Computing Winning Strategies for Poset Games

By
Craig Wilson, B.Eng

A Thesis
Submitted to the School of Graduate Studies
in partial fulfilment of the requirements for the degree of

M.A.Sc
Department of Computing and Software
McMaster University

MASTER OF APPLIED SCIENCE(2003) McMaster University
Hamilton, Ontario

TITLE: Computing Winning Strategies for Poset Games

AUTHOR: Craig Wilson, B.Eng(McMaster University)

SUPERVISOR: Dr. Michael Soltys

NUMBER OF PAGES: viii, 73

Abstract

The problem of computing winning strategies in games has yielded many important results with applications in fields such as computer science, economics, and mathematics. For example, "Online" games, where the input is received on the fly are used to model process scheduling and paging algorithms. The concept of Nash Equilibrium has implications in economic theory, and Ehrenfeuct-Frass games are used to check whether two structures are isomorphic.

In this thesis we are concerned with Partially-Ordered Set (Poset) games. We present two new methods for proving that computing winning strategies for the game of Chomp is in PSPACE. We also present the idea of "Game Completeness", and give an overview of efforts to translate any poset game into an equivalent configuration of Chomp. Finally, we show how Skelley's bounded arithmetic theory $\mathbf{W}_1^1$ can be applied to Chomp, and the consequences of expressing the existence of a winning strategy for the first player in Chomp in simpler arithmetic theories.

In short, the contributions of this thesis are as follows:

1. A new method for showing poset games in PSPACE, via a polynomial-time (logarithmic-space) reduction to the game of Geography.

2. Attempts at a reduction from Geography to poset games, and implications to whether poset games are PSPACE-complete.

3. A bounded-arithmetic formulation of the existence of a winning strategy for the first player in Chomp.

4. A definition of the concept of Game Completeness, and a translation from tree-like poset games to a modified version of Chomp.

Acknowledgements

The author would first like to acknowledge the guidance and continuous support of his parents, without whom this thesis would not be possible. The guidance and support provided by his supervisor Michael Soltys over the past two years is also greatly appreciated. Ridha Khedri and Ryszard Janicki both provided excellent feedback for improving this thesis during its draft stage, while Alan Skelley clarified many details regarding his theory $\mathbf{W}_1^1$. Finally, Grzegorz Herman provided the formulation of representing Chomp configurations as binary strings.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The problem of computing winning strategies in games has yielded many important results with applications in fields such as Computer Science, Economics, and Mathematics. For example, "Online" games, where the input is received on the fly are used to model process scheduling and paging algorithms. The concept of Nash Equilibrium has implications in Economic theory, and Ehrenfeuct-Fraïssé games are used to check whether two structures are isomorphic.

In this thesis we are concerned with Partially-Ordered Set (Poset) games. These games are simple to describe, but computing winning strategies appears intractable in all but the simplest cases. The generic poset game is played on a finite family of finite subsets of the natural numbers, ordered by inclusion. Players alternate picking a subset and removing all corresponding supersets. The player who has no sets to play on their turn loses. As in most finite, 2-player games, winning strategies in poset games are computable in polynomial space (PSPACE). Other examples of poset games include Nim, Hackendot, and Chomp. Chomp, played on a rectangular grid of squares (or "chocolate bar") is notable for having a simple nonconstructive proof of the first player having a winning strategy, while retaining the intractability of other poset games. Although it is known by a strategy-stealing argument that the first player of the game has a winning strategy for any grid size, an efficient (i.e., polynomial time or better) method to compute this strategy is presently unknown.

Our contributions begin with three different proofs (two of which are completely new methods of showing existing results) showing computing winning strategies in

Chomp being a PSPACE computation. Each of these proofs sheds a different light on the aspects of why this calculation is in PSPACE. From this we explore the idea of winning strategies in Chomp being a PSPACE-complete endeavor, and to what extent current results support or disprove this idea. We next define the concept of "Game Completeness", and give an overview of efforts to translate any poset game into configurations of Chomp. Finally, we present the third-order bounded arithmetic theory $\mathbf{W}_1^1$ and give its applications to Chomp, which include a new possibility for explicitly computing the winning strategy for the first player of the game.

The first of the three proofs (and first of the new methods) showing winning strategies in Chomp to be a PSPACE calculation involves a translation from Chomp to the game of Geography, which is known to be PSPACE-complete. The second new proof uses the aforementioned $\mathbf{W}_1^1$ theory along with results of Skelly to formalize the statement that the first player always has a winning strategy for any game of Chomp. The third proof represents Chomp as a Quantified Boolean Formula, making use of the fact that the problem of Quantified Satisfiability (QSAT) is PSPACE-complete. In exploring whether computing winning strategies in Chomp is a PSPACE-complete operation, we examine efforts to translate Geography into configurations of Chomp, what progress has been made, and what limitations these efforts reveal. The idea of "Game Completeness" states that any poset game can be translated (in polynomial time) into a configuration of another game, while preserving the winning player. Using this concept, a translation from poset games to a simplified version of Chomp, Labeled Chomp is presented. Finally, we use $\mathbf{W}_1^1$ to formalize the notion of the existence of a winning strategy for the first player in any Chomp game.

1.1 Scope of Games Covered

The games presented here are all 2-player games, and also share the following properties:

1. **Perfect Information:** Each player is aware of all actions that have taken place (and their effects) throughout the game, both their own and their opponent's.

2. **No Chance:** No random effects, dice rolls for example, are present in the game.

3. **No Draws:** Each game has both a winning and losing player, there are no instances where both players lose or win.

4. **Impartiality:** The moves available to each player is the same.

Chapter 2

Games On Posets

This section introduces the concept of a **Poset Game**, by first giving the definition of a partially-ordered set (a **poset**), then presenting different methods of representing a poset, and finally giving examples of poset games, including Chomp.

2.1 Partially-Ordered Sets

A partially ordered set (poset) is a set U together with a relation $\preceq$, which is a subset of $U \times U$, called the ordering of the elements of U. Note that for the purposes of this thesis we will only consider posets where U is finite. Assuming $a, b, c \in U$, the relation $\preceq$ must satisfy the following properties:

1. Reflexivity: $a \preceq a$.

2. Antisymmetry: if $a \preceq b$, and $b \preceq a$, then $a = b$.

3. Transitivity: if $a \preceq b$ and $b \preceq c$, then $a \preceq c$.

As the name "partially ordered set" implies, there may be elements of U for which $\preceq$ is not defined. If $\preceq$ is not defined for $a, b \in U$, i.e. neither $a \preceq b$ or $b \preceq a$ are true, we say that a and b are **incomparable**, and denote this by $a||b$. A **chain** is a subset of U in which none of the elements are incomparable.

2.2 Isomorphism to the Natural Numbers

Assuming we have a countable poset $(X, \preceq)$, we can relate it to the natural numbers in the following way. We define the set $S_a = \{x \mid x \preceq a\}$ for each $a \in X$, and the set $S_{\preceq} = \{S_a \mid a \in X\}$. Then we have the following theorem:

Theorem 1 (Folklore). $a \preceq b \leftrightarrow S_a \subseteq S_b$

Proof. For the forward ($\rightarrow$) direction, we know that $a \preceq b$. Suppose also that there is some element $x \preceq a$, thus $x \in S_a$. By the transitivity of posets this implies $x \preceq b$, and also that $x \in S_b$, meaning $S_a \subseteq S_b$.

For the reverse ($\leftarrow$) direction, if $S_a \subseteq S_b$ then we know there is an element a common to both S_a and S_b. If $a \in S_b$, then $a \preceq b$. □

From this theorem we notice that if the set X is countable, then it can be labeled with natural numbers, i.e. $X = \{x_1, x_2, \ldots\}$. If we define $N_a = \{i \mid x_i \in S_a\}$, then we have the following result:

Corollary 2. $a \preceq b \leftrightarrow N_a \subseteq N_b$

In addition to this, however, we can show an even stronger result utilizing the fact that the natural numbers are totally ordered. Let $(X, \trianglelefteq)$ be any total extension of $(X, \preceq)$, i.e. $a \preceq b \rightarrow a \trianglelefteq b$, and $\trianglelefteq$ is a total order. We can now use this to number elements of X with natural numbers such that $X = \{x_1, x_2, \ldots\}$ and

$$x_i \trianglelefteq x_j \leftrightarrow i \leq j$$

which immediately yields

$$x_i \preceq x_j \leftrightarrow i \leq j$$

And from which we have $\forall x_i \in X. \forall x_j \in S_{x_i} : j \leq i$.

2.3 String Representation

Section 2.2 showed that there is an isomorphism between countable posets and the natural numbers. This fact can be used to develop an alternate representation of posets as sets of strings. The representation only applies to finite posets, but as we

are only considering finite games in this document this restriction comes naturally. We can represent any finite subset of $\mathbb{N}$ as a finite string over $\{0, 1\}$, with the i^{th} bit of the string set to 1 if and only if i is in the subset. Thus, if we have a finite poset U, we can use the result shown in Corollary 2 to translate the poset into a subset of $\mathbb{N}$, which then can be put into the form of a binary string. From here we use the usual lexicographic ordering of binary strings ($00 < 01 < 11$, for example) as the ordering relation of the strings, to form our new poset $(U, <)$.

2.4 Poset Games

Now that posets have been introduced, we can use them to construct games between two players. Examples of these games are Nim, Hackendot, and Chomp, which will be discussed further in Section 2.5. If we have a poset $(U, \preceq)$, then the corresponding poset game is $(A, \preceq)$, and is played as follows [6]:

1. Make A equal to the set U. The poset game will use A as a record of what elements of U are currently in the game.

2. The two players take turns selecting an element $x \in A$, and removing all elements $y \in A$ such that $x \preceq y$.

3. The end of the game occurs when a player is unable to select an element of A, i.e. when $A = \emptyset$. The player who is unable to make a move loses the game.

2.5 Introduction to Chomp

Originally described by Gale in [11], Chomp is a 2-player game played on a rectangular board of n rows and m columns. Each player takes turns selecting a square from the board. When a square is selected, all other squares to the right and above it, including the selected square, are removed from the board. The player left with only the bottom-left square on their turn loses. We define a **round** of a Chomp game as a sequence of two consecutive moves, the first belonging to player 1, and the second belonging to player 2. Figure 2.1 gives an example of a Chomp grid with 4 rows and 6 columns, and "X" as the poisoned square.

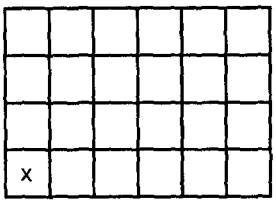

Figure 2.1: A Chomp grid with 4 rows and 6 columns

More formally, a Chomp board is a set of pairs (called cells) $\{(i,j)|1 \leq i \leq n, 1 \leq j \leq m\}$, with $(1,1)$ representing the poisoned bottom-left square. A move by either player in the game involves selecting a pair (i_0, j_0) in the set, and removing all (i,j) such that $i \geq i_0$ and $j \geq j_0$. The player who is left with only $(1,1)$ on their turn loses.

