

Game Theory

Ben Minogue

May 6, 2024

Abstract

This project explores the rich world of impartial combinatorial games, focusing on two key areas: introducing fundamental concepts and strategies of impartial games, and investigating the boundaries of decidability in subtraction games. Through a comprehensive analysis of classic games like Nim, Wythoff's Game, and Green Hackenbush, we establish a solid foundation for understanding the principles and techniques used in solving these games. We introduce powerful tools such as the Sprague-Grundy function and provide code for an interactive game that allows readers to apply their newfound knowledge in practice. In the second part of the project, we investigate undecidability in subtraction games, proving the algorithmic undecidability of general subtraction game outcomes before searching for the limit to decidability in two and three move games.

Contents

1	Introduction	2
2	Impartial Game Theory	3
2.1	Winning Strategy for Nim	3
2.1.1	The Game of Nim	3
2.1.2	P-positions and N-positions	3
2.1.3	Nim-Sum	4
2.1.4	Winning strategy for Nim using Nim-Sum	5
2.2	Graph Games	6
2.2.1	Games Played on Directed Graphs	6
2.2.2	The Sprague-Grundy Function	7
2.2.3	Sprague Grundy Function Example with Wythoff's Game	8
2.3	Sums of Combinatorial Graph Games	8
2.3.1	The Disjunctive Sum of n Graph Games	9
2.3.2	The Sprague-Grundy Theorem	9
2.3.3	Example of the Sprague-Grundy Theorem	10
2.4	Green Hackenbush	11
2.4.1	Green Hackenbush Introduction	11
2.4.2	Sprague-Grundy Theorem for Green Hackenbush	11
2.4.3	The Colon Principle	12
2.4.4	The Fusion Principle	14
2.4.5	Green Hackenbush Combined with Other Impartial Games	18

3	Vector Subtraction Games and the Limits of Computability	20
3.1	Nim with a Set of Legal Moves.	20
3.1.1	Introduction to the set \mathcal{M}	20
3.1.2	Introduction to Cellular Automata	22
3.1.3	Using Modular Games to Emulate Cellular Automata	23
3.1.4	Emulating a modular game with an invariant game	27
3.1.5	Summary of the Proof of Theorem 3	29
3.1.6	Further Questions	29
3.2	Two-move Vector Subtraction Games are Decidable	33
3.2.1	Linear Algorithm for two-move Vector Subtraction	33
3.2.2	Applying the Algorithm to deduce Decidability in two-move Vector Subtraction Games	35
3.3	Three Move Vector Subtraction Games in 2-Dimensions	36
3.3.1	Eventual Periodicity of 2-Dimensional Games	36
3.3.2	Experimental Observations on Three-move Games in 2 Dimensions	37
3.3.3	Solvability of Perfectly Periodic Rulesets	38
3.3.4	Occurrence of Perfectly Periodic Rulesets	38
3.3.5	Are Periodic Three-Move Games Decidable?	42
3.3.6	Using Outcome Segmentation to Understand Three-Region Three-move games	46
3.3.7	Recent Theory on Three-Move Subtraction Games in 2-Dimensions	49
3.3.8	Closing Discussion on Vector Subtraction Games	50
4	Conclusion	51

1 Introduction

In this project we analyse **Combinatorial Games**. These are a category of games which have the following properties:

1. There are just two players, we sometimes refer to them as Left (L) and Right (R).
2. There are finitely many positions available in the game.
3. There are clearly defined rules which specify which game positions each player can move to.
4. Left and Right move alternately.
5. Both players have complete information about the game, meaning there is no hidden element and no dependency on chance such as rolling a dice or shuffling cards.
6. The game ends when a player is unable to move.
7. The game must come to an end in a finite number of moves and cannot be drawn by repetition of moves.

When playing combinatorial games, we usually play such that when a player is unable to move, that player loses. This is known as the **Normal Play** convention. Games can also be played under the **Misere Play** convention where the first player that is unable to move wins.

In this project we will focus on a category of combinatorial games known as **impartial games**. In these games the set of allowable moves depends only on the position of the game, and not on the player whose turn it is to move. The alternative to impartial games is **partisan games**, where players can have moves that only they can play.

The project is split into two main sections. The first section provides an introduction to impartial combinatorial games for readers who are completely new to the subject. We use extended examples from Nim and Wythoff's Game to explain some key concepts in impartial combinatorial game theory such as P- and N- positions and the Sprague-Grundy Theorem. We also introduce the slightly harder impartial game Green Hackenbush and provide a full solution to the game. Finally, we discuss what it means to play impartial games simultaneously and provide code for a combinatorial game tool where Nim, Wythoff's Game and Green Hackenbush can be played simultaneously, so that the reader can practice thinking through the theory in an interactive environment. For a more comprehensive, yet entertaining introduction to the subject we recommend the highly rated Winning Ways books by John Conway, Richard Guy and Elwyn Berlekamp [1].

The second section introduces the reader to vector subtraction games and the problem of undecidability in this space. We will follow the proof from [9] to show that general vector subtraction games are undecidable and then discuss where the exact limit of computability may be.

2 Impartial Game Theory

2.1 Winning Strategy for Nim

2.1.1 The Game of Nim

Nim, first popularised by Bouton in 1901 [4], is the most famous take-away game and is played as follows. There are n piles of chips containing x_1, x_2, \dots, x_n chips respectively. Two players take it in turns to choose one pile and remove some chips from it. You can remove as many chips as you would like but only from one pile. In **Normal Play**, The winner is the player who removes the last chip. Under the **Misere Play** rule, the player who is forced to take the last chip loses. Here we focus primarily on the normal play rule. The main sources used for the following few subsections are [2] and [3].

2.1.2 P-positions and N-positions

In impartial combinatorial games, all game positions can be split into two categories that we can think of as the **perfect play outcome** of the current position, meaning who wins the game when it is played perfectly by both players.

- A game is in a **P-position** if it secures a win for the Previous player, the player who just moved, with continued perfect play.
- A game is in an **N-Position** if it secures a win for the Next player to move with continued perfect play.

We can think of perfect play as the ability to always find and play a move from an N-position to a P-position. This means that the player playing their "good" move from an N-position is winning before and after they play it. By definition every move from a P-position is to an N-position so the same player will get another opportunity to find and play a move back to a P-position and there is nothing their opponent can do about it.

In any impartial combinatorial game, it is possible to work backwards from the end of the game to define each position as either a P-position or an N-position. This process is called **backwards induction** and is outlined as follows

1. Label the terminal position (where no moves are available) as P.
2. Label each position that can reach P as N.
3. For positions that only move to N-positions, label P.
4. Return to step 2 and repeat the process until all positions are labeled.

In misere play, we just invert step one so that the terminal position is N.

This inductive system for P-positions and N-positions proves that Nim can be solved for any starting position. However, it can be difficult to generalize it to a given game position. Consider the example starting position (21, 45, 12, 9), it is not immediately obvious whether this is a P-position or an N-position and it would be difficult to find a move that takes us to a P-position if you were in an N-position. In the next section we go on to describe a way of calculating whether a position is a P-position or an N-position using the Nim-Sum.

2.1.3 Nim-Sum

The Nim-Sum of two positive integers is the bitwise addition of their binary representations modulo 2. Every positive integer x has a unique base 2 representation of the form $x = x_m 2^m + x_{m-1} 2^{m-1} + \dots + x_1 2 + x_0$ for some m , where each x_i is either one or zero. Here we use $x = (x_m x_{m-1} \dots x_1 x_0)_2$ to denote the binary representation of x .

Definition 1. The **Nim-Sum** of $(x_m \dots x_0)_2$ and $(y_m \dots y_0)_2$ is $(z_m \dots z_0)_2$, where for all k , $z_k = x_k + y_k \pmod{2}$, so $z_k = 1$ if $x_k + y_k = 1$ and $z_k = 0$ otherwise. We write the sum as $(x_m \dots x_0)_2 \oplus (y_m \dots y_0)_2 = (z_m \dots z_0)_2$.

We will now look at a simple example and compute $21 \oplus 45$. we can write this in binary as $(010101)_2 \oplus (101101)_2$. From here, we can see that the first three binary digits of each number are opposites and so sum to 1 whereas the last 3 digits match so sum to 0 (modulo 2). writing this in binary gives $(111000)_2$ which represents the number 56.

Since addition modulo 2 is associative and commutative, so is Nim-Sum. Therefore we have $x \oplus (y \oplus z) = (x \oplus y) \oplus z$ and $x \oplus y = y \oplus x$. Also, 0 is the identity for Nim-sum as $0 \oplus x = x$ and every number is its own inverse as $x \oplus x = 0$. This means that the cancellation law holds and $x \oplus y = x \oplus z \implies y = z$ just as it does for normal addition.

2.1.4 Winning strategy for Nim using Nim-Sum

The Nim-Sum of a Nim position is simply the Nim-Sum of all the piles in the position. The winning strategy for Nim is given by the following theorem which we will state, explain and prove in this section.

Theorem 1. *A position (x_1, \dots, x_m) in Nim is a P-position if and only if the Nim-Sum of its components is zero, $x_1 \oplus \dots \oplus x_m = 0$.*

This means that the winning strategy is to finish every move with a Nim-Sum of 0 if possible. If its not possible, then you must already be in a P-position with Nim-Sum 0 and you will have to play a move resulting in a positive Nim-Sum and hope that your opponent doesn't play back to a P-position on their next turn. We now prove the theorem.

Proof. Let \mathcal{P} denote the set of Nim positions with Nim-Sum zero and let \mathcal{N} denote the compliment set of game positions. We will use the definitions of P-positions and N-positions to verify that \mathcal{P} is the full set of P-positions and \mathcal{N} the full set of N-positions, thus proving the theorem. For \mathcal{P} and \mathcal{N} to be the full sets of P-positions and N-positions, they need to hold the following three conditions.

1. All terminal positions must be in \mathcal{P} .
2. From each position in \mathcal{N} , there is a move to a position in \mathcal{P} .
3. Every move from a position in \mathcal{P} is to a position in \mathcal{N} .

Lets verify these conditions.

1. The only terminal position is the position with no chips in any pile which has Nim-Sum zero and is a P-position by definition so is in \mathcal{P} .
2. Here's how we can construct a move from \mathcal{N} to \mathcal{P} . Write the Nim-Sum as a binary column addition of the size of each pile and look at the leftmost column that has an odd number of 1's. Select a pile that has a 1 in this column and change that 1 to a 0. Alter the other digits in this pile so that each column has an even number of 1's, this can be done as the number will always be smaller because we lost the 1 from the most significant position. Therefore this is a legal move that leaves the position with a Nim-Sum zero in a P-position.
3. If (x_1, \dots, x_m) is in \mathcal{P} and some x_i is altered to $x'_i < x_i$ then we cannot have $x_1 \oplus \dots \oplus x_i \oplus \dots \oplus x_m = 0 = x_1 \oplus \dots \oplus x'_i \oplus \dots \oplus x_m$ as the cancellation law implies that $x_i = x'_i$ so $x_1 \oplus \dots \oplus x'_i \oplus \dots \oplus x_m \neq 0$ which implies $(x_1, \dots, x'_i, \dots, x_m)$ is in \mathcal{N} .

\mathcal{P} and \mathcal{N} satisfy the three properties so we must have that \mathcal{P} is the full set of P-positions and \mathcal{N} is the full set of N-positions. \square

Example: Finding a Winning Move in Nim using Nim-Sum

We will now return to the starting position $(21, 45, 12, 9)$, and use the Nim-Sum strategy to ascertain whether this is a P-position or an N-position. If it is an N-position we will also find a winning move.

Firstly, we compute the Nim-Sum of the four piles, this is done in the following table:

21		0	1	0	1	0	1
45		1	0	1	1	0	1
12		0	0	1	1	0	0
9		0	0	1	0	0	1
59		1	1	1	1	0	1

The Nim-Sum is 59 which is not 0, meaning the game is in an N-position and it is possible for the next player to play a move that guarantees a win with continued perfect play. To find this move we follow the process from part 2 of the above proof. We can remove from the 45 pile such that every column in the Nim-Sum is 0. We know we can do this as the first 1 has to be removed to balance the first column. From there we can select any combination of 1s and 0s for the other columns and this number will be less than the starting number meaning it is a possible move. The easiest way to find the move explicitly is to compute $45 \oplus 59$ which is $(101101)_2 \oplus (111101)_2 = (010000)_2$, a binary representation of 16. Therefore the only winning move is to reduce the pile of 45 to 16.

2.2 Graph Games

To further analyse impartial games and alternate variations on Nim we can identify positions in a game with vertices on a graph and moves of the game with edges of a graph. This will help us define the Sprague-Grundy function which yields more information than just whether a position is a P-position or an N-position.

2.2.1 Games Played on Directed Graphs

To begin the analysis we first give the mathematical definition of a directed graph.

Definition 2. A **directed graph** is a pair (X, F) where X is a nonempty set of vertices and F is a function that gives for each $x \in X$ a subset of X , $F(x) \subset X$. For a given **position** $x \in X$, $F(x)$ represents the positions that can be moved to from x , these are known as the **followers** of x . If $F(x)$ is empty then x is a terminal position.

A two-person impartial game may be played on such a graph $G = (X, F)$ according to the following rules:

1. Left moves first from the starting position $x_0 \in X$.
2. Players make alternate moves.
3. At position x , the player whose turn it is selects a position $y \in F(x)$ to move to.
4. The first player to be confronted with a terminal position at his turn loses.

For our graphical interpretation of impartial games we require a further constrain that the graphs are **progressively bounded**. This means that there exists a number n such that every path from starting point x_0 has length less than or equal to n . A **path** is defined formally as a sequence x_0, x_1, \dots, x_m such that $x_i \in F(x_{i-1})$ for all $i = 1, \dots, m$, where m is the length of the path. The fact that all paths in progressively bounded graphs are finite means the game will always conclude in a finite number of moves, and cannot contain any **cycles**. (A cycle is a path x_0, x_1, \dots, x_m with $x_0 = x_m$ and distinct vertices x_0, x_1, \dots, x_{m-1} for $m \geq 3$.)

2.2.2 The Sprague-Grundy Function

Definition 3. The **Sprague-Grundy Function** of a graph, $G = (X, F)$, is a function, g , defined on X and taking non-negative integer values such that

$$g(x) = \min\{n \geq 0 : n \neq g(y) \text{ for } y \in F(x)\}.$$

In plain English, $g(x)$ is the smallest non-negative integer not found among the Sprague-Grundy followers of x . It may be easier to consider this function in terms of the **minimal excludant**, or **mex**, of a set of non-negative integers. This is simply the smallest non-negative integer not in the set. Then we can redefine the Sprague-Grundy function as

$$g(x) = \text{mex}\{g(y) : y \in F(x)\}.$$

We can see from this equation that $g(x)$ is defined by a self-starting recursion, meaning $g(x)$ is defined in terms of $g(y)$ for all followers y of x . For terminal positions, x , the definition implies that $g(x) = 0$, as $F(x)$ is the empty set for all terminal x . For positions x that only have terminal followers, $g(x) = 1$. We can continue in this inductive fashion to label all vertices of any progressively bounded graph.

We can use the Sprague-Grundy function g of a graph to easily analyze the corresponding graph game. Positions x with $g(x) = 0$ are P-positions and all other positions are N-positions. If the Sprague-Grundy values are known then the winning procedure is simply to choose moves that move to a vertex of Sprague-Grundy value zero. To check this we can compare the Sprague-Grundy values to the three conditions of P and N positions in the proof in Section 2.1.4:

1. If x is a terminal position, $g(x) = 0$.
2. At positions x for which $g(x) = 0$, every follower y of x is such that $g(y) \neq 0$.
3. At positions x for which $g(x) \neq 0$, there is at least one follower y such that $g(y) = 0$.

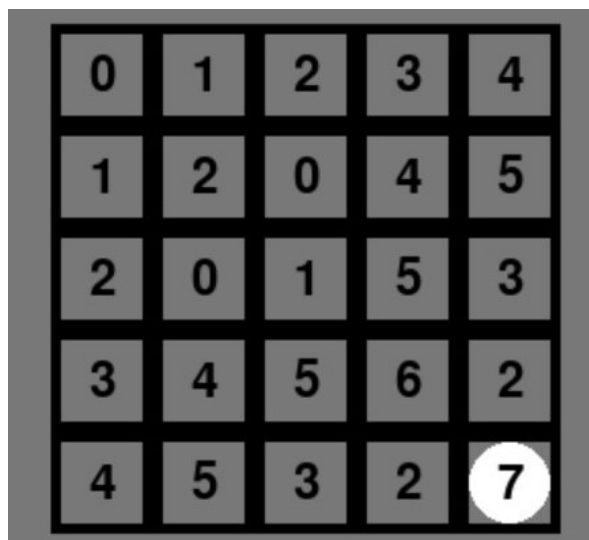
The Sprague-Grundy function therefore contains more information about a game than just the P- and N-positions. As we will see in the next section, this information allows us to analyze the disjunctive sum of graph games.

2.2.3 Sprague Grundy Function Example with Wythoff's Game

Wythoff's Game [5] is played in a similar way to Nim with just two piles of tokens. On a turn, the player can remove any positive number of tokens from a pile, as in Nim, or they can remove the same number of tokens from both piles. As usual, the first player unable to move loses and their opponent wins.

This game can be visualised on a chessboard with a Queen that can only move up, left, or diagonally up and left any amount of squares. The number of squares the queen is away from the left edge of the chessboard corresponds to one Nim pile and the number of squares the Queen is away from the top of the chessboard corresponds to the second Nim pile. The diagonal moves towards the top left correspond to removing the same number of chips from both piles.

The Sprague Grundy function of this game is non-trivial but can be computed recursively working backwards from the terminal position in the top left of the chessboard that we know to have value 0. From here any position that has all its legal moves already calculated can be computed as the mex of the Sprague Grundy values of the positions that can be moved to. The values of each position are shown in Figure 1, with the queen currently in a position of Sprague-Grundy value 7.



