How Pinball Wizards Simulate a Turing Machine

Rosemary U. Adejoh
Faculty of Media, Bauhaus-Universität Weimar, Germany

Andreas Jakoby
Faculty of Media, Bauhaus-Universität Weimar, Germany

Sneha Mohanty
Computer Networks and Telematics, University of Freiburg, Germany

Christian Schindelhauer
Computer Networks and Telematics, University of Freiburg, Germany

Abstract

We introduce and investigate the computational complexity of a novel physical problem known as the *Pinball Wizard* problem. It involves an idealized pinball moving through a maze composed of one-way gates (outswing doors), plane walls, parabolic walls, moving plane walls, and bumpers that cause acceleration or deceleration. Given the initial position and velocity of the pinball, the task is to decide whether it will hit a specified target point.

By simulating a two-stack pushdown automaton, we show that the problem is Turing-complete – even in two-dimensional space. In our construction, each step of the automaton corresponds to a constant number of reflections. Thus, deciding the *Pinball Wizard* problem is at least as hard as the Halting problem. Furthermore, our construction allows bumpers to be replaced with moving walls. In this case, even a ball moving at constant speed – a so-called ray particle – can be used, demonstrating that the *Ray Particle Tracing* problem is also Turing-complete.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Pinball Wizard problem, Halting problem, Turing-complete

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.4

Related Version *Full Version*: `https://arxiv.org/abs/2510.02560` [2]

1 Introduction and Motivation

A question of continuous interest is which physical or analog models are Turing complete. Christopher Moore [16] outlined a construction showing that, even with precisely known initial conditions, friction less movement, perfect reflection and arbitrary precision, a single billiard ball moving at constant speed can simulate a two-counter machine in 3D. Similarly, Reif et al. [20] demonstrated in their seminal work that 3D ray tracing can simulate a reversible two-counter automaton, thereby establishing its Turing-completeness. Both models rely on the use of parabolic mirrors in 3D space, supporting Moore's conjecture that three dimensions suffice to achieve Turing universality.

In this paper, we show that full 3D Euclidean space is not required: by allowing the speed of the ball to vary, we introduce a novel two-dimensional model – referred to as the 2D Pinball Wizard Problem, that can simulate a Turing complete two-stack pushdown automaton (PDA) [12].

Our *Pinball Wizard problem* is inspired by classical pinball machines and features components such as reflecting planes and parabolic walls, one-way gates (outswing doors), bumpers for accelerating and decelerating the ball, as well as a moving plane wall. Unlike in real pinball machines we do not have a tilted plane and the ball, described by a single point, moves in a straight line with constant velocity when it does not bounce into one of the components.

These components may change the direction of the ball's trajectory and the velocity of the pinball when the pinball bounces on these components in its path. See Section 3 for a complete definition of this problem.

The main components of the original pinball game that we retain here, are the one-way gates as well as the target. A one-way gate is an area that is blocked off after the ball passes through it once. It does not change the movement of a ball coming from one side, but will block and reflect the ball if it comes from the other side. We also use bumper walls in our work, which when hit by the pinball, pushes the ball away with a positive or negative change of speed. While there are various types of target in the pinball game, for simplicity, we are working with stationary target position.

The 2D Pinball Wizard model describes a decision problem, where given an initial configuration of the pinball, it is to be decided, whether it hits a given target. We show that this model facilitates the simulation of any two-stack automaton with push and pop operations.

If all stationary bumpers are replaced with a combination of moving walls, the result still holds. We consider the spatial offset interval where the ball enters the construction, and the modified speed determines the time delay (offset) at the end line. These components serve our design of a pinball machine where we want to test if the ball exits at a certain target position given a start time and space offset at the beginning of the path of a pinball having a known speed and direction. Finally, when we exchange the ball with a ray particle with constant speed, a modified construction holds up as well, showing that the *Ray Particle Tracing* problem with moving walls and one-way mirrors is also Turing-complete.

In Section 2, we discuss Related Work. Thereafter, we provide a more formal definition of the *Pinball Wizard* problem in Section 3. That Section concludes with a brief introduction of the *Ray Particle Tracing* problem, and its relationship with our *Pinball Wizard* problem. We then establish the Turing-completeness of the *Pinball Wizard* problem by simulating a two-stack pushdown automaton in Section 4. Subsequently, we modify our construction to briefly demonstrate analogous results for the *Ray Particle Tracing* problem in Section 5. Finally, Conclusions and Open Problems are presented in Section 6.

2 Related Work

The computational complexity of puzzles and games has long fascinated mathematicians and computer scientists. In his survey [21], Uehara outlines the history of this line of research and shows current trends in theoretical computer science. The simple rules found in puzzles and games not only resemble basic computational operations, but can also serve as alternative models of computation. Studying them may lead to deeper insights into the nature of computation itself.

A remarkable result in this context is the Game of Life, popularized by Martin Gardner in [11]. Confirming a long-standing conjecture, Berlekamp et al. [3] showed in 2004 that it is Turing-complete – that is, any computation performed by a Turing machine can be simulated within Conway's Game of Life.

Sometimes, games are invented to gain a better understanding of aspects of computation. In 1974, Cook and Reckhow [6] introduced the pebble game to study storage requirements. This game has proven to be a powerful tool for exploring computational complexity classes such as NL, P, NP, PSPACE, EXP, and others. It also plays an important role in understanding register allocation, parallel computation, and proof complexity.

There is not enough room to cover all the research on the computational complexity of traditional games and puzzles published since the advent of NP-completeness and higher complexity classes such as PSPACE. For example, Demaine et al. in [7] prove that push-pull block puzzles in 3D are PSPACE-complete to solve, and that push-pull block puzzles in 2D with thin walls are NP-hard, thereby settling an open question posed in [24]. Recent progress in the field of combinatorial reconfiguration, particularly through the use of the constraint logic model, has revealed many natural PSPACE-complete problems. These include a variety of sliding block puzzles whose computational complexity had remained open for nearly 40 years.

For example, in [9], Forišek analyzes the computational complexity of various two-dimensional platform games. He states and proves several meta-theorems that identify classes of games for which the set of solvable levels is NP-hard, and others for which the set is even PSPACE-hard.

Pinball Wizard is a very popular game, often found in pubs as a contrived mechanical device or as a video/computer game. In the realm of video games, speedrunning is a popular activity where the goal is to complete a game as quickly as possible. Well-known titles such as Super Mario Bros., Castlevania, and Mega Man are played by enthusiasts worldwide, with countless hours spent daily on livestreams as players refine their skills in pursuit of world records. However, human execution is not the only factor in a successful speedrun. Common techniques such as damage boosting and routing require careful planning to maximize time gains. In [13], Lafond shows that optimizing these mechanics constitutes a significant algorithmic challenge, leading to novel generalizations of classic NP-hard problems such as the knapsack and feedback arc set problems.

