Solving the Module Game with SAT

Tobias Sebastian Hahn, Norbert Manthey and Tobias Philipp
Knowledge Representation and Reasoning Group
TU Dresden, Germany

***Abstract*—We describe a CNF encoding of the Modulo game, a certain form of a combinatorial puzzle. To solve this game, tiles have to be placed on a field such that the sum of all overlaying values of a cell sums up to a multiple of a predefined value. We modify sorting networks to encode a modulo constraint for numbers that are represented unary.**

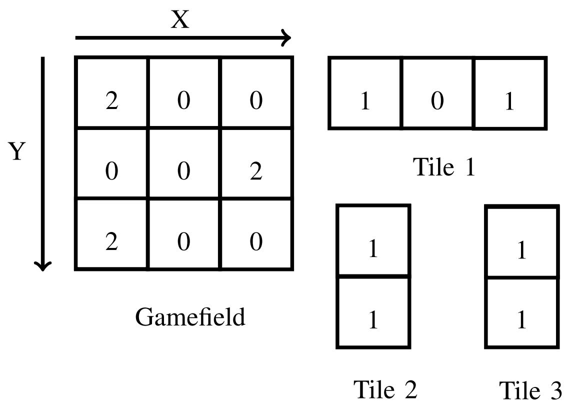


Fig. 1: A Modulo game with three tiles and modulo value 3.

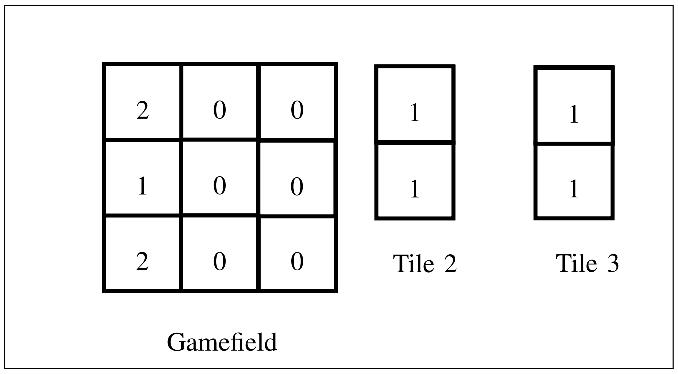
I. Introduction

The modulogame consists of a single game field, which is a rectangle of cells, and several tiles. Each cell on the game field is a modulo-k-adder and has a specified initial value. A tile is a rectangular arrangement of cells, which can hold the values 0 or 1. The goal is, that after placing all tiles on the field that each modulo-k-adder holds the value zero. Tile cells that hold the value 1 increase the modulo-k-adder value by 1. A playable version of the game can be accessed here: http://www.hacker.org/modulo/.

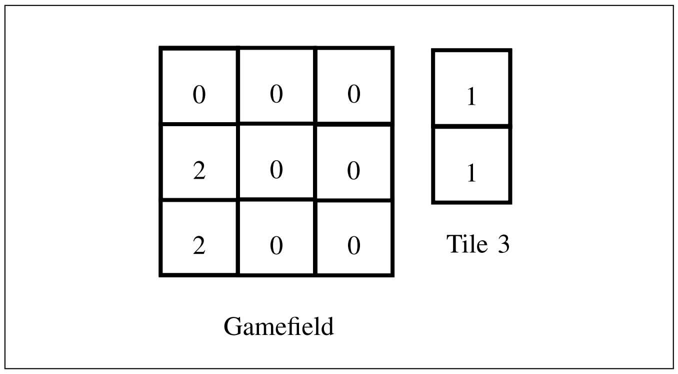
We illustrate the Modulo game by an example in Figure 1. The modulo value is 3. The field consists of 3 times 3 cells, where each cell is a modulo-3-adder, and there are three tiles.

To place a tile in the field, we use a binary vector (x, y), where x represents the column and y represents the row, where the upper left cell of the tile will be placed. We start counting in the upper left corner and start with 0. Tile 1 can be positioned at $(0, 0), (0, 1)$ and $(0, 2)$. Tile 2 and tile 3 can be positioned at $(0, 0), (1, 0), (1, 1)$ and $(0, 1)$.