2.6 A Chomp Board Configuration

A **Chomp Board Configuration** ("configuration" onwards) is a "snapshot" of the Chomp board's current state. It indicates what squares of the board have been removed, and what squares remain. The first representation of a configuration we present is an n - tuple $(a_1, a_2, \ldots, a_n)$, where each a_i represents the number of squares present in the i^{th} row of the board [17]. Thus, two properties of each a_i that can be immediately stated are:

1. $\forall i : 1 \leq i \leq n : 0 \leq a_i \leq m$

2. $\forall i : 1 \leq i < n : a_i \geq a_{i+1}$

Property 1 follows from the dimensions of the Chomp board, as each row can have at most m columns. Property 2 states that each row cannot contain less squares than the row above it. The reason why this is true is a bit more subtle, as it relies on the nature of a Chomp move. Recall from Section 1 that when a player selects a square (i,j) as their move, this square and all other squares (i_0, j_0) above and to the right are removed. Thus, having both an empty location (i,j) and a filled location $(i+1,j)$ simultaneously is impossible. Finally, the **initial configuration** of any n by m Chomp board thus has all rows containing m squares, represented by

$(m, m, \ldots, m)$. Using this definition, we can define Chomp as a language, namely as a set of encodings of configurations:

$$Chomp = \{< n, m, c_1, \ldots, c_l > \mid ValidChomp\,(c_1, \ldots, c_l)\}$$

Where $ValidChomp\,(c_1, \ldots, c_l)$ is true iff $c_1, \ldots, c_l$ encode a legal n by m game of Chomp in which player 1 has a winning strategy. Here "legal" means that each c_i can be obtained in one legal move from c_{i-1}, except for the initial configuration, naturally.

2.7 Chomp as a Poset

Not surprisingly, Chomp has a direct translation into the language of posets. In this language Chomp is a game played on a poset $(P, \preceq)$ where P has both a unique smallest element, and also a unique largest element [4]. Similar to the board game version, a move involves selecting an $x \in P$ and removing all $y \in P$ such that $x \preceq y$. This creates a poset $(P', \preceq)$ with $P' \subset P$, which is used by the next player. The player who selects or is forced to select the smallest element loses. Note that this winning condition differs from the one given in section 2.4, however we can remedy this by removing the smallest element from P, in which case the player who is unable to select an x from P loses.

2.8 Other Properties of Chomp

In addition to defining the notions of a configuration, there are also important general properties of Chomp that will be used in upcoming sections. These properties will be stated and proved below.

Lemma 3. *The maximum number of moves for an n by m game of Chomp is $(n \times m) - 1$.*

Proof. Consider an n by m Chomp board. Suppose the first player selects the top - right square of the board, (n, m). The second player then counters by playing the square adjacent to this, $(n, m-1)$. Both players continue selecting adjacent squares on the same row, and move down to the next row when the current row is emptied.

Thus, the players will make m moves on each of the rows until the last row, where only $m-1$ moves are required, as the game ends with player 2 having only the poisoned square on their turn. □

Lemma 4. *The maximum number of configurations for an n by m game of Chomp is $\binom{n+m}{n}$.*

Proof. To prove this lemma we introduce our second representation for Chomp configurations, using binary strings of length $(n+m)$ [12]. These strings correspond to configurations of an n by m Chomp game, with two additional "imaginary" rows located at positions 0 (l_1) and $n+1$ (l_2), respectively. For all possible configurations of the board, row l_1 contains m squares, while row l_2 contains 0 squares. From this augmented board our binary strings encode the differences in lengths between each row, starting (from left to right) with the different between l_2 and row n, and ending with the difference between row 1 and l_1. Figure 2.2 presents a 4 by 5 Chomp grid augmented with the two imaginary rows. To actually encode the differences, each 1 in

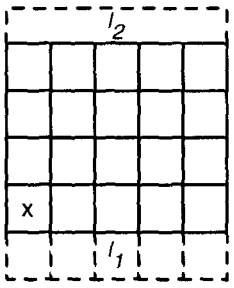

Figure 2.2: A 4 by 5 Chomp grid augmented with two additional rows l_1 and l_2.

our binary string represents a non-imaginary row in the Chomp grid, thus we have n 1s. Between each of these 1s are exactly as many 0s as the difference in length between the rows represented by the corresponding 1s. Thus, the initial configuration of a 4 by 5 Chomp grid is represented by the string in Table 2.1. As the game progresses, the 1s

0	0	0	0	0	1	1	1	1

Table 2.1: Initial Configuration for a 4 by 5 Chomp Grid

are moved leftward in the string, while the 0s move right, to represent the elimination

of squares from the board. Once all squares have been removed, the m 1s will be on the left side of the string, with the n 0s on the right. The player who is forced to place the string into this configuration loses. Table 2.2 presents this configuration for a 4 by 5 board. From this representation we notice that we have $n + m$ positions in

1	1	1	1	0	0	0	0	0

Table 2.2: End Configuration for a 4 by 5 Chomp Grid

which to play n 1s, with each arrangement of 1s representing a configuration of the game. Thus, the number of configurations is reduced to a combinatorial problem of placing n objects in $n + m$ places, which is $\binom{n+m}{n}$. □

Chapter 3

Winning Strategies in Games

In this section we present existing knowledge pertaining to winning strategies in two-person impartial games, including Chomp. A winning strategy is a method of playing or a sequence of moves that guarantees victory for a player, no matter the actions of their opponent. More formally, we can think of a winning strategy as a function f that takes as input the current configuration of the game, and returns the next move the player should make. The important property of f is that if the player plays this suggested move on their turn, and continues to adhere to f's advice for all of their subsequent moves, they will win. In the following sections we will give an example of a game for which a winning strategy is known (Nim), and present the Sprague-Grundy theorem, an important result which applies to all impartial two-person games. Finally, we will examine how the winning strategy function f is applied to Chomp, and for what initial and intermediate configurations winning strategies are known. Firstly, however, we give some terminology related to winning strategies.

3.1 $\mathcal{N}$ and $\mathcal{P}$ Positions

In the following sections we will classify configurations of games into one of two types, $\mathcal{N}$ positions or $\mathcal{P}$ positions, based on which player has a winning strategy beginning from the configuration [9]. $\mathcal{N}$ positions are configurations $C_{\mathcal{N}}$ where the next player to make a move has a winning strategy. In contrast, $\mathcal{P}$ positions are configurations $C_{\mathcal{P}}$ where the previous player, i.e. the player who put the game into $C_{\mathcal{P}}$ has a winning

strategy. From these two definitions we can extrapolate the following two properties:

Lemma 5. *At least one $\mathcal{P}$ position is reachable in one legal move from any $\mathcal{N}$ position.*

Proof. Suppose that we have an $\mathcal{N}$ position C where the only configurations reachable in one legal move are $\mathcal{N}$ configurations. Then, by the definition of $\mathcal{N}$, both players have a winning strategy, which creates a contradiction. □

Lemma 6. *Any move from a $\mathcal{P}$ position yields an $\mathcal{N}$ position.*

Proof. Suppose that from a $\mathcal{P}$ position C there is a move to another $\mathcal{P}$ position C'. This creates an immediate contradiction, as if a player makes the move from C to C', it means that C would be an $\mathcal{N}$ position, by Lemma 5. □

3.2 The Game of Nim

One of the most important games in the classical theory is Nim, which was originally analyzed by Charles L Boulton in [3], who also gave it the current monicker. The game of Nim involves two players and n distinct piles of objects (matchsticks, plates, etc.) of height h_i each. The two players take turns removing at least one object from any pile, and all objects removed must come from the same pile. The player who is left with no objects to remove loses the game; we call this normal play. The rules for winning and losing can also be modified so that the player who selects the last object loses, in which case we are playing under misère play conventions. The two types of winning conditions, normal and misère, are used to categorize most types of games. Chomp, for example, is played under misère rules. Unless otherwise stated, all further discussion of Nim is assumed to be under normal play.

Figure 3.1 gives an example Nim configuration with $n = 3, h_1 = 6, h_2 = 4$, and $h_3 = 5$, with the configuration expressed in tuple form $(n, h_1, h_2, \ldots, h_n)$. We will return to this example in the description of the winning strategy for Nim.

Nim as a game has been completely analyzed - from any configuration $(n, h_1, h_2, \ldots, h_n)$ we can compute whether it is an $\mathcal{N}$ or $\mathcal{P}$ position, and if it is an $\mathcal{N}$ position what move will put the opposing player into a $\mathcal{P}$ position. Many books contain the analysis, the method we present in the following paragraphs is taken

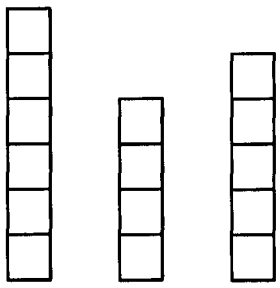

Figure 3.1: An example Nim configuration $(3, 6, 4, 5)$

from [7]. This computation of a winning strategy is efficient (polynomial time), and its components form the basis of many other important theorems in Game Theory, including the Sprague-Grundy Theorem discussed in Section 3.3.

As mentioned, the first part of the strategy is to compute whether the current configuration is an $\mathcal{N}$ or $\mathcal{P}$ position. To do this we first convert each h_i to its binary equivalent b_{h_i}, then exclusive-or (i.e., binary addition without the carry operation) all the b_{h_i}s together. This procedure is called the Nim-Sum of the individual Nim piles, while the exclusive-or of the binary representations is often called Nim-Adding or Nim-Addition. Equation 3.1 shows the Nim for our example Nim configuration in Figure 3.1, with $\oplus$ representing the exclusive-or operation.

$$\begin{aligned} \text{Nin-Sum} &= h_1 \oplus h_2 \oplus h_3 \\ &= 6 \oplus 4 \oplus 5 \\ &= 110 \oplus 100 \oplus 101 \\ &= 111 \\ &= 7 \end{aligned} \tag{3.1}$$

If the Nim-Sum is > 0, as in our example, then we are in an $\mathcal{N}$ position, otherwise we have the misfortune of residing in a $\mathcal{P}$ configuration. Obviously in the latter case our choice of move is irrelevant if our opponent is on the ball, but if we are an $\mathcal{N}$ position we need to make the correct move to give the opposing playing an $\mathcal{P}$ configuration. This means that we must move to a position where the Nim-Sum of the game is 0.

In order to determine what move will yield the desired result, we take the Nim-Sum of the current game configuration and Nim-Add it to each b_{h_i}, producing new

heights x_{h_i}, as in equation 3.2:

$$\begin{aligned} x_{h_1} &= \text{Nim-Sum} \oplus h_1 = 111 \oplus 110 = 001 = 1 \\ x_{h_2} &= \text{Nim-Sum} \oplus h_2 = 111 \oplus 100 = 011 = 3 \\ x_{h_3} &= \text{Nim-Sum} \oplus h_3 = 111 \oplus 101 = 010 = 2 \end{aligned} \tag{3.2}$$

The key observation now is that there will be at least one i for which $x_{h_i} < b_{h_i}$. This tells us not only what pile to remove objects from, but $|x_{h_i} - b_{h_i}|$, the difference between x_{h_i} and b_{h_i}, gives us how many objects to remove. In our example we can make a move on any pile and give the opposing player a $\mathcal{P}$ position. Our options are to remove $|x_{h_1} - h_1| = |1 - 6| = 5$ objects from the first pile, $|x_{h_2} - h_2| = |3 - 4| = 1$ objects from the second pile, or $|x_{h_3} - h_3| = |2 - 5| = 3$ objects from the third pile. We can quickly verify that each of these lead to $\mathcal{P}$ positions:

1. **Take from first pile:** Our new configuration is $(3, 1, 4, 5)$, and the new Nim-Sum is 0.

