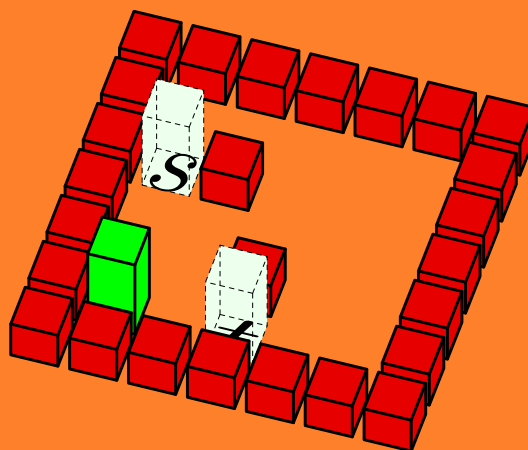




# ON THE COMPLEXITY OF ROLLING BLOCK AND ALICE MAZES

Markus Holzer      Sebastian Jakobi



IFIG RESEARCH REPORT 1202

MARCH 2012

Institut für Informatik  
JLU Gießen  
Arndtstraße 2  
35392 Giessen, Germany  
Tel: +49-641-99-32141  
Fax: +49-641-99-32149  
[mail@informatik.uni-giessen.de](mailto:mail@informatik.uni-giessen.de)  
[www.informatik.uni-giessen.de](http://www.informatik.uni-giessen.de)

IFIG RESEARCH REPORT  
IFIG RESEARCH REPORT 1202, MARCH 2012

# ON THE COMPLEXITY OF ROLLING BLOCK AND ALICE MAZES

Markus Holzer<sup>1</sup> and Sebastian Jakobi<sup>2</sup>

Institut für Informatik, Universität Giessen  
Arndtstraße 2, 35392 Giessen, Germany

**Abstract.** We investigate the computational complexity of two maze problems, namely rolling block and Alice mazes. Simply speaking, in the former game one has to roll blocks through a maze, ending in a particular game situation, and in the latter one, one has to move tokens of variable speed through a maze following some prescribed directions. It turns out that when the number of blocks or the number of tokens is not restricted (unbounded), then the problem of solving such a maze becomes PSPACE-complete. Hardness is shown *via* a reduction from the nondeterministic constraint logic (NCL) of Demaine and Hearn to the problems in question. In this way we improve on a previous PSPACE-completeness result of Buchin and Buchin on rolling block mazes to best possible. Moreover, we also consider bounded variants of these maze games, i.e., when the number of blocks or tokens is bounded by a constant, and prove close relations to variants of graph reachability problems.

Categories and Subject Descriptors: F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes—*Reducibility and completeness*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Path and circuit problems*;







Additional Key Words and Phrases: constraint logic, puzzles, rolling block maze, Alice maze

---

<sup>1</sup>E-mail: holzer@informatik.uni-giessen.de

<sup>2</sup>E-mail: jakobi@informatik.uni-giessen.de

## 1 Introduction

Puzzles where the player moves tokens over the playing board according to specific rules, trying to reach some target configuration, can be very challenging and enjoy great popularity. The number of different variants and rules of those puzzles is legion. In some puzzles, the playing board has a labeling that somehow prescribes which moves can be made. In other puzzles with an unlabeled board, the legal moves only depend on the tokens themselves. An example for the latter kind of puzzles are rolling block puzzles, where the player has to roll blocks of size  $\ell \times 1 \times 1$  over the board, and the goal is to move a designated block to a target position. Here the possible moves are determined by the dimensions of the blocks. By labeling some fields of the board as “forbidden,” mazes can be created, through whose corridors the blocks have to be rolled. A slightly different variant are colour mazes, where the fields of playing board are coloured, and the block may only come to lie on a uni-coloured area. A precursor to rolling block puzzles are the rolling cube puzzles, which were popularized by an article of Gardner in [6–8]. There, instead of arbitrary blocks, a single die, i.e., a cube with the (standard) faces , , , , , and , is used for rolling over the board, such that the top face of the die has to be the same as the label of the square of the board, assuming an appropriate labeling of the board. It was shown in [4], that these mazes can be solved efficiently in deterministic polynomial time.

A different kind of puzzles with labeled playing boards, that use simple tokens instead of particularly shaped blocks, are Alice mazes. Here each square of the board is labeled with arrows that designate in which direction a token on that square may be moved. Additionally, some squares may also change the speed of the tokens.

Rolling blocks and moving tokens is not that easy, as it looks first, because several blocks or tokens may prevent certain moves, since they may block each other’s movements. In fact a lot of such problems turn out to be extremely complicated from a complexity theoretical point of view, namely PSPACE-complete, if the number of tokens is not bounded. Here we investigate the computational complexity of some of the above mentioned puzzles, namely rolling block mazes, from which also results on colour mazes can be concluded, and Alice mazes. Detailed definitions of those games are given in the appropriate sections. We show that if the number of blocks or tokens is not bounded, these problems are PSPACE-complete. While containment in PSPACE is easy, the hardness is shown with the help of the recently introduced uniform framework for modeling games, the nondeterministic constraint logic (NCL) of Demaine and Hearn [5]. It is worth mentioning that rolling block mazes were considered before in the literature. In [2] the PSPACE-completeness of rolling block mazes was shown, even if the blocks are all of size  $2 \times 1 \times 1$  or  $3 \times 1 \times 1$ . In the journal version [3] of [2], instead of  $3 \times 1 \times 1$  blocks, forbidden squares are used. The question of the complexity with only  $2 \times 1 \times 1$  blocks, and without forbidden squares was left open—see also [9]. As a side result of our construction, we are able to prove that PSPACE-completeness also holds for this case. This is best possible, since we also show that under weak constraints rolling block mazes with blocks

of unit size—these blocks are called cubes—are trivially solvable. Further, we also consider the cases, where the number of blocks or tokens on the board is bounded by a constant. Here it shows that these problems are closely related to variants of graph reachability problems.

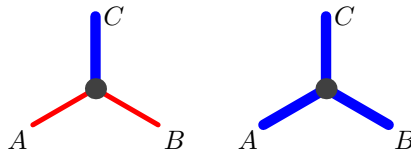
The paper is organized as follows: In the next section we introduce the necessary notations. In Section 3 we turn our attention to rolling block mazes, where we investigate the complexity of the game for an unbounded and bounded number of blocks of different sizes. In particular we improve a previous result [2, 3] on the **PSPACE**-completeness of unbounded rolling block mazes to blocks of size  $2 \times 1 \times 1$  only (without forbidden squares). In Section 4 we study Alice mazes, considering two main variants again, namely whether the number of tokens on the board is bounded or not. For the bounded case we prove **NL**-completeness, while the unbounded case is **PSPACE**-complete in general again, even without speed changing fields. The results on rolling block mazes and Alice mazes are summarized in Tables 1 and 2 in the ultimate section.

## 2 Definitions

We assume familiarity with the basic concepts of complexity theory [11] such as the inclusion chain  $\text{AC}^0 \subset \text{NC}^1 \subseteq \text{L} = \text{SL} \subseteq \text{NL} \subseteq \text{AL} = \text{P} \subseteq \text{NP} \subseteq \text{PSPACE}$ . Here  $\text{AC}^0$  and  $\text{NC}^1$  refer to the sets of problems accepted by polynomial size uniform families of Boolean {AND, OR, NOT}-circuits having, respectively, unbounded fan-in and constant depth, and bounded fan-in and logarithmic depth.  $\text{L}$  is the set of problems accepted by deterministic logarithmic-space bounded Turing machines.  $\text{SL}$  and  $\text{NL}$  can be taken to be the sets of problems logspace-reducible to the undirected graph reachability (UGR) and to the directed graph reachability (GR) problems respectively and  $\text{AL}$  is the set of problems accepted by alternating logspace bounded Turing machines.  $\text{P}$  ( $\text{NP}$ , respectively) is the set of problems accepted by deterministic (nondeterministic, respectively) polynomial time bounded Turing machines and **PSPACE** is the set of problems accepted by deterministic or nondeterministic polynomial space bounded Turing machines. All the relationships depicted in the inclusion chain have been known for a quarter of a century, except for  $\text{L} = \text{SL}$ , shown in [12].