0	1	2	3	4
1	2	0	4	5
2	0	1	5	3
3	4	5	6	2
4	5	3	2	7

Figure 1: Sprague-Grundy values for Wythoff's Game

2.3 Sums of Combinatorial Graph Games

Two-player combinatorial games can be played simultaneously which results in more complex games with harder decision making. This is done by setting an initial position in each of the games and alternating moves as before. When a player moves, they select one of the games to make a legal move in, leaving the other games untouched. The game ends when all of the games have reached a terminal position and, as before, the player who made the last move is the winner.

2.3.1 The Disjunctive Sum of n Graph Games

Definition 4. Suppose we have n progressively bounded graphs, $G_1 = (X_1, F_1), G_2 = (X_2, F_2), \dots, G_n = (X_n, F_n)$. We can combine them into a new graph, $G = (X, F)$, known as the **disjunctive sum** of G_1, G_2, \dots, G_n , denoted $G = G_1 + \dots + G_n$ in the following way. The set of vertices X is the Cartesian product $X = X_1 \times \dots \times X_n$, which is the set of all n -tuples (x_1, \dots, x_n) such that $x_i \in X_i$ for all i . For a vertex $x = (x_1, \dots, x_n) \in X$, the set of followers of x is defined as

$$\begin{aligned} F(x) = F(x_1, \dots, x_n) = & F_1(x_1) \times \{x_2\} \times \dots \times \{x_n\} \\ & \cup \{x_1\} \times F_2(x_2) \times \dots \times \{x_n\} \\ & \cup \dots \\ & \cup \{x_1\} \times \{x_2\} \times \dots \times F_n(x_n). \end{aligned}$$

Following this definition, a move from $x = (x_1, \dots, x_n)$ consists of moving in exactly one of the x_i to one of its followers in $F_i(x_i)$. The graph game played on G is the **sum** of the graph games G_1, \dots, G_n .

If each graph G_i is progressively bounded then it follows that the sum G is also progressively bounded. It also follows that the maximum number of moves from a vertex $x = (x_1, \dots, x_n)$ is the sum of the maximum numbers of moves from each of the component graphs.

2.3.2 The Sprague-Grundy Theorem

The following theorem gives a method for obtaining the Sprague-Grundy function for a sum of graph games when the Sprague-Grundy functions are known for each of the component games. It is a result that makes it much easier to analyse complex positions that arise from the disjunctive sum of multiple games. The theorem involves the notion of Nim-Sum, defined earlier, and can be considered to be a generalization of Theorem 1. In plain English, the theorem states that the Sprague-Grundy function of a sum of graph games is the Nim-Sum of the Sprague-Grundy functions of its component games.

Theorem 2. If g_i is the Sprague-Grundy function of G_i for all $i = 1, \dots, n$, then $G = G_1 + \dots + G_n$ has Sprague-Grundy function $g(x_1, \dots, x_n) = g_1(x_1) \oplus \dots \oplus g_n(x_n)$.

Proof. Let $x = (x_1, \dots, x_n)$ be an arbitrary point of X and let $b = g_1(x_1) \oplus \dots \oplus g_n(x_n)$. We need to show that $g(x_1, \dots, x_n)$ satisfies the following two statements:

1. For every non-negative integer $a < b$, there exists a follower of (x_1, \dots, x_n) that has a g-value (Grundy value) of a .
2. No follower of (x_1, \dots, x_n) has g-value b .

If these statements hold then the Sprague-Grundy value of x must be b as it is the smallest Sprague-Grundy value not assumed by one of its followers. As x is an arbitrary point in X , this is sufficient to complete the proof that the Sprague-Grundy function of a sum of games is $g_1(x_1) \oplus \dots \oplus g_n(x_n)$.

To show (1), let $d = a \oplus b$, and let k be the number of digits in the binary expansion of d . We have $2^{k-1} \leq d < 2^k$ and d must have a 1 in the k th position from the

right. Since $a < b$, b has a 1 in k th position and a must have a 0 in the k th position. Since $b = g_1(x_1) \oplus \cdots \oplus g_n(x_n)$, there is at least one x_i such that the binary expansion of $g_i(x_i)$ has a 1 in the k th position. Suppose without loss of generality that $i = 1$. Then $d \oplus g_1(x_1) < g_1(x_1)$ so there must be some move from x_1 to some x'_1 with $g_1(x'_1) = d \oplus g_1(x_1)$. The move from (x_1, \dots, x_n) to (x'_1, \dots, x_n) is a legal move in G , so $g_1(x'_1) \oplus \cdots \oplus g_n(x_n) = d \oplus g_1(x_1) \oplus \cdots \oplus g_n(x_n) = d \oplus b = a$. This proves that there is always a follower of (x_1, \dots, x_n) that has a g -value of a for arbitrary non-negative integer $a < b$.

To show (2), suppose for contradiction that (x_1, \dots, x_n) has a follower with the same g -value, and suppose without loss of generality that this follower involves a move in the first game. So (x'_1, \dots, x_n) is the follower of (x_1, \dots, x_n) and $g_1(x'_1) \oplus g_2(x_2) \oplus \cdots \oplus g_n(x_n) = g_1(x_1) \oplus g_2(x_2) \oplus \cdots \oplus g_n(x_n)$. By the cancellation law, $g_1(x_1) = g_1(x'_1)$, which is a contradiction as no position can have a follower of the same Sprague-Grundy value. \square

The Sprague-Grundy Theorem is a powerful tool for evaluating game positions when multiple impartial games are being played simultaneously and is one of the main results of this section. We will now try and solidify our understanding of the theorem by using it in the context of a simultaneous game of Nim and Wythoff's Game.

2.3.3 Example of the Sprague-Grundy Theorem

The beauty of the Sprague-Grundy Theorem is that it allows us to add together any progressively bounded impartial combinatorial games as if they were Nim-heaps of the same Sprague-Grundy value. This means that Theorem 1 still holds for any impartial combinatorial game, so we are in a P-position if the Nim-Sum of the Sprague-Grundy values of the individual games is 0, and we are in an N-position otherwise. Consider playing a game of Nim simultaneously with Wythoff's Game as set up in Figure 2:

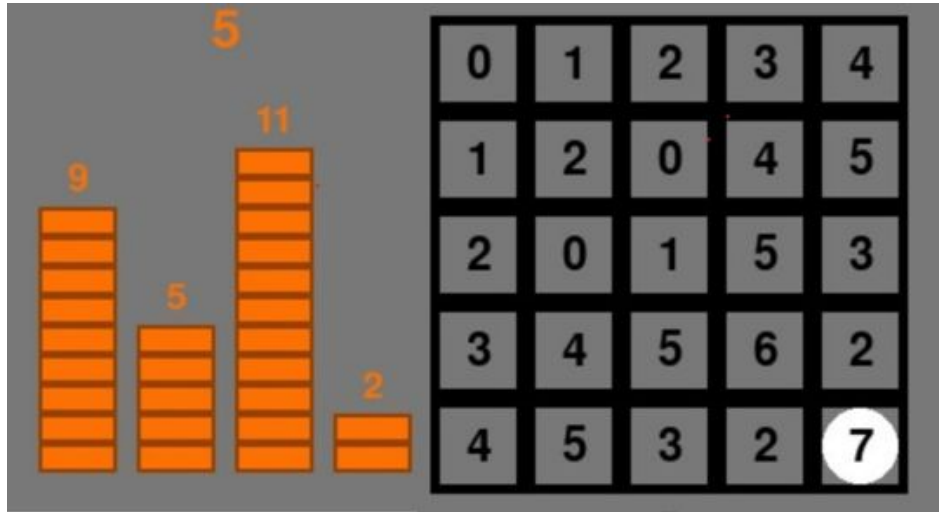


Figure 2: Set-up of Nim and Wythoff's Game played simultaneously

We can think of this game as the sum of five different components: the four Nim heaps and the Queen game. Using the Sprague-Grundy Theorem we can use binary

addition to add the value of the Queen game to the Nim-Sum of the Nim heaps to get the total value for the combined game. $5 \oplus 7 = 2$ so we are in an N-position. The two optimal moves here are moving the queen to a position of Sprague-Grundy value 5, or removing two chips from the 11-pile. Both moves would balance the Nim-Sum of the combined game leaving a P-position.

We will now introduce a more difficult impartial game called Green Hackenbush and discuss how the Sprague Grundy Theorem can still be applied.

2.4 Green Hackenbush

2.4.1 Green Hackenbush Introduction

Green Hackenbush is another example of a two player impartial combinatorial game. We begin by drawing a picture like in Figure 3, made up of green lines that are all connected to the ground. When the picture is drawn the two players, Left and Right, take it in turns to "hack" an edge of the graph and remove it from the game. Removing an edge also removes any edges that are no longer connected to the ground (the area highlighted in yellow at the bottom of the picture). The game ends when one of the players no longer has an edge they can cut and in normal play this player is the loser.

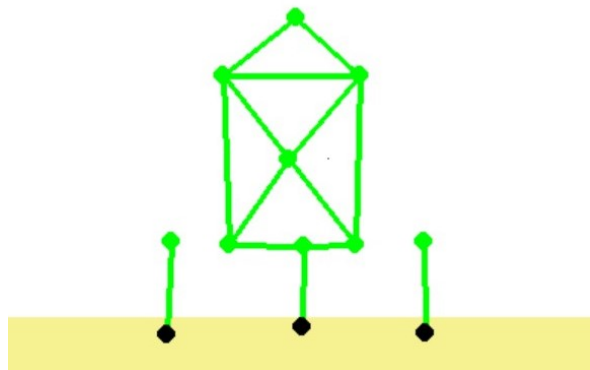


Figure 3: Simple Green Hackenbush Position.

As an example, consider the position in Figure 3. It is not too difficult to see that this is an N-position, with the obvious winning move being to remove the center structure completely. This leaves two single edges for the opponent and when one is taken we can secure the win by removing the final edge.

Now consider the same starting position but with only one stand-alone edge rather than two. Suddenly it becomes very difficult to establish whether the position is an N-position or a P-position or to find a winning move if one exists. In this section we form a complete theory for Green Hackenbush making use of the Sprague-Grundy Theorem, the Colon Principle and the Fusion Principle. Our understanding of the rather complicated proofs in this section came from [6], [7] and [8].

2.4.2 Sprague-Grundy Theorem for Green Hackenbush

As Green Hackenbush is an impartial game, we know that every position must have a Grundy value. We also know that if we break the game down into **components** by

separating the game into smaller games according to which root nodes the segments are held down by, then the Nim-Sum of the Grundy values of the components equals the Grundy value of the full game by the Sprague-Grundy Theorem. Consider the simple example Hackenbush position in Figure 2 made up of three "stalks". The position is made up of three components that can be interpreted as Nim-heaps of size 3, 1 and 4 giving the position an overall Sprague-Grundy value of 2.

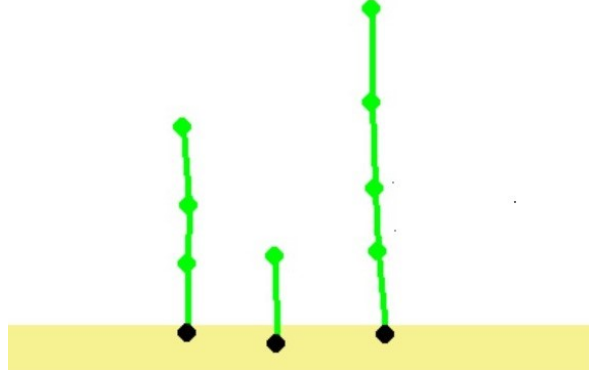


Figure 4: Simple Green Hackenbush Position.

The corresponding winning move in this position would be to remove the top two edges from the right stalk to leave the Nim-Sum zero.

Using this comparison to Nim, we completely understand any Hackenbush position that just consists of stalks from root nodes. The following two principles work to reduce any green Hackenbush component to a stalk so that it can be easily evaluated.

2.4.3 The Colon Principle

Theorem 3. *Suppose we have a picture G with some node, a , and we have pictures H and K such that H and K have the same Grundy value. Then if we "stick" picture H to G at a to create $G \cdot H_a$ This has the same Grundy value as if we had stuck K to G at a to create $G \cdot K_a$.*

To visualise the Colon Principle consider Figures 5 and 6. Figure 5 shows two general Hackenbush components H and K which we assume have the same Grundy value.

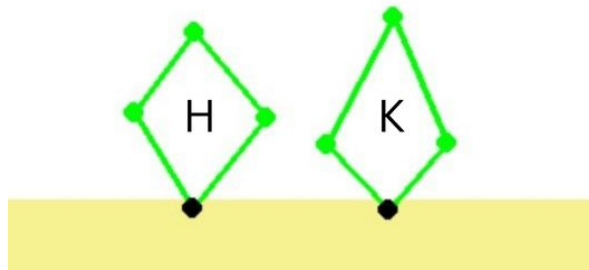


Figure 5: Simple Green Hackenbush Position.

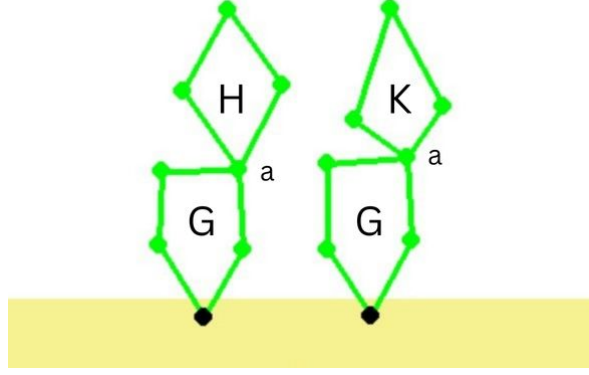


Figure 6: Simple Green Hackenbush Position.

We introduce another general component G and connect components H and K to G at node a as shown in Figure 6. The Colon Principle tells us that these two new structures still have the same Grundy value.

We now give a proof of the Colon Principle

Proof. To see this result we simply play Green Hackenbush on the sum of the two pictures described in the theorem, $G \cdot H_a + G \cdot K_a$. If the two games are equal then the value of this combined position should be zero, meaning the game is in a P-position and there is a winning strategy for the second player to win. Let the first player make a move, if this move is in G then the opponent can respond with the same move in the G section of the other component. Otherwise, if the player made a move in H or K to reduce the component to some value x , the same move must exist in the other component to re-balance the Nim-Sum as H and K have the same value. This mimicking strategy can be continued through the entire game by player 2, so this player will never not have a move and so must win the game. This shows that the position $G \cdot H_a + G \cdot K_a$ has a Nim-Sum zero, so the positions $G \cdot H_a$ and $G \cdot K_a$ must have the same value, proving the colon principle. \square

We will now demonstrate how we can use the Colon Principle to reduce "trees" to "stalks". Consider the tree in Figure 5, to reduce the tree we look for nodes that branch out into multiple stalks, identified below as a , b and c .

At node a , we have two stalks of length 1. As they are connected to the rest of the picture at the same node we can use the Colon Principle to compare the stalks. We have $1 \oplus 1 = 0$ so the stalks cancel and we can therefore remove both of them without changing the value of the picture. At node b we have stalks of length 1 and 2. $1 \oplus 2 = 3$ so we can replace these stalks with one stalk of length 3 connected at node b . Finally, at node c we have stalks of length 3 and 2. $3 \oplus 2 = 1$ so we can replace these stalks with a stalk of length 1 connected at c . The resulting tree is shown in Figure 8.

From the tree in Figure 8, all that is left is to compare the stalks coming from node d . We get $2 \oplus 4 \oplus 2 = 4$ so, connecting a stalk of length 4 back to d , we get that the total value of the tree is 5.

We now perfectly understand any Hackenbush position that has components that are "trees", we now need consider components that contain cycles.

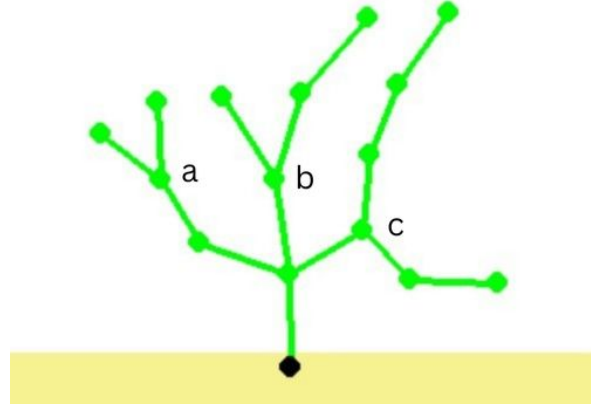


Figure 7: Simple Green Hackenbush Position.

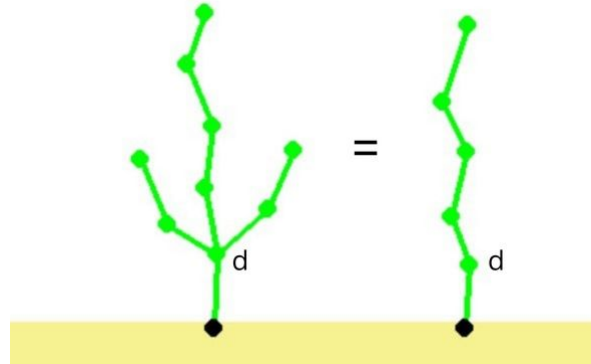


Figure 8: Simple Green Hackenbush Position.

2.4.4 The Fusion Principle

A **Cycle** in a Green Hackenbush position is a set of edges that forms a closed loop in a picture of Green Hackenbush. When considering cycles we count the ground as being one node. We can define **Fusion** as the process of taking two nodes in a cycle and **fusing** them into a single vertex, bending any edges that join the two nodes into a loop at the new node. These definitions put us in a position to define the Fusion Principle.

Theorem 4. *Let G be a position in a game of Green Hackenbush. Let H be the position obtained by fusing together some or all of the vertices in a cycle of length $k \geq 2$ in G . Then G and H have the same Sprague-Grundy value.*

Proof. Suppose, for contradiction, that this is not the case and we can find a picture such that some of its nodes cannot be fused together without changing the Grundy value. Out of all possible counter-examples there must be a picture with the smallest number of edges (lets say this picture has n edges). Out of all counter-examples with n edges there must be one with the smallest amount of nodes, we call this picture G . We know for picture G that we cannot fuse together any of its nodes as this gives a picture with less nodes and the same amount of edges, violating how we chose G . We now observe four properties that G must possess.