The fascination of Pinball Wizard may lie in its physical nature: a ball moving on a plane, representing an analog computational model not captured by traditional digital models discussed so far. Analog models offer many intriguing facets. In [4], Bonifaci et al. provide a rigorous proof that a biologically inspired mathematical model of slime mold behavior converges to the shortest path in any network, regardless of topology or initial mass distribution – formally validating classical maze experiments. They generalize the original model by Nakagaki et al. [17] to show convergence not only for shortest paths but also for undirected linear programs with non-negative cost vectors.

Analog models can display surprising limitations. Pour-El and Richards [18] construct a computable initial value problem – based on well-behaved differential equations with computable coefficients and initial conditions – whose solution is not computable. This striking result demonstrates that physical systems governed by differential equations may yield unpredictable outcomes, even with perfect knowledge of the inputs.

The reversible nature of physical computation is explored in Fredkin and Toffoli's Billiard Ball Model (BBM) [10], in which multiple balls and elastic collisions – akin to carom billiards – are used to simulate logic gates. Margolus [14] extends this model to show that conservative, reversible systems can simulate universal computation, making outcome prediction undecidable in the general case. For further constructions, see Durand-Lose's textbook [8]. As in our work, precise timing is crucial for correctness. Zhang et al. [22] optimize the mirror-based collision computing model by introducing m-counting gates, significantly reducing the number of mirrors required. A comprehensive overview of the field of collision-based computation can be found in [1]. In [5], Chattopadhyay and Gayen have shown that 3-bit XOR and XNOR logic circuits can be constructed using mechanical elements, such as; beam combiners, fixed as well as movable mirrors.

As mentioned earlier, in 1990 Moore [16] envisioned a simulation in which a single billiard ball moving at constant speed in 3D can simulate a two-counter machine. A more detailed proof can be found in the work of Reif et al. [20], showing that 3D ray tracing (the same problem with a different name) is Turing-complete. Their seminal work analyzes six 3D scenarios and was the first to investigate the computational complexity of ray tracing, establishing it as equivalent to the Halting problem. This foundational result has sparked significant theoretical discussions, such as those in [23]. The intricate nature of Ray Tracing has recently inspired a proposal in [15] to employ optical gates using reflection and refraction as foundational elements in a symmetric encryption algorithm.

We are not the first to consider the theoretical aspects of bumpers in a pinball machine. Pring et al. [19] study the dynamics of an impact oscillator with a modified reset law, inspired by a system with "active impact" – namely, the pinball machine. They show how the subtle interplay between two underlying maps gives rise to a large number of new periodic orbits, potentially explaining some of the complexity observed in the dynamics of real pinball machines.

Moore [16] and Reif et al. [20] demonstrated that three spatial dimensions are sufficient for Turing-completeness, leaving open the question of whether two dimensions would suffice. While our construction still relies on speed or time as a third dimension, we show that the spatial aspect of the system can be confined to two dimensions while preserving Turing-completeness.

3 The Pinball Wizard problem

We introduce the Pinball Wizard problem as follows: The pinball is a point moving in two dimension Euclidean space with variable speed and direction. It moves through a pinball machine and interacts with the components elaborated below under *Components*.

The ball enters the construction at timestep $t = 0$ with a given speed and direction. The task is to determine whether this pinball reaches a given target. We can analyze the movement of the ball within the construction, which consists of stationary and moving walls, one-way gates, parabola walls and bumper walls. With these components we later on construct gadgets implementing two independent stacks using space offset and time delay (offset) while changing the speed on the pinball, when it interacts with our gadgets. Some gadgets, are dedicated to inflict controlled time delays produced by pairwise bumpers or pairwise moving walls.

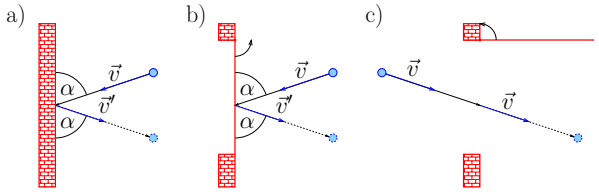

Figure 1 a) A perfectly elastic collision of a ball of speed $\vec{v}$ with a wall, after the collision the direction has changed but the speed is preserved. b) A ball has a collision with a one-way gate on the blocking side. This leads to a perfectly elastic collision. c) A ball has a collision with a one-way gate on the pass through side.

In the Pinball Wizard problem we always assume perfectly elastic collisions of the ball with walls, where the ball might change speed and direction without any impact on the wall.

Versions of perfectly elastic collisions possible for the Pinball Wizard problem

1. The wall is stationary. Here the ball changes its path after hitting the wall according to the law of reflection (see Figure 1.a).
2. The wall is moving with a constant speed. Then we can investigate the system relative to the moving wall which changes the speed and travel direction of the ball. We determine the new speed and direction of the ball relative to the wall's inertia, see Figure 2.
3. The wall is accelerated, i.e. the speed changes over time. Then we analyze the speed of the wall at the time of collision, and determine the new direction of the ball and its speed based on the speed and the direction of the movement of the wall at the time of collision. To determine the new direction and speed of the ball, we apply case 2 above.

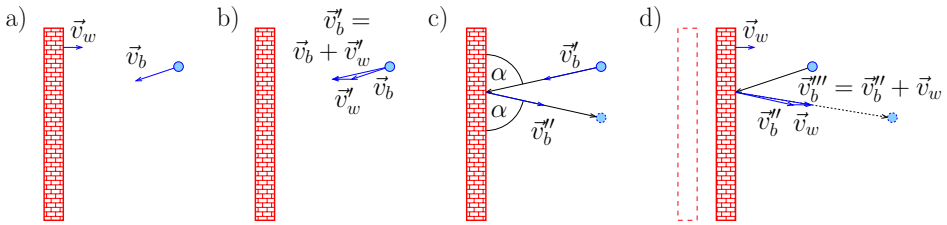

Figure 2 a) A perfectly elastic collision of a ball of speed $\vec{v}_b$ with a wall of speed $\vec{v}_w$ b) relative to the wall the ball has a speed of $\vec{v}_b' = \vec{v}_b + \vec{v}_w'$ c) the collision and the speed of the ball $\vec{v}_b''$ relative to the wall after the collision and d) the balls direction and speed $\vec{v}_b''' = \vec{v}_b'' + \vec{v}_w$ if it is not analyzed relative to the moving wall.

We also use bumper walls, which have a similar effect as straight moving walls. Bumpers increase or decrease the magnitude of the ball's velocity vector while reversing its direction, reflecting the ball to the opposite direction relative to its path before the collision. This behavior is illustrated in Figure 3.

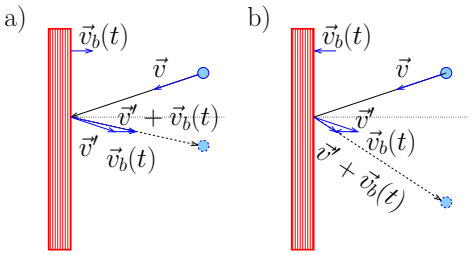

Figure 3 a) A bumper with positive (speed increasing) effect and b) a bumper with negative (speed decreasing) effect.

