \boxed{8} & \boxed{16} & 16 \\ 32 & \boxed{32} & \boxed{64} & 64 \end{bmatrix}$$

Variable Gadget: Each variable is represented by two horizontal displacers on the same *row*. This enables variables to move the portion of its row between their two displacers to the right or left. We enforce the assignment of variables in the order of their indices. A variable is assigned T or F using a $\rightarrow$ or $\leftarrow$ move, respectively. The displacers of x_1 come activated in the initial configuration to allow the game to start. No matter how variable x_i is assigned, the *connector displacers* in its row get activated and allow x_{i+1} on top of it to be activated by a $\downarrow$ move in the following turn.

This is an abstract of a presentation given at CG:YRF 2015. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear in a conference with formal proceedings and/or in a journal.

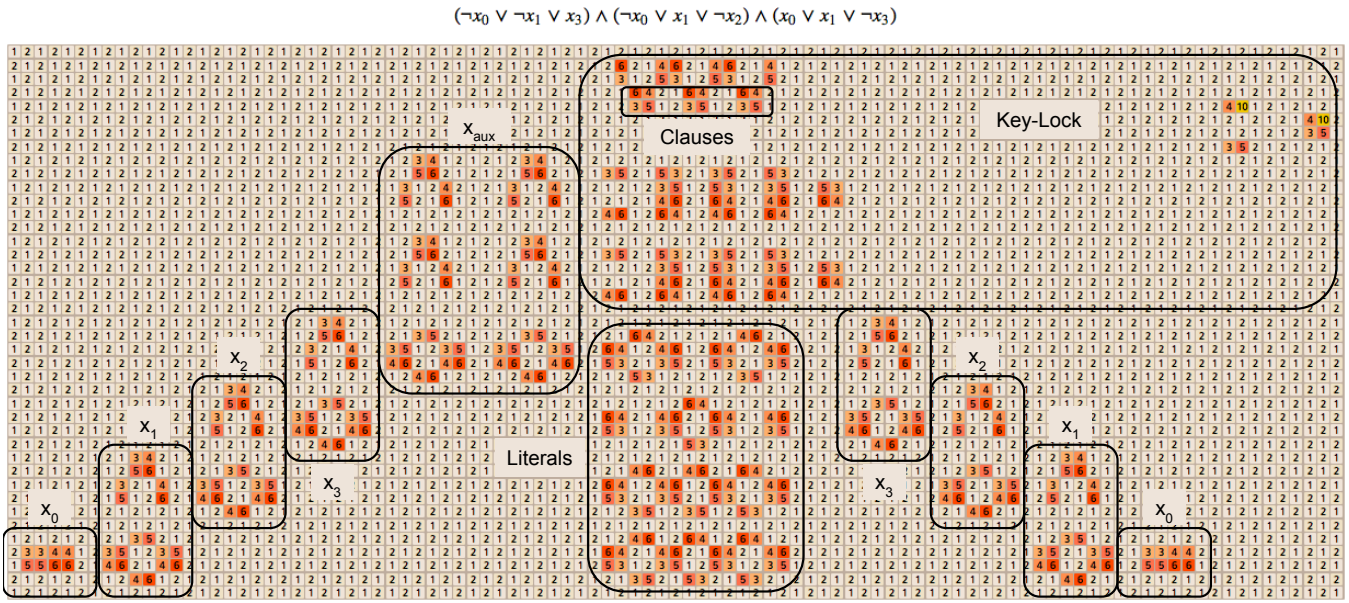


Figure 1: Annotated reduction. Only x_0 is active. We apply $\log_2$ and hide paddings to help display a large board.

Clause Gadget: Literals are encoded using a similar mechanism to the connectors in the variable gadget. but are only activated by the appropriate assignment. When a literal is activated, it allows a $\downarrow$ pull in the clause's *column* to effectively satisfy this clause. Each satisfied clause eventually contributes one horizontal displacer by providing its middle block.

Key-Lock Gadget: To check that all clauses are satisfied, it helps to arrange for a special event to happen only after all variables have been assigned. To achieve this, an auxiliary variable $x_{aux} = x_{n+1}$ activates the lock portion of this gadget. Satisfying all clauses corresponds to using the correct key. Together, the activated key-lock gadget is a sequence of displacers that can activate a unique displacer with two 1024 tiles. Collapsing that unique displacer creates the desired *2048* tile.

2.1 Properties of the Reduction

Size: As variables are stacked on top of each other all the way up to x_{aux} and the key-lock gadget, the number of rows is $O(n)$. Then, each variable has to activate the connectors to the next variable. We get a pyramid shape with variable displacers on both sides and literals in the middle, plus the unique displacer taking up $2(m+1)$ columns far to the right, for a total of $O(m+n)$ columns.

Gaps and Padding: A gap is created iff two tiles merge. The construction guarantees that gaps are only created near the edges of the board and accumulate at the corners. To make sure such gaps do not result in undesired shifts within the core, it has to be surrounded by enough padding. As the number of active gadgets is $O(m+n)$ and each gadget contributes a constant number of gaps, a padding of $O(m+n)$ thickness suffices.

Game Play: When no merges happen, two consecutive moves in opposite directions leave the board unchanged e.g. $[\leftarrow, \rightarrow, \leftarrow]$ is effectively reduced to $[\leftarrow]$. *Effective* moves alternate between horizontal and vertical. The alternation accumulates newly created gaps, resulting from the merge, at the corners so the decision encoded by the previous move cannot be altered. Furthermore, any row or column may witness merges during at most one turn. In particular, clause columns cannot experience more than one $\downarrow$ pull. This implies consistent assignments. Finally, $\uparrow$ moves are useless since they must be canceled or otherwise the player cannot win.

Hardness: Aligning the two 1024 tiles requires a $2m$ shift, which only satisfied clauses can provide with each satisfied clause contributing 2. Hence, the *2048* tile can be created iff the `3SAT` instance is satisfiable.

References

[1] Luciano Gualà, Stefano Leucci, and Emanuele Natale. Bejeweled, Candy Crush and other Match-Three Games are (NP-)Hard. *CoRR*, abs/1403.5830, 2014.

[2] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush is NP-hard in 2D. *CoRR*, cs.CG/0001019, 2000.

