CLASSICAL IMPARTIAL GAMES

ADVAITH MOPURI

1. Introduction

In this exposition, we describe some of the basics of the theory of impartial games and analyze poset games. The theory in sections 2 and 3 follows [1] and [2].

2. Impartial Games

Definition 2.1 (Impartial Games)**.** An *impartial game* is one where each player has the exact same set of moves.

One key fact about impartial games is that each game is either in $\mathcal{N}$ or in $\mathcal{P}$. This is somewhat intuitive – by definition, each player has the same moves available to them, so is it impossible for either Left or Right to always win a game. The only thing that matters is which player moves first.

In order to actually determine the outcome classes of an impartial game, we sometimes rely on the partition theorem.

Theorem 2.2 (Partition Theorem)**.**

(1) Let G be a short impartial game. Then $G \in \mathcal{N}$ if and only if there is some move from G, to G', say, with $G' \in \mathcal{P}$. Conversely, $G \in \mathcal{P}$ if and only if, for every move to some G', we have $G' \in \mathcal{N}$.

(2) Let $\mathscr{S}$ be a set of short impartial games such that if $G \in \mathscr{S}$, then all subpositions of G are also in $\mathscr{S}$. Suppose there exist subsets $\mathscr{P}$ and $\mathscr{N}$ of S that partition $\mathscr{S}$, i.e., $\mathscr{P} \cup \mathscr{N} = \mathscr{S}$ and $\mathscr{P} \cap \mathscr{N} = \emptyset$, and such that, for every $G \in \mathscr{N}$, there is a move to some game in $\mathscr{P}$, and for every $G \in \mathscr{P}$, every move is to a game in $\mathscr{N}$. Then $\mathscr{P}$ and $\mathscr{N}$ are the $\mathcal{P}$ and $\mathcal{N}$ positions, respectively, in $\mathscr{S}$.

Proof. We begin with the proof of the first statement in the first part. If $G \in \mathcal{N}$ and it is currently your move, then you must have a way to force a win by definition of $\mathcal{N}$. If there was no move G' from G such that $G' \in \mathcal{P}$, then every G' must be in $\mathcal{N}$. Then, the other player would be able to force a win no matter what move you made – a contradiction. Thus, there must exist at least one $G' \in \mathcal{P}$.

On the other hand, assuming that there exists a move G' from G such that $G' \in \mathcal{P}$, then you can simply move to G' from G and cause your opponent to play in $\mathcal{P}$ and eventually lose.

Date: October 27, 2024.

The proof of the other statement is quite similar. First, assume that it is your move and $G \in \mathcal{P}$. Then, no matter what move you make, it must be the case that your opponent wins by definition of $\mathcal{P}$. In other words, every move G' from G must be a win for your opponent; every $G' \in \mathcal{N}$.

Conversely, If every $G' \in \mathcal{N}$, then no matter what move you make, your opponent has a way to force a win. So, $G \in \mathcal{P}$.

To prove the second part, we show that $\mathscr{N} = \mathcal{N}$. Since the only outcome classes for impartial games are $\mathcal{N}$ and $\mathcal{P}$, and $\mathscr{N}$ and $\mathscr{P}$ partition $\mathscr{S}$, this implies that $\mathscr{P} = \mathcal{P}$ as well.

The second part is quite similar and thus omitted. ■

For clarity, we also provide a few examples.

Example 2.3 (2-pile Nim)**.** *Consider the game of Nim played with 2 piles. We can represent such games as (a, b), for $a, b \in \mathbb{N}_0$ – let all such representations comprise $\mathscr{S}$. Then, it is easy to see that the sets $\mathscr{P} = \{(a, b) \mid a = b\}$ and $\mathscr{N} = \{(a, b) \mid a \neq b\}$ partition $\mathscr{S}$ and satisfy the conditions outlined in the second part of 2.2. So, $\mathscr{P} = \mathcal{P}$ and $\mathscr{N} = \mathcal{N}$.*

Example 2.4 (Fibonacci Nim)**.** *Consider the game of Fibonacci Nim, which is played similarly to single pile Nim with the additional conditions that the first player can remove any number of stones from the pile (but cannot remove the whole pile), and on each subsequent turn, the player to move can remove at most twice as many stones as the previous player. Each Fibonacci Nim position can be represented as (n, r), where the pile has n stones left and the current player can remove up to r stones.*

Let $\mathscr{S}$ be the set of all Fibonacci Nim positions, $\mathscr{P}$ be the set of all positions where r is less than the smallest Fibonacci number in the Zeckendorf representation of n (see Zeckendorf's Theorem), and $\mathscr{N} = \mathscr{S} \setminus \mathscr{P}$. Clearly, $\mathscr{N}$ and $\mathscr{P}$ partition $\mathscr{S}$, and surprisingly, they also correspond to $\mathcal{P}$ and $\mathcal{N}$ by 2.2.