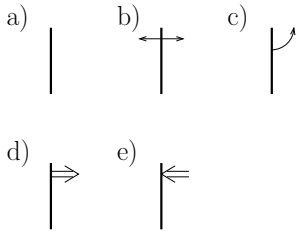

Figure 4 a) wall b) moving wall c) one-way gate d) bumper with positive effect e) bumper with negative effect.

Another component is the one-way gate. From one side, the ball passes through without any change in its motion; from the other side, it is reflected as if hitting a straight wall – behaving like a trapdoor. This is illustrated in Figure 1 b) and c). The last type of walls are parabola walls. These static walls are defined by a quadratic equation and the interval of start and end coordinates. The ball interacts with them by reflection like with a static straight wall, except the curvature has to be taken into account. We consider the tangent of the parabola at the point, where the balls hits this curve. Then, the angle of incidence of the ball is equal to its angle of reflection at this the point with respect to the tangent. Figure 4 describes the different types of walls in the problem setup.

Components

We now give a formal description of the components of our problem.

- walls : are given by the start and end positions of the wall as rational coordinates in $\mathbb{Q}^2$, see Figure 1.a).
- parabolas : are given by a quadratic function $y = a \cdot x^2 + b \cdot x + c$ where a, b, c are rational values as well as start and end values x_0 and x_1, i.e. $a, b, c, x_0, x_1 \in \mathbb{Q}$.
- one-way gates : are given by the start and end positions of the wall defined by rational coordinates. We assume that the gate opens in the clockwise direction going from the start position to the end position, see Figure 1.b) and c).
- moving walls : are given by start and end positions of the wall's initial location as rational coordinates in $\mathbb{Q}^2$. Its motion begins at a specified start time and follows a movement function over the time interval $T_0 = [0, t_1)$, with the movement function defined by a rational function in time t given by the polynomials of its numerator and denominator, i.e. the ratio of two polynomials of the time t with rational coefficients. In a second time interval $T_1 = [t_1, t_2)$, the wall moves back along a return path. We assume that after step t_2 the movement is periodically repeated. See Figure 2.
- bumpers : are given by start and end positions of the bumper given as rational coordinates, a time when the bumper effect starts, a time interval $T_0 = [0, t_1)$ with a bumper effect function for this time interval given by the quotient of two polynomials with rational coefficients depending on time t, and finally a second time interval $T_1 = [t_1, t_2)$ where the bumper rests. We assume that after step t_2 the movement is periodically repeated (Figure 3).

▶ **Definition 1** (Pinball Wizard problem). *Given a pinball with rational start and end positions, as well as its speed vector and a given set of components as described above, decide whether the ball reaches the target position.*

Given that the speed is taken as a constant, for example, the speed of light, then we obtain a similar problem to the Pinball Wizard problem called, the Ray Particle Tracing problem.

▶ **Definition 2** (Ray Particle Tracing problem). *Given a ray particle with rational start and end positions, as well as its speed vector and a given set of components as above, if we assume that walls behave the same as mirrors, decide whether the ray particle reaches the target position. The speed of the ray particle is taken as a constant, which is; the speed of light.*

The Ray Particle Tracing problem is closer to the problem analyzed by Moore in [16], as his construction for simulating a two-counter machine in 3D involves a billiard ball moving at constant speed. All proofs and observations for the Ray Particle Tracing problem can be adapted analogously to those for the Pinball Wizard problem. These adaptations are highlighted in Section 5 and discussed in detail in [2].

4 Pinball Wizard problem is Turing Complete

In this Section, we present the implementation of two independent stacks by using walls, bumper walls, moving walls, and one-way gate for the Pinball Wizard problem. For this we use the difference between the time and the position when and where the ball arrives at specific lines (i.e. a line called the starting interval or the end interval), more precisely the time and the space offset. In this Section we will present our construction using bumper walls. Note that the bumper walls can be replaced by moving walls in our constructions

(which has been discussed in [2]). Analyzing this alternative construction we can also adapt our result to the case where the ball never changes its speed, i.e. if it has reached the speed of light. This transforms the Pinball Wizard problem into the Ray Particle Tracing problem, as elaborated in [2]. Hence the Ray Particle Tracing problem is also Turing Complete if we restrict ourselves to 2D plus time.

Within our constructions we have to guarantee that the two stacks are completely independent, i.e. that push and pop operations of one stack do not change the value of the second stack (as a side effect). For the space offset stack this can be guaranteed if all the allowed paths of the ball have the same length, thus the time offset will not be affected by the stack operations of the space offset stack. Analogously for the time offset stack we have to guarantee that all allowed paths of the ball are parallel to each other. Thus, the operations of time offset stack will not influence the value of the space offset stack.

4.1 Implementing a Stack Using the Time Delay of the Pinball

The main idea to implement a stack by using a time or a space offset works as follows. Assume that the earliest time when the ball can cross a predefined (input) line is t_0 and the leftmost position on the line is s_0. And assume that the ball crosses the line at time $t_0 + \Delta t$ at position $s_0 + \Delta s$. Then we can use the offset values Δt and Δs to implement two stacks. Assume we use a second line which represents the output of a gadget implementing a stack operation. Then we use a system of walls, moving walls, bumper walls and one way gates to implement:

$$\Delta t' = \begin{cases} \frac{\Delta t}{2} & \text{to implement push 0} \\ \frac{\Delta t + 1}{2} & \text{to implement push 1} \\ 2 \cdot \Delta t & \text{to implement pop, if its output is 0} \\ 2 \cdot \Delta t - 1 & \text{to implement pop, if its output is 1} \end{cases}$$

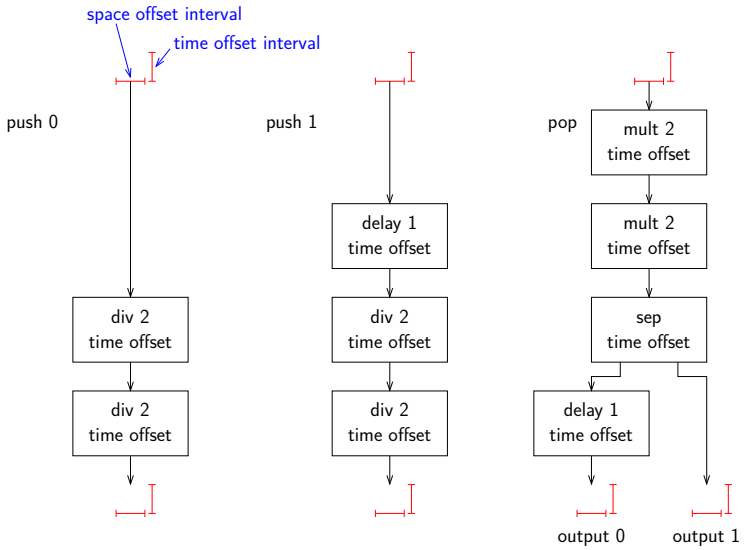

Figure 5 Illustration of the stack operations implemented by the time offset.