1. G can only have one node that touches the ground. This comes from the fact that fusing all ground nodes on any picture never changes the value.

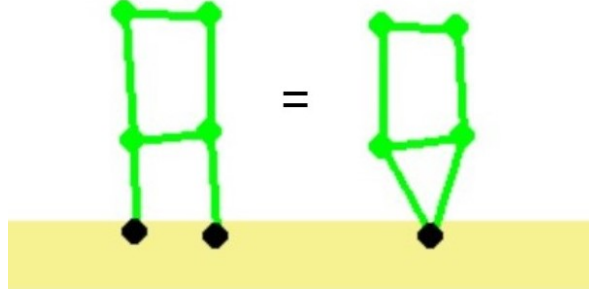


Figure 9: Fusing Ground Nodes

2. G cannot contain a pair of nodes x and y such that three or more distinct **paths** (sets of connected edges) connect x and y . This is because if G did, then we could create a new game H by fusing x and y . By the way we defined G , fusing any two nodes changes its value, so G and H have different Sprague-Grundy values. This means the game $G + H$ has a positive Nim-Sum so there exists a winning move for the first player.

Lets say the first player plays the winning move in G or H and the second player responds by cutting an edge in whichever picture the first player did not play in. This results in two new games G' and H' , which both have $n - 1$ edges. We can therefore fuse all nodes in these pictures without changing their values. As x and y started with three paths, there is still a cycle looping them together in either G' or H' as we have deleted at most one path of the three available. After fusion, we have that $G' = H'$, so $G' + H'$ is a zero game meaning the first player who played a winning move now finds them-self in a losing position. This means G and H were worth the same value, contradicting our definition of G and proving that G cannot contain a pair of nodes x and y such that three or more distinct paths.

3. No cycle in G can exclude the ground. We prove this by considering a picture G with a cycle C . We create a new picture G' by cutting every edge from G that is in C . G' must contain only one node, x , from C , as if it had two then those two nodes would be connected by three paths in G (two in C and one in G'). We therefore have that C is only connected to the rest of G at x , as shown in Figure 10. If x is above the ground, we can apply the Colon Principle to C and whatever is attached to it and consider this part separately. We can apply the fusion principle to this part of the picture as it has fewer vertices than G , however this means that the fusion principle also applies to G and so x must be on the ground.
4. G contains only one cycle that touches the ground, as otherwise it would be the sum of two smaller graphs since nodes from distinct cycles cant be joined by other parts as this violates observation 2. We could then individually apply the Fusion Principle to the smaller graphs so the Fusion Principle would work for G .

With these four observations about G taken into consideration we have massively reduced the number of pictures that G could look like. In fact, we know that G must

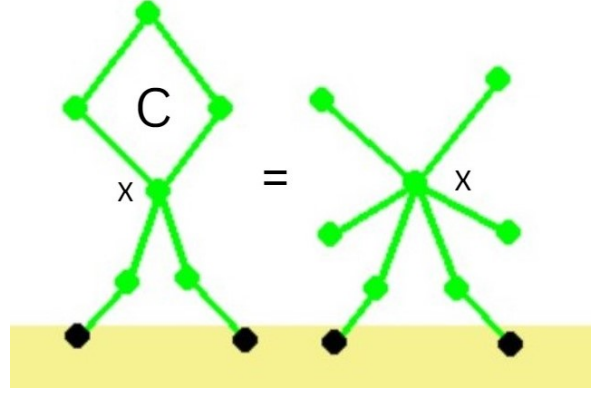


Figure 10: If x is not on the Ground we can use the Colon Principle to Fuse C

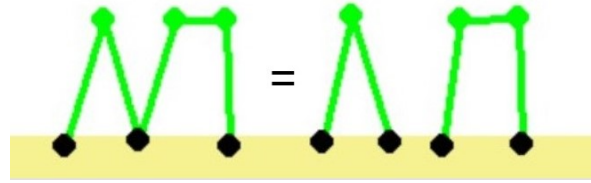


Figure 11: Clearly we can Split into Two Components

look like a bridge with "things" coming off it. By the Colon Principle we can apply fusion to the individual "things" to turn them into stalks.

If the Fusion Principle holds, the value of a bridge is the Nim-Sum of its stalks $+1$ if the bridge is an odd length and $+0$ if the bridge is an even length. This is because fusing a bridge of length n takes all the bridge nodes to the same node, creating n single loops equivalent to stalks of length 1.

If n is even, by our theory taking a bridge with an arbitrary number of stalks and the same stalks connected to the ground (as shown in Figure) results in a zero game. If the first player moves in a stalk then the second player can cut the other identical stalk in the same place to return to a zero game, we can then apply the Fusion Principle on this smaller game to show it is in fact a zero game. If the first player instead moves in the bridge, this will always lead to a game with non-zero value as it will leave an odd number of edges in the picture and Nim-addition respects parity. Using the Colon Principle we can reduce what remains of the bridge into a stalk of non-zero length and there will exist a move for the second player to return the position to a zero game, securing the win. So Fusion holds on even bridges.

If n is odd we can use the same argument where we add a stalk of length 1 to compensate for the $+1$ given by fusing an odd bridge using our theory. This means we expect Figure 13 to be a zero game. Again, if the first player moves in a stalk the second player can move in the opposite stalk and the Fusion Principle then shows us we are still in a zero game. Instead, if player 1 moves in the bridge, the resulting picture must be odd as we have an even number of edges in the bridge left, an even number of edges in the stalks (as there are two of each) and the one extra edge to turn the Nim-Sum odd. Again this result holds as Nim-addition respects parity. Using the Colon Principle the full picture can be reduced to a stalk of odd height and so there

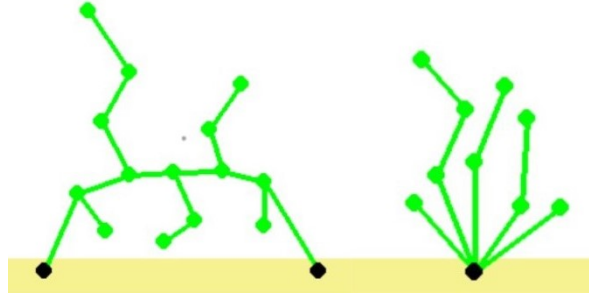


Figure 12: A Zero Position with an Even Bridge

exists a move for player 2 to return the position to a zero game. Therefore the position in Figure X is a zero position and so Fusion also holds on odd bridges as well.

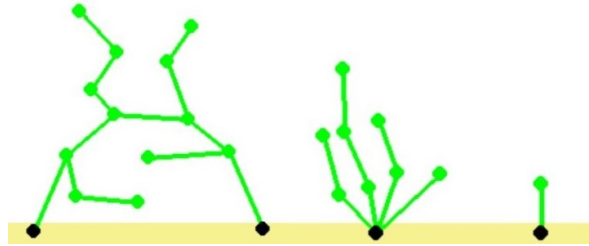


Figure 13: A Zero Position with an Odd Bridge

To sum up the proof we first showed that if there exists a position G such that the Fusion Principle does not hold then it must look like a bridge. We then showed that Fusion holds for all bridges so we can conclude that Fusion holds for all possible game positions. \square

2.4.5 Green Hackenbush Combined with Other Impartial Games

Now that we have a full understanding of computing the Sprague Grundy value for any Green Hackenbush component we can play it simultaneously with other games like we did before with Nim and Wythoff's game and still find the winning move in any N-position. Consider the game in Figure 14 which is the disjunctive sum of a game of Nim, Wythoff's Game and Green Hackenbush.

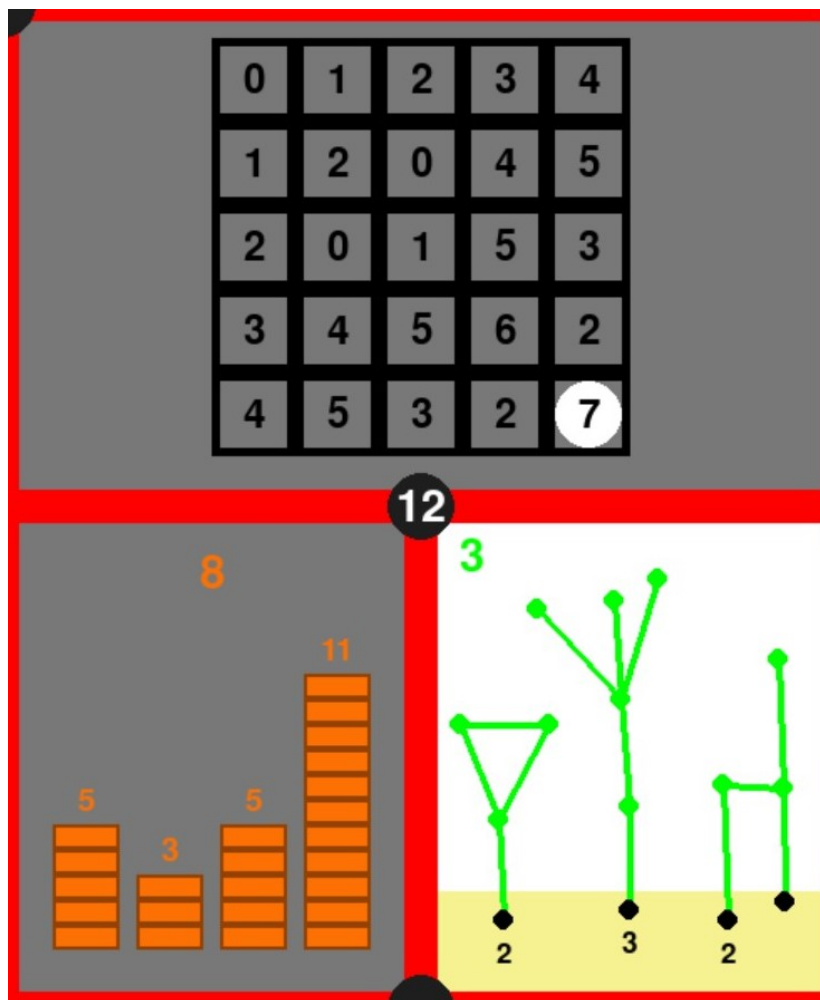


Figure 14: A Combined game of Wythoff's Game, Nim and Green Hackenbush

The components in the Hackenbush picture follow the rules of all Impartial game components in that we can treat them as Nim-heaps of the size of their Sprague Grundy value. Adding the three Hackenbush components, the four Nim blocks and the Wythoff Game gives a value of 12 so we are in an N-position. A visual trick we can use to quickly balance the Nim-sum and find a winning move is to cancel out pairs of components with the same Sprague-Grundy value. We can spot that the two Nim-heaps of size 5 cancel with each other, the two Hackenbush components with value 2 cancel and the Nim-heap of size 3 cancels with the Hackenbush components of value 3. This leaves just the Nim-heap of size 11 and the Wythoff Game that has value 7 and we can see that the obvious winning move is to reduce the 11-heap to size 7 so that it cancels with the Wythoff Game. If we are able to keep this pairing of components in our head, then when our

opponent responds with a move we can return the game to a P-position by playing the equivalent move in the other component of the pair. This gives us a strategy to beat an opponent in real time without too much thinking.

A fully functional tool for visualising and playing these games simultaneously is available at the project's [Github repository](#). The game features set up tools for the three games and an optional opponent that will find a winning move in any given N-position.

3 Vector Subtraction Games and the Limits of Computability

3.1 Nim with a Set of Legal Moves.

3.1.1 Introduction to the set \mathcal{M}

We will now study a different variation on the game of Nim. As previously discussed, a game of Nim with a fixed number of heaps, d , can be represented as a d -dimensional vector $\mathbf{x} = (x_1, \dots, x_d)$ of non-negative integers. We will refer to the number of heaps, d , as the **dimension** of the game.

In this variation, we introduce a finite set, \mathcal{M} , containing integer vectors that specify the legal moves available to both players and define the game. For example, if $(m_1, \dots, m_d) \in \mathcal{M}$ then a player can move from a position (x_1, \dots, x_d) to a new position $(x_1 - m_1, \dots, x_d - m_d)$ provided all numbers $x_1 - m_1, \dots, x_d - m_d$ are non-negative. With this construction, m_i is the number of chips removed from heap i if the move is played. If m_i is negative then we are adding chips to heap i when this move is played. To prevent cycles and ensure the game will eventually terminate we apply the condition that for each $(m_1, \dots, m_d) \in \mathcal{M}$, we have that $m_1 + \dots + m_d > 0$. This ensures that the total number of matches after a move is played will always decrease.

For each game \mathcal{M} , there exists an infinite set of P-positions which we denote $P(\mathcal{M})$. In this section we investigate to what extent we can understand $P(\mathcal{M})$ as a perfect understanding of $P(\mathcal{M})$ gives us complete understanding of how to win the game \mathcal{M} . When first approaching this problem, we note that for any position \mathbf{x} in a game \mathcal{M} it is possible to recursively determine if \mathbf{x} is a P-position or not by first computing the status of all positions that can be moved to from \mathbf{x} . We know that $\mathbf{x} \in P(\mathcal{M})$ if and only if there is no move from \mathbf{x} to a position in $P(\mathcal{M})$. Using this recursive technique we can work backwards from the terminal positions of a game to get a map of P-positions.

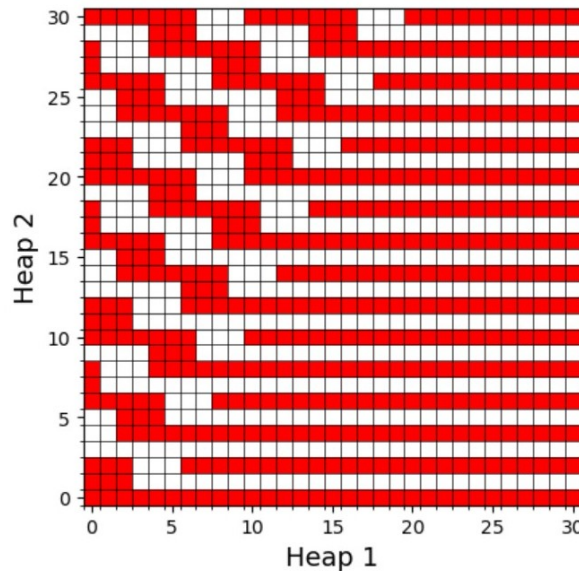


Figure 15: P-positions for the game $\mathcal{M} = \{(3, 1), (-1, 3)\}$

In the figure above we can see two clear **regions/segments** with their own periodic P-position pattern. In an example like this it is possible to state the pattern explicitly and prove by induction that it holds for heaps of any size. This would give us a complete understanding of $P(\mathcal{M})$ for this game.

Unfortunately, the same cannot be said for all games. In many cases, the P-positions show no signs of regularity even after extensive computation. The P-positions for the game $\mathcal{M} = \{(0, 2), (2, 0), (-2, 3), (3, -2), (5, -4), (5, 2), (4, 3), (1, 4)\}$ are shown in Figure 16, going some way towards demonstrating the "chaotic" behaviour along borders of different regions.

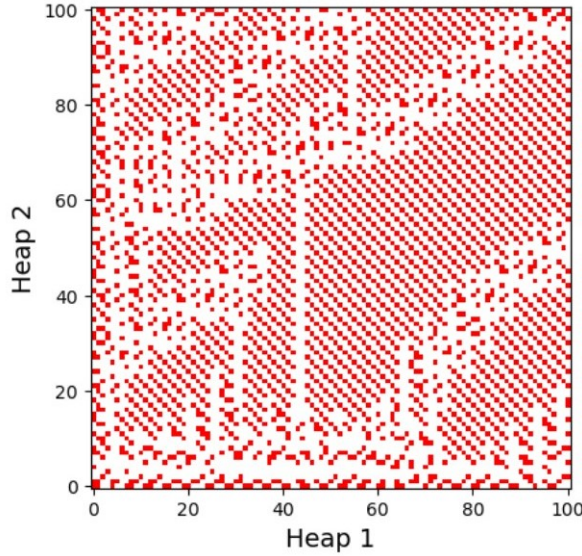


Figure 16: P-positions for the game $\mathcal{M} = \{(0, 2), (2, 0), (-2, 3), (3, -2), (5, -4), (5, 2), (4, 3), (1, 4)\}$, an example game also used in [9]. If we extend the map further we get new regions appearing making the task of calculating an explicit pattern even harder.

Despite the simplicity in the rules of these games, it has been proven in [9] that it is generally impossible to fully understand the behaviour of the set $P(\mathcal{M})$ given a list of legal moves \mathcal{M} . The formal definition of "full understanding" is ambiguous here, but in [9] it is argued that full understanding at least implies the ability to recognise when two games are P-equivalent, meaning they have the same P-positions.

Theorem 5. *There is no algorithm that, given as input the number d of heaps and two finite sets \mathcal{M} and \mathcal{M}' of integer vectors specifying the rules of two d -heap games, decides whether or not $P(\mathcal{M}) = P(\mathcal{M}')$.*

If Theorem 3 holds, then clearly we do not have full understanding of the set $P(\mathcal{M})$ given legal moves \mathcal{M} . We will now spend some time carefully proving Theorem 3 following the extended proof in [9] using Cellular Automata, adding additional insights and explanations where necessary.

3.1.2 Introduction to Cellular Automata

A Cellular Automata (CA) consists of a grid of cells, each of which can exist in a finite number of states. The system evolves over discrete time steps, where the state of each cell is determined by a set of local rules based on its own state and the states of neighboring cells. Despite its simple formulation, CA exhibits remarkably complex behavior, capable of generating intricate patterns and dynamics from elementary rules. Here, we restrict CAs to only having 2 states (0 and 1) with the state of cell i at time t being $x_{t,i}$. We begin with the starting configuration

$$x_{0,i} = \begin{cases} 1 & \text{if } i \geq 0, \\ 0 & \text{if } i < 0. \end{cases}$$

The update rule of a CA is determined by two main components: a parameter n representing the neighborhood size, and a Boolean function f which operates on n bits of input. Put formally,

$$x_{t+1,i} = f(x_{t,i-n+1}, x_{t,i-n+2}, \dots, x_{t,i}).$$

which means $x_{t+1,i}$ depends on $x_{t,i}$ and the $n - 1$ cells immediately to the left of $x_{t,i}$. We put a further requirement on f that $f(0, \dots, 0) = 0$ which ensures that $x_{t,i}$ remains at 0 for all t when $i < 0$.