Two other particularly relevant problems are undirected grid graph reachability (UGGR) and constraint logic (CL). The former problem is defined as follows: given an  $n \times n$  grid of nodes such that an edge only connects immediate vertical or horizontal neighbors, is there a path from node  $s$  to node  $t$ , where  $s$  and  $t$  are designated nodes from the grid? UGGR is  $\text{NC}^1$ -hard under  $\text{AC}^0$  reducibility, it belongs to  $\text{L}$ , yet it is not known to be  $\text{L}$ -hard [1]. The latter problem, i.e., constraint logic or more precisely nondeterministic constraint logic (NCL), is defined as follows: given a constraint graph  $G$  and an edge  $e$  of  $G$ , is there a sequence of legal moves on  $G$  that eventually reverses  $e$ ? Here a constraint graph is a directed graph with edge weights from the set  $\{1, 2\}$  and where each vertex has a non-negative minimum inflow. Here the inflow of a vertex is the sum of the weight of inward-directed edges. A legal configuration of a constraint graph has an inflow of at least the minimum inflow at each vertex

(these are the constraints that have to be satisfied), and a legal move on a constraint graph is the reversal of a single edge that results in a legal configuration. NCL is PSPACE-complete, even for planar constraint graphs built by AND- and OR-vertices only [5]—see Figure 1 for AND- and OR-vertices. Thus in order



**Fig. 1.** Nondeterministic constraint logic (NCL): red edges have weight 1, blue edges have weight 2, and vertices have minimum in-flow constraint of 2. (Left:) AND-vertex: edge  $C$  may be directed outward if and only if both edges  $A$  and  $B$  are directed inward. (Right:) OR-vertex: edge  $C$  may be directed outward if and only if either edge  $A$  or edge  $B$  is directed inward.

to prove PSPACE-hardness it suffices to construct AND- and OR-gadgets that simulate the behaviour of the corresponding vertices and wiring capabilities.

### 3 Rolling Block Mazes

A rolling block maze is played on an  $n \times n$  rectangular board with an arbitrary number of blocks initially placed on the board (up-right or lying). The blocks are of sizes  $\ell \times 1 \times 1$ . In particular, a  $1 \times 1 \times 1$  block is also called a cube. Some of the board cells (squares) may be marked as forbidden territory—instead of forbidden territories one can also use non-movable unit cubes on the appropriate board squares. The objective of the game is to roll a distinguished block from a given starting position  $s$  to a target position  $t$  without rolling off the grid (here the forbidden squares count as off the grid). It is *not* required to visit all squares of the board. A rolling block maze and its solution is shown in Figure 2. There the cubes on the border of the maze are non-movable and only used to engrid the rolling block maze; all non-movable blocks are not counted as blocks in the forthcoming.

#### 3.1 Rolling Block Mazes with an Unbounded Number of Blocks

We first prove that solving rolling block mazes is PSPACE-complete, if the number of blocks is not bounded by a constant. This result improves a recent result [2, 3] on the PSPACE-completeness of rolling block mazes by using blocks of size  $2 \times 1 \times 1$  only and disallowing forbidden squares. In [2] forbidden squares can be simulated by larger blocks of size  $3 \times 1 \times 1$ .

**Theorem 1.** *Solving a rolling block maze with an unbounded number of blocks is PSPACE-complete, even if all blocks are of size  $2 \times 1 \times 1$ . The result holds true even without forbidden squares.*

*Proof.* Given a rolling block maze, a polynomial space bounded Turing machine can store the configuration and may simulate the sequence of movements of

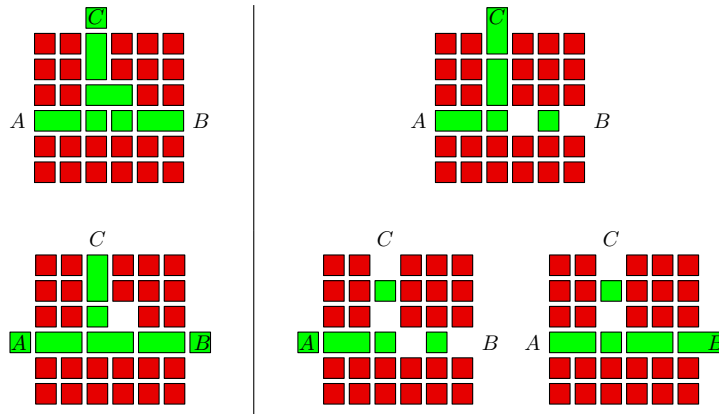


**Fig. 2.** Rolling block maze of size  $7 \times 7$  with single block (green) of size  $2 \times 1 \times 1$ , not counting the non-movable cubes (red). The leftmost image in the first row shows the initial configuration of the maze in topview, and the others are shown three-dimensional. The solution moving the single  $2 \times 1 \times 1$  block from  $s$  to  $t$  (starting and ending in an up-right position) down from left to right and top to bottom with the following movements: down, down, right, right, up, right, up, left, down, right, up, left, left, down, right, down, down, and left. This example comes from [www.puzzlebeast.com/rollingblock](http://www.puzzlebeast.com/rollingblock).

the blocks by simply guessing the sequence step by step. Since determinism and nondeterminism coincides for polynomial space by Savitch's theorem [13], the containment within PSPACE follows. It remains to show PSPACE-hardness, for which we reduce NCL to rolling block mazes with an unbounded number of blocks. Therefore, we construct submazes depicted in Figure 3, simulating AND- and OR-vertices and show how to properly connect these.

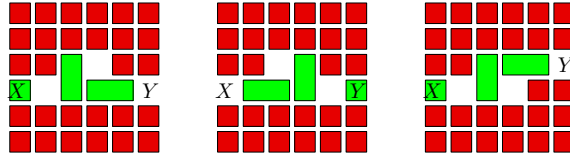
In all cases, the red blocks cannot be moved at all, except for the two blocks left and right of the horizontal lying block in the middle of the AND-submaze (left-hand side of Figure 3) and for the four blocks above and under the four center squares in the connection-submaze (Figure 4). If in the AND-submaze, the middle block gets rolled downwards, one of the two neighbouring red blocks may be toppled onto this two free squares, freeing exactly one square, which cannot be used to move any other block. We can only move the red block back to its initial position. Similarly, two neighbouring free squares in the connection-submaze gives the opportunity to tip a standing red block from above or from below onto this place, which again only leaves one free square, that cannot be used to move any other block. Thus, these standing red blocks can be seen just as forbidden squares.

The AND-submaze is shown on the left-hand side of Figure 3. The block standing on exit  $C$  may be rolled downwards into the submaze, which means that the corresponding edge gets turned outward, if and only if both lying blocks at  $A$  and  $B$  are rolled out of the submaze, meaning the corresponding edges being turned inward. In the OR-submaze, shown on the right-hand side of Figure 3, the lying block at exit  $C$  may be rolled further into the submaze, if and only if at least one of the blocks near  $A$  and  $B$  are rolled outwards, occupying the corresponding exit. The other cases behave symmetrically. So these submazes behave like NCL AND- and OR-vertices. Note that all inward-facing edges, that is, all exits where a block reaches out of the submaze, occupy exactly the first square outside the submaze.



**Fig. 3.** (Left:) Rolling block maze subgame simulating an AND-vertex. (Right:) Rolling block maze simulating an OR-vertex. If a label  $A$ ,  $B$ , or  $C$  is occupied by some block, then the corresponding edge faces inward, otherwise, the edge faces outward. So edge  $C$  is facing inward in the upper and gets turned outward in the lower row.

The mazes that connect these AND- and OR-vertex-mazes are shown in Figure 4. The labeled squares  $X$  and  $Y$  correspond to some labeled squares  $A$ ,  $B$ , or  $C$  of the neighbouring vertex-submaze. These connection-submazes also ensure, that no block may leave its corresponding submaze for more than one roll, since the vertical block in the middle can only be moved by one step. All submazes have the size  $6 \times 6$  and so can be rotated and arranged easily into a grid. Connecting rotated vertex-mazes may cause offsets on the corresponding exits, which can be adjusted by using an offset-connector, as shown on the right-hand side of Figure 4, which of course can be mirrored, if needed. This completes the proof of PSPACE-completeness.  $\square$



**Fig. 4.** Connection-submazes: (Left:)  $Y$  is free to take a block from the right neighbour vertex maze and the block on  $X$  prevents the left neighbour vertex maze from rolling a block outwards. Thus, the corresponding edge faces rightwards. (Middle:) This positioning describes a leftward facing edge. (Right:) An offset-connection.