2. **Take from second pile:** Our new configuration is $(3, 6, 3, 5)$, and the new Nim-Sum is 0.

3. **Take from third pile:** Our new configuration is $(3, 6, 4, 2)$, and the new Nim-Sum is 0.

Thus, any of the three possible moves lead to a $\mathcal{P}$ position for the other player. Following this pattern, eventually the second player will be left in a configuration where they are left with no objects to select, and thus will lose the game. Further information on the winning strategy for Nim can be found in [7] and [2].

3.3 The Sprague-Grundy Theorem

Having given an overview of the game of Nim, we can now introduce one of the classic results in game theory, the Sprague-Grundy Theorem. Discovered independently by R. P. Sprague in 1936 and P.M. Grundy in 1939 [2], the theorem is stated as follows (taken from [2] and [7]):

Theorem 7. *Any finite, impartial, 2-player game is equivalent to a configuration of a Nim game with a single pile.*

The main result of this theorem is the Sprague-Grundy Function, which provides a method of labeling nodes on the Game Graph of a given game in order to calculate a winning strategy in polynomial-time. The game graph of a game is a graph whose nodes are all possible configurations C_i of the game, and a directed edge exists from node C_i to C_j if any only if C_j can be reached from C_i in one legal move. Note that because the game graph requires all possible configurations to be computed, it usually requires exponential time to create.

The labels given to each node of the game graph are called Grundy Numbers, and are calculated as follows [9]:

1. For every sink the in the graph (i.e., nodes with no outgoing edges), assign a label of 0.

2. For every node connected via a single edge to a sink, assign a label of 1.

3. For every other node in the graph, the label is the least nonnegative integer not in the set of labels of its children. This is called the minimum excluded value, or $mex - value$.

By giving each node a label we link them to a corresponding Nim pile. Thus, we can use the strategy given for Nim in section 3.2 to determine which configurations to move to in the graph in order to win. Although this does provide a polynomial-time winning strategy, the need to construct the game graph, which is usually an exponential-time task limits its uses.

3.4 Fixed-Point Method for Winning Strategies

Another method for computing winning strategies in games is to calculate the fixed point of a function, as given by Backhouse and Michaelis in [1]. Here the game is characterized by a binary relation M that indicates the moves which can be made from any given position, analogous to the game graph for a game. A winning strategy is a relation W on positions, with the following three properties [1]:

1. $W \subseteq M$

2. $W \bullet M$ is well-founded, with $\bullet$ being composition of relations

3. $\forall t : t \in range(W) : (\forall u : (t,u) \in M : u \in dom(W))$, with $range(W)$ and $dom(W)$ being functions giving the range and domain of W, respectively

The first property is intuitively obvious - the moves of the winning strategy must be a subset of the moves which can be made in the actual game. The second property implies that by following the moves of the strategy the game will eventually terminate. Finally, the third property states that for all states in the range of the winning strategy (i.e., states that the opponent makes their moves on), any move is to a state of the game in the domain of the strategy, i.e. states that we make our moves on. Thus, positions in the domain of the strategy are $\mathcal{N}$ positions, while positions in the range are $\mathcal{P}$ positions.

The categorization of configurations into winning and losing positions is formally done via the predicates *win* and *lose*, which are defined as follows:

$$win \Rightarrow \mu(\text{Some}.M \circ \text{All}.M)$$
$$lose \Rightarrow \mu(\text{All}.M \circ \text{Some}.M)$$

where $\circ$ denotes composition of functions, Some and All are predicate transformers, given by the definitions

$$\text{All}.R.p.s \equiv \langle \forall t : sRt : p.t \rangle$$
$$\text{Some}.R.p.s \equiv \langle \exists t : sRt : p.t \rangle$$

with R being a relation on positions, p a predicate on positions, and s and t as positions. Finally, "μ denotes the function which maps a monotonic endofunction to its least fixed point" [1]. Intuitively, these definitions correspond to Lemmas 5 and 6, respectively. Thus, the winning positions of the game can be computed by finding the least fixed point of the function $(\text{Some}.M \circ \text{All}.M)$.

The algorithm for constructing the winning strategy continues this idea by iterating over all possible positions in the game to approximate the least fixed point of non-monotonic function g. As the number of positions in the game is assumed to be finite, the algorithm is guaranteed to terminate. The algorithm computes the winning strategy incrementally, beginning with the empty relation, and utilizes the following

two auxiliary functions f and g, defined for any relation R [1]:

$$f.R = \neg(\text{dom}.R) \bullet M \bullet \text{All}.M.(\text{dom}.R)$$
$$g.R = R \cup f.R$$

Note that positions satisfying All.M.(dom.W) are also shown to be losing positions, thus moves from positions not in dom(W) to positions where All.M.(dom.W) is true are winning, and thus added to W in the algorithm. Using this fact along with the above functions, the algorithm itself is as follows [1]:

{number of positions is finite}
$W := \emptyset$ { $\emptyset$ denotes the empty relation}
; {**Invariant:** W is a winning strategy}
do $\neg(W = g.W) \rightarrow W := g.W$
od
{W is a winning strategy $\wedge$ [dom.W = win = $\{\mathcal{N}\}$]}

Although unique in the fact that it uses relations and fixed-point theory instead of the usual set theory, like the Sprague-Grundy function an actual implementation of the algorithm would require the enumeration of all possible configurations in the game to represent the relation M. As has been stated before, this is usually an exponential-time task and thus not very practical.

3.5 Winning Strategies in Chomp

We now look to apply a strategy function f to Chomp. Defining $x_k = (i_x, j_y)$ and $y_k = (i_y, j_y)$ as the moves of player 1 and 2 in round k respectively, C as the current configuration and l as the last round of the game, f can be defined as:

$$f(C) = \{(i,j) | \forall y_k \exists x_{k+1} \forall y_{k+1} \exists x_{k+2} \forall y_{k+2} \ldots \exists x_l : \text{player 1 wins}\} \qquad (3.3)$$

Note that we can define the function in terms of player 2 by switching the meaning of the x_ks and y_ks. We will see later that this is not necessary, however, as only the first player has a winning strategy in Chomp. A second observation to make about this definition is that it allows for the possibility of f returning multiple pairs of indices.

There could very well be situations where more than one move will still lead to a winning strategy, although it would be an interesting adventure to prove that for any Chomp winning strategy the move that must be played at any given configuration is unique.

At this point it is important to discuss the difference between a winning strategy for a specific player, and a winning strategy from a specific board configuration. When stating that "player 1 has a winning strategy", it is meant that from the initial configuration of the Chomp board, the first player has a sequence of moves that will win them the game, regardless of what the second player does. If, on the other hand, a player has a winning strategy starting from a specific configuration, then we say "from this configuration, player 1 can always win", for example.

3.5.1 Existence of a Winning Strategy for the First Player

It is known that either the first or second player has a winning strategy for Chomp. To show this, we can formulate an instance of the game (for a particular n and m) as a fully Quantified Boolean Formula (QBF):

$$\exists x_1 \forall y_1 \exists x_2 \forall y_2 \ldots \exists x_l R(x_1, y_1, x_2, y_2, \ldots, x_l) \tag{3.4}$$

Where $x_1, x_2, \ldots x_l$ represent moves of the first player, $y_1, y_2, \ldots y_l$ represent moves of the second player ($2 \leq l \leq (n \times m)$ in both cases), and R is a predicate that checks whether the given set of moves constitute an instance of a Chomp game, according to the rules of Chomp. This formula is true if and only if the first player has a winning strategy. If it is false, then the second player has a winning strategy.

Further to this, it is also known that the **first** player has a winning strategy, by a "strategy - stealing" argument [11]. To show this, consider the situation where player 1's first move of the game is simply removing the top - rightmost square of the board, (n, m). This move leads to a winning strategy for one of the two players. If it leads to a winning strategy for player 1 then our claim is proven. If it does not, then the next (second) move will lead to a winning strategy for player 2. But, by the nature of the moves of Chomp, any configuration that is induced by this second move would have been reachable from the initial board configuration, and thus could have been played by the first player originally.

Equation 3.4 is also the first method of showing that computing winning strategies for Chomp is in PSPACE. By expressing the existence of a winning strategy as a QBF, we can utilize the fact that the language

$$TQBF = \{\langle\Phi\rangle | \Phi \text{ is a true fully quantified Boolean formula}\}$$

is PSPACE complete (see [14]) to show directly that winning strategies in Chomp is in PSPACE.

3.5.2 Known Winning Strategies from Initial Configurations

Although a general method for computing the winning strategy in Chomp is not known, winning strategies are known for certain initial configurations. The list below presents some examples:

1. If the Chomp board is a row (1 by m) or column (n by 1) vector, play the square adjacent to the poisoned block. Thus, if the board is a single row, play the position $(1, 2)$. If the board is a column, play $(2, 1)$. This will leave the second player with the poisoned square, making them lose.

2. If the Chomp board is made up of 2 equal - length rows, the results of Zeilberger in [17] dictate that the first player must always leave the second player with configurations of the form $(a - 1, a)$, as these are $\mathcal{P}$ positions. This can be easily accomplished by playing the square $(2, m)$ on the first turn, and playing accordingly for all remaining turns. If the second player (foolishly) reduces the grid to a single row, the first player simply follows the previously mentioned strategy to victory.

3. The strategy for Chomp boards of 2 columns of height h is analogous to the strategy for 2 rows. The first player's goal is to leave the second player in configurations where the leftmost column is 1 square taller than the right column. The first move which accomplishes this is the square $(n, 2)$.

4. If the Chomp board is square (i.e., n by n), then the strategy for the first player is as follows:

(a) Chomp the square diagonally up and to the right of the poisoned square c_p. This will leave the second player with an L-formation board.

(b) For every move the second player does, "mimic" the move on the "opposite" side of the L-square.

5. If the Chomp grid is 3 by m, a probabilistic method for selecting the winning first move is given by Friedman and Landsberg in [10]. The winning move is unique for each m, and is either to configuration $0^{m/\sqrt{2}}10^{m(2-\sqrt{2})/2}11$ or $0^{m(2-\sqrt{2})}110^{m(\sqrt{2}-1)}$, using our representation of configurations as binary strings. The probability that the winning move is to the former configuration is $\sqrt{2}-1$, the probability that it is to the latter is $2-\sqrt{2}$.

In the next section we exam intermediate Chomp configurations which have been identified as $\mathcal{N}$ or $\mathcal{P}$ positions.

3.5.3 Known Winning Strategies from Intermediate Configurations

In addition to analyzing the first player's strategies from initial configurations, seeing which player can win from intermediate configurations (that is, configurations that cannot be initial ones, but do not have only the poisoned square) can yield valuable information and insight into how Chomp "works".