3. Nim Sums and Sprague-Grundy Theory

The nim sum, also known as xor, provides a great deal of insight into analyzing Nim positions. For any two numbers a and b, we define $a \oplus b$ as the result of writing a and b in binary, lining the numbers up as in addition, and then applying the rules $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, and $1 \oplus 1 = 0$ to each column of binary digits. Note that $\oplus$ is a commutative and associative operation, so these rules are easily extended to compute the nim sum of multiple numbers.

To make use of this operation, recall the following theorem from [2, Chapter 4].

Theorem 3.1. *The position with piles of size $a_1, a_2, \ldots, a_k$ is a $\mathcal{P}$ position if and only if $a_1 \oplus a_2 \oplus \cdots \oplus a_k = 0$.*

The proof of this theorem is provided in its entirety in [2], so we will omit it here. Note that 2.2 is used in the proof.

Additionally, the nim sum allows us to reduce any multi-pile Nim game into a single pile game.

Theorem 3.2. *A Nim game G with piles of size $a_1, a_2, \ldots, a_k$ is equivalent to a game H with a single pile of size $a_1 \oplus a_2 \oplus \ldots \oplus a_k$.*

Proof. From the definition of equivalent games, we have that G and H are equivalent if and only if $G - H \in \mathcal{P}$. Since H is impartial, its set of left and right options are the same. So, $-H = H$ and it suffices to show that $G + H \in \mathcal{P}$.

The nim sum of $G + H$ is equivalent to the nim sum of G nim summed with H due to associativity. In other words, this is equivalent to

$$(a_1 \oplus a_2 \oplus \ldots \oplus a_k) \oplus (a_1 \oplus a_2 \oplus \ldots \oplus a_k),$$

which equals 0 since the nim sum of 2 equal values is 0. Thus, by 3.1, $G + H \in \mathcal{P}$. ■

This theorem motivates us to define the *nimbers* as representation of single pile nim games, since every Nim game can in fact be expressed as a single pile.

Definition 3.3 (Nimbers)**.** The *nimbers* are recursively defined by

$$*n = \{*0, *1, \ldots, *(n-1)\},$$

for all nonnegative integers n.

Nimbers allow us to easily represent options of games in a way that allows us to apply our previous theorems. Additionally, we also have the following rule to evaluate positions whose options are nimbers.

Theorem 3.4 (Mex Rule)**.** *For a set of nonnegative integers S, define $mex(S)$ to be the smallest nonnegative integer not in S. Then, we have*

$$\{*a_1, *a_2, \ldots, *a_k\} = *m,$$

where $m = mex(a_1, a_2, \ldots, a_k)$.

In fact, 3.4 allows us to compute a value n for *any* impartial game G such that G is equivalent to to the Nim game with a single pile of n stones.

Theorem 3.5 (Sprague-Grundy)**.** *Let G be any short normal-play impartial game. Then $G = *n$ for some nonnegative integer n.*

We omit the proofs for the previous 2 theorems since they are entirely stated in the textbook.

Note that 3.5 allows us to extend the idea of a nim sum to other impartial games than Nim. Combined with 3.1, this implies that any impartial game is $\mathcal{P}$ if it has nim sum 0, and is $\mathcal{N}$ otherwise.

4. Poset Games

Definition 4.1 (Posets)**.** A *partially ordered set*, or *poset*, is a set P along with a partial order $\preceq$ which is reflexive ($a \preceq a$ for all $a \in P$), antisymmetric ($a \preceq b$ and $b \preceq a$ implies $a = b$ for $a, b \in P$), and transitive ($a \preceq b$ and $b \preceq c$ implies that $a \preceq c$ for $a, b, c \in P$). The term "partial" comes from the fact that two elements $a, b \in P$ are not necessarily comparable under $\preceq$.

For the rest of this section, we will refer to arbitrary posets as $(P, \preceq)$.

Definition 4.2 (Poset Games)**.** A *poset game* begins with a poset $(P, \preceq)$. Players take turns picking an element $a \in P$, and then removing a and any elements in P greater than a from the set. We denote the poset resulting from this move by P_a. In this section, we consider the normal play variant of poset games.

Note that due to 3.5 and 3.4, we can represent the Grundy value of a poset game P as

$$\mathcal{G}(P) = \text{mex}(\{\mathcal{G}(P_a) \mid a \in P\}).$$

Poset games are particularly interesting because they extend more well known impartial games such as the following 3 examples.