Example:

We demonstrate the process by taking $n = 2$ and letting the Boolean function be

$$f(x, y) = x \oplus y,$$

so that $f(x, y)$ is equal to 0 if $x = y$ and 1 if $x \neq y$. The 1's are represented by red squares and we have time flowing upwards so that each updated tape is stacked on top of the previous tape. We do not need to include positions $x_{t,i}$ for negative i because, as we previously stated, these positions are 0 for all t . Therefore we start the CA at $i = 0$.

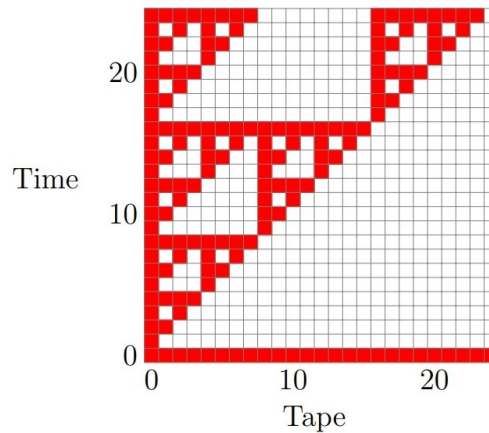


Figure 17: The CA given by $f(x, y) = x \oplus y$, positions on the grid are explicitly calculated recursively as $x_{t+1,i} = x_{t,i} \oplus x_{t,i-1}$. (Figure taken from [9])

Although the pattern is non-periodic, we still understand it perfectly as Pascal's triangle modulo 2. There are Boolean functions that give much more difficult behaviour to understand.

With the basics of cellular automata now described, we are in a position to introduce the first of three lemmas that we will use to prove Theorem 3.

Lemma 1. *There is no algorithm that takes as input an n -ary Boolean function f , and answers whether or not the string 101 ever occurs in $CA(f)$.*

We will prove this lemma by relating the situation to The Halting Problem. The Halting Problem can be stated simply as follows: Given a description of a Turing Machine M , and an input w , will M halt (finish its computation) given input w ? It was famously proved by Alan Turing in 1936 [11] that it is impossible to create a general algorithm that can accurately determine whether any given computer program will halt or continue indefinitely.

To emulate the halting problem with CA we need to construct a CA that behaves analogously to a universal Turing machine. This can be done by designing a binary CA $CA(f)$ that takes in the description of a Turing machine M and its input w to simulate the computation of M on w . In the CA, we introduce the concept of halting by programming $CA(f)$ to transition to the halting symbol 101 if and only if the emulated Turing machine M halts on input w .

By construction, the $CA(f)$ is capable of determining whether the simulated Turing machine M halts on input w meaning $CA(f)$ effectively solves the halting problem. This contradicts the undecidability of the halting problem, so there cannot exist an algorithm that determines whether the string "101" (or any other specific pattern) occurs in the evolution of the CA, proving lemma 1.

We will now go on to construct a game emulating a generic cellular automaton using a class of non-invariant games known as **Modular** games.

3.1.3 Using Modular Games to Emulate Cellular Automata

A modular game \mathcal{G} has only two heaps, (x_1, x_2) . We call x_1 the tape-heap and x_2 the time-heap, corresponding to the tape and time axes in the CA of Figure 3. In modular games, instead of one set \mathcal{M} containing the legal moves for the game we have k finite sets of integer vectors $\mathcal{M}_0, \dots, \mathcal{M}_{k-1}$ specifying the legal moves. For a given game position (x_1, x_2) the set of available moves is given by \mathcal{M}_i , where $0 \leq i < k$ and

$$i \equiv x_2 \pmod{k}.$$

For modular games we require that the time-heap decreases with every move and that the tape-heap does not increase. This means that for $(m_1, m_2) \in \mathcal{M}_i$ we have $m_1 \leq 0$ and $m_2 < 0$ and we can move from (x_1, x_2) to $(x_1 + m_1, x_2 + m_2)$ provided $x_1 + m_1 \geq 0$, $x_2 + m_2 \geq 0$ and $i \equiv x_2 \pmod{k}$. We now want to prove the following lemma relating to the P-equivalence of modular games.

Lemma 2. *For each Boolean function f we can effectively construct two modular games \mathcal{G} and \mathcal{H} that are P-equivalent if and only if the word 101 never occurs in $CA(f)$.*

We will prove lemma 2 by constructing a modular game that emulates $CA(f)$ for an arbitrary boolean function f .

When emulating $CA(f)$, the number of chips in the tape-heap will match with the position on the tape of the CA. However on the time axis there will be some space to allow for the rules of the modular game to compute the Boolean function f . We will choose k such that positions with kt chips in the time-heap will correspond to the state of $CA(f)$ at time t .

We now need a more "interactive" way of expressing f so we can visualise the construction from the modular games. We do this using square bracket notation. Let square brackets $[\cdot]$ denote $1 - \max(\cdot)$, so we have

$$[xyz] = 1 - \max(x, y, z)$$

and, by convention, $[] = 1$. We can express every boolean function in terms of nested brackets as the set consisting of $\&$ and \neg is a complete set of connectives in propositional logic. As an example we will consider emulating $f(x, y) = x \oplus y$ to show that we can recreate the CA from earlier.

x	y	$[x]$	$[y]$	$[xy]$	$[[x][y]]$	$[[xy][x][y]]$	$f(x, y)$
0	0	1	1	1	0	0	0
0	1	1	0	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	0	0	1	0	0

Table 1: Truth table for the function $f(x, y) = x \oplus y = [[xy][x][y]]$

From the truth table above we can see that $f(x, y)$ can be expressed as $f(x, y) = x \oplus y = [[xy][x][y]]$.

For a modular game, we can let the P-positions correspond to the value 1 and N-positions to the value 0. We can use the bracket function to recursively compute the value of positions in the following way: If a position has moves to values x_1, \dots, x_k then its value is $[x_1x_2 \dots x_k]$. We can check this is correct as a P-position can only move to N-positions meaning $x_1 = \dots = x_k = 0$ which implies $[x_1x_2 \dots x_k] = 1$. An N-position must have some move to a P-position meaning there exists an $x_i = 1$ so $[x_1x_2 \dots x_k] = 0$.

We can construct a game that computes the function f by giving intermediate positions the values $[x]$, $[y]$, $[xy]$ and $[[x][y]]$, as these are the move options needed to get to $[[xy][x][y]]$. We can visualise how this is done using the following figure.

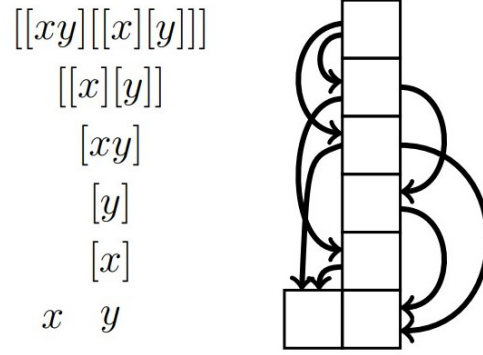


Figure 18: A modular game computing $f(x, y) = x \oplus y$ with $k = 5$. (Figure taken from [9])

The arrows in Figure 18 represent the move options from the position and we can see that they match the contents of the bracket for that position. The construction in Figure 18 corresponds to the modular game:

$$\begin{aligned}\mathcal{M}_0 &= \{(0, 1), (0, 2)\} \\ \mathcal{M}_1 &= \{(1, 1)\} \\ \mathcal{M}_2 &= \{(0, 2)\} \\ \mathcal{M}_3 &= \{(0, 3), (1, 3)\} \\ \mathcal{M}_4 &= \{(0, 2), (0, 3)\}\end{aligned}$$

Figure 19 then shows how this game emulates the CA given by $f(x, y) = x \oplus y$ with every fifth row corresponding to a row in Figure 17.

As any Boolean function f can be expressed in terms of nested brackets, we can always find a k to construct a modular game to emulate the function in this way. Now we can use the modular game to check for the word 101.

Given an arbitrary Boolean function f and its corresponding modular game \mathcal{G} , we can modify \mathcal{G} to build in a check for whether or not the input to f ends with the substring 101. We can do the check before even computing f by introducing boxes b_1 and b_2 that invert the last and third-from-last positions of the input. Essentially, these two boxes check for the 1's in the string 101. Then the third box b_3 checks the penultimate position is 0 and the status of b_1 and b_2 , completing the check for 101 in the input. We call the modified game \mathcal{G}' and illustrate the process below.

We now also construct a dummy game \mathcal{G}'' identical to \mathcal{G}' but without the check for 101. Instead of connecting b_3 to the penultimate input position, we connect it to the third-to-last input position and remove the connection to b_1 . This means b_3 will always be an N-position in \mathcal{G}'' but the values of the game positions are otherwise identical to the game \mathcal{G}' .

As the positions of \mathcal{G}' and \mathcal{G}'' are identical except for b_3 , we have that \mathcal{G}' and \mathcal{G}'' are

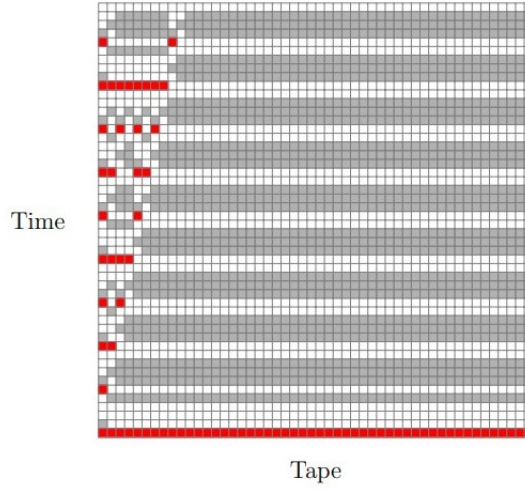


Figure 19: A modular game emulating $f(x, y) = x \oplus y$ with grey squares representing P-positions and red squares representing P-positions with $x_2 \equiv 0(\text{mod}5)$, matching the rows in Figure 17. (Figure taken from [9])

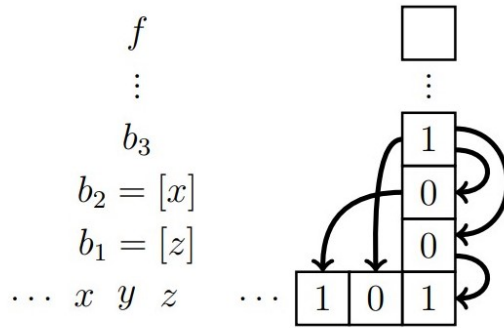


Figure 20: The modular game \mathcal{G}' . The fact that b_3 has moves to b_2 , b_1 and the penultimate input position means all three of these positions must be 0 for b_3 to be a P-position. In other words, b_3 has value 1 if and only if $xyz = 101$, making it a valid check for the substring. (Figure taken from [9])

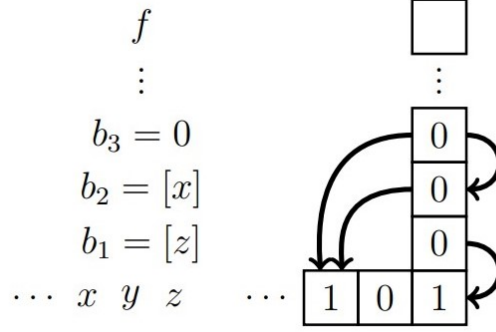


Figure 21: The modular game \mathcal{G}'' which computes an identical output function f to the one computed in the game \mathcal{G}' . The only difference is that \mathcal{G}' doesn't contain a check for 101 as the box b_3 has value 0 regardless of the input to f . (Figure taken from [9])

P-equivalent if and only if the string 101 never occurs in $CA(f)$. This completes the proof of lemma 3.

3.1.4 Emulating a modular game with an invariant game

The final lemma we introduce in this section provides the final link from modular games to invariant games that we need to complete the proof of Theorem 1.

Lemma 3. *For two modular games \mathcal{G} and \mathcal{H} , we can effectively construct invariant games \mathcal{M} and \mathcal{M}_0 that are P-equivalent if and only if \mathcal{G} and \mathcal{H} are P-equivalent.*

By nature, modular games are more complex than invariant games as they have k sets of legal moves $\mathcal{M}_0, \dots, \mathcal{M}_{k-1}$ whereas invariant games have only one set of legal moves \mathcal{M} . We can emulate this increase in complexity with invariant games by adding k more heaps than the modular game which strictly has only 2 heaps. We call these k added heaps the **gadget**. The k heaps of the gadget (labeled $0, \dots, k-1$) emulate the modular game when $k-1$ of the heaps are empty and one heap contains a single chip. The heap i containing the chip corresponds to the move set \mathcal{M}_i of the modular game. We can then construct moves in the invariant game by taking moves from the modular game and applying them to the tape-heap and time-heap as well as removing the chip from heap i and adding one to the new modulus of the time-heap. Continuing with our example from before, we can use the modular game:

$$\begin{aligned}\mathcal{M}_0 &= \{(0, 1), (0, 2)\} \\ \mathcal{M}_1 &= \{(1, 1)\} \\ \mathcal{M}_2 &= \{(0, 2)\} \\ \mathcal{M}_3 &= \{(0, 3), (1, 3)\} \\ \mathcal{M}_4 &= \{(0, 2), (0, 3)\}\end{aligned}$$

to construct the invariant game:

$$\mathcal{M} = \left\{ \begin{array}{l} (0, 1, 1, 0, 0, 0, -1) \\ (0, 2, 1, 0, 0, -1, 0) \\ (1, 1, -1, 1, 0, 0, 0) \\ (0, 2, -1, 0, 1, 0, 0) \\ (0, 3, -1, 0, 0, 1, 0) \\ (1, 3, -1, 0, 0, 1, 0) \\ (0, 2, 0, 0, -1, 0, 1) \\ (0, 3, 0, -1, 0, 0, 1) \end{array} \right\}$$

It is quite clear how this invariant game emulates the modular game as the moves in the invariant game only become available when there is a chip in the heap corresponding to the move set in the modular game. The emulation works perfectly as we described it when the starting position contains only one chip in the gadget which is in the position corresponding to the value of the time-heap modulo k . There are some slight adjustments to this invariant game that we still have to make to ensure the emulation holds for all possible starting positions of the game.

The first technicality to address is that for the construction to work we would need every move in the modular game to change the value of $x_2 \pmod k$ so that we enter into a different legal move set after every move. We want the emulation to also hold for modular games which have moves to the same move set. When these situations arise we can extend the gadget by a multiple of k which is larger than the number of matches removed from the time-heap in any of the moves. We know this can always be done as any k -modular game can be described as an lk -modular game for every positive integer l .

The next possible issue occurs when the gadget is out of phase, meaning the chip in the gadget is not in the correct position corresponding to the modulus of the time-heap. There will therefore be a first row $0 < i < k$ which corresponds to a finished computation. The gadget treats this row as if it were congruent to 0 modulo k , meaning we permit moves in \mathcal{M}_0 . Since $i - k < 0$ and $\square = 1$, $f(x_1, \dots, x_n)$ is defined uniquely by its empty input and will produce an output that is independent of the position of the tape-heap. This means that the information from row i must be constant as an N-position or a P-position. If the row is a constant N-position then the following pattern will be periodic as the computation restarts as if it had started on a tape of only zeros. If instead the row is constantly in a P-position then the behaviour from row i and onwards would be the same as if the gadget was in phase, since the computation now starts from a row identical to the starting configuration of the corresponding CA to the modular game.

The final issue we deal with is when the gadget contains chips in more than one position, say positions i and j , then a legal move can be obtained from either \mathcal{M}_i or \mathcal{M}_j independent of its number of matches. This means that the invariant game in this position no longer emulates a modular game. To combat this issue we trivialise situations with more than one match in the gadget by choosing some large N and allowing any

move that transfers two matches from the gadget to two other heaps and removes any number smaller than N from the two main heaps. These moves will never change the amount of chips in the gadget, only the other moves will, so we will get a trivial periodic pattern of P-positions in the main two heaps when these moves are selected.

Following this process we can emulate any modular game with an invariant game in such a way that P-equivalence is conserved. This means that if we took P-equivalent modular games \mathcal{G} and \mathcal{H} and constructed the invariant games \mathcal{M} and \mathcal{M}' to emulate them, \mathcal{M} and \mathcal{M}' would also be P-equivalent games. This proves lemma 4 as we can construct \mathcal{M} and \mathcal{M}' to be P-equivalent if and only if the corresponding modular games \mathcal{G} and \mathcal{H} are P-equivalent.

3.1.5 Summary of the Proof of Theorem 3

With the three rather abstract looking lemmas proved we now explain how this proves the undecidability of P-equivalence in invariant games.

We have shown with lemma 1 that no algorithm exists that decides whether or not the string 101 occurs in the CA of an n -ary boolean function. Lemma 2 tells us that for any boolean function f we can construct two modular games \mathcal{G} and \mathcal{H} that are P-equivalent if and only if the word 101 never occurs in $CA(f)$. Combining these lemmas we know that there cannot exist an algorithm that decides whether or not two modular games \mathcal{G} and \mathcal{H} are P-equivalent as then we would have an algorithm telling us if 101 ever occurs in $CA(f)$, contradicting lemma 1.

By the same argument, because lemma 3 tells us that we can construct invariant games \mathcal{M} and \mathcal{M}_0 that are P-equivalent if and only if corresponding modular games \mathcal{G} and \mathcal{H} are P-equivalent, we know that there cannot be an algorithm to decide whether \mathcal{M} and \mathcal{M}_0 are P-equivalent. This is because if there was, we would be able to construct an algorithm using lemma 3 to decide if modular games \mathcal{G} and \mathcal{H} are P-equivalent, something we just said was impossible.