A variant of rolling block mazes with some additional rules are colour mazes. A colour maze is an  $n \times n$  grid where the cells are coloured by red or blue. There the challenge is that the blocks must be moved from the starting positions to the target positions without rolling off the grid and in addition, the block must always lie entirely within one colour zone. An example of a colour maze and its 14 step solution is shown in Figure 5—here the block is of size  $2 \times 1 \times 1$  and it must start and end in the vertical position. Of course, the rolling block mazes from above can be seen as uni-colour colour mazes and we immediately gain the following:

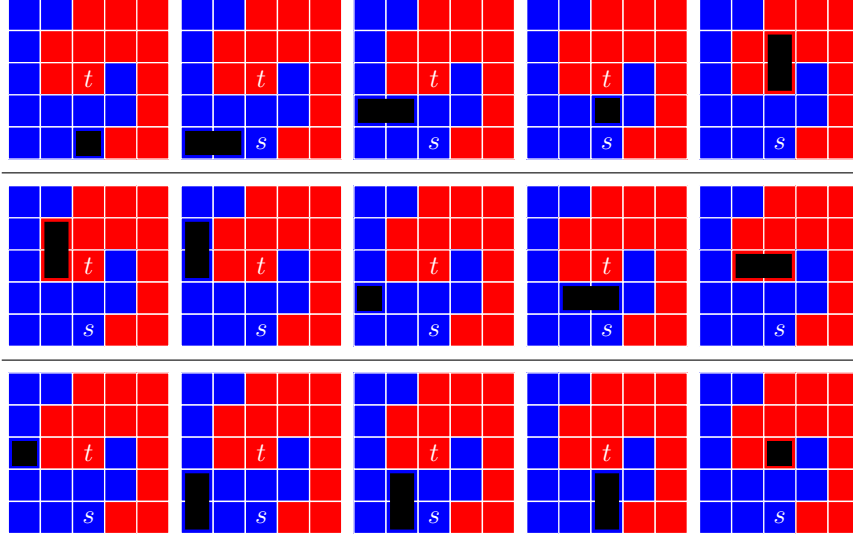
**Corollary 2.** *Solving a colour maze with an unbounded number of blocks is PSPACE-complete, even if all blocks are of size  $2 \times 1 \times 1$ .*  $\square$

Of course, in colour mazes with block size  $1 \times 1 \times 1$ , the colour does not matter—these mazes are just rolling block mazes. Furthermore, rolling block mazes with block size  $1 \times 1 \times 1$  can also be seen as sliding block puzzles with  $1 \times 1$  sized blocks. The next theorem shows that the PSPACE-completeness result for  $2 \times 1 \times 1$  blocks is best possible in the case with no forbidden squares.

**Theorem 3.** *Solving a rolling block maze without forbidden squares and with an unbounded number of cubes, i.e., blocks of size  $1 \times 1 \times 1$ , is trivial.*

*Proof.* If all squares are filled with cubes, the maze is unsolvable since no block can move. Thus assume that there is at least one square, where no cube resides on. We call a square not holding a cube a hole. In order to roll an arbitrary cube to either direction we do as follows: if a hole is in the chosen direction





**Fig. 5.** Colour maze of size  $5 \times 5$  and its solution shown from left to right and top to bottom with the following movement of the  $2 \times 1 \times 1$  block: left, up, right, up, left, left, down, right, up, left, down, right, right, and up. Note that the block is not displayed in 3D, in effect one sees only its shadow (vertical at  $s$  and  $t$ ). This example comes from [www.clickmazes.com/roll/ixroll.htm](http://www.clickmazes.com/roll/ixroll.htm).

on the neighbouring square, we simply roll the cube in the desired direction. Otherwise, we first must bring a hole to the appropriate neighbouring position by not changing the position of the cube under consideration, and then roll the cube to the designated direction. Since there are no forbidden squares we can always perform this task. Bringing some hole to a distinguished position, excluding one square position of the maze, is easy because one has to search for a path starting at the hole, excluding the mentioned square position, and ending at the distinguished target, such that every square, except the first one, contains a cube. It is easy to see that such a path always exists. Thus, one can move a distinguished block from  $s$  to  $t$  by repeatedly applying the above strategy. This shows that solving a rolling block maze *without* forbidden squares and with an unbounded number of cubes is trivially solvable.  $\square$

In the proof of the previous theorem it was essential that no forbidden squares exist. In case they do, one can show that solving a rolling block maze with forbidden squares and an unbounded number of cubes (a single cube, respectively) is at least as hard as UGGR under  $AC^0$  reducibility. The straight forward construction is left to the interested reader.

### 3.2 Rolling Block Mazes with a Bounded Number of Blocks

In rolling block mazes with a bounded number of blocks it turns out that forbidden squares are very important for the computational complexity, since without these squares the problem under consideration becomes trivial, not only for the case that only cubes are rolled, but in general.

**Theorem 4.** *Solving a rolling block maze without forbidden squares and a bounded number of blocks is trivial.*

*Proof.* Let  $k$  refer to the number of blocks and assume the largest block to be of size  $\ell \times 1 \times 1$ . Let  $n = 2 \cdot k \cdot \ell$ . We claim that every rolling block maze of size at least  $n \times n$  is trivially solvable. Since the number of rolling block mazes that are smaller in size is finite, we can make a precompiled list, that contains the answer to whether the given rolling block maze is solvable or not. It remains to prove the claim above. The case  $k = 1$  can easily be verified, thus let  $k \geq 2$ . Let an  $n \times n$  rolling block maze be given and without loss of generality we assume that the target position is to the lower right—if not, we rotate the maze accordingly. The solution runs in three phases: (1) First all blocks are moved into the lower right corner of size  $k \cdot \ell \times k \cdot \ell$ , (2) then all blocks are arranged side by side in the upper left corner and (3) finally, the target block is moved onto its target position.

First we scan the maze from right to left until we see a block, we have not moved yet, choosing the topmost, whenever two or more blocks are encountered simultaneously. We roll this block rightwards as far as possible. Note that the leftmost part of the block, independent of its orientation, will occupy a square at a distance of at most  $\ell$  squares from the next obstacle (wall or block) on the right. Thus, after moving all  $k$  blocks rightwards as far as possible, they occupy at most  $k \cdot \ell$  columns from the right. Then we move all blocks downwards as far as possible in the same manner. Except for the lower right quarter of size  $k \cdot \ell$ , the maze is now free of blocks.

Then we take the leftmost block from the lower right quarter of the maze, choosing the topmost if there are more than one, and roll it to the left as far as possible. Since the lower left quarter is free of blocks, the block comes to lie or stands in the leftmost  $\ell$  columns. Then we change its orientation, so that it lies vertically. If the block is standing, we just have to topple it upwards, whereas a horizontal lying block has to be set upright first. Since  $k \geq 2$ , we have at least  $\ell$  free squares above the block even if a block already lies in the top  $\ell$  rows of this column, so this can be done. If this is the  $i$ th block we moved in this phase, we now roll it into the  $i$ th column and there, roll it upwards as far as possible. It will then stand or lie in the top  $\ell$  rows of the  $i$ th column. This is done until all  $k$  blocks are arranged side by side in the first  $k$  columns and the upper  $\ell$  rows of the maze.

Finally, we can now easily move the target block to its target position on the lower right quarter of the maze. Let the target square be in column  $x$ , line  $y$  and let  $\ell_0 \leq \ell$  be the length of the target block. We first roll the target block downwards until it lies vertically and has free space to its left and right, and then roll it rightwards or leftwards into a column  $c$  such that  $x - c$  is a multiple of  $\ell_0 + 1$ . Then we roll it down (it now stands upright) and left (or right, if we are too close to the leftmost wall), so that the block lies horizontally in columns  $c - \ell_0$  to  $c - 1$ . Now we can roll it downwards into the  $y$ th line and then roll it rightwards onto its target position.  $\square$

Thus, in the forthcoming we only consider rolling block mazes with forbidden squares.

**Theorem 5.** *Solving a rolling block maze with a constant number of blocks can be done in deterministic logspace.*

*Proof.* Given a rolling block maze with a constant number  $k$  of blocks. Then the number of configurations in the rolling block maze game is polynomially bounded by the number of grid cells to the power of  $k$  times the possible orientations of the blocks. It is easy to see that the construction of the configuration graph can be done in deterministic logspace. Moreover, since every move in the game can be undone, the edge relation of the configuration graph is symmetric and thus it is an undirected graph. Since UGR can be solved in deterministic logspace, the stated claim follows.  $\square$

In case we consider rolling block mazes with a single *cube* the problem is trivially equivalent to UGGR, because the labyrinth is an undirected grid graph itself and the single cube mimics the searching of the maze. Thus we can state the following result:

**Theorem 6.** *Solving a rolling block maze with a single cube is equivalent to UGGR under  $AC^0$  reducibility.*  $\square$

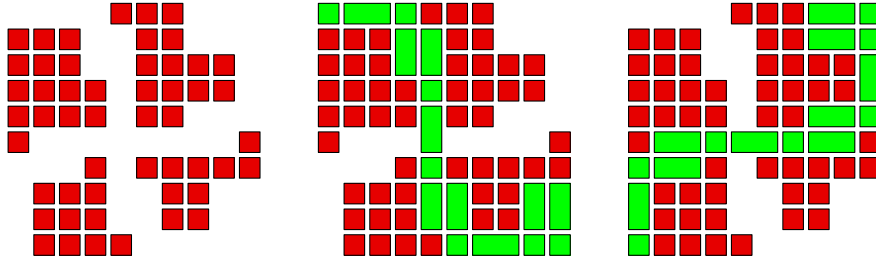
If the block size is larger, the complexity slightly increases to UGR equivalence and thus in turn to L-completeness.