[3] Ahmed Abdelkader, Aditya Acharya, and Philip Dasler. On the complexity of slide-and-merge games. *CoRR*, abs/1501.03837, 2015.

[4] Rahul Mehta. 2048 is (PSPACE) Hard, but Sometimes Easy. *CoRR*, abs/1408.6315, 2014.

[5] Ahmed Abdelkader. 2048 gadgets. `http://cs.umd.edu/~akader/projects/2048/index.html`.

[6] Christopher Chen. 2048 is in NP. `http://blog.openendings.net/2014/03/2048-is-in-np.html`.

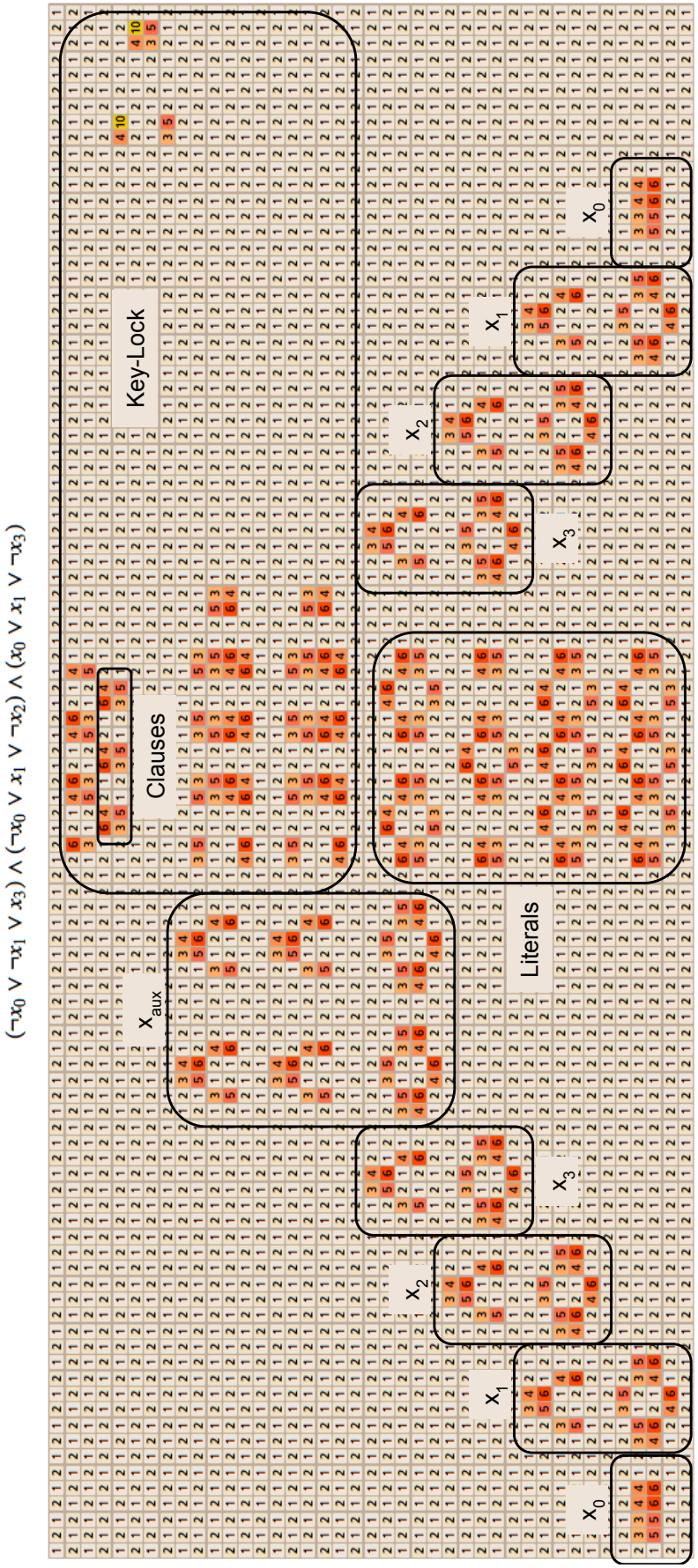


Figure 2: Annotated reduction of a given boolean formula.