Example 4.3. *Nim is a very straightforward example of a poset game. For a poset P of k-pile games, we define $\preceq$ such that for two games $A = (a_1, a_2, \ldots, a_k)$ and $B = (b_1, b_2, \ldots, b_k)$, $A \preceq B$ if and only if $a_i \leq b_i$ for all $1 \leq i \leq k$.*

Example 4.4. *Green Hackenbush is another example of a poset game. For green Hackenbush games A and B, we say that $A \preceq B$ if and only if it is possible to obtain A from a series of moves in B.*

Example 4.5. *A slightly modified version of a game known as Chomp is also a poset game. Chomp begins with a rectangular grid which is missing its bottom left square. The player to move selects any square in the grid and removes it along with any squares above it and to its right. In the normal play variant of this game, the last player that can make a move wins.*

We can define a very natural partial order $\preceq$ on the squares of a Chomp board. Formally, if we assign a coordinate system to an $m \times n$ Chomp grid such that the bottom left corner is $(0, 0)$ and the top right corner is (m, n), then we say that $(a, b) \preceq (c, d)$ (where $0 \leq a, c \leq m$ and $0 \leq b, d \leq m$) if $a \leq c$ and $b \leq d$.

In some specific cases, analyzing the outcome class of poset games is quite simple. Consider the following lemmas which apply for P with an infimm or a supremum.

Lemma 4.6 (Posets With an Infimum)**.** *In a poset game $(P, \preceq)$, if P has an infimum, then $P \in \mathcal{N}$.*

Proof. Let the infimum of P be a. The first player to play simply moves to P_a. This removes every element in the poset, so the other player loses. ∎

Lemma 4.7 (Posets With a Supremum)**.** *In a poset game $(P, \preceq)$, if P has a supremum, then $P \in \mathcal{N}$.*

Proof. Let the supremum of P be b. If $\mathcal{G}(P_b) = 0$, then $\mathcal{G}(P) > 0$ by our above expression for the Grundy value of P. So, it must be the case that $P \in \mathcal{N}$. Otherwise, if $\mathcal{G}(P_b) > 0$, then $P_b \in \mathcal{N}$ so there must exist some $a \in P_b$ such that $\mathcal{G}((P_b)_a) = 0$ by 2.2. However, b is a supremum of P so $a \leq b$ meaning that P_a is a valid move from P. Thus, P must have a Grundy value greater than 0 in this case as well. ■

The strategy stealing argument presented in the proof of the previous lemma can also be used to show that the first player to move in Chomp can guarantee a win, as long as the starting position is nonempty.

5. Time Complexity of Poset Games

In general, however, finding the winning strategy or even determining the winner of a poset game is computationally difficult. To see just how difficult this is, we first define some standard notions of computational complexity, beginning with Turing machines. The following definitions and theory come from [2, Chapter 18].

Definition 5.1 (Turing Machines)**.** At a high level, a Turing machine M is a machine that uses a *set of states* Q and an *alphabet* Γ to solve problems. The problem to be solved is written on a tape using Γ (this is known as the *input string*), followed by infinitely many blank cells of the tape. This tape is then given to M.

The machine begins solving the problem on the first cell of the tape, say, x_0, at a designated *start state* q_0. The *transition function* δ tells M how to progress from the current state/alphabet pair. In particular, if $\delta(q, x) = (q', x', L)$, then M replaces the value of the cell it is currently on, x, with x'. The machine then moves one cell left (hence the L – if M is to move right, this would be R) and transitions to a new state, q'.

In the process of repeatedly applying δ, M might transition to one of 2 special states – q_a or q_r. If the machine enters the former state, then the input is said to be *accepted*. If it enters the latter state, it is *rejected*.

At every point in the problem solving process, note that the tape can be represented by a finite string of non-blank characters from Γ. We denote by $\mathcal{L}(M)$ the set of such strings which, when provided as input strings to a Turing machine M, cause it to accept. This set is known as the language of M, or the language accepted by M.

We are now ready to define time complexity and big O notation.

Definition 5.2 (Time Complexity)**.** The running time time complexity of a Turing machine M is the function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n)$ is the maximum number of steps that M uses on any input of length n. We say that M runs in time $f(n)$.

Definition 5.3 (Big O)**.** Let f and g be functions from $\mathbb{N}$ to $\mathbb{R}$. We say that $f(n) = O(g(n))$ if there exist positive numbers C and N such that for all $n \geq N$, we have

$f(n) \leq Cg(n)$. If this is the case, we say that $g(n)$ is an asymptotic upper bound for $f(n)$.

Example 5.4. *Note that for polynomials f of degree d, we have $f(n) = O(n^d)$.*

Using big O notation, we are able to group various time complexities into the same class.