**Theorem 7.** *Solving a rolling block maze with a bounded number of blocks is L-complete, even for a single block of size  $2 \times 1 \times 1$ .*

*Proof.* For a constant number of blocks, there is only a polynomial number of game configurations, inducing a graph by the legal move relation. This graph is undirected since every move can immediately be undone. Thus, an appropriate UGR question solves the rolling block maze, which proves containment within L by [12]. For L-hardness, we reduce the reachability problem for undirected grid graphs with diagonal edges to rolling block mazes with forbidden squares and a single block of size  $2 \times 1 \times 1$ . Since it is shown in [10] that this UGGR problem with diagonal edges is  $AC^0$  equivalent to UGR, which in turn is L-complete, the result follows.

Given a UGGR instance with X-crossings  $G$  of size  $n \times n$ , we construct a rolling block maze  $G'$  as follows: Assume line-column coordinates for the vertices in  $G$ . Then  $G'$  has size  $9n \times 9n$  and at first, we mark all squares as forbidden. For each vertex  $(i, j)$  in  $G$ , we free the square  $(9i, 9j)$  in  $G'$ . Let  $s$  and  $t$  be the starting and target vertices of  $G$ , then the movable  $2 \times 1 \times 1$  block is placed upright on  $9s$  and has to be rolled onto  $9t$  facing upright. For each horizontal edge  $((i, j), (i, j + 1))$  in  $G$  we free the squares  $(9i, 9j + x)$ , for  $1 \leq x \leq 8$ , forming a lane between the corresponding vertex-squares of length 8. Here the block can be rolled exactly six times, resulting in an up-right orientation on the next vertex-square. For vertical edges  $((i, j), (i + 1, j))$  we proceed analogously, freeing the squares  $(9i + x, 9j)$ , for all  $1 \leq x \leq 8$ . The submaze of size  $10 \times 10$  for the X-crossing and the paths for the block are depicted in Figure 6.

Note that the block always reaches the crossing center in a lying position, so it cannot turn off its way. Although the block could leave this submaze in



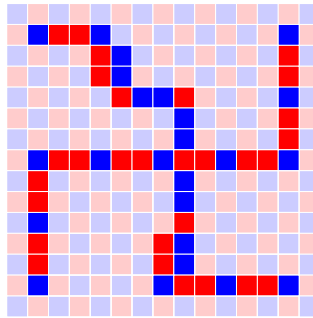
**Fig. 6.** The X-crossing submaze: Red squares are forbidden squares, green fields show the rolling path of the block.

a “wrong” way, e.g., lying and not standing on the lower right vertex-square, it cannot enter other X-crossing submazes in any other than the intended way and even cannot take bends from vertical to horizontal edges. Thus, the block can be rolled from its initial to the target position if and only if there is a path from  $s$  to  $t$  in the UGGR instance with X-crossings  $G$ .  $\square$

A very similar construction proves L-hardness for colour mazes. Here we cannot use forbidden squares, but a grid colouring as shown in Figure 7 has about the same effect on a non-cube block as forbidden squares do. The figure shows a colour maze that simulates an X-crossing. It should be clear that by using this submaze, one could design an appropriate colour maze from an UGGR instance with X-crossings.

**Corollary 8.** *Solving a colour maze with a bounded number of blocks is L-complete, even if all blocks are of size  $2 \times 1 \times 1$ .*

*Proof.* For containment in L, the same argumentation as for rolling block mazes applies to colour mazes, too. For L-hardness, UGGR with X-crossings can be reduced to rolling block mazes in a very similar way as for rolling block mazes. Therefore, each vertex  $(i, j)$  in the UGGR instance is transformed to a blue square at position  $(12i, 12j)$  in the to be constructed colour maze. Horizontal and vertical edges are represented as a path of alternating  $2 \times 1$  sized red and  $1 \times 1$  sized blue areas between the corresponding two blue vertex-squares. An appropriate submaze for simulating an X-crossing is depicted in Figure 7.



**Fig. 7.** A colour maze subgame simulating an X-crossing.

As also seen there, the remaining colouring of the maze is an alternation of red and blue squares, such that the vertex-squares match this colouring. Since all paths, the block should roll along, are embedded by this alternating colour structure, the block cannot leave the path from an upright position, for it needs two uni-coloured squares to lie on. From a lying position, the block could leave the path onto a “forbidden” square, but from there on, it only can be rolled back onto the path.  $\square$

## 4 Alice Mazes

An Alice<sup>1</sup> maze is played on an  $m \times n$  rectangular board where each square is labeled with a set of vertical, horizontal, and diagonal arrows, further a square can be marked as speed increasing (green arrows) or speed decreasing (red arrows). An arbitrary number of tokens is initially placed on the board, each with its own initial speed. The goal is to move a distinguished token from its starting position  $s$  to some target position  $t$ . A token can leave a square only in one of the directions designated by the arrows on the square, and it moves the number of squares according to the token’s speed. All squares over which the token moves must be free of other tokens. If a token finishes its movement on a speed increasing or decreasing square, its speed is increased or, respectively, decreased by 1. An example maze with its solution is shown in Figure 8.

### 4.1 Alice Mazes with an Unbounded Number of Tokens

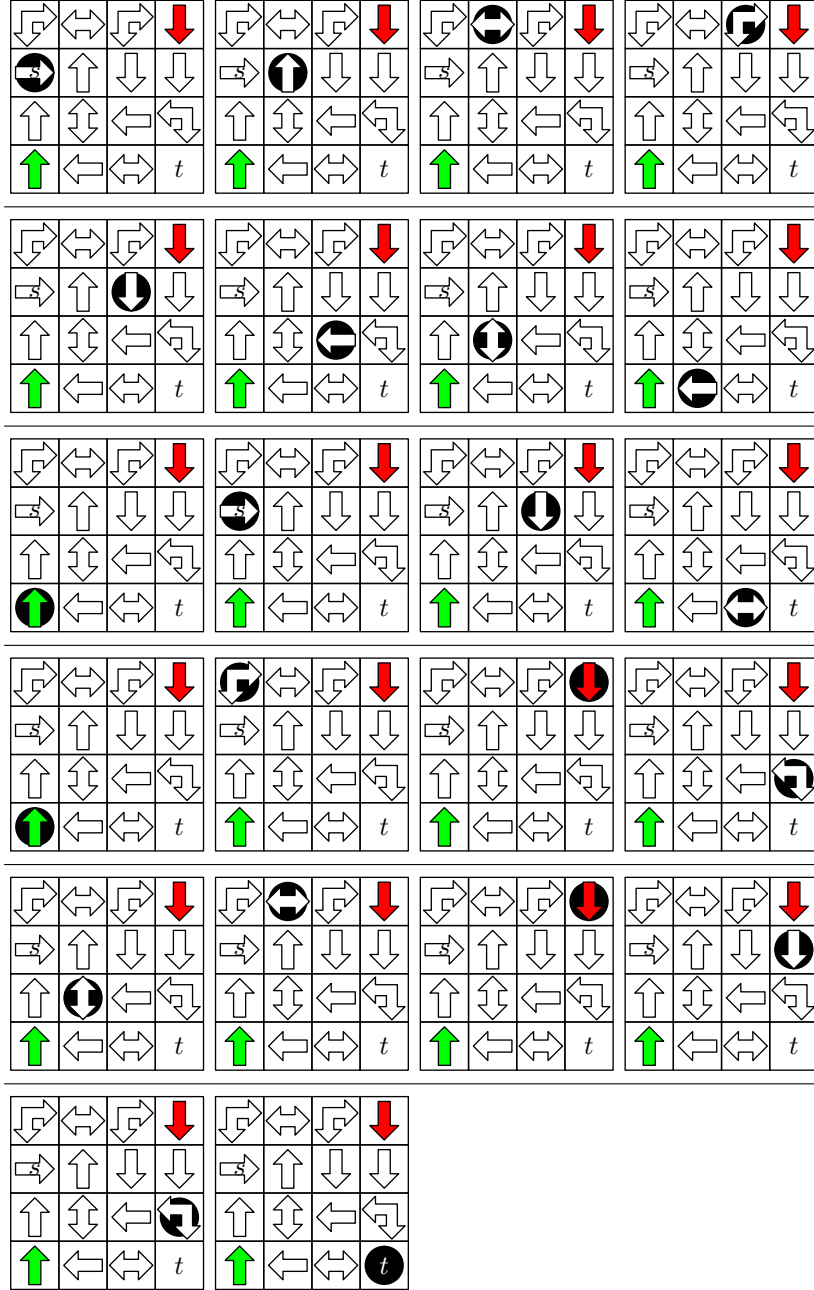
As in the case of rolling block mazes we first show PSPACE-completeness for Alice mazes with an unbounded number of tokens, where we use the NCL framework again.