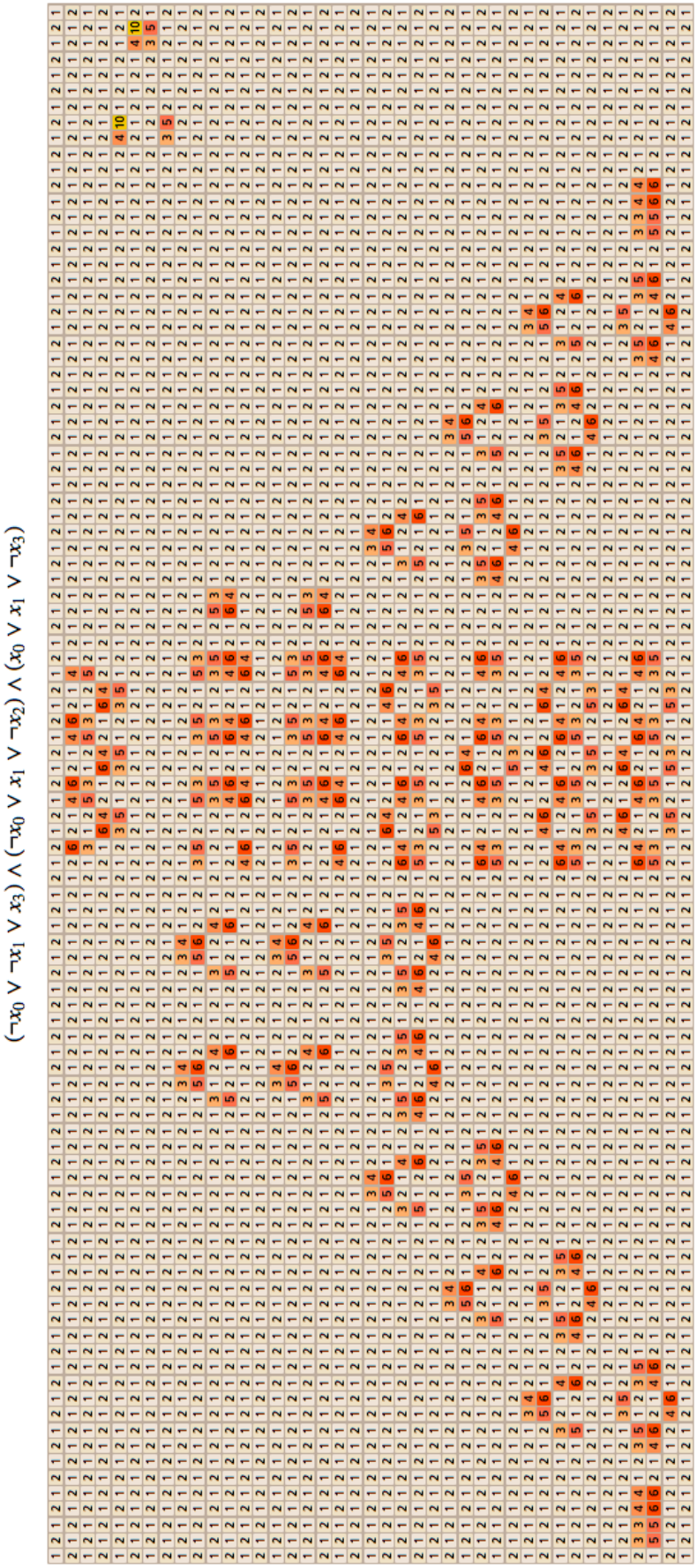
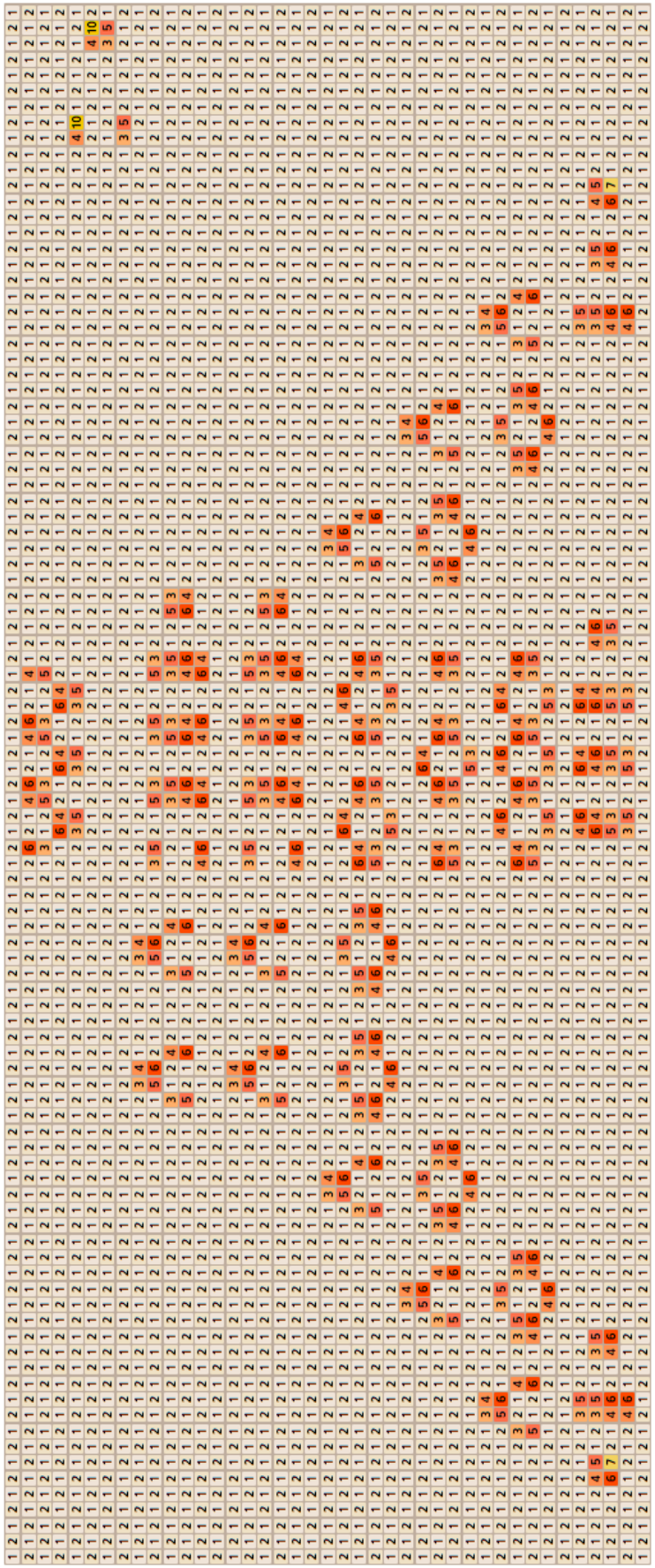


Figure 3: Board(0): Initial configuration.



$$(\neg x_0 \vee \neg x_1 \vee x_3) \wedge (\neg x_0 \vee x_1 \vee \neg x_2) \wedge (x_0 \vee x_1 \vee \neg x_3)$$

Figure 4: Board(1): $move(\rightarrow)$. x_0 assigned T.