1. **Symmetric L**: These are configurations consisting of a single column of height n together with a single row of length m, with $n = m$. These are $\mathcal{P}$ configurations. To see why, recall that for a square initial configuration, the first player's winning strategy was to select the square at location $(2, 2)$ (giving the second player a symmetric L configuration), then mimic the second player's movements until victory. From this strategy we see that once the second player receives the L configuration, they have lost. Figure 3.2 gives an example of a Symmetric L configuration.

2. **Asymmetric L**: These configurations have a similar structure as Symmetric L configurations, but here $n \neq m$. Consequently, these are $\mathcal{N}$ configurations. The player who receives an asymmetric L configuration first removes the exact

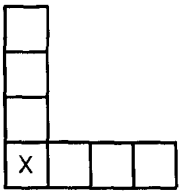

Figure 3.2: A Symmetric-L Chomp configuration

amount of squares from the longer "end" (row or column), which turns the board into a symmetric L, and thus gives the second player a $\mathcal{P}$ configuration. An example of the winning play for an Asymmetric L configuration is giving in figure 3.3.

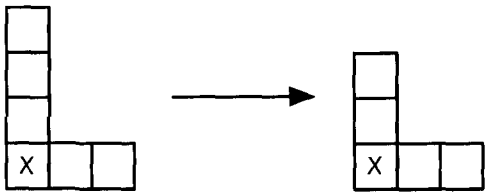

Figure 3.3: The winning play for an Asymmetric-L Chomp configuration

3. **Symmetric L with Additional Squares**: This type of configuration has $n = m$, but is not a full square grid. Nevertheless, these configurations are of the $\mathcal{N}$ type, by playing the strategy given for square configurations. Because any "extra" squares present on top of a symmetric L configuration must include the square $(2, 2)$, the player plays this square, which leaves the second player with a symmetric L configuration, and thus a losing position. Figure 3.4 gives the winning play for an example Symmetric L configuration with additional squares.

The Asymmetric L with Addition Squares configurations remain open for solving, as it is not clear whether the optimal strategy lies in selecting one of the additional squares, or a square along either side of the "L". Figure 3.5 gives an example of an Asymmetric L with Additional Squares configuration.

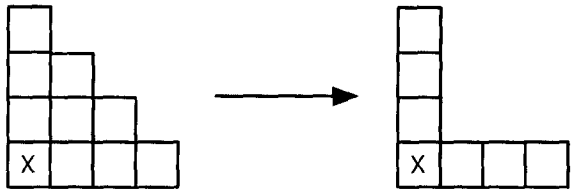

Figure 3.4: The winning move for a Symmetric-L configuration with additional squares

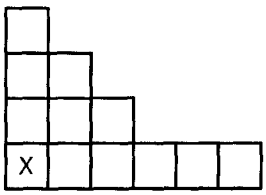

Figure 3.5: An Asymmetric L with Additional Squares Chomp configuration

3.5.4 Three-Rowed Intermediate Configurations

Zeilberger ([17]) and Sun ([16]) both examined winning positions for Chomp configurations of three rows. Zeilberger states that for configurations where the length of the third row is restricted to a single square, the only $\mathcal{P}$ positions are when the first two rows are each of length 2, or the first row is of length 3, and the second row of length 1 (Figure 3.6). Zeilberger then outlines the four forms of $\mathcal{N}$ positions:

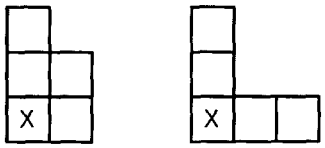

Figure 3.6: The two $\mathcal{P}$ positions for three-row Chomp where the third row is a single square

$(3,2,1), (3,3,1), (4+\alpha,1,1)$, and $(2+\alpha,2+\beta,1)$, where $(\alpha \geq \beta \geq 0)$ and $(\alpha+\beta > 0)$. These are given in Figure 3.7.

Sun extended Zeilberger's results to develop a formula which characterizes $\mathcal{P}$ positions for configurations with n rows, where the first two rows contain a and b squares

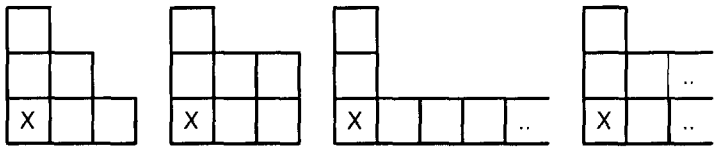

Figure 3.7: The four forms of $\mathcal{N}$ positions for three-row Chomp with third row a single square

respectively, the third row has 2 squares, and every row thereafter contains only a single square. This formula is given in equation 3.5, and gives the n corresponding to the conditions on a, b which yield a $\mathcal{P}$ position.

$$n = \begin{cases} 1 & \text{if } a = 1 \\ 2 & \text{if } a = b + 1 \\ \lfloor \frac{2a+b}{2} \rfloor & \text{if } a + b \text{ is odd, and } a \neq b + 1 \\ \lfloor \frac{3a}{2} \rfloor + 1 & \text{if } a = b \\ \min\left(\lceil \frac{2a-b}{2} \rceil, \frac{3(a-b)}{2} \right) & \text{if } a + b \text{ is even, and } a \neq b \end{cases} \tag{3.5}$$

Chapter 4

Translations between Chomp and Geography

In this section we examine translations from Chomp to the game of Geography, and vice-versa. *GG*, the language based on Geography is known to be PSPACE-complete (see Section 4.1), thus a translation from Chomp to Geography shows that Chomp is in PSPACE. Section 3.5.1 gave the first proof of Chomp $\in$ PSPACE, however this method requires the definition of a predicate R to decide whether a given set of moves constitute a legal game of Chomp, an operation which can be very expensive. In sections 4.2.1-4.2.2 we give a translation from any poset game (and thus, Chomp) to Geography which is polynomial-time (logarithmic space) in the size of the poset game. Conversely, a translation from Geography to Chomp shows winning strategies in Chomp to be PSPACE-complete. We begin by introducing the game of Geography itself, then detail the reduction from Chomp to Geography. Finally, we examine what progress has been made in translating Geography to Chomp.

4.1 The Game of Geography

Informally, the game of Geography involves two players naming cities of the world, starting with a designated first city. Each player is restricted to naming a city whose name begins with the letter that the city named by the previous player ended with. Thus, if the first player names "Toronto", then a legal move for the second player

would be "Orlando", but not "Mississauga". Also, cities that have been previously named cannot be repeated.

Mathematically, Geography can be seen as a directed graph G, with one of its nodes s specified as the starting node of the game. The first player selects this node on the first turn of the game, then for every turn thereafter (beginning with the second player), the players alternately select a node that is connected to the previous node via an incoming edge. Play ends when a player is on a node that either has no outgoing edges, or only outgoing edges that lead to previously - visited nodes. This formulation of Geography is known as Generalized Geography (GenGeo, see [13], [14]), and corresponds to the following language:

$GG = \{\langle (G, s) \rangle : \text{player 1 has a winning strategy for the GenGeo game on } G \text{ starting at } s\}$

It has been shown that GG is PSPACE-complete, with both [14] and [13] detailing proofs of this fact. Section 4.2 uses this fact to give a logspace translation from poset games to Geography, thus showing that computing winning strategies for poset games (and thus Chomp) is in PSPACE.

4.2 Reducing Chomp to Geography

In this section we show how to reduce Chomp to the game of Generalized Geography (GG) presented in Section 4.1. The reduction is actually larger in scope, as it reduces any poset game to an instance of Geography, thus showing that computing winning strategies in poset games is in PSPACE.

For the reduction itself, we let

$$POSETGAMES = \{\langle (U, \prec) \rangle : \text{player 1 has a winning strategy}\}$$

i.e. the language of encodings of Poset Games where the first player wins In order to transform $POSETGAMES$ into GG, we will describe a function f such that $\langle (U, \preceq) \rangle \overset{f}{\mapsto} \langle (G = (V, E), s) \rangle$. We define $\preceq$ as a list of pairs, and note that this list has length at most $O(|U|^2 \log(|U|))$, as each element of U can be represented in its binary form ($\log(|U|)$), and each element may be paired with every other element in U ($|U|^2$).

4.2.1 The Base Construction

To construct the graph G from $(U, \preceq)$, for every element x of U we add a node x to V. For every pair of elements $x, y \in U$ such that $x \preceq y$, we add the directed edge (y, x) in E. For every pair of elements $x, z \in U$ such that $x||z$, we add the edges (x, z) and (z, x). If the source poset game has a supremum, then the node with no incoming edges is the starting node s. For poset games without supremums (i.e., multiple largest elements), we create an additional node s and add outgoing edges from s to each of the "largest" elements. We call this the base construction of G. While this forms the basic groundwork of G, we also need a method to ensure that once a node x has been visited in the graph, all other nodes y such that $x \preceq y$ in the poset game can no longer be visited. To accomplish this, we link each node x in V with 4 other nodes x_1, x_2, x_3, x_4. These auxiliary nodes allow a player to challenge the other's move, if they suspect that the player selected a node y on their turn such that $\exists x : x \preceq y$, and node x has already been visited. As such a scenario cannot occur in the poset game, it should not occur in our translated Geography game. Section 4.2.2 describes the structure of each 5 - part node in detail, and shows that not only do players gain nothing from making illegal moves, but they also make no gains by making false challenges, that is challenges to legal moves.

4.2.2 Construction of the 5-node gadgets of G

Figure 4.1 shows the structure for a given node x of G, along with its 4 auxiliary nodes. As the figure shows, for every incoming edge (y, x) that was present between comparable elements in the base construction of G, we add the edge (y, x_1). Nodes z such that $x||z$ only have the edge (z, x). For every outgoing edge (x, z) in the base construction, we now have (x_3, z) - this applies to nodes comparable and incomparable to x. Finally, we add the edges $(x_2, x_3), (x_2, x_4), (x, x_2), (x_4, x_1)$, and (x_1, x). Additionally, we add two more nodes s and s' to G, along with the edges (s, s'), and (s', x), for all x in the base construction of G. These two nodes are used to emulate the "open-endedness" of the first move in a poset game, where the first player can select any of the elements of U. In a game of Geography the first player must select the designated first node s, so in order to ensure that the first player has free selection of all the nodes corresponding to elements of U, we force the second player to

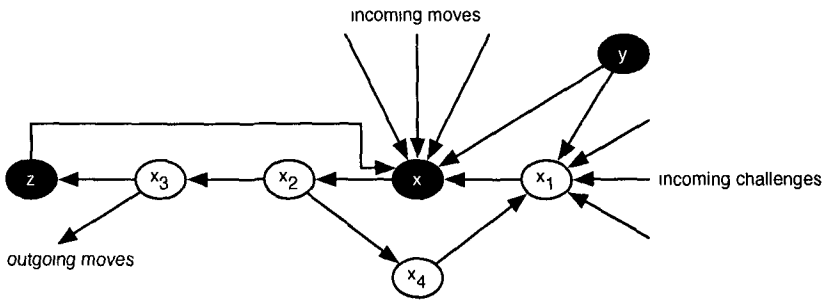

Figure 4.1: The 5 - node gadget in G, representing an $x \in U$

select the node s' on their first move, then player 1 can select from any $x \in U$ via s'. Finally, we note that this translation can be carried out in logspace. Section 4.2.3 details how this construction forces players to play correctly, by showing that neither player makes gains by playing an illegal move.