**Theorem 9.** *Solving an Alice maze with an unbounded number of tokens is PSPACE-complete.*

*Proof.* For Alice mazes, a similar argumentation as in the case of rolling block mazes applies. The problem is in PSPACE, because it can be solved nondeterministically using polynomial space. For proving PSPACE-hardness, we reduce NCL to Alice mazes with an unbounded number of tokens, by constructing submazes simulating AND- and OR-vertices and their connections. We use the notation  $(X1 \leftarrow X2 \leftarrow X3 \leftarrow \dots \leftarrow Xk)$  to describe the sequence of moves where a token lying on square  $X2$  is moved onto square  $X1$ , then the token on  $X3$  is moved onto square  $X2$ , and so on.

We start by describing the AND-submaze. Figure 9 shows an Alice maze corresponding to an AND-vertex, where the “blue edge” on the left is faced inward (because there is a token on  $A6$ ) and the “red edges” on the right are

<sup>1</sup> On his web site [www.logicmazes.com](http://www.logicmazes.com), Robert Abbott explains the name of this game as follows: “These are called ‘Alice’ mazes because they recall the scene in *Alice in Wonderland* where Alice eats a piece of cake with the sign ‘Eat Me’ and grows larger, then she drinks from a bottle marked ‘Drink Me’ and becomes smaller. These mazes won’t make you larger or smaller, but the distance you travel in a move will get larger or smaller.”



**Fig. 8.** An Alice maze with speed increasing (green) and speed decreasing (red) squares. The maze can be solved by moving the token, which has an initial speed of 1, from square  $s$  to square  $t$ , by taking the following moves: right, up, right, down, down, left, down, left, up, right, down, left, up, right, down, down, and down. This example is the fourth maze from [www.logicmazes.com/alice.html](http://www.logicmazes.com/alice.html).

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
								↻	↻	↻		↻		↻	1
						↻	↻	↻		↻	↻		↻		2
						↻	↻	↻							3
						↻		↻	↻	↻		↻		↻	4
	↻		↻	↻		↻				↻	↻		↻		5
↻		↻		↻	↻										6

**Fig. 9.** Alice maze simulating an AND-vertex. The squares  $A6$ ,  $B5$ , and  $C6$  correspond to the “blue” edge, which is facing inward because there is a token on  $A6$ . Squares  $O1$ ,  $N2$ , and  $M1$ , and squares  $O4$ ,  $N5$ , and  $M4$  correspond to the “red edges,” which are facing outward because there are no tokens on  $O1$  and  $O4$ .

faced outward (because  $O1$  and  $O4$  are free). To reverse edges in this AND-submaze, one can first take the following moves to free  $I2$  and  $H3$ : ( $O1 \leftarrow M1 \leftarrow L2 \leftarrow K2 \leftarrow J1 \leftarrow I1 \leftarrow I2$ ) and ( $O4 \leftarrow M4 \leftarrow L5 \leftarrow K5 \leftarrow J4 \leftarrow I4 \leftarrow I3 \leftarrow H3$ ). The resulting configuration is depicted in Figure 10. Since now all three squares  $O1$ ,  $O4$ , and  $A6$  carry a token, that configuration corresponds to an AND-vertex where all three edges are turned inward.

With  $I2$  and  $H3$  being free, the token on  $G4$  can now go to  $I2$  and the left edge can be reversed as follows: ( $I2 \leftarrow G4 \leftarrow G5 \leftarrow F6 \leftarrow E6 \leftarrow C6 \leftarrow B5 \leftarrow A6$ ). This results in the configuration shown in Figure 11. Note, that all tokens on the red arrows and the token on  $I2$  have speed 2 and the others have speed 1. In this configuration,  $A6$  is free, which means that the “blue edge” on the left is turned outward, and  $O1$  and  $O4$  are occupied, which means that the corresponding “red edges” are facing inward. The token on  $I2$  is blocking both ways from  $O1$  and  $O4$ , preventing that any one of the “red edges” is turned outward unless the “blue edge” at  $A6$  is turned inward again.

To reverse the edges again, first take the following moves to free  $I2$ : ( $H3 \leftarrow H2$ ), ( $A6 \leftarrow C6 \leftarrow D5 \leftarrow E5 \leftarrow E6 \leftarrow G4 \leftarrow G3 \leftarrow G2 \leftarrow I2$ ). This results in the configuration depicted in Figure 12. Now we can free  $O1$  and  $O2$  again by taking the following moves, resulting in the initial configuration: ( $I2 \leftarrow I3 \leftarrow$

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
								↻	↻	↻		↻		↻	1
						↻	↻	↻		↻	↻		↻		2
						↻	↻	↻							3
						↻		↻	↻	↻		↻		↻	4
	↻		↻	↻		↻				↻	↻		↻		5
↻		↻		↻	↻										6

**Fig. 10.** Alice maze simulating an AND-vertex, where all edges are facing inwards.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
								↻	↻	↻		↻		↻	1
						↻	↻	↻		↻	↻		↻		2
						↻	↻	↻							3
						↻		↻	↻	↻		↻		↻	4
	↻		↻	↻		↻				↻	↻		↻		5
↻		↻		↻	↻										6

**Fig. 11.** Alice maze simulating an AND-vertex, where the “red edges” face inward (there are tokens on  $O1$  and  $O4$ ), and the blue edge is facing outward ( $A6$  is free).

$J4 \leftarrow K4 \leftarrow M4 \leftarrow N5 \leftarrow O4$ ), ( $H2 \leftarrow I2$ ), and ( $I2 \leftarrow J1 \leftarrow K1 \leftarrow M1 \leftarrow N2 \leftarrow O1$ ).

Next we argue, that we cannot cheat in the following sense: no token can ever leave its submaze and we cannot reach the situation where  $A6$  and  $O1$  or  $O4$  are free at the same time. First note, that each white and red arrows directly point at an adjacent gadget field, so that tokens having speed 1 cannot leave the submaze from these fields. Tokens leaving green arrows (with speed 2) come to lie on red arrows (slowing them down to speed 1 again) except for the case where a token leaves  $G4$  for  $I2$ . This is the only situation where a token lies on a white arrow field and has speed 2. But from  $I2$  the token can only go to  $G2$  where it gets slowed down again. It cannot take the other direction, because if  $I2$  is occupied by a speed 2 token, then this token must have come from  $G4$ , which means that then  $G4$  and  $H3$  are the only two free fields of the submaze, in particular,  $I1$  is occupied and thus, blocks the token on  $I2$ . So, no token ever leaves its submaze.

We have already seen that for freeing  $A6$ , one must first occupy  $O1$  and  $O4$  and move the token from  $G4$  to  $I2$ . As long as this token, which has speed 2, sits on  $I2$ , it blocks both fields  $J1$  and  $I3$ , so none of the fields  $O1$  and  $O4$  can be freed. In order to free  $I2$  again, we have to clear  $H2$  and  $G2$ , but for clearing  $G2$ , we finally have to occupy  $A6$  again. This means, that before directing a “red edge” ( $O1$  or  $O4$ ) outward, we must first direct the “blue edge” ( $O6$ ) inward.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
								↻	↻	↻		↻		↻	1
						↻	↻	↻		↻	↻		↻		2
						↻	↻	↻							3
						↻		↻	↻	↻		↻		↻	4
	↻		↻	↻		↻				↻	↻		↻		5
↻		↻		↻	↻										6

**Fig. 12.** Alice maze simulating an AND-vertex, all edges are facing inwards.



Now we discuss the OR-submaze, shown in Figure 13 in its initial configuration where the left edge is facing inward ( $A5$  is occupied) and both edges on the right face outward ( $M1$  and  $M4$  are vacant). Since in the whole submaze there are only two free fields, at most two edges can be directed outward at the same time. To see that no token can leave the submaze, note that all red and white arrows directly point to gadget fields and tokens leaving green arrows always come to lie on a red arrow.