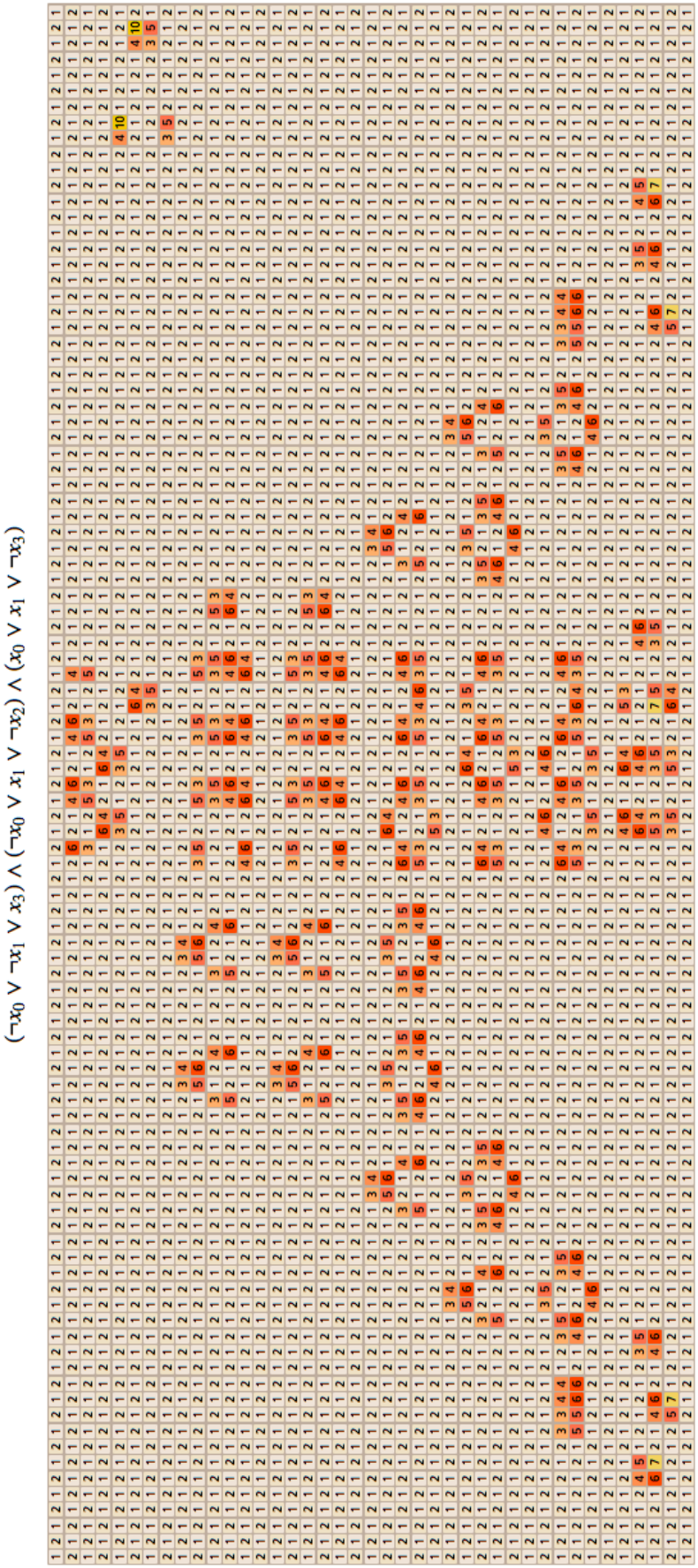


Figure 5: Board(2): $move(\downarrow)$. c_2 satisfied. x_0 fixed to T. x_1 activated.

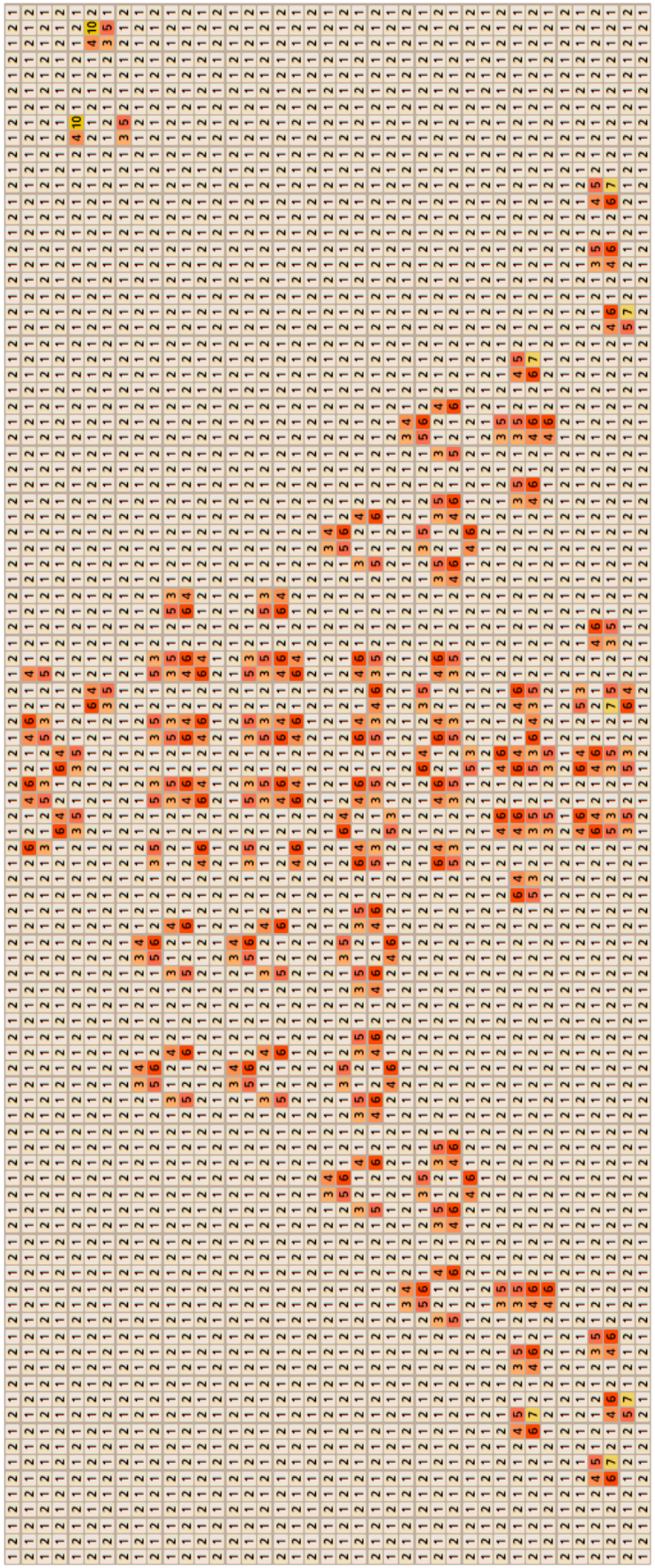


Figure 6: Board(3): $move(\leftarrow)$. x_1 assigned F.

Figure 7: Board(4): $move(\downarrow)$. c_0 satisfied. x_1 fixed to F. x_2 activated.