In the same way we can implement the stack operations using Δs for the space offset. For our time offset stack the multiplication with $\frac{1}{2}$ or with 2 is not sufficient. Since we do not assume infinite speed for the walls, a gap between the two offset intervals is therefore

necessary. This gap is implemented by multiplying the time offset two times with $\frac{1}{4}$ or 4 as required, as illustrated in Figure 5. A detailed explanation is included later in this section. We can draw a similar illustration for the space offset stack (by replacing "time" with "space", deleting the second multiplication box, and replacing the extra time offset with a space shift in Figure 5).

Let us assume that the ball enters the construction with a speed of v, then the speed will be modified by hitting the first bumper wall. By using a second bumper wall, we undo the effect of the speed modification. Hence the ball leaves the construction with a speed of v. The multiplication factor of the time offset will be generated by the modified speed within the time where the ball has a modified speed and the differed distances of the bumper walls. Recall, if the first bumper wall has a positive effect on the speed of the ball, the second has to have a negative effect and vice versa. We assume that the effect can by modified over time, analogously to acceleration. Therefore we use the acceleration of a bumper to describe this behavior of a bumper.

So the bumpers are used to manipulate the speed of the ball, and the changed speed over a given fixed distance δ_2 is used to achieve a multiplication of 2 (or $\frac{1}{2}$, resp.) with the time offset. We assume that outside of the construction blocks, the ball maintains a given speed v. This allows us to keep track of the time interval in which the ball arrives at a specific construction block.

The construction is illustrated in Figure 6. We now claim that:

▶ **Lemma 3.** *The 2D Pinball Wizard problem with one-way gates, plane and parabolic walls, moving walls, and bumpers simulates a stack using the time offset of the ball.*

Proof. We show the proof by describing how the multiplication of the time offset by 2 is implemented. Let us assume the following values where t_s is the starting time of the ball relative to a base time line. We can divide the way of the ball into three parts. Each of these parts have been described in Figures 6 (ways 1, 2 and 3).

Part 1. In the first part the ball goes from the starting line to the first moving wall.

- δ_1 is the distance between the starting line of the ball and the start line of the moving interval of the first moving wall.
- v is the initial speed of ball and its final speed.
- The ball reaches first the bumper wall at time step $t_1(t_s)$. Note that $t_1(t_s) = t_s + \frac{\delta_1}{v}$. We use $t_1 = t_1(0)$.
- $a_1(t)$ is the acceleration of the first bumper wall. We assume that the acceleration is 0 at time $t \leq t_1$. The effect of the bumper wall will be $v_1(t) = (t - t_1) \cdot a_1(t)$.
- $v_{b,1}(t_s)$ is the speed of the ball after the collision with the first bumper wall. Note that $v_{b,1}(t_s) = v - v_1(t_1 + t_s)$.

Note that

$$t_1 = \frac{\delta_1}{v} \qquad \text{and} \qquad v_1(0) = v \ .$$

Part 2. In the second part the ball goes from the the first bumper wall to the second bumper wall.

- δ_2 is the distance between the first bumper wall and the second bumper wall.
- $v_{b,1}(t_s)$ is the speed of ball within this part.
- The ball reaches the second bumper wall at time step $t_2(t_s)$. Note that $t_2(t_s) = t_1(t_s) + \frac{\delta_2}{v_{b,1}(t_s)} = t_s + \frac{\delta_1}{v} + \frac{\delta_2}{v_{b,1}(t_s)}$. We use $t_2 = t_2(0)$.

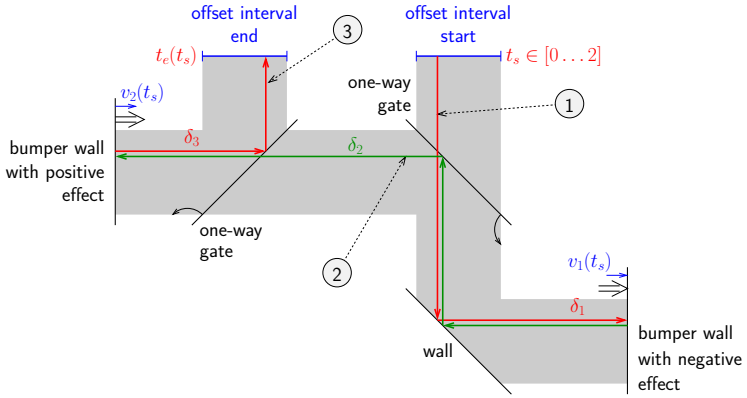

Figure 6 Multiplying the time offset by a factor using bumpers: The first red way (way 1) from the input (at offset interval start) to the right hand side bumper wall illustrates the the length δ_1. The green way (way 2) between the two bumper walls illustrates the the length δ_2. This is the only way where the ball has a changed speed. The second red way (way 3) from the left hand side bumper wall to the output (at offset interval end) illustrates the the length δ_3. The way between the two bumper walls is used to implement the multiplication of the time offset by a constant. The vertical shift of the 3 ways is used to increase readability.

- $a_2(t)$ is the acceleration of the second bumper wall. We assume that the acceleration is 0 at time $t \leq t_2$. The effect of the bumper wall will be $v_2(t) = (t - t_2) \cdot a_2(t)$.
- $v_{b,2}(t_s) = v$ is the speed of the ball after the collision with the second bumper wall. Note that $v_{b,2}(t_s) = v_{b,1}(t_s) + v_2(t_2(t_s))$.

Note that

$$t_2 = \frac{\delta_1 + \delta_2}{v} \qquad \text{and} \qquad v_{b,2}(0) = v_{b,1}(0) \;=\; v \;.$$

Part 3. In the third part the ball goes from the the second bumper wall to the end line.

- δ_3 is the distance between the second bumper wall and the end line of the ball.
- The ball reaches the end line at time step $t_c(t_s)$. Note that $t_e(t_s) = t_2(t_s) + \frac{\delta_3}{v}$. We use $t_e = t_e(0)$.

Note that

$$t_e = \frac{\delta_1 + \delta_2 + \delta_3}{v} \qquad \text{and} \qquad t_e(t_s) = t_s + \frac{\delta_1 + \delta_3}{v} + \frac{\delta_2}{v_{b,1}(t_s)} \;.$$

Our goal is now to determine functions for the acceleration $a_1(t_s)$ and $a_2(t_s)$ such that $t_e(t_s) = 2 \cdot t_s + c$ for a value $c = t_e$ which does not depend in t_s. This leads to

$$t_s + \frac{\delta_2}{v} \;=\; \frac{\delta_2}{v_{b,1}(t_s)}$$

$$\text{and therefore} \qquad v_{b,1}(t_s) \;=\; \frac{v \cdot \delta_2}{v \cdot t_s + \delta_2} \;.$$

Hence

$$a_1(t_1 + t_s) \;=\; \frac{v - v_{b,1}(t_s)}{t_s} \;=\; \frac{v^2}{v \cdot t_s + \delta_2}$$

$$a_2(t_2 + 2 \cdot t_s) \;=\; \frac{v - v_{b,1}(t_s)}{2 \cdot t_s} \;=\; \frac{v^2}{2 \cdot (v \cdot t_s + \delta_2)}$$

◀

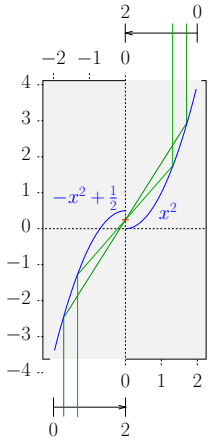

Figure 7 After performing a multiplication step, the order of the offset is reversed (the zero offset is moved from the left of the pinball interval to the right of the interval). This tool moves the pinball to its original order, i.e. the zero offset is moved back to the left side of the interval. Depending on the widths of the pinball interval, one might have to implement a divergent version of this tool, i.e. with x coordinates going from -4 to 4 instead of the presented version with x coordinates going from -2 to 2. Within the following construction only offsets between 0 and 2 (excluding 0 and 2) are used.