If it is algorithmically undecidable for given invariant games \mathcal{M} and \mathcal{M}_0 whether $P(\mathcal{M}) = P(\mathcal{M}_0)$ then it is also algorithmically undecidable whether a given finite pattern occurs in the set of P-positions of an invariant game.

3.1.6 Further Questions

After such a difficult and involved proof it may be worth taking a step back at this point and thinking about what Theorem 3 tells us at a higher level. The theorem tells us that we will never perfectly understand subtraction games that have a finite number of moves. Immediately this poses questions about where exactly our understanding of these games stops. At what dimension and for what number of moves will these games become undecidable? We will finish this section by taking some time to reflect on these questions before diving deeper into two and three move games in the search for the limit of computability.

How Many Heaps are Required for Undecidability?

We have proved that there is no algorithm that can take in a general invariant game and decide whether a given finite pattern occurs in the set of the games P-positions. This does not mean that an algorithm doesn't exist for invariant games of only d heaps for some finite integer d . It seems natural to suspect that there must be some $\delta \in \mathbb{N}$ such that for $d > \delta$ the P-positions of invariant games with this many heaps becomes algorithmically undecidable. We will see in the next section that this is not the case. The proof of Theorem 3 we have used offers little help in verifying or disproving this suspicion as it relies on our ability to construct an arbitrarily large gadget to emulate modular games. The authors of [9] suspect that three heaps will suffice for undecidability and suggest that even 2 heaps might suffice given the unpredictable nature of games such as the one displayed in Figure 16. Intuitively, this claim is hard to believe as the rules of the games are so simple.

Even for 1-heap games there has been difficulty understanding the perfect play outcomes for certain games with infinite moves. In 1966, Golomb [12] investigated the infinite one-dimensional ruleset $\mathcal{M} = \{1, 4, 9, 16, \dots\}$ where players can remove any square number from a single pile. It was claimed in the paper that studying the distribution of P-positions for this set may be as difficult as studying the distribution of prime numbers. However, with finite moves the outcome space is simple enough for us calculate recursively until we find a period that describes the entire outcome picture. For 2-heap games with only two legal moves, periodic patterns for different regions can be found and proved quite easily as we will soon see in more detail. For 2-heap games with eight or more moves, such as that in Figure 16, our intuition tells us that periodic patterns will exist for the countably many different regions on the grid. These periodic patterns that describe the placement of regions and the behaviour inside of regions will likely have periods that are undeterminable from the starting ruleset. If we know that finding a periodic pattern in 1-heap Nim is always possible and we are arguing that periodic patterns will always exist in 2-heap Nim then we can make an inductive argument that periodic patterns will exist for games with 3-heaps and more. Obviously as the dimension d increases, the task of finding periodic patterns by means of computation becomes even harder and the ability to visualise the space of P-positions becomes impossible. This does not mean that periodic patterns do not exist in these spaces and it could very well be true that periodic patterns may exist for any finite number of heaps d , however massive their periods may be.

Do we Need to be Able to add Matches to Heaps in Order to Achieve Undecidability?

This question is also unanswered by our proof of Theorem 3 as we need to be able to add matches to heaps in an invariant game in order to construct the gadget to simulate moving between move sets of the emulated modular game. The source paper guesses that 2-heaps with the addition of matches allowed suffices for undecidability as "it is easy to generate complicated patterns of P-positions with small finite sets of moves". This argument holds equally well for invariant games where we cannot add matches because, as demonstrated in Figure 22, complicated patterns can still emerge.

First lets consider the top left impartial game \mathcal{M}_1 . At first glance it looks like the

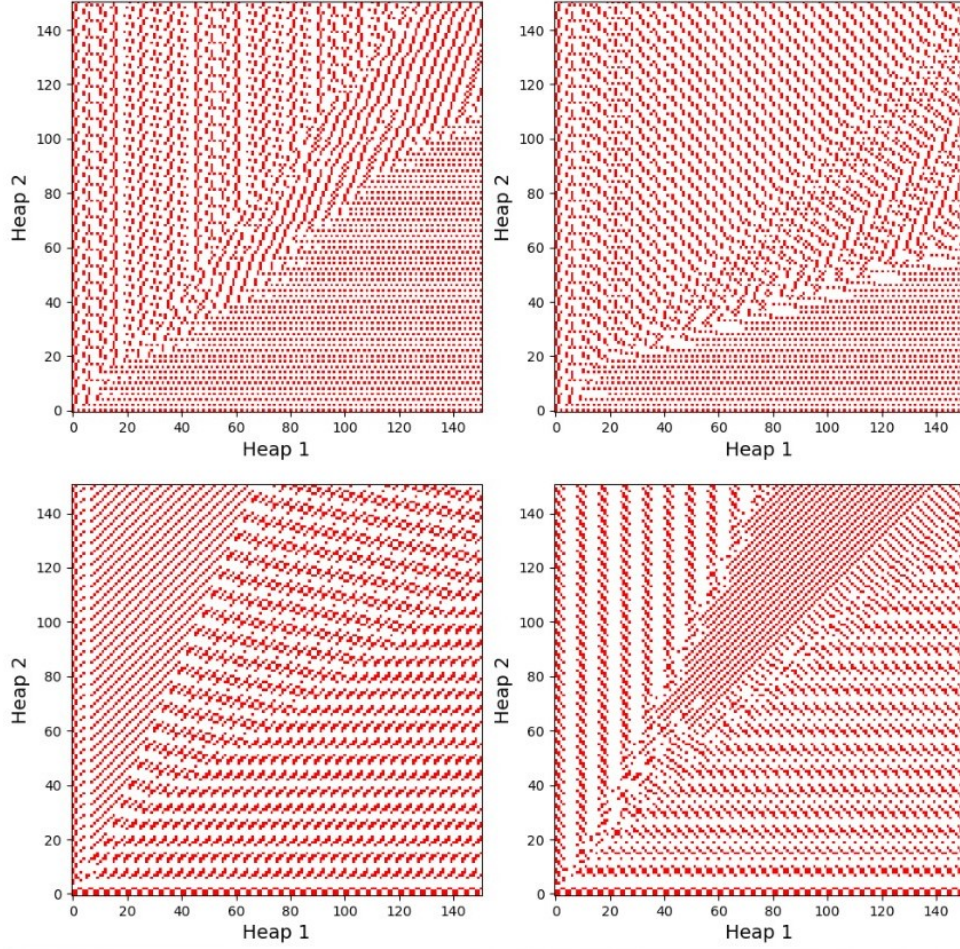


Figure 22: The P-positions of four invariant games with the addition of matches to a heap not allowed:

$\mathcal{M}_1 = \{(3, 1), (1, 0), (2, 1), (4, 5), (0, 3), (7, 1), (5, 6)\}$ (Top Left),

$\mathcal{M}_2 = \{(3, 1), (1, 0), (2, 1), (4, 5), (0, 3), (7, 1), (5, 6), (9, 1)\}$ (Top Right),

$\mathcal{M}_3 = \{(0, 2), (2, 0), (2, 3), (4, 2), (5, 4), (6, 5), (4, 3), (1, 4)\}$ (Bottom Left),

$\mathcal{M}_4 = \{(0, 2), (2, 0), (2, 3), (4, 2), (5, 4), (6, 5), (4, 3), (1, 5)\}$ (Bottom Right).

patterns have separated into three distinct regions. There is some predictability to the intricate patterns within these regions but it would be naive to assume these patterns will continue forever. The birth of the center region seems to occur at about (20,20) when the top left region and the bottom right region line up in such a way that a new region is created. this new region seems highly chaotic with the appearance of random "line of circles" seen at (95,80), (115,100) and (145,120). Its very possible that if a part of this chaotic center region lined up in a certain way with another region then a fourth region could be created for positions with much larger heaps. It is this creation of regions that give the locations of P-positions their unpredictability and therefore their undecidability from the starting position. If new regions can be continuously created then its very possible that the full picture of P-positions never settles down into a periodic pattern.

If we add the vector (9,1) to \mathcal{M}_1 we get \mathcal{M}_2 and we can see from the top right picture that the regions are now quite difficult to distinguish. Clearly we do not need to be able to add matches to heaps in order to get complicated patterns of P-positions with small finite sets of moves. Granted, when adding matches is allowed we often get more regions occurring randomly and more chaotic patterns, but the fact that even when adding matches isn't allowed we still get new regions occurring is good evidence for the claim that we don't need to be able to add matches in order to achieve undecidability.

A final comment its worth mentioning about Figure 22 is how much slight changes in an Invariant game affect the location of P-positions. The only difference between the top left game and the top right game is the addition of the move (9,1) and the only change between the bottom two pictures is is the move (1,5) is allowed in the bottom right picture instead of (1,4) in the bottom left. The dramatic changes in regions and region placement for small variations in the games implies that its difficult and maybe even impossible to construct theory for analysing behaviour of neighbouring games with this many move options. It is also unlikely that we will be able to decide if a given games P-positions are eventually periodic using neighbouring games that we know to be periodic as the behaviour of neighbouring games is too unpredictable and chaotic.

How Many Move Options are Required for Undecidability?

This question was surprisingly not asked in the source paper but it is nevertheless quite interesting to think about. If we cant even find periodic patterns for games such as that in Figure 16 with only 8 moves then maybe there exists some finite μ such that for any game with $n < \mu$ moves we can perfectly understand the placement of the games P-positions. This μ may depend on the number of heaps in the vector. Again, our proof of Theorem 3 offers little help in finding this μ as we need to be able to add arbitrarily many moves to an invariant game to emulate the modular game and complete the proof in this way. It has been proved in [10] that we can perfectly understand two-move games with any amount of heaps however problems seem to arise when going on to understanding 3-move games. We will go over this proof and then attempt to find a value for μ in the next section.

3.2 Two-move Vector Subtraction Games are Decidable

3.2.1 Linear Algorithm for two-move Vector Subtraction

In this section we will devise a linear time algorithm to solve two move vector subtraction in any dimension using results from Section 6 of [10]. The first lemma we need that is fundamental to this section is the two-move \mathcal{P} - \mathcal{P} rule for vector subtraction.

Lemma 4. *Consider the two move vector subtraction game $\mathcal{M} = \{\mathbf{m}_1, \mathbf{m}_2\}$. We have that $\mathbf{x} \in \mathcal{P}$ if and only if $\mathbf{x} + \mathbf{m}_1 + \mathbf{m}_2 \in \mathcal{P}$.*

Proof. (\implies) Suppose that $\mathbf{x} \in \mathcal{P}$. Then $\{\mathbf{x} + \mathbf{m}_1, \mathbf{x} + \mathbf{m}_2\} \subset \mathcal{N}$ as we must have come from an N-position to now be in a P-position. This is the same set as the set of options from $\mathbf{x} + \mathbf{m}_1 + \mathbf{m}_2$ so we have that $\mathbf{x} + \mathbf{m}_1 + \mathbf{m}_2 \in \mathcal{P}$, as a position is a P-position if all of its moves take it to an N-position.

(\impliedby) Suppose $\mathbf{x} + \mathbf{m}_1 + \mathbf{m}_2 \in \mathcal{P}$. All moves from a P-position are to N-positions, so the set of move options is $\{\mathbf{x} + \mathbf{m}_1, \mathbf{x} + \mathbf{m}_2\} \subset \mathcal{N}$. Both of these positions have at least one move to a P-position. So if $(\mathbf{x} + \mathbf{m}_1) - \mathbf{m}_2 \in \mathcal{N}$ or $\mathbf{x} + \mathbf{m}_2 - \mathbf{m}_2 \in \mathcal{N}$ then we must have $\mathbf{x} \in \mathcal{P}$. Otherwise we have $\{\mathbf{x} + \mathbf{m}_1 - \mathbf{m}_2, \mathbf{x} + \mathbf{m}_2 - \mathbf{m}_1\} \subset \mathcal{P}$. This implies that $\{\mathbf{x} - \mathbf{m}_2, \mathbf{x} - \mathbf{m}_1\} \subset \mathcal{N}$. Hence, $\mathbf{x} \in \mathcal{P}$ as applying either move takes it to an N-position. \square

This lemma also implies that $\mathbf{x} \in \mathcal{N}$ if and only if $\mathbf{x} + \mathbf{m}_1 + \mathbf{m}_2 \in \mathcal{N}$.

We now define the translation function, an important tool used in the algorithm.

Definition 5. *Consider the two move vector subtraction game $\mathcal{M} = \{\mathbf{m}_1, \mathbf{m}_2\}$. We define the **Translation Function** as $t(\mathbf{x}) = t_{\mathcal{M}}(\mathbf{x}) = \mathbf{x} + \mathbf{m}_1 + \mathbf{m}_2$.*

We can use this function iteratively, for example $t^3(\mathbf{x}) = t(t(t(\mathbf{x})))$. We have for all k that $t^k(\mathbf{x}) = \mathbf{x} + k(\mathbf{m}_1 + \mathbf{m}_2)$.

We know directly from lemma 4 that $\mathbf{x} \in \mathcal{P}$ if and only if $\mathbf{x} - t^k(\mathbf{x}) \in \mathcal{P}$ where $k \in \mathbb{N}_0$ such that $\exists i \in (1, \dots, d)$ with $(\mathbf{x} - t^{k+1}(\mathbf{x}))_i < 0$. This means that $\mathbf{x} - t^k(\mathbf{x})$ is one iteration of the translation function away from being an illegal position with a negative heap. We will refer to this as the **General Two-Move Rule**.

From here it is remarkably simple to construct an algorithm that can determine the perfect play outcome of any position in the two dimensional game space \mathcal{B} . We know that any position $\mathbf{x} \in \mathcal{B}$ has the same outcome as its smallest representative, say $\mathbf{x}' \in \mathcal{B}(\text{mod } \mathbf{m}_1 + \mathbf{m}_2)$, which can be easily computed. When $\mathbf{x}' = (x'_1, x'_2)$ is calculated we have two possible situations:

1. Both heaps of \mathbf{x}' are too small for either move to be played, so \mathbf{x}' is a terminal position.
2. There is a move \mathbf{m}_i such that $\mathbf{x} - \mathbf{m}_i$ is a terminal position.

In the first case, if \mathbf{x}' is a P-position then so is \mathbf{x} by the general two-move rule. In the second case there is a move to a terminal P-position from \mathbf{x}' so $\mathbf{x}' \in \mathcal{N}$ and again by the general two-move rule $\mathbf{x} \in \mathcal{N}$.

The core of this algorithm lies in finding the smallest representative \mathbf{x}' of \mathbf{x} modulo $(\mathbf{m}_1 + \mathbf{m}_2)$, which is essentially a remainder computation in d dimensions. This involves repeatedly applying the translation function $t^k(\mathbf{x})$, incrementing k until we have a heap i such that $x_i - t^{k+1}(x_i) < 0$, thereby ensuring \mathbf{x}' is the smallest representative. While the standard naive algorithm for remainder computation has exponential time complexity in the succinct input size, for the special case of two-move vector subtraction, this can be done in linear time. After finding \mathbf{x}' , the algorithm only needs to check the constant number of cases discussed above, which has constant time complexity. Therefore, the overall time complexity of the algorithm is linear in the succinct input size.

Example: To demonstrate this algorithm, consider the invariant game $\mathcal{M} = \{(5, 3), (1, 7)\}$ and suppose we want to find the perfect play outcome of the position $(50, 50)$. We have that $\mathbf{m}_1 + \mathbf{m}_2 = (6, 10)$ and it is clear to see that we can take five lots of this vector away to get the minimum representative $\mathbf{x}' = (20, 0)$. We cannot play either move in this position so \mathbf{x}' is a terminal P-position meaning that $(50, 50)$ is also a P-position. Now consider finding the perfect play outcome for the position $(50, 49)$. This time we can only take four lots of $(6, 10)$ from the starting position to get a minimum representative of $\mathbf{x}' = (26, 9)$. The two possible moves from this position are to $(21, 6)$ and $(25, 2)$. The latter is a terminal P-position meaning $(26, 9)$ must be an N-position. This means that $(50, 49)$ must be an N-position as we have demonstrated by the general two-move rule.

We can see with a glance at Figure 23 that these results, $(50, 50) \in \mathcal{P}$ and $(50, 49) \in \mathcal{N}$, match the results found when the outcomes are computed recursively.

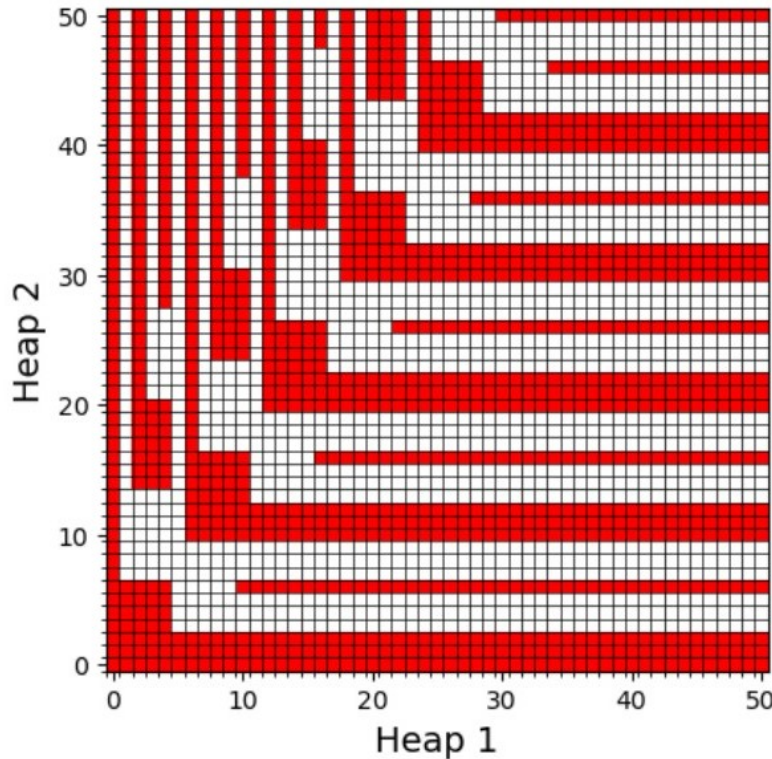


Figure 23: The P-positions of the invariant game $\mathcal{M} = \{(5, 3), (1, 7)\}$

3.2.2 Applying the Algorithm to deduce Decidability in two-move Vector Subtraction Games

As we have stated, if it is algorithmically undecidable for given invariant games \mathcal{M} and \mathcal{M}' whether $P(\mathcal{M}) = P(\mathcal{M}')$ then it is also algorithmically undecidable whether a given finite pattern occurs in the set of P-positions of an invariant game. We know that for general invariant games with arbitrary rulesets, deciding pattern occurrence in $P(\mathcal{M})$ is undecidable. However, we can use this statement to show that two-move invariant subtraction games are decidable as we can decide if a finite pattern occurs in $P(\mathcal{M})$ using our efficient algorithm.