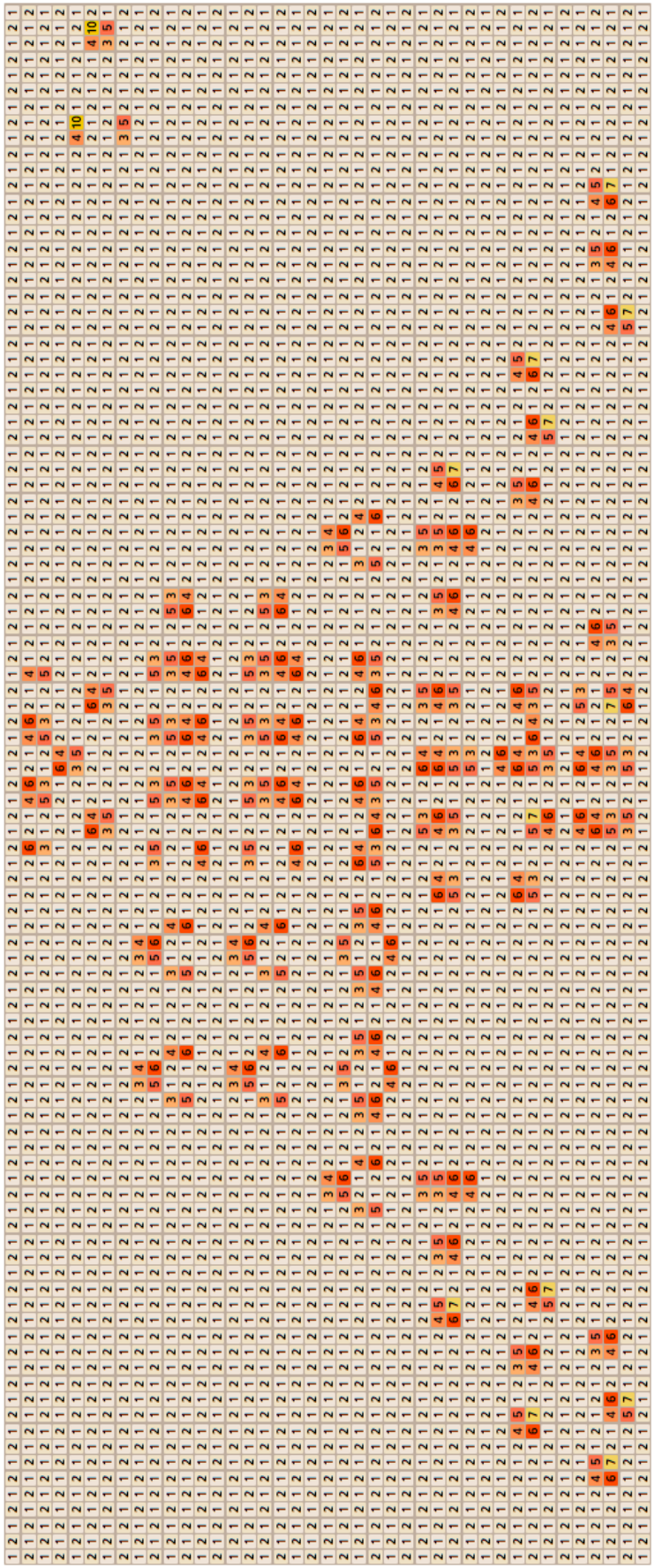


Figure 8: Board(5): $move(\leftarrow)$. x_2 assigned F.

Figure 9: Board(6): $move(\downarrow)$. c_1 satisfied. x_2 fixed to F. x_3 activated.

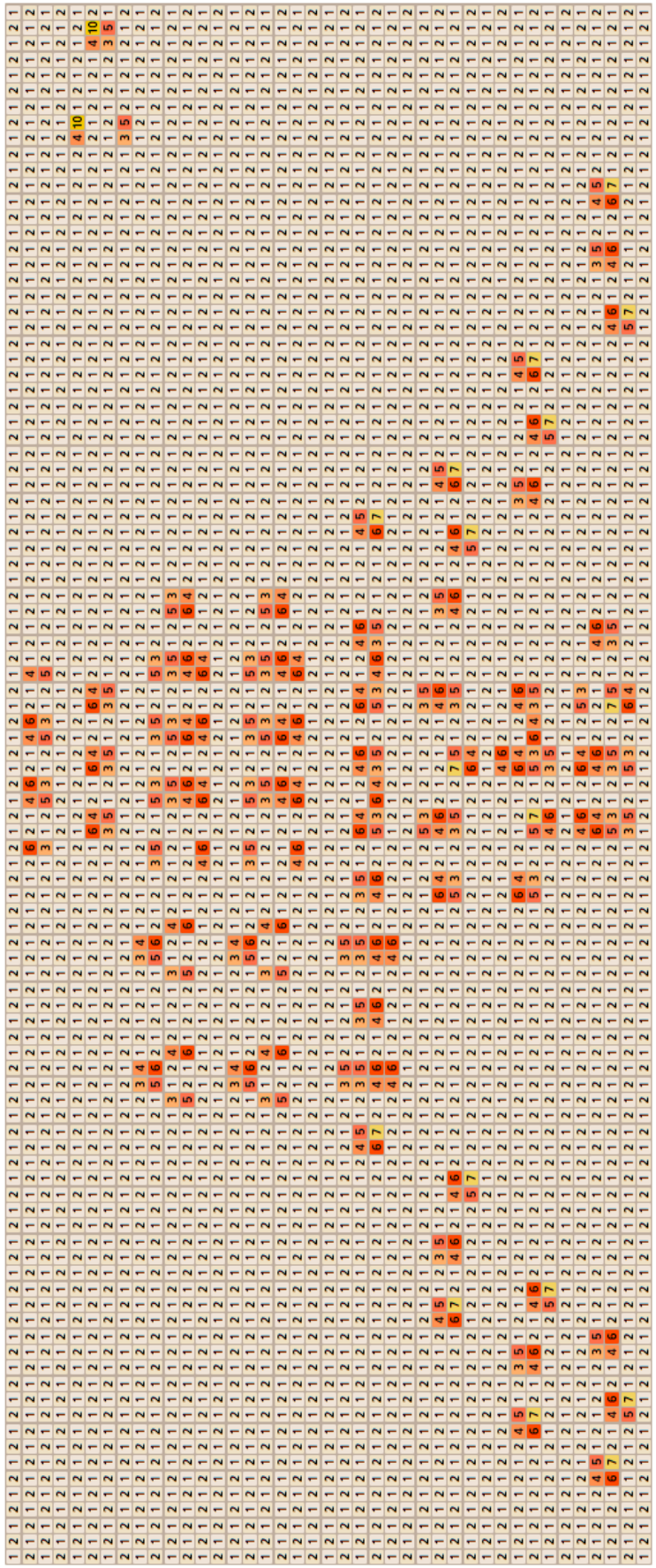


Figure 10: Board(7): $move(\rightarrow)$, x_3 assigned T.