4.2.3 Analysis of Challenges

Having detailed the 5-node gadgets of G, we now examine the various ways the players can challenge each other's moves in the Geography game, and show that neither play can make gains by making an illegal move, or challenging a legal move (false challenge).

1. **Challenging an illegal move:** Suppose player 1 moves to a node y in G, but there is a node x such that $x \preceq y$ and x has already been visited on a previous turn of the game. This would be an invalid move in the associated poset game (as $y \notin A$), so player 2 will challenge the move to exposure player 2's treachery. To do this, player 2 moves from node y to the auxiliary node x_1 associated with x. Now, player 2's only move is from x_1 to x, which causes them to lose the game, as x has already been played.

2. **False Challenge:** Suppose player 1 makes a legal move to node y in G, but player 2 thinks they can cause player 1 to lose by challenging the move. Thus, player 2 moves from y to x_1, the auxiliary node of some $x \in G$, where $x \preceq y$ but x has not been previously played. From here player 1 moves from x_1 to x, then player 2 from x to x_2. At this point player 1 can show player 2's false challenge,

by moving from x_2 to x_4. This forces player 2's next move to be from x_4 back to x_1, which causes player 2 to lose the game.

3. **False Counter - Challenge:** This situation can be described as a player challenging their own move, in the hopes of causing the other to lose. Suppose player 1 moves directly to node y (i.e., not through the auxiliary node y_1). Then player 2 makes their move to y_2, but instead of moving to node y_3, player 1 moves to node y_4. This forces player 2 to move to node y_1, but now player 2's only move is to revisit y, which causes them to lose. This also shows that a player challenging a move to an incomparable node yields no benefit.

Thus, our construction not only ensures that illegal moves result in the offending player losing the game, but that there is no incentive for a player to falsely challenge a legal move. These facts prove the following theorem and corollary:

Theorem 8. *There is a logspace reduction from poset games to the game of Geography.*

Corollary 9. *Computing winning strategies for poset games is in PSPACE.*

4.3 Reductions from Geography to Poset Games

In Section 4.2 we showed that poset games can be reduced to instances of Geography. But what about the other direction? If we could show the existence of a logspace reduction from Geography to poset games, this would show poset games to be PSPACE-complete. This result may yield further insight into the classification of restricted poset games, such as Chomp and Subset-Sum. On the other hand, if such a reduction highlights limitations in the power of Chomp, we may gain some insight into why Chomp may not be PSPACE-complete.

In the following sections we begin tackling the idea of such a reduction by first introducing a restricted version of Geography (Treeography) played on tree - like graphs, then present a simple poset games translation. Properties of this translation are then analyzed, beginning with simple tree structures and moving to forests of larger trees.

4.4 Treeography

In this section we introduce a restricted version of the Generalized Geography game presented in Section 4.1, called Treeography. In Treeography, the graph G must be tree - like, meaning it is connected and acyclic, in addition to being directed. The "root" of G is defined to be the node with no incoming edges, while a "leaf" of G is defined to be a node with no outgoing edges. The root of the tree is the starting node s of the Treeography game. We define the height h of a tree to be the number of nodes on the longest path from the root to a leaf, and define a level ℓ of the tree to be the set of nodes ℓ edges away from the root, with the root at level 0. Figure 4.2 shows a sample G with height 4. This tree structure allows for interesting analysis to

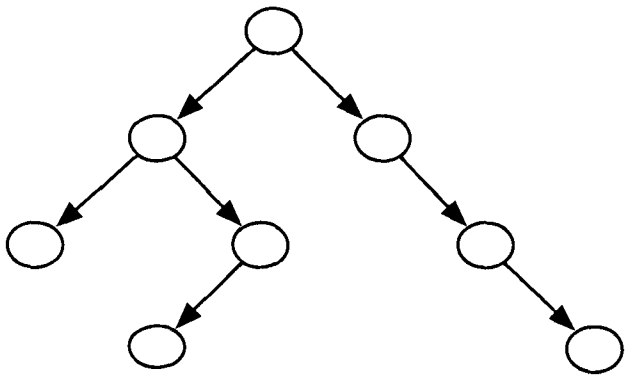

Figure 4.2: A sample tree - like graph G

be conducted. Namely, it allows the use of the following algorithm to determine the winner of the game from the initial position:

1. Label each of the leaves of G with P, indicating that they are winning positions for the previous player.

2. For every remaining node i of G, recursively do the following:

   (a) If **at least one** of i's children is a P node, label i as an N node.

   (b) If **all** of i's children are N nodes, label i as a P node.

3. If, after all nodes have been labeled, the root of the tree is a P node, the first player has a winning strategy. An N root corresponds to the analogous situation for the second player.

Using this algorithm, we can prove the following Lemma:

Lemma 10. *The first player has a winning strategy in games of Treeography played on balanced binary trees iff n is* ***odd****.*

Proof. From the first step of our algorithm, we have all of the leaves of the tree labeled as P nodes. The nodes on the next level up in the tree are each connected to P nodes only, so they become N nodes. The level above that has nodes that are all connected to only N nodes, so they become P - labeled. This alternation of labeling continues for the rest of the nodes of the tree.

Because the labels of the tree alternate (starting with P at the bottom), it follows directly that an instance of Treeography with height n being **odd** results in a winning strategy for the first player, while an **even** n allows the second player to win. □

4.5 Initial Translation from Treeography to Poset Games

In this section we present a translation from Treeography to a Poset Game where the first player always has a winning strategy. Our translation is as follows: we first use a breadth - first search to assign a number to each of the nodes in the tree, with the root being node 1. Figure 4.3 illustrates an example of this for an instance of Treeography with $n = 3$. Next, we generate our set A by creating a subset for each node of the original tree. These subsets contain the numeric identifier of the node, along with the identifiers of all nodes in its subtree. Thus, a tree of height 2 would have its root be represented in the post game with the set $\{1, 2, 3\}$. Figure 4.4 shows the tree from figure 4.3 with the corresponding elements of A adjacent to each node. The second part of our poset formulation requires a partial ordering on the elements of A. we define our ordering $\preceq$ as follows:

$$x \preceq y \text{ iff } x \subseteq y$$

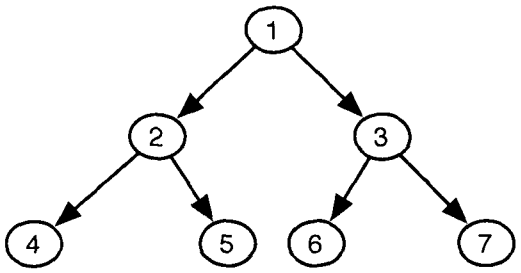

Figure 4.3: The labeling algorithm applied to a binary tree with $n = 3$

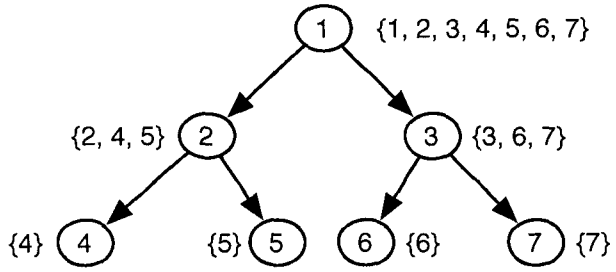

Figure 4.4: A 3-level binary tree with corresponding subsets for the poset game.

Then, when a player selects an $x \in A$, we remove all other elements $y \in A$ such that $x \subseteq y$. Because a poset game is played on a collection of sets rather than a directed graph, there is no prescribed first element that must be chosen. Figure 4.5 illustrates the result of selecting $\{7\}$ as the first move of the game - that is, elements $\{7\}, \{3, 6, 7\}$ and $\{1, 2, 3, 4, 5, 6, 7\}$ are all removed from the set A. However, as the following result shows, the winning strategy for the poset game involves making the same move as in the original Treeography instance.

Lemma 11. *In the poset version of Treeography played on a balanced binary tree the first player has a winning strategy, irrespective of the height of the source tree.*

Proof. We first describe the strategy employed by the first player, then prove that it always works. The strategy of the first player follows these steps:

1. Select the element of A corresponding to the root of the original tree.

2. For every move of the second player, make the corresponding move on the opposite subtree, i.e., "mirror" the second player's move. Here opposite refers

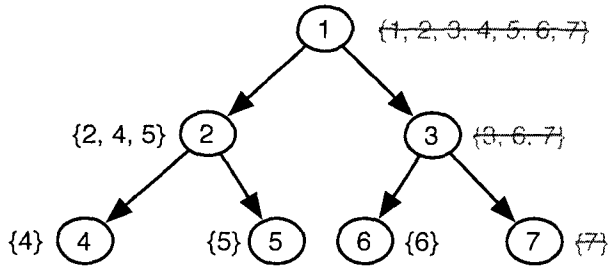

Figure 4.5: The result of selecting $\{7\}$ as the first move of the post game - all elements marked in red are removed from the set A.

to the two subtrees which are rooted at the children of the root node of the entire tree. Figure 4.6 illustrates this division explicitly.

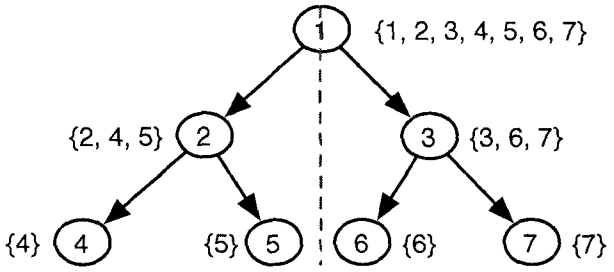

Figure 4.6: The tree of figure 4.4, with a line marking the two main subtrees.

The thing to prove now is that at any point in the game, the first player will be able to mirror the move of the second player on the opposite subtree. To do this we will use the Least Number Principle to show that the instance where player 1 cannot mirror the move of the second player cannot occur. Before doing so, however, we define a **round** i, $1 \leq i \leq (2^n - 1)$ of the game is a sequence of two consecutive moves, beginning with a move of the second player. As the first move of our strategy involves the first player selecting the $x \in A$ which corresponds to the root of the original tree, we designate a special round, round 0 to contain this move.

Assume that for all rounds j, $(j < i)$ player 1 has been following the "mirroring" strategy. Consider the instance where, in round i of the poset game, player 1 is unable to mirror the move just made by player 2. Let y be the element of A played by player

2. This means that the element $z \in A$ player 1 needed to play no longer exists, which in turn means that either player 1 or player 2 played z or an element corresponding to a node in z's subtree in a previous round j. By the Least Number Principle, if this were to occur, there must have been the first instance of it occurring. Consider the first round where this occurred, round i_0. As stated before, either player 1 or 2 must have played element z (or an element in its subtree) for it not to exist in round i_0. We analyze each of the cases separately.