As we increase k in our translation function we are effectively producing a finite pattern of P-positions and N-positions. This can be visualised on Figure 23 as we can see that if we started the image at position $k((5, 3) + (1, 7)) = (6k, 10k)$ for any $k \in \mathbb{N}_0$ and extended to show the next 50×50 positions we would get the very same image, a direct consequence of the $\mathcal{P} - \mathcal{P}$ rule. Therefore we can apply the linear time algorithm to compute a modular full-set of P-positions for any two-move ruleset. Doing this for arbitrary two-move games \mathcal{M} and \mathcal{M}' and then comparing the sets of P-positions is a viable algorithm for deciding whether $P(\mathcal{M}) = P(\mathcal{M}')$ so we have that two-move vector subtraction games are decidable.

3.3 Three Move Vector Subtraction Games in 2-Dimensions

3.3.1 Eventual Periodicity of 2-Dimensional Games

Before we start looking into three move games for decidability we will prove a more general result for all 2-dimensional games. We start this section by introducing some notation and definitions. Given a two dimensional vector subtraction game let $o(i, j)$ denote the perfect play outcome of position (i, j) , so $o(i, j) \in \{\mathcal{N}, \mathcal{P}\}$. We can use this notation to define Row Periodicity and Row Eventual Periodicity.

Definition 6. A two dimensional vector subtraction game is **Row Periodic** if $\forall j \geq 0$, $\exists T > 0$ such that $\forall i \geq 0$ we have $o(i, j) = o(i + T, j)$

In English this just means that every row of the P-position map is periodic. In practice row periodicity is very rare and non of our examples so far have had this property.

Definition 7. A two dimensional vector subtraction game is **Row Eventually Periodic** if $\forall j \geq 0$, \exists a column i' and a $T > 0$ such that $\forall i \geq i'$ we have $o(i, j) = o(i + T, j)$. We call this column i' the **preperiod length** of row j .

This means that every row has a point i' after which the row becomes periodic. The definitions for column periodic and column eventually periodic are analogous

Theorem 6. The perfect play outcome of any two-dimensional subtraction game with a finite number of moves is row (and column) eventually periodic.

Proof. We denote an arbitrary subtraction game with k moves as $\mathcal{M} = \{(a_i, b_i)\}_{1 \leq i \leq k}$ and let $A = \max_i \{a_i\}$ and $B = \max_i \{b_i\}$ be the maximum of the x- and y-coordinates of the subtraction set respectively.

Fix a row y and define the window $W(x, y) = (o(x, y), o(x + 1, y), \dots, o(x + A - 1, y))$ using the outcome function $o(i, j) \in \{\mathcal{N}, \mathcal{P}\}$. The binary nature of the outcome function means that there are 2^A possible outcome patterns within the range of the window. We will now prove the theorem by induction on the rows of the outcome map.

For the base case we prove that the outcomes of row 0 are eventually periodic. Since 2^A is finite, there exists a smallest x and an α such that $W(x + \alpha, 0) = W(x, 0)$, so the outcome patterns are repeated in the window. By computing the next outcome we find that $W(x + \alpha + i, 0) = W(x + i, 0)$ for all $i \geq 0$ due to the choice of window size. This means that the repetition occurs before $A2^A$, so the period length p_0 is at most $A2^A$.

Now we fix a row y' and assume that the outcomes of all rows $0 \leq y < y'$ are eventually periodic. Let the period lengths of the rows $0, 1, \dots, y' - 1$ be denoted as $p_0, p_1, \dots, p_{y'-1}$. We want to show that the y' -th row is eventually periodic to prove the theorem by induction.

To do this we define the super-period length

$$\pi_{y'} = \prod_{i=y'-B}^{y'-1} p_i.$$

For sufficiently large x , consider the windows $W(x + m\pi_{y'}, y')$ where $m \in \mathbb{N}_0$. As m increases there will at some point be repetition as the number of patterns the window

can be is finite, lets say that m' and m'' give the same window pattern. When this repetition occurs we know that all the lower rows have become periodic as we picked x to be sufficiently large. It follows that row y' is eventually periodic because $o(x + m'\pi_{y'} + 1, y') = o(x + m''\pi_{y'} + 1, y')$ and we can use the same argument as in the base case. \square

Does This mean 2-dimensional games are decidable?

The fact that the rows and columns of all 2-dimensional subtraction games become eventually periodic leads us to believe that we may be able to decide whether a given finite pattern occurs in the set of P-positions of these games, thus proving 2-dimensional games are decidable. Unfortunately, this is not the case. The preperiod lengths and periods can be extremely large and computing these is still an undecidable problem in general. Also this result offers no understanding of the infinitely large diagonal region found in the middle of all subtraction game pictures. Finite patterns may span across multiple rows and columns in this region with its occurrence depending on the specific alignment of the periods.

Even though we cannot use the eventual periodicity of 2-dimensional games to prove decidability directly, it is still a useful structural insight that we can use to make arguments about decidability in general. It is also useful to know that when we see periodic behaviour at the top left and bottom right of our game's outcome matrix we know that this behaviour continues and no nasty surprises await us.

3.3.2 Experimental Observations on Three-move Games in 2 Dimensions

After extensive experimentation we observe that three move subtraction games seem to fall into two categories. Either their diagonal regions remain a constant size and are periodic/eventually periodic, or their diagonal regions grow as the size of the two heaps grow, spawning a third and sometimes even a fourth segment in the middle. We know that the rows and columns become eventually periodic so the diagonal region we are referring to is the area of the outcome grid before this periodicity happens.

For rulesets that have diagonal regions with constant size we have two possibilities regarding the periodicity's. We introduce these possibilities with two original definitions

Definition 8. A ruleset $\mathcal{M} = \{m_1, m_2, m_3\}$ is **Perfectly Periodic** if there exists $p_1, p_2 \in \mathcal{M}$ such that for all $x \in \mathcal{B}$ we have $o(x) = o(x + p_1 + p_2)$. We will refer to these $p_1, p_2 \in \mathcal{M}$ as the **dominant** move options.

We have shown already that this definition applies to all two-move rulesets as a direct consequence of the $\mathcal{P} - \mathcal{P}$ rule. However, it does not apply for most three-move rulesets.

Definition 9. A ruleset $\mathcal{M} = \{m_1, m_2, m_3\}$ is **Periodic** if there exists $p, q \in \mathcal{B}$ such that for all $x \in \mathcal{B}$ with $x_1 \geq q_1, x_2 \geq q_2$ we have $o(x) = o(x + p)$.

In this definition of periodic, the two dimensional period p does not have to depend on the ruleset \mathcal{M} in any way. We have that q acts as the preperiod so we get periodic behaviour for positions beyond q .

If a ruleset \mathcal{M} is not periodic, then the middle region between the eventually periodic rows and columns must be expanding as we increase the combined size of the two heaps. We see this behaviour later in this section in Figure 26 and analyse these cases in a different way. For now let us consider periodic rulesets.

3.3.3 Solvability of Perfectly Periodic Rulesets

If we know that a ruleset is perfectly periodic then we are free to use the translation function as we did in the last section to construct a linear time algorithm that will determine the outcome of any $\mathbf{x} \in \mathcal{B}$.

Consider a perfectly periodic ruleset $\mathcal{M} = \{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3\}$ with period vectors $\mathbf{p}_1, \mathbf{p}_2 \in \mathcal{M}$. Define the translation function $t(\mathbf{x}) = \mathbf{x} + \mathbf{p}_1 + \mathbf{p}_2$, and as before, let $t^k(\mathbf{x})$ denote the k -fold application of t to \mathbf{x} . By the definition of perfect periodicity, we have: $o(\mathbf{x}) = o(t^k(\mathbf{x}))$ for all $k \in \mathbb{N}_0$ and $\mathbf{x} \in \mathcal{B}$.

Now, consider an arbitrary game position $\mathbf{x} \in \mathcal{B}$. To determine its outcome, we can do the following:

1. Compute the smallest non-negative integer k such that $\mathbf{x} - t^{k+1}(\mathbf{x}) \not\geq \mathbf{0}$. This means that $\mathbf{x} - t^k(\mathbf{x})$ is the smallest representative of \mathbf{x} in $\mathcal{B} \pmod{\mathbf{p}_1 + \mathbf{p}_2}$.
2. Let $\mathbf{x}' = \mathbf{x} - t^k(\mathbf{x})$. By perfect periodicity, $o(\mathbf{x}) = o(\mathbf{x}')$.
3. Determine the outcome of \mathbf{x}' by checking the following:
 If $\mathbf{x}' \not\geq \mathbf{m}_i$ for all $i \in \{1, 2, 3\}$, then \mathbf{x}' is a terminal position, and thus $o(\mathbf{x}') = P$.
 If there exists an $i \in \{1, 2, 3\}$ such that $\mathbf{x}' - \mathbf{m}_i$ is a terminal position, then $o(\mathbf{x}') = N$.

Steps 1 and 2 can be performed in linear time with respect to the input size, as they involve simple vector addition and subtraction. Step 3 only requires a constant number of comparisons, so it also runs in constant time. Therefore, this algorithm determines the outcome of any position \mathbf{x} in a perfectly periodic ruleset \mathcal{M} in linear time.

We can now repeat the argument in Section 3.2.2 and say that all perfectly periodic rulesets are decidable as we can compute a modular full-set of P-positions to determine if $P(\mathcal{M}) = P(\mathcal{M}')$ for any perfectly periodic rulesets $\mathcal{M}, \mathcal{M}'$.

3.3.4 Occurrence of Perfectly Periodic Rulesets

The perfect periodicity of a ruleset may seem like a rather strict condition to impose in our search for decidability. However we find through experimental observations that the property occurs in three move games perhaps more often than one may expect. It would be nice if we could formulate some rule that tells us if a ruleset is going to be perfectly periodic without us having to plot and inspect the outcome matrix. It would also be nice if there was a rule that told us which of the moves in the ruleset are the dominant rule options that form the period of a perfectly periodic ruleset. Below we give two tables of observations on outcomes for when a third move is added to the same two-move game in an attempt to uncover some patterns that could be clues as to what these rules are or if they are even likely to exist.

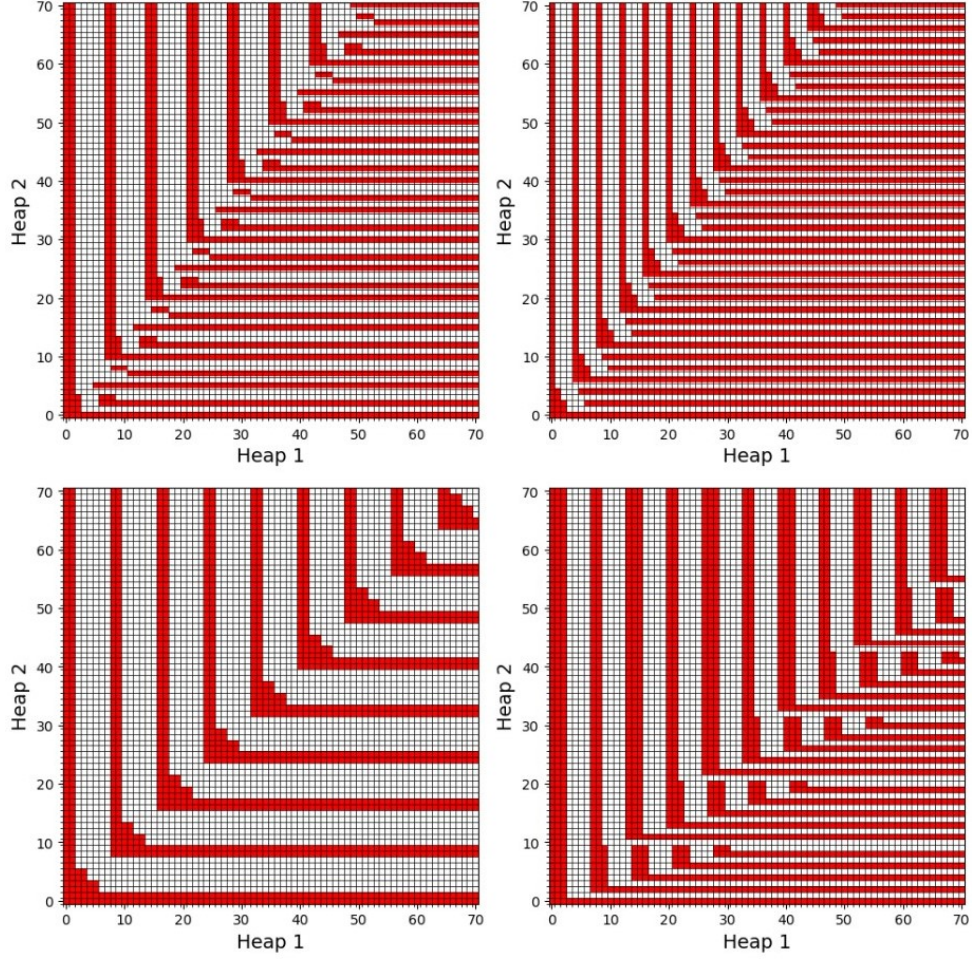


Figure 24: The P-positions of four three-move games that are perfectly periodic. The sum of the first two moves in the ruleset is the period:

- $\mathcal{M}_1 = \{(2, 4), (5, 6), (3, 1)\}$ Period: (7, 10) (Top Left),
- $\mathcal{M}_2 = \{(1, 5), (3, 1), (2, 3)\}$ Period: (4, 6) (Top Right),
- $\mathcal{M}_3 = \{(6, 2), (2, 6), (4, 4)\}$ Period: (8, 8) (Bottom Left),
- $\mathcal{M}_4 = \{(9, 10), (4, 1), (3, 1)\}$ Period: (13, 11) (Bottom Right).

In the tables below we consider the games $\mathcal{M}_1 = \{(2, 4), (5, 6), (x, y)\}$ (Table 2) and $\mathcal{M}_2 = \{(1, 3), (2, 4), (x, y)\}$ (Table 3) for all possible third moves up to $(12, 12)$. If the outcome matrix is perfectly periodic, the period is shown. Game outcomes that are just periodic are represented with a "P" and outcomes that have three or more segments are represented with an "X".

$y \backslash x$	0	1	2	3	4	5	6	7	8	9	10	11	12
0		P	(7,10)	(7,10)	(7,10)	X	X	X	X	X	X	X	X
1	(7,10)	P	(7,10)	(7,10)	(7,10)	X	X	X	P	X	X	P	X
2	X	X	(7,10)	X	X	X	P	X	X	X	X	X	X
3	X	X	(7,9)	X	X	X	P	X	X	X	X	X	X
4	(7,10)	X	(7,10)	(7,10)	(7,10)	X	P	X	X	X	X	X	X
5	(7,10)	X	(7,10)	(7,10)	(7,10)	X	X	X	X	X	X	X	X
6	(7,10)	X	(7,10)	(7,10)	(7,10)	(7,10)	P	X	(10,10)	X	X	P	X
7	X	X	(7,11)	(7,11)	(7,11)	X	X	X	(10,11)	X	X	P	X
8	X	X	(7,12)	(7,12)	(7,12)	X	P	X	(10,12)	X	X	P	X
9	X	X	X	X	X	X	X	X	X	X	X	P	X
10	X	X	X	X	X	X	X	X	X	X	X	P	X
11	X	X	X	X	X	X	X	X	P	P	P	P	X
12	X	X	X	X	X	X	P	X	X	(14,18)	(15,18)	X	X

Table 2: A visualisation of the type of games we get when we add the move (x, y) to the ruleset of $\mathcal{M} = \{(2, 4), (5, 6)\}$ to create a three-move game.

As we can see the placement of perfectly periodic and periodic rulesets is very random in this space and doesn't seem to bear any noticeable relation to the ever-present moves $(2, 4)$ and $(5, 6)$. It would be very hard, and perhaps impossible, to predict the location and periods of three-move games of this kind if we added a move larger than $(12, 12)$. Unsurprisingly, the most common period is $(7, 10)$ which is the sum of our constant moves. What is surprising though is when this period changes. When we add the move $(2, 3)$ this move takes dominance from $(2, 4)$ to give the period of $(7, 9)$, very strange behaviour as normally we expect the larger vectors to be dominant. For the games with periods $(10, 10)$, $(10, 11)$ and $(10, 12)$ the new move is dominant with $(2, 4)$. However in the games with periods $(14, 18)$ and $(15, 18)$ the new move is dominant with $(5, 6)$.

The random behaviour in terms of dominant move options continues with our second example in Table 3. However, the placements of the perfectly periodic games seem to have slightly more structure, with "clusters" of perfectly periodic games forming around specific values of x and y . These clusters suggest that certain move combinations are more likely to yield perfectly periodic behavior, hinting at a potential underlying pattern. Nevertheless, even with these glimmers of structure, the overall distribution of perfectly periodic games remains largely unpredictable. Ultimately, these tables demonstrate the randomness of perfectly periodic rulesets and dent our hopes of finding comprehensive rules governing their occurrence. Any rule that appears logical always seems to fall victim to a counter-example at some point.