For the multiplication of the time offset by $\frac{1}{2}$ we exchange the two bumpers and the analysis will follow the same ideas as above.

Note that at the output interval the order of the space offset is reversed. To correct the order we have to add the construction of Figure 7 at the output interval of the time offset multiplier.

It is also possible to implement a multiplier for the time offset by using only moving walls (see Figure 8). The analysis can be found in [2]. The implementation of the multiplier by moving walls can be adapted such that it can be used to handle the case where the ball does not change its speed (leading to the Ray Particle Tracing Problem). If we simulate bumpers with moving walls the movement of both walls has to be coordinated in such a way that the different speeds and delays add up to the same effect as two bumpers.

Finally we have to describe a gadget for adding a constant delay to the ball and a gadget for separating early balls (arriving at a specific position with a time offset of $[0 \dots 1)$) from late balls (arriving at a specific position with a time offset of $[1 \dots 2)$). The constant delay can easily be generated by using some 45° walls. Hence we will now focus on the construction of the time offset separator.

The main idea of the time offset separator is to use a moving wall which leads a ball arriving with a time offset $[0 \dots 1)$ in one distinct direction and a ball arriving with a time offset $[1 \dots 2)$ in a second distinct direction. Since we do not assume that a wall can move with an infinite speed, we have to implement a gap between the two offset intervals. This can be implemented be doubling the multipliers, i.e. instead of multiplying the time offset once with $\frac{1}{2}$ (or with 2, respectively), we multiply the time offset two times with $\frac{1}{2}$ (or with 2, respectively). Hence we multiply the time offset either with $\frac{1}{4}$ or with 4.

As illustrated in Figure 5 the time offset separator only occurs after the multiplication with 4 within the pop operation. Thus we can assume that the ball will arrive with a time offset within the interval $[0 \dots 2)$ (instead of within the standard interval $[0 \dots 1)$). Using double multiplication guarantees that the ball will either arrive with a time offset in $[0 \dots \frac{1}{2})$ or with a time offset in $[1 \dots \frac{3}{2})$.

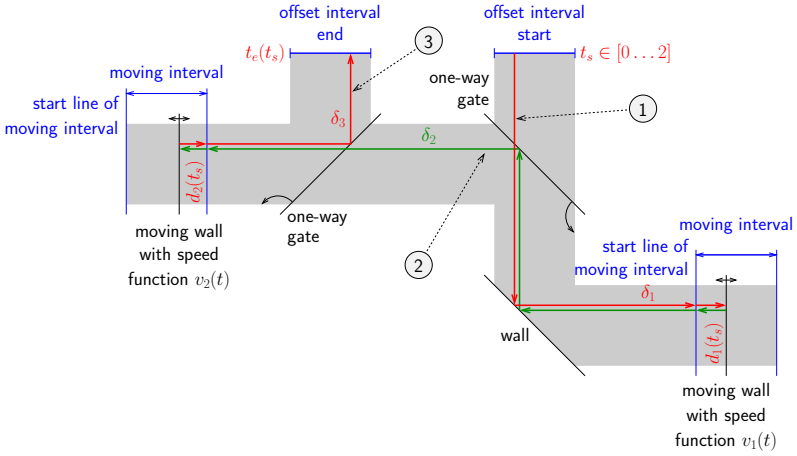

Figure 8 As is the case with the multiplication using bumpers, we can split the way from the starting interval to the end interval into 3 part: The first red way (way 1) from the input (at offset interval start) to the right hand side moving wall, the green way (way 2) between the two moving walls, and the second red way (way 3) from the left hand side moving wall to the output (at offset interval end). Way 2 is used to implement the multiplication of the time offset by a constant.

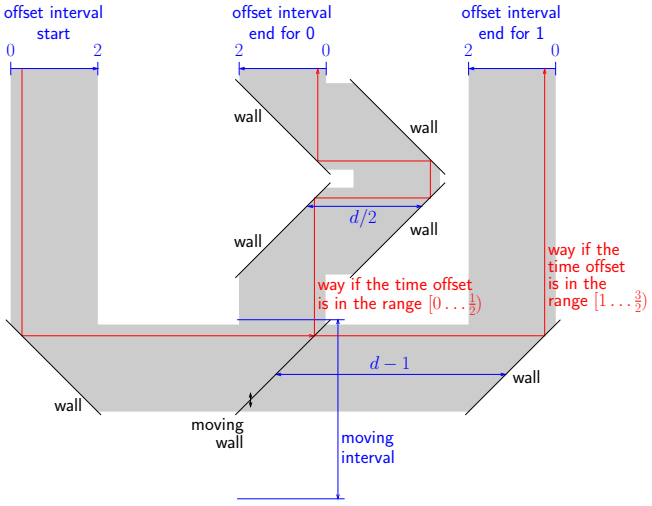

Figure 9 Construction for separating the ball arriving with a time offset in the time interval $[0 \dots \frac{1}{2})$ from the ball arriving with a time offset in the time interval $[1 \dots \frac{3}{2})$. Using the distance of $d-1$ instead of d shifts the base of the time interval of the second interval $[1 \dots \frac{3}{2})$ from 1 to 0. The time gaps in the offset between $\frac{1}{2}$ to 1 (and from $\frac{3}{2}$ to 2) are necessary to give the moving walls the time either to move out of the way of the ball or to move into its initial position.

This allows the separating moving wall to move out of the way and to move back. The construction is illustrated in Figure 9. We refer the reader to [2] for a more detailed analysis.

4.2 Implementing a Stack Using the Space Offset of the Pinball

The space offset stack can be implemented using a system of plane walls, one-way gates and parabola walls. Note that the distance the ball has to pass to reach the focus is independent of its starting position on the starting interval. This distance is given by the distance from the starting line segment to the point where the ball hits the parabola wall plus the distance between this point to the focus.