1. If player 1 caused z to not be present in round i_0, this can happen in two ways:

   (a) Suppose player 1 played element z prior to round i_0. But, as we assumed that player 1 had been following the "mirroring" strategy in all the rounds prior to i_0, this means that player 2 must have played the opposite element y in that same round. But then this results in a contradiction, as player 2 played y at the beginning of the current round i_0.

   (b) Suppose player 1 played an element of A corresponding to a node in the subtree rooted at z, prior to round i_0. This eliminates z from play. However, by our assumption this means that the move was the mirror of one made by player 2. As z is the mirror node of y, this means that if player 1 played an element in z's subtree, player 2 played an element in y's subtree, which would have eliminated y from play. But this is a contradiction, as player 2's most recent move was assumed to be y.

2. Analogous to the situation for player 1, player 2 can cause z to be absent in two ways:

   (a) Suppose player 2 played z in an earlier round. This also results in a contradiction though, as then by our original assumption player 1 would have played y in response (to mirror player 2's action), but we have player 2 selecting y in the current round.

   (b) If player 2 played an element in z's subtree, this would also have eliminated z from play. However, by our original assumption player 1 would have countered this move by playing an element from the mirror subtree, that is the subtree of y, which would have also eliminated y from play, resulting in a contradiction.

Thus, as there is no way a first instance of player 1 being unable to mirror player 2's move can occur, by the Least Number Principle it cannot occur at all, meaning player 1 will always be able to mirror the move of the second player in each round.

As we have shown that player 1 can always mirror a move of the second player, and begins the game by eliminating the element of A corresponding to the root of the original tree, we have the invariant that, after each round i of the game, each subtree is a mirror image of the other. This is easy to see - as shown above, not only will player 1 always be able to mirror the move of player 2, but no other "intermediate" moves can be made that would cause the structures of the two trees to be different. If that were to happen, then at some point player 1 would be unable to mirror a move, which we showed above to be impossible.

The consequence of this invariant is at the end of each round we are left with an even number of nodes. To see this, recall that any balanced binary tree of height n has $(2^n - 1)$ - many nodes, which is an odd number. After round 0 we are left with an even number, as player 1 removes only the "root" element. For each round thereafter, player 2 eliminates some number of elements k on their turn, while player 1 eliminates the same number with the mirror move, so the total number of elements removed is thus $2k$, which is even. Thus, eventually a round will begin with $|A| = 0$, meaning player 2 loses, as they go first in each round. If the number of nodes left was always odd, on the other hand, eventually there would be a round that began with $|A| = 1$, so player 2 would select the last element, and player 1 would lose. □

The next logical step is to extend this result for balanced trees of uniform degree > 2. Fortunately, this is not too difficult. The following theorem summarizes the results for the poset translation on any balanced binary tree.

Theorem 12. *In the translation of Treeography to a poset game given in section 4.5, the first player of the poset game has a winning strategy, regardless of the degree or height of the original tree.*

Proof. We divide the proof into two cases, based on whether the degree d of the original tree is **even** or **odd**.

1. If d is **even**, then we can simply extend the result shown in lemma 11. For an even degree we are guaranteed to have an odd number of elements N in the

poset game, as each level of the tree except for the root has an even number of elements (due to the even degree), and the addition of the root node makes N odd. The first player's initial move is still the $x \in A$ which corresponds to the root of the original tree, and then to mirror every subsequent move of the second player, until victory.

2. If d is **odd**, then the structure of the original tree causes problems for mirroring. The original tree will have a "centerline" path from the root to a leaf - that is, a series of nodes that lie on the axis of symmetry for the tree. Figure 4.7 illustrates this for $d = 3$ and height $h = 3$. Thus, the first player can no longer simply play

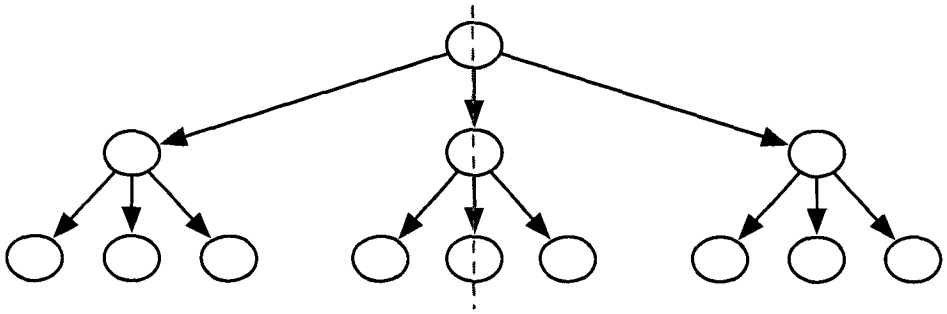

Figure 4.7: Illustrating the centerline path of nodes for a tree with $d = 3$ and $h = 3$

the "root" element of A on their turn, as this leaves the tree in an un-mirror-able state and with a possibly odd number of nodes, which is beneficial for the second player. Fortunately, it turns out that by playing the leaf element on the centerline path, the first player can always put the poset game in a mirror-able state, with an even number of elements for the second player. The first part of this claim is easy to see, as if the centerline is eliminated that the nodes of the original tree have been divided into two disjoint but symmetric subsets. The second part of the statement requires more justification, however.

Lemma 13. *Selecting the leaf node of the centerline path in a balanced tree of height h and **odd** degree d results in a tree with an even number of nodes.*

Proof. We know that for any balanced tree with height h and uniform degree

d, the formula for the number of nodes N in the tree is

$$N = d^{(h-1)} + d^{(h-2)} + \ldots + d + 1, \text{for } h \geq 1$$

Note that this means all of the d^i terms will be odd, as this is just repeated multiplication of odd terms. Then we have $h - 1$ odd terms to add together, along with the 1 term for the root of the tree, for a total of h odd terms summed together. To see whether N will be even or odd in this case, we turn to the rules for parity arithmetic and multiplication. From these we can see that if we have an odd number of (odd) terms to sum, the result will be odd, whereas an even number of terms will generate an even result. Thus, suppose the height h of our tree is **odd**. Then we have an odd number of odd terms, resulting in N being odd. The first player's move is to select the element of A corresponding to the leaf of the centerline path, which removes h elements from $|A| = N$, and thus leaves an even number of mirror-able elements for the second player. Similarly, if h is even, then so will be N, and thus $N - h$ will again leave player 2 with an even number of elements in a mirror-able configuration, ensuring victory for the first player. □

As the first player has a winning strategy in both cases for the parity of the degree of the original tree, we can conclude that the poset translation results in a winning strategy for the first player on balanced trees of uniform degree. □

4.6 Treeography on Forests of Balanced Binary Trees

Having shown that the first player of poset Treeography has a winning strategy on any balanced tree of uniform degree, we now turn our attention to forests of balanced, uniform degree binary trees. The following subsections detail results for forests with maximum height 1 and 2, then explorations into forests with maximum height ≥ 3 are presented.

4.6.1 Forests of Height 1

If the maximum height of the trees in the forest is 1, then determining who wins reduces to the parity of the number of trees. As all of the trees consist of a single node and are thus incomparable with respect to the ordering of the poset game, the game consists of each player selecting a single element and removing it from the set A of the game. Thus, if the number of trees is **odd**, then the first player wins. Having an **even** number of trees results in a winning situation for the second player.

4.6.2 Forests of Height 2

If our forest contains trees with two levels, then the winning condition depends first on the number of height 2 trees (n_2). The first player has a winning strategy if n_2 is odd, regardless of the number of height 1 trees n_1. When n_2 is even, a winning strategy exists for the first player if the number of height 1 trees n_1 is **odd**. The first player's general strategy is to play moves such that they are able to play on the last remaining height 2 tree. By doing this, the first player is able to ensure that n_1 is even on the next turn of the second player, thus putting player 2 in a losing position. Thus, similar to the case for forests of maximum height 1, any configuration of a forest with height 2 with n_2 odd is a winning position for player 1. However, because of the presence of trees of height 1, the game is not as simple as each player alternately selecting nodes of height 2 trees. As mentioned previously, a configuration with n_1 odd and n_2 even is a winning one for player 1. Thus, when presented with an odd number of height 2 trees, the first player cannot simply select any node of a height 2 tree to player.

To show the strategy the first player must employ, we start from the base conditions of $n_2 = 1$, and $n_2 = 2$, to arrive at base cases for winning and losing positions.

1. $n_2 = 1$: As stated previously, this configuration is a winning position for player 1. With a tree of height 2, player 1 can choose whether to create one new tree of height 1, by playing a leaf, or two new trees, by playing the root. This freedom means that no matter the value of n_1, player 1 can force n_1 to be even after their move, and thus put the opposing player in a losing configuration.

2. $n_2 = 2, n_1 =$ **odd**: This is also a winning configuration for player 1. The odd

number of trees of height 1 allow the player to force the other into selecting one of the height 2 trees on the opposing player's turn, thus leaving player 1 with one height 2 tree, and thus a winning configuration. In order to do this, player 1 selects a height 1 tree, making n_1 even. Now, the opposing player can either select one of the two height 2 trees, or select another height 1 tree. If they select a height 2 tree, then the game is put into a winning configuration for player 1, and our objective is reached.

If the opposing player selects one of the height 1 trees, then the first player can simply keep selecting trees of height 1 until the other player is forced to select one of the two height 2 trees. The first player can do this because n_1 is guaranteed to be odd on their turn, if the second player's move was a height 1 tree. Thus, either the opposing player will select a height 2 tree on their own, or will be left with $n_1 = 0$ on their turn, and be forced to play a height 2 tree.

3. $n_2 = 2, n_1 =$ **even**: This is a losing configuration for player 1. As was shown in point 2 above, player 1 can only win when n_2 is even if n_1 is odd, as it allows the first player to keep playing height 1 trees, with the knowledge that eventually the second player will have $n_1 = 0$ on their turn, and be forced to play one of the two height 2 trees (if they haven't already). With n_1 even, however, following this strategy will result in $n_1 = 0$ on player 1's turn, meaning player 2 will be placed in a configuration with $n_2 = 1$, and thus win the game.

We can take this three situations and apply the winning conditions to any values of n_1 and n_2, with one slight modification. We showed that $n_2 = 2$ (meaning even) and n_1 odd was a winning configuration for player 1. Thus, when playing on a configuration with n_2 odd, player 1 must ensure that the move they make does not leave n_1 odd, as then the opposing player is left in a winning situation. The description of the situation for $n_2 = 1$ details how to accomplish this, and maintains the winning condition for player 1 when playing on an odd number of height 2 trees.

4.6.3 Forests of Height 3 and Beyond

As was shown in the previous section, the first player's strategy for winning on a forest with trees of different heights is to play the last remaining tree for each height

in the forest, beginning with the tallest subset of trees. It was also shown that there is one configuration on which the first player cannot win. For forests of height 3 and higher these general rules (and the strategies shown previously) seem to apply, however with more types of trees to account for, more complications arise.