Definition 5.5 (Time Complexity Classes)**.** Let $t : \mathbb{N} \to \mathbb{N}$ be a function. The time complexity class $\text{Time}(t(n))$ is the set

$$\{\mathcal{L} : \mathcal{L} \text{ is a language accepted by a } O(t(n)) \text{ time Turing machine}\}.$$

We are finally able to define common complexity classes.

Definition 5.6 (Polynomial Time)**.** We define the time complexity class P, also known as the polynomial time class, as

$$\bigcup_{k=1}^{\infty} \text{Time}(n^k).$$

At an intuitive level, the polynomial time class consists of all languages for which it is fairly easy (hence, polynomial time) to decide whether or not a given string is in the language or not. For example, consider a language which represents $\mathcal{P}$ Nim positions. This language is in P as 3.1 provides a polynomial time method to determine whether or not a given Nim game is in $\mathcal{P}$.

Given our previous definitions, it is also easy enough to define another important time class, NP.

Definition 5.7 (Nondeterministic Polynomial Time)**.** Let L be a language. A verifier for L is a Turing machine V such that

$$L = \{w \mid V \text{ accepts } (w, c) \text{ for some string } c\}.$$

The general idea here is that c acts as a proof certificate for w. The Turing machine V reads the pair (w, c) as input, and then checks if this pair is in some other language.

We call V a polynomial time verifier if V runs in polynomial time in the length of w. The language L is said to be polynomially verifiable if it has a polynomial time verifier. The nondeterministic polynomial time class, NP, is the set of all languages that have polynomial time verifiers. From this definition, it is clear that P $\subseteq$ NP. However, it is still a famous open problem to show whether or not the two complexity classes are equal.

We have thus far seen complexity classes based on the number of steps that a Turing machine takes to either accept or reject. Another natural way of defining complexity is based on the number of cells of the tape required for the machine to resolve. In fact, replacing "time" in 5.1, 5.2, 5.5, and 5.6 with "space" yields the space complexity class PSPACE. It is not too hard to show that P $\subseteq$ NP $\subseteq PSPACE$.

With the prerequisite definitions out of the way, we are free to discuss interesting results regarding the computation of winning strategies in poset games.

In particular, [3] shows that this task is in PSPACE. The authors go about this by providing a reduction in polynomial time (and hence, polynomial space) between a general poset game, and a special game known as Geography. Though we have not formally defined reductions, they can intuitively be thought of as a polynomial time function that translates one language to another.

This reduction is important since Geography is a well studied game. In particular, Geography is in PSPACE, so general poset games must be in this space complexity class as well.

Geography consists of a directed graph G and a node s of G. A marker is placed on s to begin the game, and each turn, the marker is moved via an edge to an adjacent node of G. Once an edge is used, it may not be used again. As usual, the first player unable to make any moves loses. The reduction from general poset games to Geography is a bit too long for this paper, but it is not too hard to follow the argument provided in [3].

Additionally, this paper mentions that except for a few common examples including Nim, algorithms for finding a winning strategy for a poset game which run in polynomial time are unknown.

Extending this result, [4] shows that for an arbitrary finite poset game, deciding the winner of the game is in a complexity class known as PSPACE-complete. This new class is defined as follows.

Definition 5.8 (PSPACE-complete)**.** A language B is said to be PSPACE-complete if $B \in$ PSPACE and every language $A \in$ PSPACE is reducible in polynomial time to B.

It is surprising that a definition primarily concerning space complexity relies on a reduction in polynomial time, but this is due to the fact that time reductions are generally much easier to perform than space reductions.

A final interesting remark about poset games is their periodicity of Grundy values. Though it is a weaker result than the periodicity theorem for octal games, the theorem presented in [5] is still quite interesting. In particular, it shows that if there are an infinite number of positions of a certain type in a poset game, then the Grundy values of the game must be eventually periodic. The specific details of the theorem and proof require many preliminaries and are quite complex, though the paper is still an enjoyable read for anyone looking to learn more about the subject.

References

[1] Elwyn Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for Your Mathematical Plays (4 vols.)*. A. K. Peters, 2 edition, 2000.
[2] Simon Rubinstein-Salzedo. *Combinatorial Game Theory*. 2024.
[3] Michael Soltys and Craig Wilson. On the complexity of computing winning strategies for finite poset games. *Theory of Computing Systems*, 2011.

[4] Daniel Grier. *Deciding the Winner of an Arbitrary Finite Poset Game Is PSPACE-Complete*, page 497–503. Springer Berlin Heidelberg, 2013.
[5] Steven Byrnes. Poset game periodicity. *Integers: The Electronic Journal of Combinatorial Number Theory*, (3), 2003.