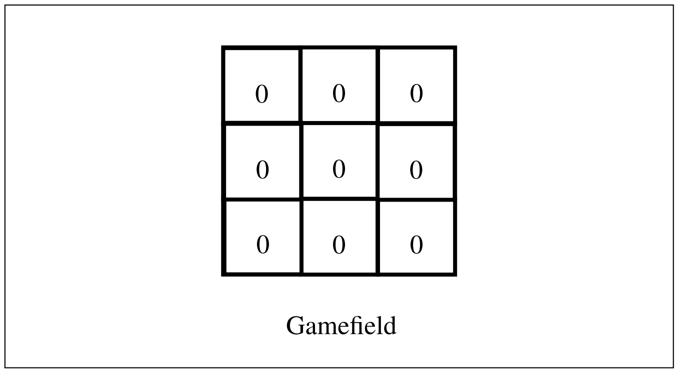
We place tile 1 on $(0, 1)$, and apply the addition. The resulting state of the field is given next, where tile 1 does not occur any more, as it has been placed already.



In this figure, one can already see that the current state of the example game has two solutions. The first solution is to put tile 2 at position $(0, 0)$ and to place tile 3 at position $(0, 1)$. The second solution is to swap the positions of the two tiles, as they have the same shape and all corresponding cells hold the same values. In the example, we follow the first solution, and place tile 2 at $(0, 0)$.



Finally, we place the last tile at position $(0, 1)$, as discussed above, and reach a solution.



II. Encoding the Modulo Game

In this section we briefly describe how we encode the game into CNF.

A. Placing Tiles

For each tile and possible tile position such that the tile fits on the game field we introduce a propositional variable $p(t, x, y)$. As each tile must be positioned once to the field, for each tile we add an *exactly-one constraint* for all these variables. The at-least-one part of this constraint is encoded as a clause, and the at-most-one part is encoded naively without introducing auxiliary variables.

B. Encoding Cells

For each cell of the field we encode a modulo constraint

$$(x_1 + \cdots + x_n) \mod k = 0,$$

where x_i are propositional literals, and k is a natural number. The constraint ensures that the number of satisfied literals in $\{x_1, \ldots, x_n\}$ is 0 modulo k. We use a sorting-network for, and represent the number of the cell unary. The input to the sorting network of a cell is the initial value of the cell, and all the tile placement variables $p(t, x, y)$. By requiring that each tile has to be placed on the field, and requiring that all numbers that are added to the field cells are handled properly, we obtain a CNF encoding of the game.

1) Sorting Network Input: Consider the game in Figure 1 once more. Given a cell $(2, 0)$, then the sorting network of cell $(2, 1)$ has two inputs that are initialized to $\top$, as the cell initially holds the value 2. Next, when tile 1 contributed a 1 to this cell, if this tile would have been placed on position $(0, 1)$. Hence, the variable $p(1, 0, 1)$ is another input to the network. Similarly, for the other tiles the variables $p(2, 2, 0)$, $p(2, 2, 1)$ as well as $p(3, 2, 0)$ and $p(2, 2, 1)$ are added.

2) Sorting Network Output: The output of a sorting network is a sorted sequence of the input truth values, where all satisfied values are *printed* first, and afterwards, the corresponding number of falsified values is given. Hence, the output of this sorting network is the unary representation of the number of satisfied inputs. For the modulo game, this number is the value that the cell holds without applying the modulo operation. Assume that the network has n inputs (including the initialization inputs), the modulo value is k and the output is the unary number o with the variables o_0 to o_{n-1}. Then, $o \mod k = 0$, if for some $0 \leq i \leq k$ with $i \mod k = 0$ the bit o_i is satisfied and o_{i+1} is falsified, such that o represents the number i. To check whether the pair $o_i \wedge \overline{o_{i+1}}$ is satisfied, we add another auxiliary variable, one for each bit pair of the sorting network output. Then, we add another clause that requires that one of the bit pairs for $i \mod k = 0$ has to be satisfied, including the case $i = 0$ and $i = n$, if $n \mod k = 0$.

III. The Formulas

The submitted formulas have been generated from games that are available at http://www.hacker.org/modulo/ up to a hardness level of 38. We submitted two flavors. In the first set, the sorting networks have been encoded with the sorting networks [2]. In the second set, we replaced this encoding with the encoding presented in [1].

References

[1] Ignasi Abo, Robert Nieuwenhuis, A.O., Rodrguez-Carbonell, E.: A parametric approach for smaller and better encodings of cardinality constraints. In: Schulte, C. (ed.) Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, vol. 8124, pp. 80–96. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-40627-0_9

[2] Roberto Asn, Robert Nieuwenhuis, A.O., Rodrguez-Carbonell, E.: Cardinality networks and their applications. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing - SAT 2009, Lecture Notes in Computer Science, vol. 5584, pp. 167–180. Springer Berlin Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-02777-2_18