Let us assume that a ball going down in the direction of the y-axis starts at an arbitrary point ℓ on the line segment defined from point $(0, h)$ to point $(2, h)$ at time t. Assume again that the ball bounces off a parabola $(x, a \cdot x^2)$ with focus $(0, \frac{1}{4a})$. If the ball starts at point (ℓ, h) then the distance traveled by the ball to the focus is given by

$$h - a \cdot \ell^2 \quad + \quad a \cdot \ell^2 + \frac{1}{4a} \quad = \quad h + \frac{1}{4a}$$

which is independent from the starting point at the starting line segment (see Figure 10). If we assume that within each time unit the ball can pass through one space unit, then within a construction like in Figure 11, it will reach the focus at time $t + h + \frac{1}{4a}$, independent from ℓ.

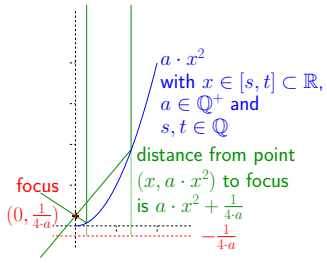

Figure 10 A parabola with points $(x, a \cdot x^2)$ and focus $(0, \frac{1}{4a})$.

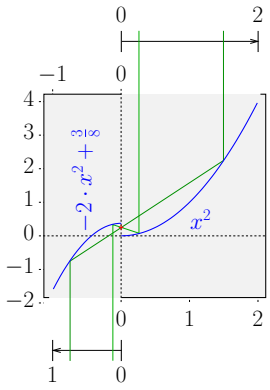

Figure 11 Multiplication of the offset by $\frac{1}{2}$ is a main tool for implementing a push operation. Here we show the reverse ordering when two separate balls enter the starting interval line at different positions.

▶ **Lemma 4.** *The Pinball Wizard problem can simulate a stack which is implemented by the space offset of the ball within a system of parabola walls.*

Proof. We achieve push and pop operations using the multiplication of the offset by 2^{-1} or by 2. We assume starting interval between 0 and 2. For the multiplication of the offset by $\frac{1}{2}$, two parabolas are placed such that the second parabola is flipped as seen in Figure 11. Analogously, multiplying the offset by 2 can be done using a similar construction (see [2]). As with the time offset stack, both constructions reverses the ordering of space offset of the ball. Thus a similar construction as shown in Figure 7 can be used to correct the ordering.

Larger offsets are allowed to deal with pushing 1 to or popping 1 from the stack, as both operations require the addition of a 1 to the offset (that is, a shift in the offset interval).

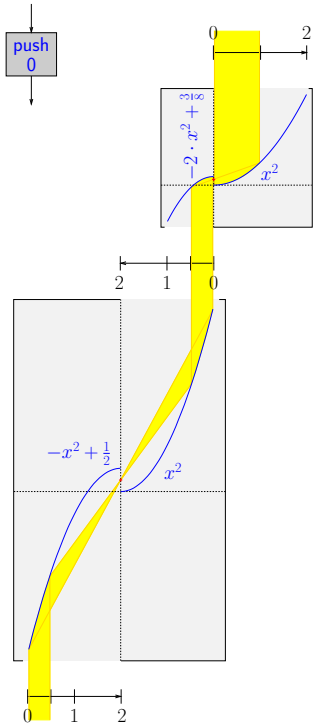

Figure 12 Multiplying the offset from the interval $[0, 1)$ by $\frac{1}{2}$ implements a push of the Boolean value 0 to the stack. The yellow area represents area of possible offsets.

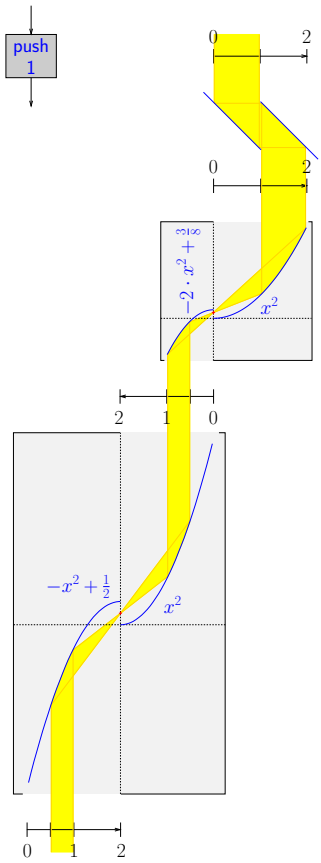

Figure 13 Shifting the offset from the interval $[0, 1)$ to the interval $[1, 2)$ and multiplying the resulting offset by $\frac{1}{2}$ implements a push of the Boolean value 1 to the stack. The yellow area represents area of possible offsets.

For push 0 we multiply the space offset by $\frac{1}{2}$ (see Figure 12). To push 1 we first shift the starting offset interval from $[0 \ldots 1)$ to $[1 \ldots 2)$ before multiplying the space offset by $\frac{1}{2}$ (see Figure 13). For pop 1, after multiplying by 2, the possible output offset values are split in two intervals, each going from 0 to 1 as seen in Figure 14. Thus if the top entry of the stack is a 0, the ball with the modified offset will show up on the left exit. In the case where it is a 1, the ball will accrue to the right exit. To separate these two cases we have to split the interval after the multiplication into two intervals to get the resulting stack and the output of the pop operation.

By assuming the bottom of the stack always contains at least two 1 digits, the offset can never have the values 0, $\frac{1}{2}$, and 1. To guarantee that these edge cases never occur we can assume a bottom symbol on the stacks which will never be removed. ◀

The full proof as well as constructions associated with Lemma 4 are given in [2].

Note that both pop operations (for the time offset stack as well as for the space offset stack) results in two potential output intervals (one for the pop result 0 and one for the pop result 1). When we would like to continue with our simulation of the two-stack PDA we have to combine the two intervals again. For this we can use a one-way gate, which guarantees a

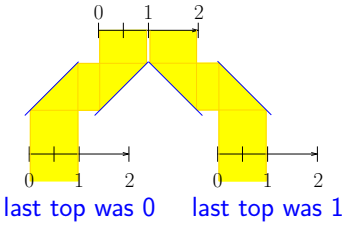

Figure 14 Construction for separating the arrival of the ball within the space interval $(0 \ldots 1)$ from the ball's arrival within the space interval $(1 \ldots 2)$.

perfect overlapping of the two intervals (the 0 offset of the first interval will overlap with the 0 offset of the second interval and the 1 offset of the first interval will overlap with the 1 offset of the second interval).

To illustrate the behavior of space as well as time offset intervals w.r.t. push and pop operations we will showcase a short example in the following :

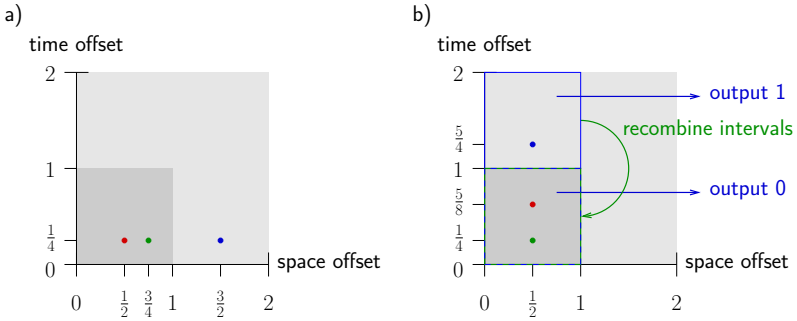

Figure 15 Behaviour of the time and space offset intervals according to the (a) push(1) and (b) pop operations. The red dot denotes the initial position of the stack, the blue dot shows the intermediate position and the green dot shows the final position of the stack.