A	B	C	D	E	F	G	H	I	J	K	L	M	
								●		●		●	1
						↓	↔	↔	↔		↖		2
						↖							3
	↘		↖	↔	↔	↖		●		●		●	4
●		●		●		↖	↔	↔	↖		↖		5

**Fig. 13.** Alice maze simulating an OR-vertex. The left “blue edge” faces inwards, because there is a token on  $A5$ , the other “blue edges” face outward, because  $M1$  and  $M4$  are vacant.

It can easily be checked, that for freeing  $A5$ , one has to move tokens towards  $M1$  or  $M4$ . For the remaining situations, suppose  $A5$  has to stay free, i.e. the left edge needs to face outward. Then one can switch from the configurations where  $M1$  is occupied while  $M4$  is free to the state where  $M1$  is free while  $M4$  is occupied by moving as follows: ( $M4 \leftarrow K4 \leftarrow \dots \leftarrow G4 \leftarrow F4 \leftarrow G3 \leftarrow G2 \leftarrow \dots \leftarrow K1 \leftarrow L2 \leftarrow M1$ ). The backward sequence is similar.

Finally we explain how the information, if a vertex-submaze’s edge is turned outward or inward, is propagated to the submaze on the other side of the edge. Figure 14 shows a connection-submaze that simulates an edge that is directed to the left, which is represented by  $C2$  being free and  $I2$  being occupied. The fields  $I2, J1, K2$  correspond to the fields  $A6, B5, C6$  in an AND-submaze or to the fields  $A5, B4, C5$  of an OR-submaze respectively. The fields  $C2, B3, A2$  correspond to the fields in the columns  $M, N, O$  in an AND-submaze or the columns  $K, L, M$  in an OR-submaze. In particular, this means that tokens on these fields can only leave towards the connected submazes and cannot go further into the connection-submaze. Also note, that the other tokens in the connection-submaze cannot leave it either, so there are three disjoint cycles in this gadget. The field  $G1$  ( $E3$ ) is connected to  $E1$  ( $G3$ , respectively) by one-way white arrow fields, each occupied with a token. These paths can be used for bended and arbitrary long connections.

Suppose, the left vertex-submaze wants the edge to be turned inward, which means that  $A2$  has to be moved to  $C2$ . As long as  $B2$  is occupied, this move is forbidden, and in order to move  $B2$  we have to shift  $D2, D3, \dots, H3$  and  $I3$ . The token on  $I3$  however, is blocked by the token on  $I2$  which can only be moved if  $K2$  moves out of the connection-submaze, triggering the vertex-submaze on the right side of the edge. Then the necessary moves can be done and the resulting configuration is shown in Figure 15.

A	B	C	D	E	F	G	H	I	J	K	
		↓	←	←		←	←	↘	↘		1
→	→	↖	↓				↓	↖	→	→	2
	↖	↖	←	←		←	←	↑			3

**Fig. 14.** Alice maze for connecting the vertex-submazes. The three squares  $A2$ ,  $B3$ , and  $C2$  and the three squares  $K2$ ,  $J1$ , and  $I2$  correspond to the three squares simulating an edge in the AND-vertex and OR-vertex submazes. Since there is a token on  $I2$ , and the square  $C2$  is free, the edge faces towards the vertex-submaze on the right hand side of the connection.

The token on  $A2$  is now able to go to  $C2$  (and in the connected gadget, another token can now go to  $A2$ ). Then, moving the token on  $C3$  to  $B2$  results in the reversed initial configuration. If we move  $C3$  to  $B2$  first, we cannot move  $A2$  to  $C2$  anymore but can only take cycling moves ( $C3 \leftarrow C1 \leftarrow D1 \leftarrow \dots \leftarrow H2 \leftarrow J2 \leftarrow I1 \leftarrow I3 \leftarrow H3 \leftarrow \dots \leftarrow D3 \leftarrow B2$ ) and so on until we free  $B2$  so that  $A2$  can move right.  $\square$

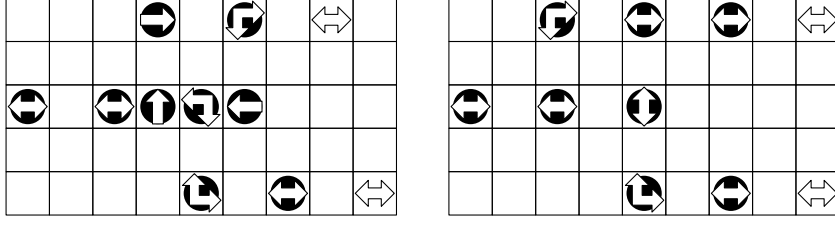
In the previous proof we used a lot of speed changing fields. One could also consider Alice mazes without such fields, so the speed of the tokens cannot change. In such a case one could still define an individual initial speed for each of the tokens, that stays constant throughout the game. In this setting, we still can prove PSPACE-hardness, if each token is assigned an initial speed of two, and we even do not need diagonal arrows for the constructions. Figure 16 shows how NCL AND- and OR-vertices can be simulated in such an Alice maze, and mazes that are used to connect these vertex-mazes are shown in Figure 17. Again, these mazes also ensure, that no token can leave its submaze. Note that because all tokens have speed two, the edges between the vertex-mazes basically lie only on even numbered rows and columns. Only the upper right edge of the AND-vertex maze is shifted onto odd numbered column fields. This can be corrected using the shifting connection-submaze on the left-hand side of Figure 17. Thus, we can state the following.

**Theorem 10.** *Solving an Alice maze with an unbounded number of tokens is PSPACE-complete, even for mazes without any speed changing fields, and without diagonal arrows.*  $\square$

If we set speed two only for the single distinguished token, that has to be moved onto the target square, and speed one for all other tokens, the solvability

A	B	C	D	E	F	G	H	I	J	K	
		↓	←	←		←	←	↘	↘		1
→	→	↖	↓				↓	↖	→	→	2
	↖	↖	←	←		←	←	↑			3

**Fig. 15.** Alice maze for connecting the vertex-submazes. The right vertex gadget turned its edge outward by freeing  $I2$  (this is why there is a token “outside” the connection), and the left vertex gadget may now turn its edge inward by occupying  $C2$ .



**Fig. 16.** Alice mazes where all tokens have a constant speed of two. The maze on the left simulates an AND-vertex with the “blue edge” on the left facing inward (because the leftmost square is occupied by a token) and the “red edges” on the right facing outward (because the corresponding squares are free of tokens). The right maze simulates an OR-vertex, with its left edge facing inward and its right edges facing outward.

problem stays at least NP-hard, which is shown in the following. Whether this problem can also be solved in NP stays open.

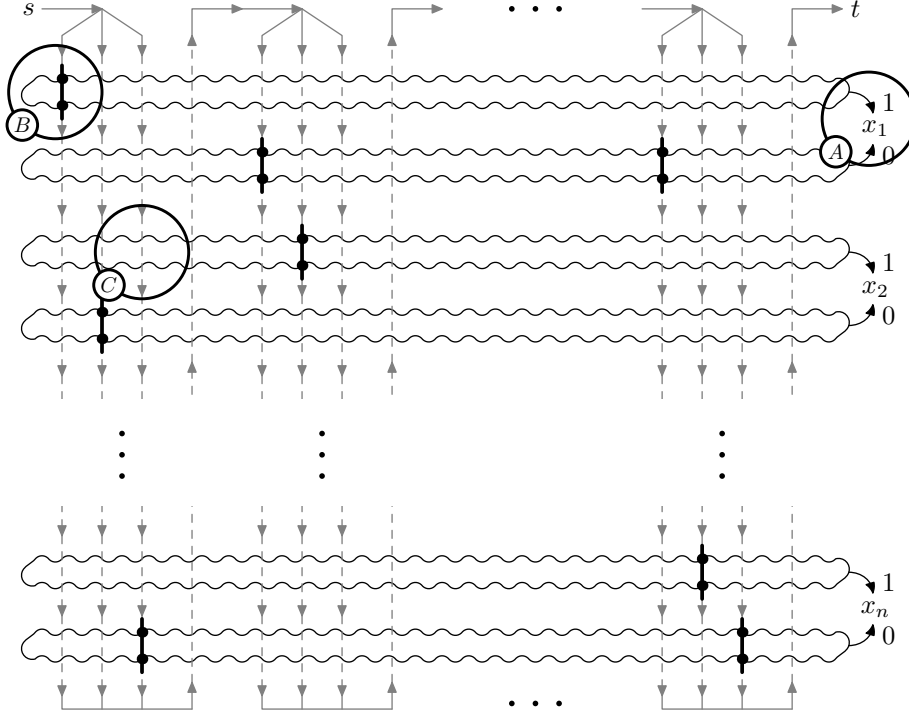
**Theorem 11.** *Solving an Alice maze with an unbounded number of tokens and without any speed changing squares, where at most one token has a speed greater than 1, is NP-hard.*

*Proof.* We reduce 3-SAT to our problem: given a Boolean formula  $\varphi$  in 3-CNF with  $k$  clauses  $c_1, c_2, \dots, c_k$  over a set of  $n$  variables  $x_1, x_2, \dots, x_n$ , we construct an Alice maze such that a distinguished block can be moved to a target square if and only if  $\varphi$  is satisfiable. The scheme of the constructed maze is shown in Figure 18. For each variable, we have a horizontal variable gadget consisting of two variable cycles, which correspond to the two possible truth assignments. For each clause we have a vertical clause gadget, consisting of three paths corresponding to the literals of the clause. These literal paths cross each variable cycle in such a way, that the literal path checks exactly the truth value of its corresponding literal. The distinguished token in the upper left corner has speed 2 and has to pass through each clause gadget in order to reach the marked square in the upper right corner. Further, in each variable gadget there is only one free square, which is used to assign a truth value to the variable, all the other squares of the variable gadgets are occupied by tokens of speed 1.