The first complication affects the so - far cardinal rule in player 1's strategy, that being always aim to play the last remaining tree of the tallest (remaining) height in the forest. Suppose, however that on player 1's turn the configuration of the game is $n_3 = 1$, n_2 even, and n_1 odd. From the previous section we know that n_2 even, n_1 odd is a winning configuration for the next player. Thus, player 1 must create either zero or two new trees of height 2, and one tree of height 1 to put the opposing player in a losing configuration. From looking at the possible moves on a tree of height 3, however, we see that no such moves are possible, and such a configuration is then a losing one for the next player. This result has a ripple effect - which configurations with n_3 even are now winning configurations for the next player, in that they lead to the previous example?

The second complication arises in the case when n_3 is even. For forests of height 2, the first player could use a strategy of "waiting - out" the opposing player by selecting nodes of height 1, if the initial number n_1 was odd. For forests of height 3, having trees of height 2 further complicates matters, as moves on these trees create trees of height 1, and the player who makes such a move can select how many trees are created. With these complications it may result that either player has no definitive winning strategy from a given configuration, or that the scheme of always playing on the tallest subset of trees is no longer valid.

4.7 General Implications to Poset Games

As has been shown, translating Geography into poset games requires great restrictions in order to conduct analysis. Even then, once we allow trees of height 3 in the forest, determining the winning player becomes difficult. These attempts at reducing Geography to poset games have highlighted a significant difference between Geography and poset games, though, which may yield weight to the idea that poset games are not PSPACE-complete.

More specifically, in Geography one is able to restrict the moves available to a

player on each turn, by only allowing players to move to nodes connected by a single edge. This ability is not present in poset games, as a player can select any element from the poset on their turn. Thus, emulating the strict ordering present in Geography becomes the main stumbling block. During the course of the author's research attempts were made at modifying general poset games to introduce a mechanism to emulate strict ordering, but these introduced a new problem of showing the relationship between the modified version of poset games and the general version. It is the lack of expressing strict ordering that leads the author to suspect that poset games are not PSPACE-complete. Note that in Geography we can emulate the free choice that exists in poset games, the translation in sections 4.2.1 and 4.2.2 demonstrates this. Thus, the game of Geography appears to have more "power" than poset games, which again leads the author to believe that poset games are not PSPACE-complete.

Chapter 5

Translating General Poset Games to Chomp

Given the difficulties in reducing Geography to Chomp, we now try a different approach in attempting to further understand the properties of Chomp. We introduce the concept of **Game-Completeness**, and attempt to show that Chomp is game-complete for class of poset games. To begin, we give the definition for Game-Completeness:

Definition 14. *A game A is **Game-Complete** for a class of games GC iff every game $B \in GC$ can be translated into an instance of A in polynomial time (logarithmic space), and this translation preserves the winning player from B.*

In the following sections we will give a translation from any poset game to Chomp, and show that this translation preserves the winning player from the original poset game.

5.1 Basic Overview

In translating a poset game into an instance of the Chomp game, the most basic issue we must be concerned with is preserving the structure of the poset. That is, ensuring that elements which are (in)comparable in the poset remain (in)comparable in Chomp. Translating the rules for eliminating squares on a Chomp grid in terms

of posets, a square a is greater than another square b if and only if a lies either above b, to the right of b, or both. If a lies above and **to the left** of b, a and b are incomparable.

5.2 Representing Posets as Graphs

Having identified the basic principle behind the translation, we now note that a poset can be given as a directed, acyclic graph (DAG), where the nodes of the graph are the elements of the poset, and the directions of the edges represent the relation $\preceq$. For this document we use the convention that in the DAG representation of a poset $(P, \preceq)$, there is a directed edge from node a to b (where $a, b \in P$) if and only if $a \preceq b$. This also captures the transitivity of posts. If there is an edge from node a to node b in the DAG, as well as an edge from node b to node c, then there must also exist an edge from node a to node c, meaning $a \preceq c$. Thus, the problem becomes transforming a DAG into an instance of Chomp. For the moment we will restrict our focus to those posets whose ordering relation $\preceq$ can be described by a tree. Figure 5.1 gives an example of a poset in tree form. Note that for the sake of clarity we do not draw the edges representing transitive connections (such as between nodes 1 and 5). These are implicitly assumed, as described previously.

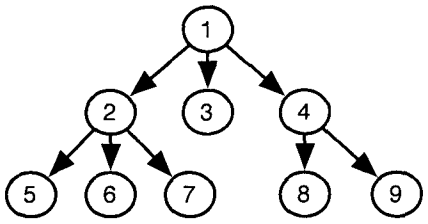

Figure 5.1: The DAG representation of poset $(C, \preceq)$

5.3 Definitions of Inscription and Labeled Chomp

Now that we have given a high-level overview of our algorithm, we present the two stages of the translation from a tree T to a configuration C of a new type of Chomp,

Labeled Chomp. Figure 5.2 gives an example of a Labeled Chomp configuration, obtained from the poset given in Figure 5.1.

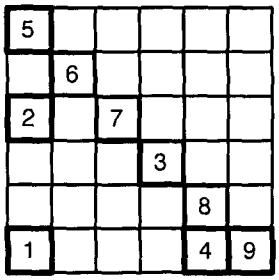

Figure 5.2: The Labeled Chomp representation of the poset given in Figure 5.1

From this example we define exactly what we mean by "Labeled Chomp". Intuitively, configurations of Labeled Chomp are Chomp configurations where the only moves available to the players are on nodes with labels, or the poisoned square. All other positions in the grid cannot be selected. The conditions for termination are unchanged - the player who is (forced) to select the poisoned square loses. Looking again at Figure 5.2, only squares 1 through 9 can be selected by the players, all other squares are out of play. Also note that this configuration of Chomp preserves the (in)comparabilities between elements in the source tree. Finally, for the sake of simplicity whenever we say "Chomp" in the remainder of the document (until Section 6) we mean "Labeled Chomp".

In addition to defining the type of Chomp configurations we will be generating, we also must define exactly what it means to translate the tree representation of a poset game into a configuration of Chomp. The following definition formalizes the high-level description given in section 5.1:

Definition 15. *A tree T is translated or* ***inscribed*** *by a function f into a Chomp configuration C if the following properties hold (assume i, j, k are nodes of T):*

1. If i is an ancestor of j, then $f(i) \leq f(j)$.

2. If i is not an ancestor of j and j is not an ancestor of i, then neither $f(i) \leq f(j)$, or $f(j) \leq f(i)$.

Now we need an algorithm which adheres to the above definition. We first present an algorithm that is simple to describe, however not the most efficient in terms of the size of Chomp grid required. A second, incomplete algorithm is presented as an appendix. This algorithm is more difficult to describe (and also incomplete), yet is aimed at being more efficient in using more compact Chomp grids. For now, however, Section 5.4 details our first inscription algorithm.

5.4 First Inscription Algorithm

Our first inscription algorithm is based on a simple iterative approach of constructing an initial Chomp grid, then continually subdividing the grid until no nodes from the source tree remain. We describe the algorithm in further detail below, and use the tree given in Figure 5.1 as our source tree.

Algorithm 16. *Algorithm for inscribing a poset in tree form into a Chomp grid.*

1. Obtain the largest out-degree in the graph, call it o_degree. Construct an initial grid of size o_degree^h by o_degree^h, with h being the height of the source tree. Place the root of the source tree in the bottom-left square of the grid. Figure 5.3 gives the initial grid for our source tree of height 3, and with o_degree = 3.

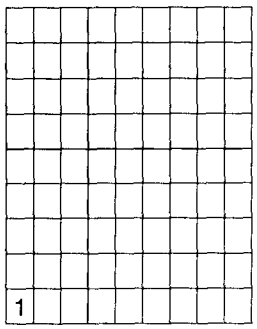

Figure 5.3: Initial Chomp grid for source tree in Figure 5.1

2. Subdivide the grid vertically and horizontally into *o_degree*-many sections. Figure 5.4 gives the subdivided grid for source tree.

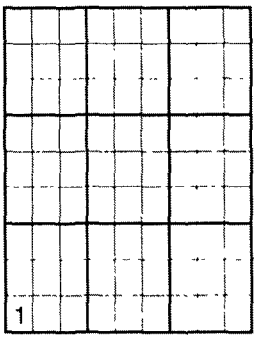

Figure 5.4: Subdivided Chomp grid for source tree in Figure 5.1

3. Place the children of the root in the bottom-left square of each section along the main diagonal of the subdivided grid, thus putting each subtree from the root in its own section. Figure 5.5 shows the result for our example.

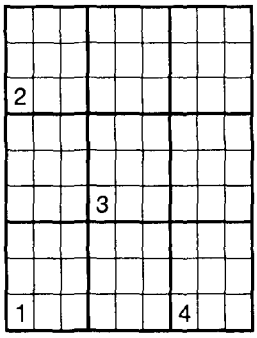

Figure 5.5: Placing children of the root node into the subdivided grid

4. Repeat the previous two steps in each section of the grid. As our source tree is of height 3, each subsection now contains only 1 square. Placing the children of each node from the previous step along each section's main diagonal, we obtain the grid in Figure 5.6.

5. Continue repeating step 4 until all nodes in the source tree have been place in the grid. As our example is a tree of height 3 we have no more nodes to place.

6. Once all nodes have been placed, add an additional L-shaped set of squares to

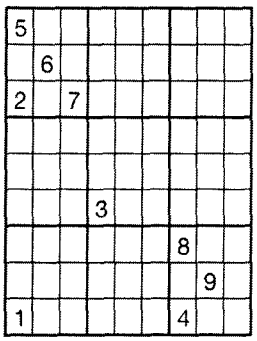

Figure 5.6: Further subdividing the grid for the next level of nodes in the source tree

the left of the grid, with the bottom-left square in this L marked with an "X" to denote it as the poisoned square for the Chomp game. Figure 5.7 shows this step applied to our example.

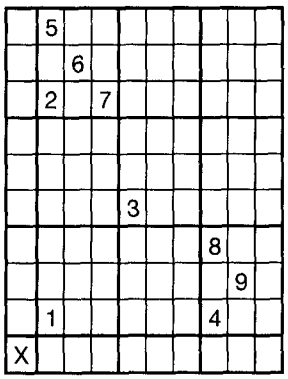

Figure 5.7: Adding the L-shaped set of squares to Figure 5.6

The L-shaped set of squares is necessary to have our Chomp board correspond to the rules of the original poset game, where the player who selects the last remaining node wins. This conflicts with the rules of Chomp, where the player who selects the last remaining node (presumably the poisoned square) loses. Adding the extra set of squares does not change who wins the original poset game in this case, as the first player has a winning strategy for every tree-like poset games (select the root of the tree on the first move of the game).

As can be seen by looking at the end result in Figure 5.7 this algorithm is inefficient with regards to the size of the grid. It does not take into account nodes without children, or nodes with less children than others. Referring back to our example, as node 3 in the source tree has no children, it does not require a full section to itself, only a single square. The algorithm attached as an appendix to this document aims to be more efficient with space usage.