Even though there are no steadfast rules that tell us whether a ruleset is perfectly periodic or the period of such a ruleset, we can construct algorithms that take in a rule-

$y \backslash x$	0	1	2	3	4	5	6	7	8	9	10	11	12
0		(3,7)	P	X	P	P	X	P	P	X	P	P	X
1	(3,7)	X	X	X	X	X	X	X	X	X	X	X	X
2	(2,6)	(3,6)	X	X	X	X	X	X	X	X	X	X	X
3	(3,7)	(3,7)	(3,7)	(4,7)	P	P	X	P	P	X	P	P	X
4	(3,7)	(3,7)	(3,7)	(4,7)	X	X	X	X	X	X	X	X	X
5	P	(3,8)	(3,8)	(4,8)	X	X	X	X	X	X	X	X	X
6	P	(3,9)	(3,9)	(4,9)	X	X	X	X	X	X	X	X	X
7	P	(3,10)	(3,10)	(4,10)	X	X	X	X	X	X	X	X	X
8	X	P	X	X	(6,12)	X	X	X	X	X	X	X	X
9	X	P	X	X	(3,7)	X	X	X	X	X	X	X	X
10	X	X	X	X	(3,7)	(3,7)	(3,7)	P	P	X	P	P	X
11	X	X	X	X	(3,7)	(3,7)	(3,7)	X	X	X	X	X	X
12	X	X	P	P	(6,15)	(6,15)	(7,15)	X	X	X	X	X	X

Table 3: A visualisation of the type of games we get when we add the move (x, y) to the ruleset of $\mathcal{M} = \{(1, 3), (2, 4)\}$ to create a three-move game.

set and derive the existence and value of a period by considering the outcome matrix. We will now give a high level example of such an algorithm.

Consider a three move game $\mathcal{M} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$ and assume we can calculate the outcome matrix using a standard recursive function up to $(2\max\{x_1, x_2, x_3\}, 2\max\{y_1, y_2, y_3\})$. We could check for perfect periodicity as follows:

1. Iterate over all possible pairs of dominant moves. With only three moves in the ruleset this loop contributes only a constant factor of 3 to the time complexity.
2. If we are considering $\mathbf{p}_1, \mathbf{p}_2 \in \mathcal{M}$ as the dominant moves then consider the game in $(\text{mod } \mathbf{p}_1 + \mathbf{p}_2)$ and check $o(\mathbf{x}) = o(\mathbf{x}(\text{mod } \mathbf{p}_1 + \mathbf{p}_2))$ for all \mathbf{x} in the outcome matrix.
3. If the equality does hold for all \mathbf{x} in the outcome matrix then the game is perfectly periodic with dominant moves \mathbf{p}_1 and \mathbf{p}_2 . If it does not hold for all \mathbf{x} then we try the next potential pair of dominant moves.
4. If none of the pairs of moves pass the criteria in step 2 the the game is not perfectly periodic.

The time complexity of this algorithm is $O(\text{rows} * \text{cols})$, which is linear in the size of the outcome matrix. This is because the algorithm iterates over each position in the matrix once for each pair of potential dominant moves.

It is necessary to check the outcome matrix all the way up to $(2\max\{x_1, x_2, x_3\}, 2\max\{y_1, y_2, y_3\})$ rather than just up to double the size of the period. This is because if the third move is much bigger than the other two we might see two-move perfectly periodic behaviour up until the third move begins to be available changing the behaviour of the outcome matrix and often spouting a third segment. As we can see from the tables, perfect periodicity in games with a large move are rare and when they do occur the large move is often one of the dominant moves.

With a linear time algorithm such as this we have shown that given a three-move ruleset \mathcal{M} we can algorithmically determine if the ruleset is perfectly periodic. If it is, we can then determine its period and understand the outcome map completely using our process in Section 8.2. We now turn our attention to rulesets that are only periodic and discuss if these are perhaps the limit of computability.

3.3.5 Are Periodic Three-Move Games Decidable?

When making the jump from two-move games to three-move games we get the first occurrence of games that are not perfectly periodic, so its very possible that this is our limit for computability and the first instance of games we cannot fully understand. For the vast majority of positions labeled "P" in our tables, the periods and pre-periods relate to the move options in some intimate way. Often the period is some linear combination of the move options, as is the case with the four periodic games shown in Figure 25. Again, it would be nice to find some governing rules here for what the linear combinations are and discover if one always exists for periodic games. Other games, such as that shown in Figure 26, are not so easy to dissect. The pre-period is likely impossible to understand without directly computing it and its hard to see visually where the period starts and how long it is, as there are many "false-periods" that don't quite work. We tentatively label this game as "P" in Table 2, even though we cant be completely sure periodicity has started. The size of the center region remains the same up to 500x500 but without proving its periodicity there is always a chance it could sprout a third region at some point.

In an ideal world, we would also want rules that determine questions such as: When is a pre-period involved? How big is the pre-period? Can a pre-period be periodic in its own way? What are the periods of the individual segments? When is a three-move game periodic?

Unfortunately, the questions we pose may not have simple, universal answers. The decidability of periodic three-move games would require the existence of an algorithm that can determine, for any given ruleset, whether the game is periodic or not. If the periods and pre-periods are arbitrarily large then we may not be able to bail ourselves out with a brute force recursive computation of the outcome matrix like we did when deciding perfectly periodic games. Fortunately, There are techniques we can use such as outcome segmentation, which we will explore in the next section, that could contribute to algorithmically deciding if a three-move game is periodic.

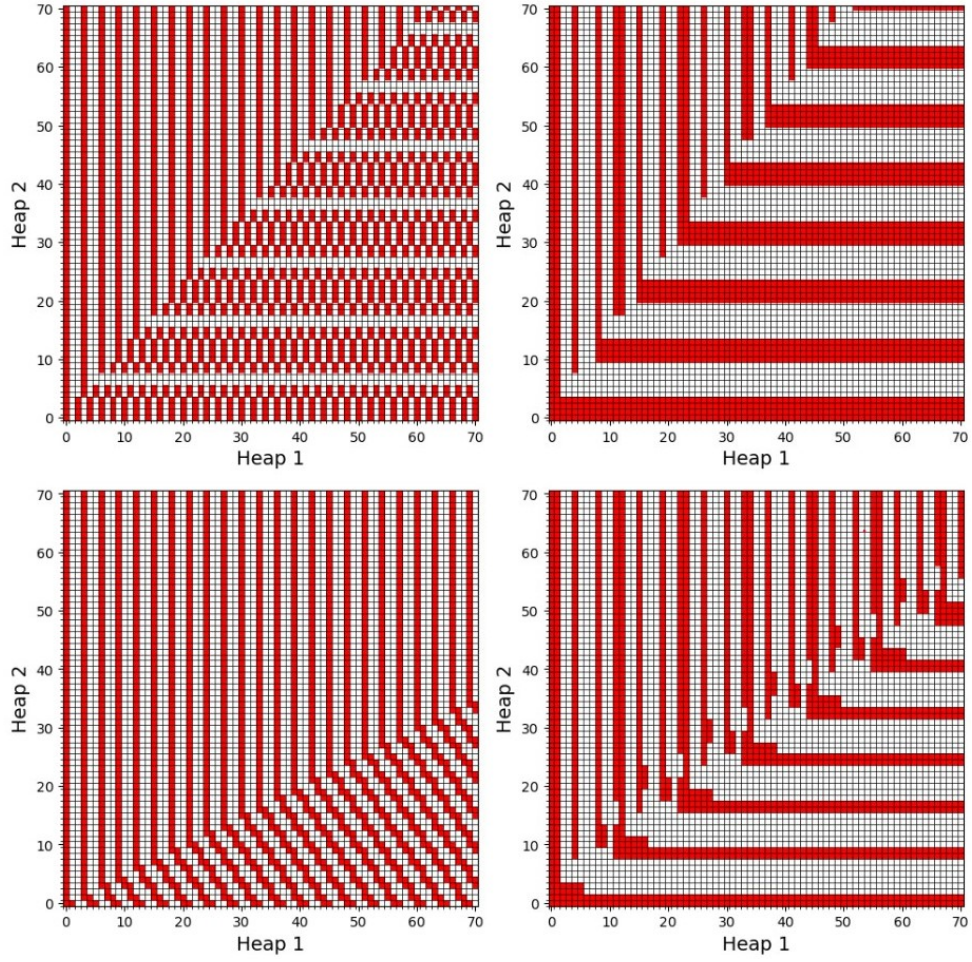


Figure 25: The P-positions of four three-move games that are periodic but not perfectly periodic

$\mathcal{M}_1 = \{(1, 0), (2, 4), (5, 6)\}$ Period: $(9, 10)$ (Top Left),

$\mathcal{M}_2 = \{(6, 6), (2, 4), (5, 6)\}$ Period: $(22, 30)$ (Top Right),

$\mathcal{M}_3 = \{(1, 1), (1, 2), (2, 0)\}$ Period: $(6, 3)$ (Bottom Left),

$\mathcal{M}_4 = \{(6, 2), (2, 4), (5, 6)\}$ Period: $(11, 8)$ With Preperiod: $(11, 8)$ (Bottom Right).

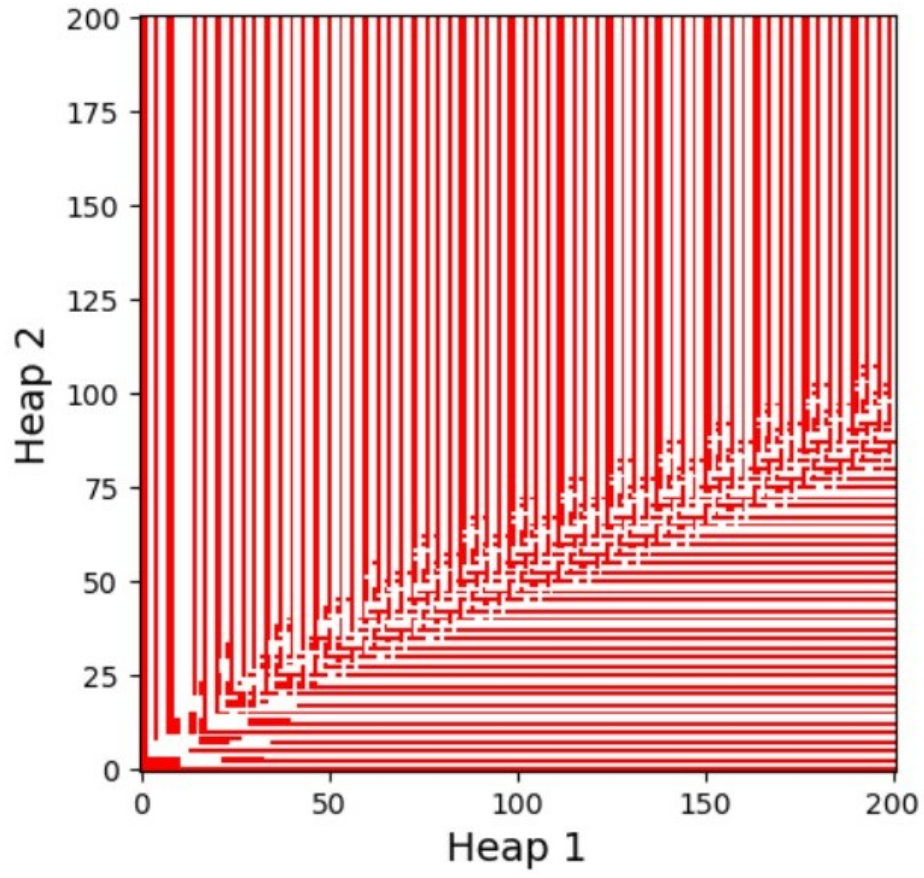


Figure 26: The P-positions of the periodic game $\mathcal{M} = \{(2, 4), (5, 6), (11, 1)\}$

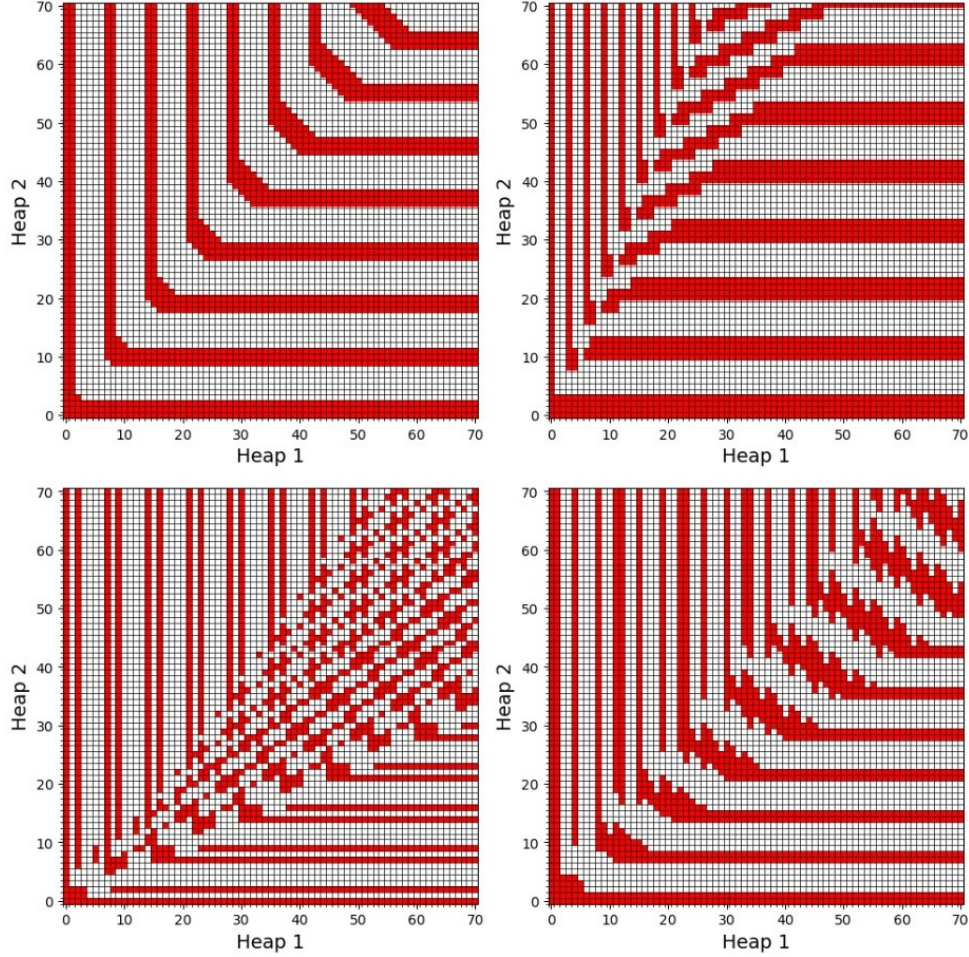


Figure 27: The P-positions of four three-move games that are not periodic and have expanding center segments. Notice the bottom left appears to have a fourth segment appearing. The games are as follows:

- $\mathcal{M}_1 = \{(3, 3), (2, 4), (5, 6)\}$ (Top Left),
- $\mathcal{M}_2 = \{(1, 4), (2, 4), (5, 6)\}$ (Top Right),
- $\mathcal{M}_3 = \{(1, 3), (3, 4), (4, 1)\}$ (Bottom Left),
- $\mathcal{M}_4 = \{(6, 2), (2, 5), (5, 4)\}$ (Bottom Right).

Notice the bottom left appears to have a fourth segment appearing.

3.3.6 Using Outcome Segmentation to Understand Three-Region Three-move games

This section is motivated by the Outcome Segmentation techniques introduced in [10]

Definition 10. A subset Γ of the first quadrant may permit a **Colouring Scheme** as follows. An update rule $u = (u_1, u_2)$ takes as input a coloured position (x, y, c) , where $c \in C_u \subset C$ (the set of colours) and $(x, y) \in \mathcal{B}$ (the set of positions). The output is a coloured position $(x + u_1, y + u_2, c') \in \Gamma \times C$.

In this scheme, multiple update rules can correspond to a single colour. A legal colouring scheme (X_0, U, C) assigns at most one colour to each position in Γ . Here, X_0 is a subset of Γ representing the initially coloured positions, and U is a finite set of update rules.

A colouring scheme set up in this way can be regarded as a simple cellular automaton, where the colour of each cell depends on only one younger cell. This makes it simpler than known Turing complete cellular automata that depend on several younger cells. Colouring schemes allow cells to be updated multiple times if they retain the same colour.

Definition 11. Consider a subset of the first quadrant $\Gamma = \{(x, y) | \alpha x + k < y < \beta x + m\} \subset \mathcal{B}$, for some fixed rationals α, β, k, m and a finite set of update rules U . Γ is a U -segment if it permits a colouring scheme via the rules in U . Suppose that a U -segment Γ partitions the \mathcal{P} - and \mathcal{N} -positions of a given ruleset \mathcal{M} such that the coloured positions correspond to the \mathcal{P} -positions. Then Γ is an **Outcome Segment**.

Definition 12. Consider a ruleset \mathcal{M} . A finite union of outcome segments $\cup_i \Gamma_i$ is an **Outcome Segmentation** of \mathcal{M} if for all i, j we have $\Gamma_i \cap \Gamma_j = \emptyset$ and for sufficiently large games of n positions they contain at least $n - o(n)$ positions. A segmentation is considered **perfect** if it contains all positions. A k -segmentation has k outcome segments.