We now come to the details of the construction. A variable  $x_i$  is assigned *true* (*false*), by moving the three tokens leading to the upper (*lower*, respectively) variable cycle one step towards the free square labeled  $x_i$  (cf. Figure 19, A). This gives one hole that can be shifted through the variable cycle for that truth assignment, and also makes all tokens on the other variable cycle unmovable. The token with speed two can pass a clause gadget through any one of the



**Fig. 17.** Alice mazes that can be used to connect the vertex simulating mazes from Figure 16, and to prevent tokens from leaving their submazes. In the depicted configurations, the edge points from left to right, because the leftmost square is occupied, while the rightmost square is free. The maze on the left is used to rectify the shift of one of the edges from the AND-vertex simulating maze.



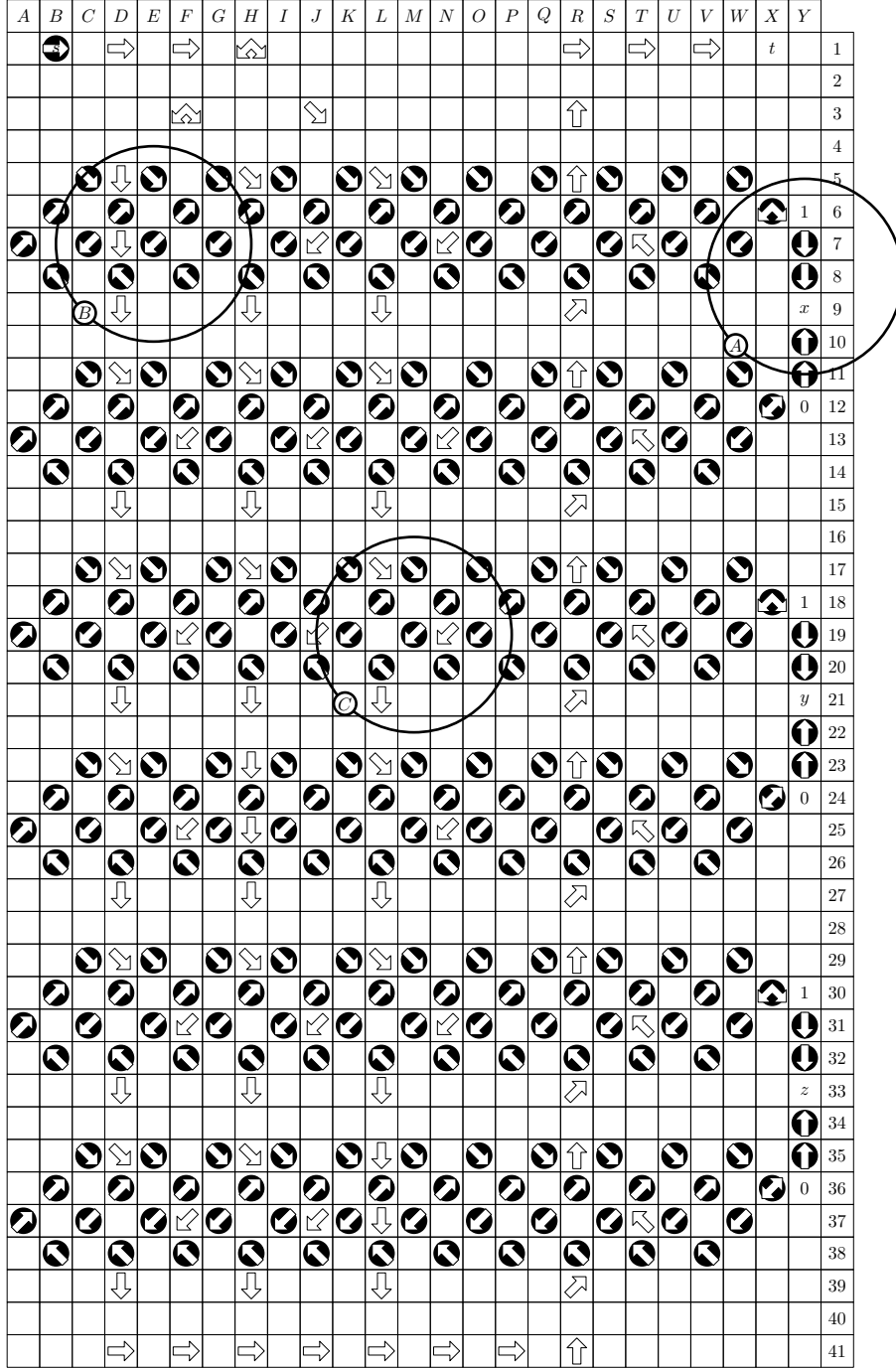
**Fig. 18.** Scheme of an Alice maze constructed from a 3-SAT instance. The exact realizations of a variable setting part A, a literal checking B, and a literal bypass C can be seen in the example in Figure 19.

three literal paths. A path corresponding to the literal  $x_i$  ( $\bar{x}_i$ ) can only pass the upper (lower, respectively) cycle of the variable gadget  $x_i$ , if this cycle has a hole (cf. Figure 19, B): if the token sits on  $D5$ , it can only move to  $D7$ , if there is no token on  $D6$ , which can be done, if and only if the variable  $x_i$  is set *true* (*false*, respectively). Similarly, from  $D7$  the token can then move to  $D9$ , by shifting the hole through the cycle to position  $D8$ . All other variable cycles can be bypassed as shown in Figure 19, C. So the token with speed 2 can pass a clause gadget if and only if there is a truth assignment to the variables that satisfies the respective clause. Since the token has to pass each clause gadget, the constructed Alice maze is solvable if and only if the underlying Boolean formula  $\varphi$  is satisfiable.  $\square$

The complexity of Alice mazes without any speed changing squares, and where each token has speed 1 stays open. We only have a PSPACE upper bound and, as shown in the next section, an NL lower bound.

#### 4.2 Alice Mazes with a Bounded Number of Tokens

Similar to rolling block mazes, if the number of tokens in an Alice maze is bounded by some constant, the problem becomes easier. In contrast to rolling blocks, a move in an Alice maze can in general not be undone. This is reflected in a slightly increased complexity, since an Alice maze can simulate directed graphs, whereas rolling block mazes only allow simulation of undirected graphs.



**Fig. 19.** Alice maze constructed from the formula  $\varphi = (x \vee \bar{y} \vee \bar{z})$ . The highlighted parts implement the variable setting (A), the literal checking (B), and the literal bypass (C).

Block $\ell \times 1 \times 1$	Rolling block maze variant			
	bounded ( $k$ blocks)		unbounded	
	no obstacles	obstacles	no obstacles	obstacles
$\ell = 1$	trivial	$\cdot \equiv \text{UGGR}$ , for $k = 1$	trivial	
		$\text{UGGR} \leq \cdot \in \mathbf{L}$		
$\ell$ with $\ell \geq 2$		$\mathbf{L}$ -complete, even for $k = 1$	$\mathbf{PSPACE}$ -complete	

**Table 1.** Rolling block maze results for bounded and unbounded number of blocks of size  $\ell \times 1 \times 1$ —here  $\equiv$  and  $\leq$  refer to  $\mathbf{AC}^0$  equivalence and  $\mathbf{AC}^0$  reducibility, respectively.

**Theorem 12.** *Solving an Alice maze with a bounded number of blocks is  $\mathbf{NL}$ -complete, even without any speed changing squares and for a single block of speed 1.*

*Proof.* Note that the speed of each token is bounded by the size of the maze, because faster tokens would not be able to move, and thus have speed 0. This means that for a constant number of blocks there is only a polynomial number of game configurations. Then the problem can be solved by an appropriate graph reachability question on the underlying (directed) configuration graph, and thus is contained in  $\mathbf{NL}$ .