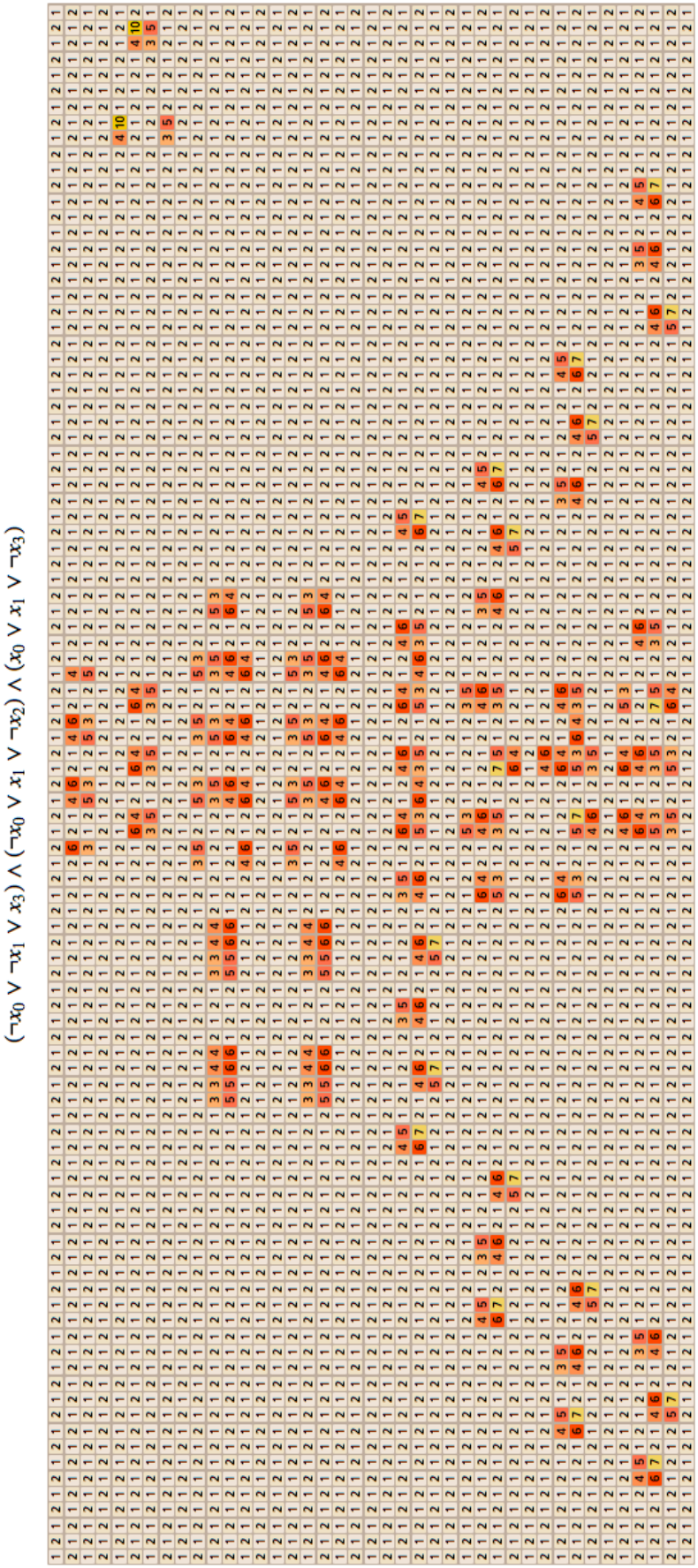
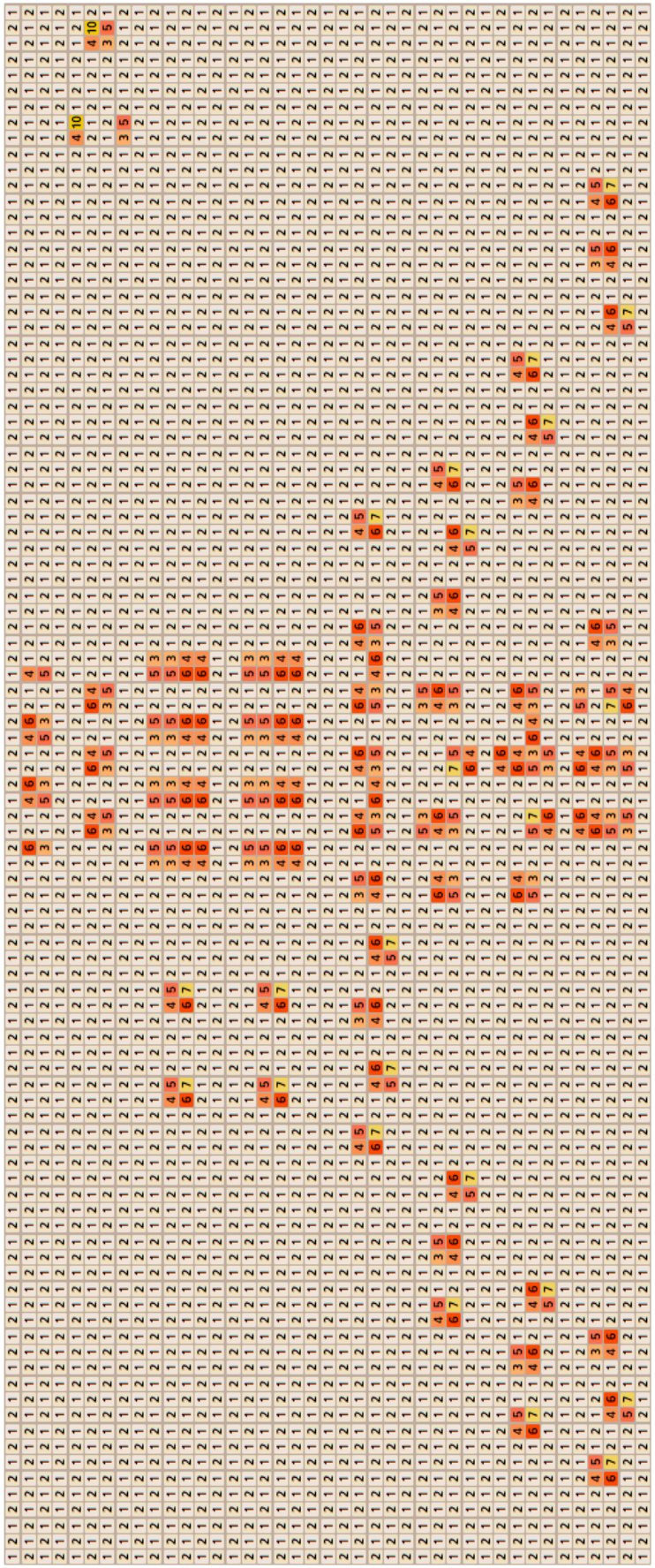