For the behaviour of the offset intervals according to stack operations (see Figure 15 a), starting with the stack values 01 (represented by the binary value 0.01_2) for the time offset stack and 1 (represented by the binary value 0.1_2) for the space offset stack (red dot), we begin by performing a push(1) operation to the space offset stack: First add 1 to the value of the space offset stack (blue dot), and then multiply by 2^{-1} (green dot). This leads to the stack content 11 (represented by the binary value 0.11_2).

In Figure 15 b), we start with stack values 101 (represented by the binary value 0.101_2) for the time offset stack and 1 (represented by the binary value 0.1_2) for the space offset stack (red dot) and perform a pop operation on the time offset stack as follows: First multiply the value of the time offset stack by 2 (blue dot). If the result is in the range between 1 and 2, the output will be 1, otherwise if the result is in the range between 0 and 1, the output will be 0. Finally we recombine the two possible intervals (by moving the upper interval to map on the lower interval, resulting in the green dot). This leads to the stack content 01 (represented by the binary value 0.01_2) in the time offset stack. Note that all the operations should not modify the space offset value.

One should highlight here that from Figure 10, it can be seen that for every offset of the ball that might occur within each of the push 0, push 1 and pop constructions, the ball will always need the same amount time to go from the horizontal start line to the finish line.

This is possible because an extra delay can be introduced on the ball's movement by an additional substructure such that its time of arrival at the end line is well-defined. Hence, the operations of the space offset stack is independent of the time offset stack.

▶ **Lemma 5.** *The 2D Pinball Wizard problem with one-way gates, walls and moving walls simulates a pushdown automaton where the stack is represented by a time offset such that each step is represented by a constant number of reflections.*

Recall that a pop operation leads to 2 possible alternative ways the ball might go (see Figure 9 and 14). The two alternative ways represent the two possible different calculation steps of the simulated two-stack pushdown automaton depending on the top element of one stack. To continue with the simulation the two alternative ways have to rejoined. This will be done by the construction presented in Figure 16.

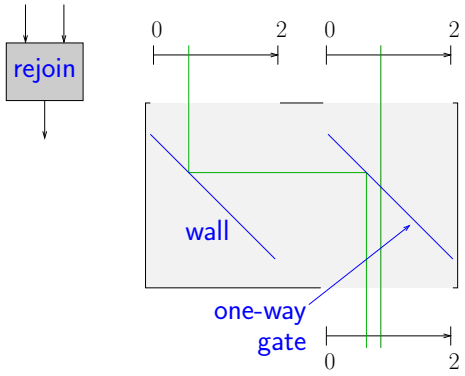

Figure 16 A simple rejoin tool to be used after the splitting at a pop operation.

Now, using the proofs of Lemma 3, Lemma 4 as well as Lemma 5 (proof available in [2]), we conclude that

▶ **Theorem 6.** *The 2D Pinball Wizard problem with one-way gates, plane, parabolic walls and moving walls simulate a two-stack pushdown automaton (PDA) such that each step is represented by a constant number of reflections. This also holds if we replace the moving walls by bumpers.*

Since a two-stack PDA can simulate any deterministic Turing machine, we conclude the following:

▶ **Corollary 7.** *Deciding the 2D Pinball Wizard problem with one-way gates, plane, parabolic and moving walls is as hard as solving the Halting problem.*

5 Ray Particle Tracing Problem is Turing Complete

Let us now assume that bumpers and moving walls do not change the speed of the ball, i.e. our ball has always the same speed v. This is for example the case if we assume that the ball has the speed of light, leading to the Ray Particle Tracing problem. Note that bumpers behave like usual walls if the speed of a ball cannot be changed. The only effect of a moving wall on the ball might be a change in the distance which the ball has to go. Within this problem we usually use mirrors instead of walls. However, for uniformity in the analyses, we stay with the word "wall".

Since the distance a ball has to travel influences the traveling time one could implement multipliers for the space and time offset only by using simple (non-moving) and by moving walls. For the implementation of the space offset we do not need any changes in our construction. For dealing with the time offset stack some changes have to be done to original construction. If the moving wall can change the speed of the ball, we have to analyze the changed speed and the changed travel distance (see the construction for such a multiplier in Figure 8 and its analysis in [2]).

In the special case of a ball traveling with the speed of light, one could only assume that such a multiplier multiplies the time offset with a value of $1 + \varepsilon$ or $1 - \varepsilon$, respectively, for a small but constant value of ε. Otherwise the walls have to move with nearly the half of the speed of light. To reach a multiplication with 2 or $\frac{1}{2}$ one has to iterate the multiplier (for the analysis see [2]). On the other hand, if the ball never changes its speed by a collision with a moving wall, we do not have to undo such changes. Thus the resulting construction for the multiplication of the time offset looks simpler than the construction presented in the previous section.

6 Conclusions and Open Problems

Inspired by pinball machines, the Pinball Wizard problem involves navigating a maze composed of stationary and moving walls, one-way gates, parabola walls and bumper walls. The objective is to determine whether a pinball can reach a designated target. We show that the Pinball Wizard problem in two dimensions can simulate a two-stack PDA, which is equivalent in power to a single-tape deterministic Turing machine. Thus, solving instances of this problem is at least as hard as deciding the Halting problem.

While the billiard problem with static walls – also known as ray tracing with mirrors – is known to be Turing-complete in three dimensions, as established independently by Moore [16] and Reif et al. [20], our work may contribute to resolving the open question of whether Turing-completeness can also be achieved in two dimensions.

Another interesting direction for future research is to investigate whether a simplified model with a reduced set of components can still achieve Turing-completeness. Currently, our construction relies on idealized assumptions such as arbitrary precision and perfect reflection. It remains an open question how the computational complexity would change under more realistic physical constraints, including damping, friction, finite ball diameters and random disturbances. Future work could also explore whether a smaller set of components suffices for the Pinball Wizard problem specifically and what precision bounds are necessary for reliable computation.

It is an open problem whether the presented constructions can be modified such that no one-way gate for rejoining the different simulation steps after the splitting at a pop operation are necessary anymore. This leads to the question of simulating reversible Turing Machines by the presented Pinball Wizard problem.

References

1 Andrew Adamatzky and Jérôme Durand-Lose. Collision-based Computing. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors, *Handbook of Natural Computing, Section VII: Broader Perspective — Alternative Models of Computation*, pages 1949–1978. Springer, 2012. doi:10.1007/978-3-540-92910-9_58.

2 Rosemary Adejoh, Andreas Jakoby, Sneha Mohanty, and Christian Schindelhauer. How pinball wizards simulate a turing machine, 2025. arXiv:2510.02560.