Extended Example:

We will now put these tricky definitions into context by presenting an outcome segmentation and proving that it corresponds exactly to the \mathcal{P} -positions of the ruleset

$$\mathcal{M} = \{(3, 3), (2, 4), (5, 6)\}.$$

The outcomes of this ruleset are shown in grey in Figure 28. There are three segments. Positions in the upper segment can be treated as perfectly periodic with period $(7, 10)$ and positions in the lower segment can be treated as perfectly periodic with period $(8, 9)$. The middle segment is described by a blue/red colouring scheme mapping the \mathcal{P} -positions. This colouring scheme is described as follows:

Initialise by setting **red** colours at $\{(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (2, 3)\}$ and then iterate over the following:

- for $(x, y) \in \mathbf{red}$ turn $(x + 7, y + 10)$ **red**
- for $(x, y) \in \mathbf{red} \cup \mathbf{blue}$ turn $(x + 9, y + 8)$ **blue**

Up until this point our notion of regions/segments has been based entirely on observations. To formalise the outcome segmentation we need hard borders on these segments so we can show the outcome segmentation corresponds to the \mathcal{P} -positions of the subtraction game. For our example we define the segment borders by the lines

$$f(x) = \frac{10x}{7} + \frac{1}{7} \text{ and } g(x) = \frac{9x}{8} - \frac{19}{8}$$

so the segments can be defined as follows. A position (x, y) is in

- the **upper** segment if $y \geq \lceil f(x) \rceil$
- the **middle** segment if $\lceil g(x) \rceil < y < \lceil f(x) \rceil$
- the **lower** segment if $y \leq \lceil g(x) \rceil$

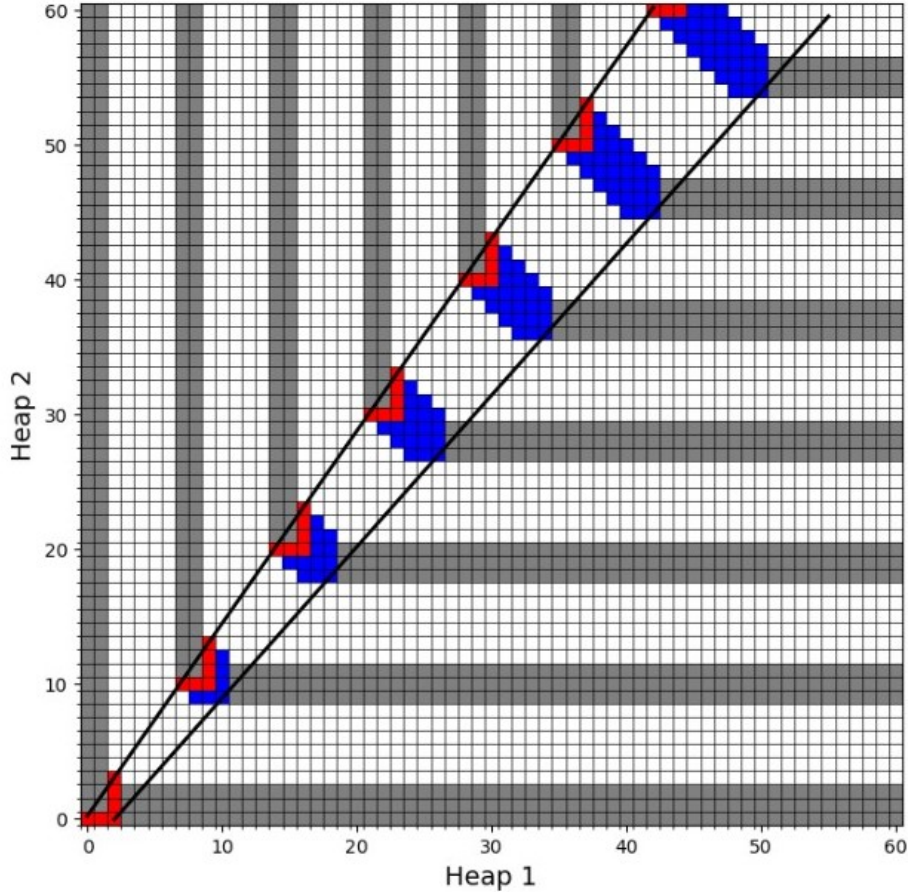


Figure 28: Visualisation of how the colouring scheme matches perfectly to the expanding center segment of the game $\mathcal{M} = \{(3, 3), (2, 4), (5, 6)\}$.

This segmentation and colouring scheme is shown in Figure 28. We will now prove that the game $\mathcal{M} = \{(3, 3), (2, 4), (5, 6)\}$ has a perfect outcome segmentation given by the colouring scheme described

Proof. First, consider the lower segment defined by $y \leq \lceil g(x) \rceil$ where $g(x) = \frac{9x}{8} - \frac{19}{8}$. The update rule for this segment is $(8, 9)$, which is a \mathcal{P} -to- \mathcal{P} combination of the moves $(3, 3)$ and $(5, 6)$. Starting from the initial \mathcal{P} -positions $(x, 0) | x \geq 8$, the update rule generates all \mathcal{P} -positions in this segment. Since no rule has y -coordinate greater than 6, no coloured position in this segment can reach another coloured position, satisfying the \mathcal{P} -to- \mathcal{N} property. The \mathcal{N} -to- \mathcal{P} property holds because from any non-coloured position, one of the moves $(2, 4)$, $(3, 3)$ or $(5, 6)$ leads to a coloured \mathcal{P} -position.

Next, consider the upper segment defined by $y \geq \lceil f(x) \rceil$ where $f(x) = \frac{10x}{7} + \frac{1}{7}$. Similar to the lower segment, the update rule $(7, 10)$ is a \mathcal{P} -to- \mathcal{P} combination of $(2, 4)$ and $(5, 6)$. Starting from initial \mathcal{P} -positions $(0, y) | y \geq 1$, the update rule generates all \mathcal{P} -positions satisfying the \mathcal{P} -to- \mathcal{N} property. The \mathcal{N} -to- \mathcal{P} property holds by one of the moves $(2, 4)$, $(3, 3)$ or $(5, 6)$ leading from any non-coloured position to a \mathcal{P} -position.

Finally, consider the middle segment. The initial red colourings correspond to terminal \mathcal{P} -positions. The red update rule $(7, 10) = (2, 4) + (5, 6)$ leads from red \mathcal{P} -positions to red \mathcal{P} -positions. The shared update rule $(8, 9) = (3, 3) + (5, 6)$ leads from both red and blue \mathcal{P} -positions to blue \mathcal{P} -positions. This satisfies the \mathcal{P} -to- \mathcal{P} property. For the \mathcal{P} -to- \mathcal{N} property, note that from any red position, $(3, 3)$ is not used in the update rules and leads to an \mathcal{N} -position. Similarly, from any blue position, $(2, 4)$ leads to an \mathcal{N} -position. The \mathcal{N} -to- \mathcal{P} property can be verified by induction - at each level between updates, the non-coloured cells are reached by the moves $(2, 4)$, $(3, 3)$ or $(5, 6)$ from the coloured \mathcal{P} -positions of the previous level.

Since the segment boundaries are disjoint and contain all positions, this constitutes a perfect outcome segmentation of the game $\mathcal{M} = (3, 3), (2, 4), (5, 6)$, with the described colouring scheme capturing exactly the \mathcal{P} -positions. \square

If we can find a perfect outcome segmentation for a game and describe it using a colouring scheme, as demonstrated in the extended example, then we have a complete understanding of the game's outcomes. Given a perfect outcome segmentation, we can determine the outcome of any position $(x, y) \in \mathcal{B}$ by following these steps:

1. Identify the segment to which the position (x, y) belongs, based on the segment boundaries defined in the outcome segmentation.
2. If the position (x, y) is in a perfectly periodic segment (the upper or lower segment in the example), we can calculate its outcome using the periodic pattern. In the example, positions in the upper segment have period $(7, 10)$, and positions in the lower segment have period $(8, 9)$. We can use modular arithmetic to determine if the position corresponds to a \mathcal{P} -position or an \mathcal{N} -position based on the periodic pattern.
3. If the position (x, y) is in a segment described by a colouring scheme, we can use modular arithmetic to decide if the position is coloured according to the scheme. If it is, it is a \mathcal{P} -position; otherwise, it is an \mathcal{N} -position.

In examples like this, outcome segmentation serves us very well when attempting to understand three move games. Naturally our thoughts turn towards a generalisation of the approach. It is conjectured in Section 10 of the source paper that there is an outcome segmentation for all three-move games and that three move games have at most four segments. If this is true then it will be possible to develop a brute force

algorithm to determine where the middle segments start, and their colouring scheme. We can be encouraged by the fact that the colouring schemes tend to resemble the move options of the game in some way so perhaps there are rules we could develop that limit the number of colouring schemes we would need to try to find one that works. This would also solve our problem of determining periodicity, as if we had a procedure to generate the segment boundaries then we would know the outcome map is periodic if the boundaries had the same gradient.

3.3.7 Recent Theory on Three-Move Subtraction Games in 2-Dimensions

During our earlier discussion on perfectly periodic and periodic games we talked about how difficult it is to find underlying rules that govern the behaviour of three-move subtraction games. For special rulesets, the source paper finds the following three lemmas that serve as \mathcal{P} -to- \mathcal{P} update rules. Here we state the lemmas, omitting the proofs that can be found in Section 7 of [10], and discuss their potential usefulness in solving two dimensional three move games.

Definition 13. The ruleset $\mathcal{M} = \{(a, b), (c, d), (e, f)\}$ is **Additive** if $e = a + c$ and $f = b + d$

Definition 14. The ruleset $\mathcal{M} = \{(a, b), (c, d), (e, f)\}$ is **Asymmetric Additive** if $e = a + c$ and $b = d + f$

Lemma 5. Let $\mathcal{M} = \{(a, b), (c, d), (a + c, b + d)\}$ be an additive ruleset. If $(x + 2a + c, y + 2b + d) \in \mathcal{P}$ and $(x + 2c + a, y + 2d + b) \in \mathcal{P}$ then $(x, y) \in \mathcal{P}$.

In vector notation, $\mathcal{M} = \{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3\}$ with $\mathbf{m}_1 + \mathbf{m}_2 = \mathbf{m}_3$. If $\mathbf{x} + \mathbf{m}_1 + \mathbf{m}_3 \in \mathcal{P}$ and $\mathbf{x} + \mathbf{m}_2 + \mathbf{m}_3 \in \mathcal{P}$ then $\mathbf{x} \in \mathcal{P}$.

Lemma 6. Let $\mathcal{M} = \{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3\}$ with $\mathbf{m}_1 + \mathbf{m}_2 = \mathbf{m}_3$ be an additive ruleset and suppose $\mathbf{x} \in \mathcal{P}$.

- (i) If $\mathbf{x} + 2\mathbf{m}_3 \in \mathcal{N}$ then $\mathbf{x} + \mathbf{m}_1 + \mathbf{m}_3 \in \mathcal{P}$ or $\mathbf{x} + \mathbf{m}_2 + \mathbf{m}_3 \in \mathcal{P}$
- (ii) If $\mathbf{x} + \mathbf{m}_1 + \mathbf{m}_3 \in \mathcal{N}$ then $\mathbf{x} + 2\mathbf{m}_1 \in \mathcal{P}$
- (iii) If $\mathbf{x} + \mathbf{m}_2 + \mathbf{m}_3 \in \mathcal{N}$ then $\mathbf{x} + 2\mathbf{m}_2 \in \mathcal{P}$
- (iv) If $\mathbf{x} + \mathbf{m}_2 + \mathbf{m}_3 \in \mathcal{N}$ and $\mathbf{x} + \mathbf{m}_1 + \mathbf{m}_3 \in \mathcal{N}$ then $\mathbf{x} + 2\mathbf{m}_3 \in \mathcal{P}$

Lemma 7. Let $\mathcal{M} = \{(a, b), (c, d + b), (a + c, d)\}$ be an asymmetric additive ruleset and suppose $(x, y) \in \mathcal{P}$.

- (i) If $(x + 2a + 2c, y + 2d) \in \mathcal{N}$ then $(x + a + 2c, y + 2d - b) \in \mathcal{P}$ or $(x + 2a + c, y + d - b) \in \mathcal{P}$
- (ii) If $(x + a + c, y + 2b + d) \in \mathcal{N}$ then $(x, y + 2b) \in \mathcal{P}$
- (iii) If $(x + 2a + c, y + b + d) \in \mathcal{N}$ then $(x + 2a, y) \in \mathcal{P}$
- (iv) If $(x + a + 2c, y + 2d - b) \in \mathcal{N}$ and $(x + 2a + c, y + d - b) \in \mathcal{N}$ then $(x + 2a + 2c, y + 2d) \in \mathcal{P}$

The main limitation of these lemmas is that they hold only in one direction. They provide conditions under which certain positions can be identified as P-positions based on the knowledge of other P-positions. However, the converse of these lemmas is not necessarily true. This unidirectional nature of the lemmas means that we cannot directly apply them to form a complete P-to-P principle, as we did for two-move games. In the case of two-move games in any dimension, the simple P-to-P principle was sufficient to fully solve the game and determine the outcome of any position. In contrast,

the lemmas presented for three-move games only work for very specific classes of three-move games, and only in 2 dimensions, highlighting the significant step up in difficulty when moving from two-move games to three-move games.

The fact that these lemmas are barely scratching the surface in terms of understanding three-move games suggests that finding a universal solving mechanism, similar to the \mathcal{P} -to- \mathcal{P} principle, may be extremely challenging or even impossible. The authors of the paper predict that the lemmas "will be useful in future work". If this is to be the case, it is likely that they will need to be extended or combined with additional rules that work in both directions. The ultimate goal would be to develop a set of rules that can be applied iteratively, similar to how the \mathcal{P} -to- \mathcal{P} principle is used in two-move games. By establishing bidirectional rules, we could potentially propagate the information about P-positions and N-positions throughout the game state space, building up a complete solution.

3.3.8 Closing Discussion on Vector Subtraction Games

In our exploration of three-move subtraction games in two dimensions, we have seen that the outcome segmentation approach holds promise in solving these games. By dividing the game state space into distinct segments, each with its own periodic pattern or colouring scheme, we can potentially develop a comprehensive understanding of the game's outcomes. The lemmas introduced in the recent theory section, although limited in their scope, could be valuable in a comprehensive proof of outcome segmentation as a way of understanding three-move games.

Casting our minds back to the questions we posed in Section 3.1.6, we considered the crucial parameters δ and μ , which represent the number of heaps and the number of move options, respectively, required for undecidability. Our analysis has shown that the value of δ may not be as critical as initially thought. The fact that two-move games are decidable in all dimensions suggests that the number of heaps alone does not determine the boundary between decidability and undecidability. On the other hand, the value of μ appears to be a key factor. The emergence of periodic games in the three-move case hints at the possibility that $\mu = 3$ could be the threshold for undecidability, as the periods threaten to get arbitrarily large. Obviously three move games are solvable in one dimension so perhaps if we can construct a proof of decidability in two dimensions it would generalise for 3 move games with more heaps. In truth, we are no closer to answering the main question in this project as to whether three-move two-dimensional games are decidable and we await further research in this area with great interest.

As a final hypothetical point, for games that have enough moves and enough heaps to be undecidable, can we consider the outcome matrices to be truly random? If the outcomes are undecidable then it seems natural that these subtraction game outcomes could have potential use cases in cryptography, perhaps in pseudorandom number generators (PRNGs). By leveraging the unpredictable patterns and complex interactions within these games, we may be able to create robust and efficient PRNGs with desirable statistical properties. This exciting prospect is just one of many areas for potential future research.

4 Conclusion

The purpose of this thesis is two-fold. Firstly, it aims to introduce the reader to the fundamental concepts and strategies of impartial combinatorial games. Through a detailed exploration of classic games like Nim, Wythoff's Game, and Green Hackenbush, we have provided a solid foundation for understanding the key principles and techniques used in analyzing these games. The introduction of the Sprague-Grundy function and its application in solving complex game positions, along with the development of a game where readers can put their newfound knowledge to the test, offers an engaging and interactive learning experience.

Secondly, this thesis delves into the intriguing realm of undecidability in subtraction games. We have proven that determining the outcome of a general subtraction game is algorithmically undecidable, showcasing the inherent complexity and unpredictability of these games. By investigating two-move and three-move games in two dimensions, we have pushed the boundaries of our understanding, seeking to uncover the precise limits of decidability. The exploration of periodic and recursively enumerable games, along with the use of innovative techniques like outcome segmentation, offers a glimpse into the exciting frontiers of this field. We hope that the reader will find this journey through the landscape of undecidability both intellectually stimulating and rewarding, as it challenges our intuitions and invites further exploration into the depths of combinatorial game theory.

The code used to generate the outcome matrices and the outcome segmentation can be found at the [Github Repository](#) for the project if the reader would like to try some two-dimensional game examples of their own.

References

- [1] Conway, J. H., Berlekamp, E. R., and Guy, R. K. (1982). *Winning Ways for Your Mathematical Plays, Volume 1*. Academic Press.
- [2] Thomas S. Ferguson *A Course in Game Theory*, Part 1: Impartial Combinatorial Games <https://www.mina.moe/wp-content/uploads/2018/05/GAME-THEORY-Thomas-S.Ferguson.pdf> (2018)
- [3] Lecture Notes *Theory of Impartial Games*. The Mathematics of Toys and Games, SP.268, February 3, 2009. <https://web.mit.edu/sp.268/www/nim.pdf>
- [4] C. L. Bouton *Nim, a game with a complete mathematical theory*, Ann. Math. 3, 35-39.
- [5] W. A. Wythoff (1907) *A modification of the game of nim*, Nieuw Archief voor Wiskunde 7, 199-202
- [6] Dr A. N. Walker. *G13GAM – Game Theory – Impartial Hackenbush*. <https://www.cs.cmu.edu/afs/cs/academic/class/15859-s05/www/lecture-notes/green-hackenbush.html>, (1997).
- [7] Padraic Bartlett. *A Short Guide to Hackenbush*. <https://www.math.uchicago.edu/~may/VIGRE/VIGRE2006/PAPERS/Bartlett.pdf>, (2006).
- [8] Davorin Stajsic. *Combinatorial Game Theory*. <https://libres.uncg.edu/ir/uncg/f/Stajsic-uncg-0154M-10489.pdf>, (2010).
- [9] Urban Larsson and Johan Wastlund *From Heaps of Matches to the Limits of Computability* Electron. J. Combin., 20(P41), 2013.
- [10] Urban Larsson, Indrajit Saha and Makoto Yokoo. *Subtraction games in more than one dimension*. <https://arxiv.org/abs/2307.12458>, (2024).
- [11] A. M. Turing, *On Computable Numbers, with an Application to the Entscheidungs problem*. Proc. London Math. Soc., series 2, 42 (1936-37), 230-265.
- [12] Solomon W Golomb, *A mathematical investigation of games of "take-away"*. J. Combinatorial Theory, 1(4):443-458, 1966.