Figure 11: Board(8): $move(\downarrow)$. x_3 fixed to T. x_{aux} activated.



$$(\neg x_0 \vee \neg x_1 \vee x_3) \wedge (\neg x_0 \vee x_1 \vee \neg x_2) \wedge (x_0 \vee x_1 \vee \neg x_3)$$

Figure 12: Board(9): $move(\leftarrow)$. Key-lock sequence initiated.

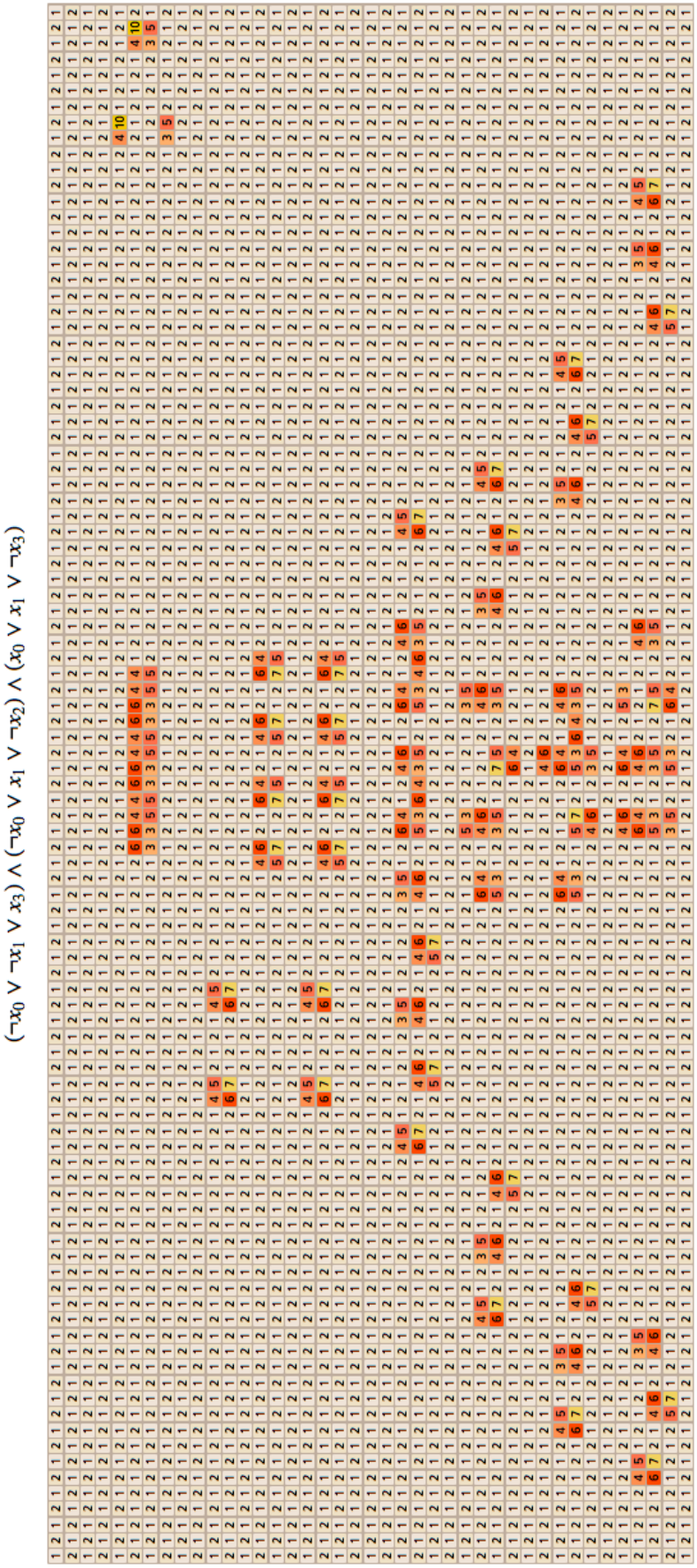


Figure 13: Board(10): $move(\downarrow)$. Key-lock ready.