For  $\mathbf{NL}$ -hardness one can easily reduce the reachability problem for (directed) grid graphs with diagonal edges to Alice mazes as described below. Since in [10] this problem is shown to be  $\mathbf{NL}$ -complete, the result follows. Let  $G = (V, E)$  be a grid graph with diagonal edges and  $s, t \in V$  be the starting and target nodes. For each node  $v = (i, j) \in V$  we have a square in the Alice maze. The arrows on the squares correspond in a natural way to the outgoing edges of  $v$ . For example, if there is an edge to node  $(i, j + 1)$  to the right, then the corresponding squares carries an arrow pointing to the right, if there is an edge to the node  $(i + 1, j - 1)$  to the lower left of  $v$ , then the squares carries an arrow to the lower left, and so on. Then a token of speed 1 placed on the square corresponding to the starting node  $s$  can be moved to the target square corresponding to node  $t$  if and only if in  $G$  there is a path from  $s$  to  $t$ .  $\square$

## 5 Conclusions

We have investigated the computational complexity of rolling block mazes and Alice mazes. It turns out that when the number of blocks in a rolling block or colour maze, or the number of tokens in an Alice maze is unbounded, then the problem of solving such a maze becomes  $\mathbf{PSPACE}$ -complete. Here,  $\mathbf{NCL}$  turned out to be a great framework for proving  $\mathbf{PSPACE}$ -hardness. On the other hand, if the number of blocks or tokens is bounded by a constant, these problems are closely related to graph reachability questions. Our findings are summarized in Tables 1 and 2.

As the reader can see, in both tables the exact complexity of some problems remains open: in some unbounded cases we have  $\mathbf{PSPACE}$  upper bounds, but non matching lower bounds. In particular, for rolling block mazes with  $1 \times 1 \times 1$  sized blocks we have a lower bound of  $\text{UGGR}$ -hardness, for Alice mazes without

speed of tokens	Alice maze variant	
	bounded ( $k$ tokens)	unbounded
all tokens have speed 1	NL-complete, even for $k = 1$	$\text{NP} \leq \cdot \in \text{PSPACE}$
all tokens speed 1, except a single token of speed 2		
all tokens have speed 2		PSPACE-complete
speed changing		

**Table 2.** Alice maze results for bounded and unbounded number of tokens and different speed rules—here  $\leq$  refers to  $\text{AC}^0$  reducibility.

speed changing squares and only tokens of speed 1 we have an NL lower bound, and Alice mazes without speed changing squares but with at least one token of initial speed 2 are at least NP-hard.

The rolling block problem for  $1 \times 1 \times 1$  sized blocks, may seem similar to the  $1 \times 1$  Rush Hour problem, but a main difference is that the blocks in Rush Hour puzzles have an orientation, allowing a block only to move either horizontally or vertically. Also the unbounded Alice maze variant without speed changing squares, where all tokens have speed 1 is different from  $1 \times 1$  Rush Hour, because the orientation in Alice mazes is given on the squares of the board, while in Rush Hour, the blocks themselves carry the orientation.

## References

1. E. Allender, D. A. Mix Barrington, T. Chakraborty, S. Datta, and S. Roy. Planar and grid graph reachability problems. *Theory of Computing Systems*, 45(4):675–723, July 2009.
2. K. Buchin and M. Buchin. Rolling Block Mazes are PSPACE-Complete. 2007, unpublished manuscript.
3. K. Buchin and M. Buchin. Rolling block mazes are PSPACE-complete. *J. Inform. Proc.* (2012), to appear.
4. K. Buchin and M. Buchin and E. D. Demaine and M. L. Demaine and D. El-Khechen and S. P. Fekete and C. Knauer and A. Schulz and P. Taslakian. On rolling cube puzzles. In *Proceedings of the 19th Canadian Conference on Computational Geometry*, pages 141–144, Ottawa, Ontario, Canada, August 2007.
5. E. D. Demaine and R. A. Hearn. A uniform framework for modeling computations as games. In *Proceedings of the 23th Annual IEEE Conference on Computational Complexity*, pages 149–162, College Park, Maryland, USA, June 2008. IEEE Computer Society Press.
6. M. Gardner. Mathematical Games Column. In *Scientific American*, 209(6), December 1963.
7. M. Gardner. Mathematical Games Column. In *Scientific American*, 213(5), November 1965.
8. M. Gardner. Mathematical Games Column. In *Scientific American*, 232(3), March 1975.
9. R. A. Hearn and E. D. Demaine. *Games, Puzzles, and Computation*. A K Peters, 2009.
10. M. Holzer and S. Jakobi. Grid Graphs with Diagonal Edges and the Complexity of Xmas Mazes. IFIG Research Report 1201, Institut für Informatik, Justus-Liebig-Universität Gießen, Arndtstr. 2, D-35392 Gießen, Germany, January 2012.
11. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
12. O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):Article 17 (24 pages), September 2008.
13. W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, April 1970.



## Recent Reports

(Further reports are available at [www.informatik.uni-giessen.de](http://www.informatik.uni-giessen.de).)

- M. Holzer, S. Jakobi, *Grid Graphs with Diagonal Edges and the Complexity of Xmas Mazes*, Report 1201, January 2012.
- H. Gruber, S. Gulan, *Simplifying Regular Expressions: A Quantitative Perspective*, Report 0904, August 2009.
- M. Kutrib, A. Malcher, *Cellular Automata with Sparse Communication*, Report 0903, May 2009.
- M. Holzer, A. Maletti, *An  $n \log n$  Algorithm for Hyper-Minimizing States in a (Minimized) Deterministic Automaton*, Report 0902, April 2009.
- H. Gruber, M. Holzer, *Tight Bounds on the Descriptive Complexity of Regular Expressions*, Report 0901, February 2009.
- M. Holzer, M. Kutrib, and A. Malcher (Eds.), *18. Theorietag Automaten und Formale Sprachen*, Report 0801, September 2008.
- M. Holzer, M. Kutrib, *Flip-Pushdown Automata: Nondeterminism is Better than Determinism*, Report 0301, February 2003.
- M. Holzer, M. Kutrib, *Flip-Pushdown Automata:  $k + 1$  Pushdown Reversals are Better Than  $k$* , Report 0206, November 2002.
- M. Holzer, M. Kutrib, *Nondeterministic Descriptive Complexity of Regular Languages*, Report 0205, September 2002.
- H. Bordihn, M. Holzer, M. Kutrib, *Economy of Description for Basic Constructions on Rational Transductions*, Report 0204, July 2002.
- M. Kutrib, J.-T. Löwe, *String Transformation for  $n$ -dimensional Image Compression*, Report 0203, May 2002.
- A. Klein, M. Kutrib, *Grammars with Scattered Nonterminals*, Report 0202, February 2002.
- A. Klein, M. Kutrib, *Self-Assembling Finite Automata*, Report 0201, January 2002.
- M. Holzer, M. Kutrib, *Unary Language Operations and its Nondeterministic State Complexity*, Report 0107, November 2001.
- A. Klein, M. Kutrib, *Fast One-Way Cellular Automata*, Report 0106, September 2001.
- M. Holzer, M. Kutrib, *Improving Raster Image Run-Length Encoding Using Data Order*, Report 0105, July 2001.
- M. Kutrib, *Refining Nondeterminism Below Linear-Time*, Report 0104, June 2001.
- M. Holzer, M. Kutrib, *State Complexity of Basic Operations on Nondeterministic Finite Automata*, Report 0103, April 2001.
- M. Kutrib, J.-T. Löwe, *Massively Parallel Fault Tolerant Computations on Syntactical Patterns*, Report 0102, March 2001.
- A. Klein, M. Kutrib, *A Time Hierarchy for Bounded One-Way Cellular Automata*, Report 0101, January 2001.
- M. Kutrib, *Below Linear-Time: Dimensions versus Time*, Report 0005, November 2000.
- M. Kutrib, *Efficient Universal Pushdown Cellular Automata and their Application to Complexity*, Report 0004, August 2000.
- M. Kutrib, J.-T. Löwe, *Massively Parallel Pattern Recognition with Link Failures*, Report 0003, June 2000.
- A. Klein, M. Kutrib, *Deterministic Turing Machines in the Range between Real-Time and Linear-Time*, Report 0002, February 2000.
- M. Kutrib, J.-T. Löwe, *Fault Tolerant Parallel Pattern Recognition*, Report 0001, February 2000.
- M. Kutrib, *Automata Arrays and Context-Free Languages*, Report 9907, October 1999.