3 Elwyn R Berlekamp, John H Conway, and Richard K Guy. *Winning ways for your mathematical plays, volume 4*. AK Peters/CRC Press, 2004. doi:10.1201/9780429487309.

4 Vincenzo Bonifaci, Kurt Mehlhorn, and Girish Varma. Physarum can compute shortest paths. *Journal of Theoretical Biology*, 309:121–133, 2012. doi:10.1016/j.jtbi.2012.06.017.

5 Tanay Chattopadhyay and Dilip Kumar Gayen. Optical xor-xnor logic circuits using mechanical movable mirrors. In *2021 Devices for Integrated Circuit (DevIC)*, pages 65–70, 2021. doi:10.1109/DevIC50843.2021.9455909.

6 Stephen Cook and Ravi Sethi. Storage requirements for deterministic / polynomial time recognizable languages. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 33–39, New York, NY, USA, 1974. Association for Computing Machinery. doi:10.1145/800119.803882.

7 Erik Demaine, Isaac Grosof, and Jayson Lynch. Push-pull block puzzles are hard. In *Algorithms and Complexity*, pages 177–195, April 2017. doi:10.1007/978-3-319-57586-5_16.

8 Jérôme Durand-Lose and Andrew Adamatzky. *Computing Inside the Billiard Ball Model*, pages 135–160. Springer London, London, 2002. doi:10.1007/978-1-4471-0129-1_6.

9 Michal Forišek. Computational complexity of two-dimensional platform games. In Paolo Boldi and Luisa Gargano, editors, *Fun with Algorithms*, pages 214–227, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-13122-6_22.

10 Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of theoretical physics*, 21(3-4):219–253, 1982. doi:10.1007/BF01857727.

11 Martin Gardner. Mathematical games. *Scientific American*, 223(4):120–123, 1970. URL: http://www.jstor.org/stable/24927642.

12 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Co., New York, 1980. doi:10.1016/0096-0551(80)90011-9.

13 Manuel Lafond. The complexity of speedrunning video games. In Hiro Ito, Stefano Leonardi, Linda Pagli, and Giuseppe Prencipe, editors, *9th International Conference on Fun with Algorithms (FUN 2018)*, volume 100 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:19, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FUN.2018.27.

14 Norman Margolus. Physics-like models of computation. *Physica D: Nonlinear Phenomena*, 10(1):81–95, 1984. doi:10.1016/0167-2789(84)90252-5.

15 Sneha Mohanty and Christian Schindelhauer. Cryptography based on 2d ray tracing. In *2025 9th International Conference on Cryptography, Security and Privacy (CSP)*, pages 33–40, 2025. doi:10.1109/CSP66295.2025.00014.

16 Cristopher Moore. Unpredictability and undecidability in dynamical systems. *Phys. Rev. Lett.*, 64:2354–2357, May 1990. doi:10.1103/PhysRevLett.64.2354.

17 Toshiyuki Nakagaki. Smart behavior of true slime mold in a labyrinth. *Research in Microbiology*, 152(9):767–770, 2001. doi:10.1016/S0923-2508(01)01259-1.

18 Marian Boylan Pour-el and Ian Richards. A computable ordinary differential equation which possesses no computable solution. *Annals of Mathematical Logic*, 17(1):61–90, 1979. doi:10.1016/0003-4843(79)90021-4.

19 Stephen R Pring and Christopher J Budd. The dynamics of a simplified pinball machine. *IMA Journal of Applied Mathematics*, 76(1):67–84, 2011. doi:10.1093/imamat/hxq064.

20 John H. Reif, J. Doug Tygar, and A. Yoshida. Computability and complexity of ray tracing. *Discrete & Computational Geometry*, 11:265–288, 1994. doi:10.1007/BF02574009.

21 Ryuhei Uehara. Computational complexity of puzzles and related topics. *Interdisciplinary Information Sciences*, 29(2):119–140, 2023. doi:10.4036/iis.2022.R.06.

22 Liang Zhang. Computing naturally in the billiard ball model. In *International Conference on Unconventional Computation*, pages 277–286. Springer, 2009. doi:10.1007/978-3-642-03745-0_29.

23 Martin Ziegler. Physically-relativized church–turing hypotheses: Physical foundations of computing and complexity theory of computational physics. *Applied Mathematics and Computation*, 215(4):1431–1447, 2009. Physics and Computation. doi:10.1016/j.amc.2009.04.062.

24 Tadeu Zubaran and Marcus Ritt. Agent motion planning with pull and push moves. In *Anais do VIII Encontro Nacional de Inteligência Artificial*, pages 358–369, Porto Alegre, RS, Brasil, 2011. SBC. URL: https://sol.sbc.org.br/index.php/eniac/article/view/33707.

A Simulating a DTM

Here, we will sketch how the Pinball Wizard problem and the Ray Particle Tracing problem can be used to simulate the computation of a 1-tape DTM (equivalently, a two-stack PDA). Hence both problems are Turing complete. Hence both problems are at least as hard as the Halting problem.

Assume a configuration of a 1-tape DTM (equivalently, a two-stack PDA). To represent the tape content left of the head we can use the time offset stack and to represent the tape content right of the head we can use the space offset stack. Given a 1-tape DTM with a binary input string w, let $S^1 = \langle q_0, s_0^1, w\rangle$ and $S^2 = \langle s_0^2\rangle$ be the initial configuration of the space and time offset stacks, respectively where q_0 is the initial state of the DTM. We assume that the states of the DTM are represented by binary numbers of length ℓ. s_0^1 and s_0^2 are the bottom symbols encoded in the stacks.

At the start of every step in the simulation, if the current state is q_i and a_i is the symbol in the cell read by the DTM. Then we assume that a_i, all the elements to the right of a_i and q_i are in the space offset stack and the elements to the left of a_i are in the time offset stack.

In each step, the current state q_i and cell symbol a_i are read from the space offset stack via a sequence of pop operations, each of which will result in a component which uniquely describe the starting point of the next step. Recall that each pop operation yields a binary decision tree over the possible output intervals regions, see Figure 14. In order to perform the transition function, we use a binary decision tree, where we branch out using the pop operation for every of the possibilities defined by the starting state and symbol read from the space offset stack. Since this is a DTM, the next state q_j, the new cell symbol a_j, and the direction τ in which the head will move are uniquely defined. Next, we write a_j to the current cell and move the tape head again by pop operations and push operations. Finally, we write the encoding of the new state q_j to the space offset stack, combine the space offset intervals and time offset intervals (recall that the separators after the multiplications with 2 result in several alternative offset intervals) to a single unique output interval. Hence, the branches of the decision tree are combined again into a single space and time offset using one-way gates. The tool for combining intervals consisting of a wall and a one-way gate is shown in Figure 16.

If necessary, additional delay on the particle can be introduced by strategically placed walls. Now we can start with the next simulation step.

For the moving walls we schedule the return time to the initial position according to the constant simulation time of each step of the Turing machine. So, we can reuse them for simulating in the next step.