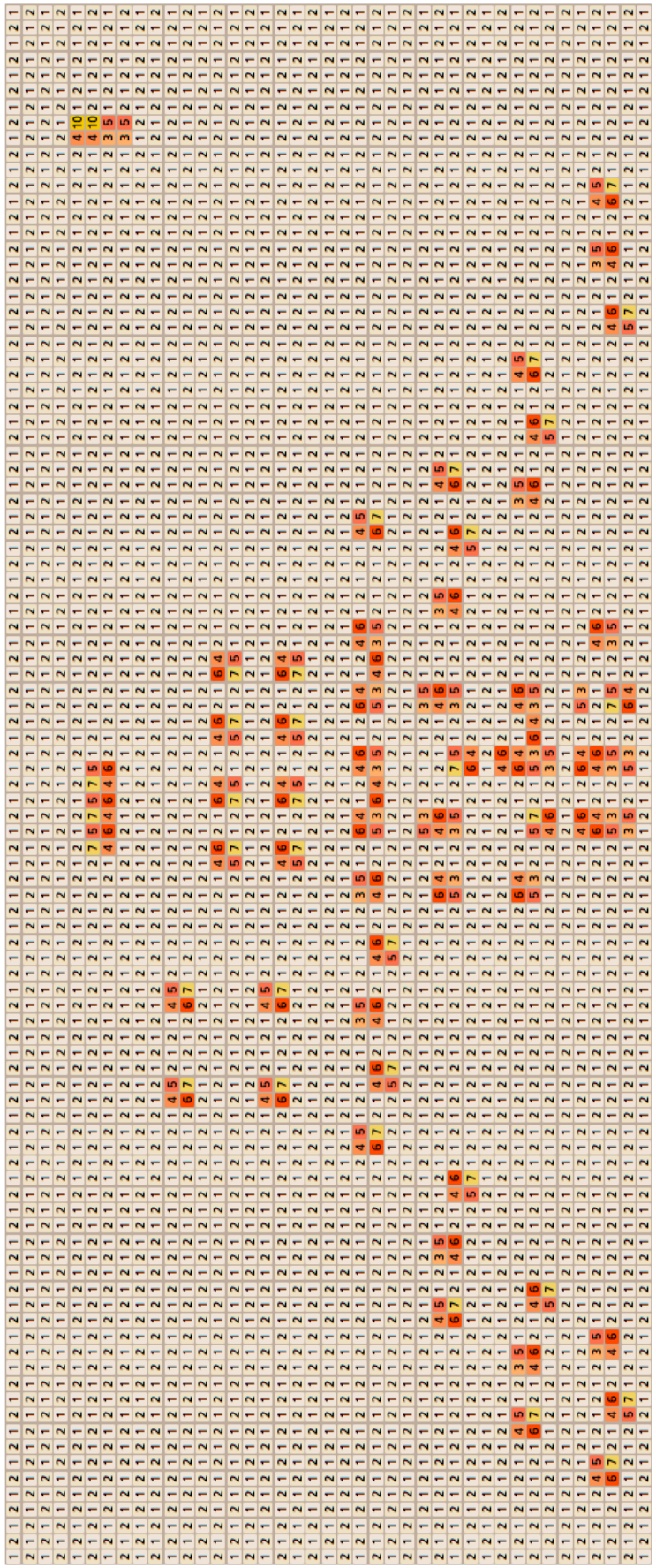


Figure 14: Board(11): $move(\leftarrow)$. All clauses satisfied? Unlock.

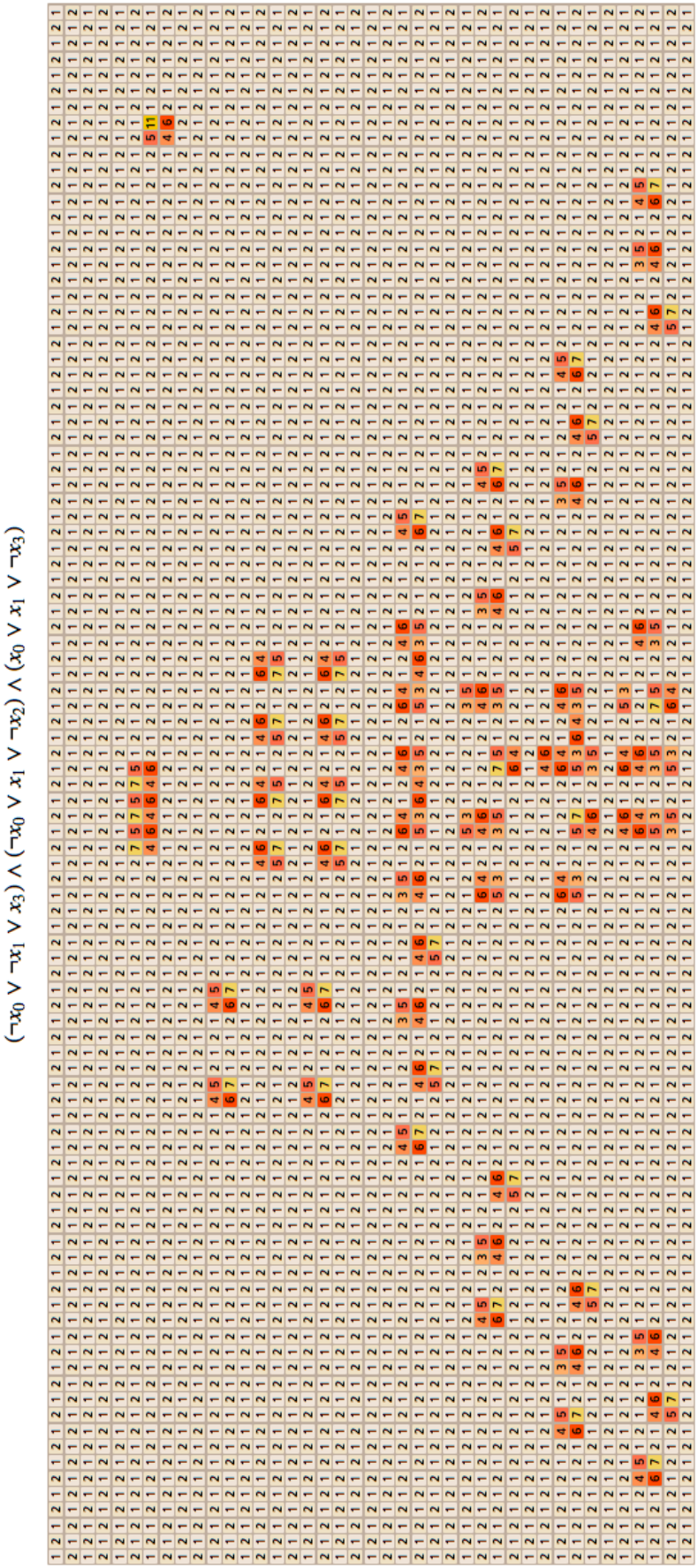


Figure 15: Board(12): $move(\downarrow)$. Win.