Chapter 6

Chomp and Proof Complexity

In this last set of sections we present both our final proof of the computation of winning strategies in Chomp being in PSPACE, and a method to extract the algorithm to compute the winning strategy from this proof. We do this using the bounded arithmetic theory $\mathbf{W}_1^1$ developed by Skelley in [15]. As with our translation from Chomp to Geography in Section 4.2, we will see that the proof we present is applicable to not only Chomp, but any poset game with a unique largest element (supremum). $\mathbf{W}_1^1$ is a three-sorted theory for reasoning on sets of strings, which allows us to develop and prove properties of formulas that quantify over configurations of Chomp, which can be represented as binary strings (see proof of Lemma 4). Using the theorems developed by Skelley we can once again show that computing winning strategies in Chomp is in PSPACE, and hope that by doing so in $\mathbf{W}_1^1$ methods of formalizing the existence of a winning strategy in weaker formulas can be developed, which will lead to lower complexity algorithms for computing the winning strategy in Chomp.

Before introducing $\mathbf{W}_1^1$, however, we will give an overview of the question we wish to answer, and how bounded arithmetic will be used to help in finding a solution. We also introduce the set of second-sorted theories $\mathbf{V}^i$, which allows us to detail concepts central to bounded arithmetic, including Comprehension Axioms and classes of formulas, both of which apply to $\mathbf{W}_1^1$.

6.1 Introduction to Proof Complexity and Bounded Arithmetic

As shown in sections 3.5 and 3.5.1, the definition of a winning strategy and question of the existence of one in Chomp can be formulated as Quantified Boolean Formulas (QBFs). What else can we use these formulas for? As Buss asks in [5], does the existence of a proof of $(\forall x)(\exists y)A(x,y)$, a simple QBF, imply the existence of a feasible (i.e. polynomial time) algorithm f for computing y from x? Using Bounded Arithmetic, it is possible to not only show that the implication is true, but to give the algorithm computing f, and say exactly the computational complexity of f. We do this by placing conditions on the formula A, and by specifying the formal theory that $(\forall x)(\exists y)A(x,y)$ must be provable in.

6.1.1 Review of Quantification

The formulas we will be constructing make use of the usual quantifiers $\forall$ and $\exists$. Recall that an instance of a variable x in formula A is called bound if and only if it is quantified over by an external quantifier, and occurs in the scope of the quantifier. Otherwise we say that x occurs free in A. As an example, in the formula $\forall x(x \vee y)$, x is bound by the $\forall$ quantifier, while y is free.

As the name implies, bounded arithmetic places limits on the values of bound variables in formulas. Quantifiers of the form $(\forall x)$ and $(\exists x)$ are called unbounded quantifiers, which quantifiers of the form $(\forall x \leq t)$ and $(\exists x \leq t)$ are bounded. Note that t does not contain any occurrences of x [5]. The semantics of $(\forall x \leq t)$ are $(\forall x)(x \leq t \rightarrow A)$, while $(\exists x \leq t)$ means $(\exists x)(x \leq t \wedge A)$. A formula A is bounded if and only if it contains no unbounded quantifiers. Now that we have defined the types of quantification we allow in our formulas, we can describe the types of variables used in the formulas (Section 6.1.2), and classify them into a hierarchy, based on the number of alternations of bounded quantifiers (Section 6.1.3).

6.1.2 The Different "Sorts" of Variables

Theories of Bounded Arithmetic may involve different types of variables called sorts. Intuitively, the hierarchy of sorts progresses as follows: the first sort consists of indi-

vidual elements (numbers, characters, etc.), while every sort above the first consists of sets of elements of the previous sort. Thus, elements of the second sort are sets of individual elements, elements of the third sort are sets of sets of individual elements, etc.

The labels for free variables from each of the three sorts are as follows, with the bound variables are given in parenthesis : $a, b, c, \ldots (x, y, z, \ldots)$ for the first sort (natural numbers), $A, B, C, \ldots (X, Y, Z, \ldots)$ for the second sort (finite sets of natural numbers), and $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots (\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \ldots)$ for the third sort (sets of sets of natural numbers). From first to third sort, variables from each sort are known as number variables, set variables, and string variables, respectively [15]. The theory $\mathbf{V^i}$ contains formulas utilizing variables from the first two sorts, while $\mathbf{W_1^1}$ involves formulas with variables from all three sorts.

6.1.3 The Hierarchy of Formulas Σ_i

Having introduced the types of variables we can use in formulas, we can now organize the formulas used in Bounded Arithmetic into a single hierarchy, based on the number of alternations of bounded quantifiers the formula contains. For the purposes of this paper we will be concerned with the formula classes Σ_i^b, Σ_i^B, and $\Sigma_i^{\mathcal{B}}$. In each of these classes i indicates the number of alternations of bounded quantifiers, while the character in superscript details the level of variable sort which can be quantified over: b for variables of the first sort, B for second-sort variables, and $\mathcal{B}$ for variables of the third sort. With each of these classes, only bounded quantifiers over the highest sort are counted in the calculation of i [15]. Thus, a Σ_0^B formula such as

$$(\forall x)\,(\exists y)\,A\,(x, y) \tag{6.1}$$

has no quantifiers over variables representing sets, but may have any number of quantifiers over first-sort variables, while a $\Sigma_1^{\mathcal{B}}$ formula such as

$$(\exists \mathcal{X})\,(\forall Y)\,(\exists Z)\ldots(\forall y)\,A\,(\mathcal{X}, Y, Z, \ldots, y) \tag{6.2}$$

has one quantifier over a third-sort variable, and an arbitrary number of first and second-sorted quantifiers. Also note that classes can have more free variables of the largest sort than alternations of quantifiers. Looking again at equation 6.1, the

formula A could have free variables of the second sort, even though there are no second-sorted quantifiers. More examples of Σ_i^b and Σ_i^B formulas will be given in Section 6.2.1, while Section 6.3 will give more detail regarding the class of Σ_i^B formulas, and the scope in which they are covered for this document.

6.1.4 Comprehension Axioms

Although different theories of Bounded Arithmetic are given in this document, each one provides a Comprehension Axiom. This axiom is used to define strings, and sets of strings. As an example, we can define the set X to be the set of prime numbers:

$$X(i) \leftrightarrow (\forall j < i)\,[((i \bmod j) = 0) \rightarrow (j = 1 \vee j = i)] \tag{6.3}$$

Note that formula 6.3 is a Σ_0^B formula. In the same way, we can define more complicated sets.

Finally, note that some theories can have more than one Comprehension Axiom. The theory $\mathbf{W_1^1}$ has two, to describe second and third-sorted formulas, respectively. Having given the general properties common to all theories of Bounded Arithmetic, we will now describe the second-sorted theory $\mathbf{V^i}$, as a "prelude" to the third-sorted theory $\mathbf{W_1^1}$.

6.2 The Theory $\mathbf{V^i}$

Before we detail the theory $\mathbf{W_1^1}$, we give a brief introduction to the family of theories $\mathbf{V^i}$ given by Cook and Ngyuen in [8]. This family operates over second-sorted elements (sets of natural numbers), and for $i \geq 1$ each $\mathbf{V^i}$ characterizes the functions which belong to the i-th level of the polynomial time hierarchy, starting with $\mathbf{V^1}$ characterizing $\mathbf{P}$. Here when we say "characterizes" we mean that all functions provably total in $\mathbf{V^i}$ are exactly the functions in the i-th level of the polynomial time hierarchy. The theory $\mathbf{V^0}$, meanwhile, characterizes functions of the class $\mathbf{AC^0}$. Many of the axioms used to define $\mathbf{V^i}$ are also used to define $\mathbf{W_1^1}$, and the language $\mathcal{L}_A^2$ used to give the symbols for $\mathbf{V^i}$ is the basis for the language $\mathcal{L}_A^3$ used for $\mathbf{W_1^1}$.

6.2.1 Definition and Basic Properties

As mentioned previously the family of theories $\mathbf{V^i}$ uses symbols from the language $\mathcal{L}_A^2$, and axioms from the set **2-BASIC** over these symbols. The language $\mathcal{L}_A^2$ is given by the set

$$\{0, 1, +, \times, ||, \in, \leq, =_1, =_2\} \tag{6.4}$$

where the elements 0 and 1 are numbers, $+$ is the addition operator for numbers, $\times$ represents multiplication of numbers, and $|X|$ gives the largest element of the set (second sort) X. The symbol $\in$ is a set membership operator for second-order sorts, while $\leq$ and $=_1$ compare numbers using each symbol's usual meaning. The $=_2$ symbol compares two second-sort variables for equality. For the rest of this document we will only write $=$, and let the meaning be determined by the context in which it is used.

The $|X|$ operator also allows us to give another representation of elements of the second sort, in the form of finite binary strings. The set X represents the binary string $\mathbb{X}$ of length $|X| - 1$, where $\mathbb{X}(i) = 1 \leftrightarrow i \in X$. When X is the empty set, we map this to the empty string. Notice also that the set $\{1\}$ is mapped to the empty string. In this mapping $|X|$ marks the end of $\mathbb{X}$, i.e. $|X|$ is one greater than the largest element of $\mathbb{X}$, which allows for strings with leading zeros, 0110 for example, to be expressed in set form. If X is the empty string, then we define $|X|$ to be 0. Finally, this also implies that the $=_2$ operator for second-sort equality is used to compare binary strings.

6.2.2 Axioms for $\mathbf{V^i}$

Table 6.1 gives the set of axioms **2-BASIC** which axiomatize $\mathbf{V^i}$. The "B#" axioms deal with elements of the first sort, while axioms **L1** and **L2** formalize the meaning of $|X|$ mentioned earlier. Axiom **SE** states that sets X and Y are the same if and only if they have the same elements.

Now that we have defined the axioms which govern our theories, we can now state what formulas $\mathbf{V^i}$ can generate, via the Comprehension Axiom:

Definition 17 (Comprehension Axiom for $\mathbf{V^i}$). *If Φ is a set of formulas, then the **comprehension axiom scheme** for Φ, denoted by Φ – **COMP**, is the set of*

B1. $x+1 \neq 0$	**B7.** $(x \leq y \wedge y \leq x) \rightarrow x = y$
B2. $(x+1 = y+1) \rightarrow x = y$	**B8.** $x\ lex + y$
B3. $x+0 = x$	**B9.** $0 \leq x$
B4. $x+(y+1) = (x+y)+1$	**B10.** $x \leq y \vee y \leq x$
B5. $x \times 0 = 0$	**B11.** $x \leq y \leftrightarrow x < y+1$
B6. $x \times (y+1) = (x \times y)+x$	**B12.** $x \neq 0 \rightarrow \exists y \leq x(y+1 = x)$
L1. $X(y) \rightarrow y < \|X\|$	**L2.** $y+1 = \|X\| \rightarrow X(y)$
SE. $[\|X\| = \|Y\| \wedge \forall i < \|X\|(X(i) \leftrightarrow Y(i))] \rightarrow X = Y$	

Table 6.1: The set **2-BASIC**

all formulas

$$\exists X \leq y \forall z < y\,(X(z) \leftrightarrow \varphi(z)) \tag{6.5}$$

where $\varphi(z)$ *is any formula in* Φ*, and* X *does not occur free in* φ*.*

In the above definition $\varphi(z)$ may have free variables of both sorts, in addition to z. The types of formula sets Φ we are concerned with belong to the class Σ_i^B, originally presented in Section 6.1.3.

6.3 Description of $\mathbf{W_1^1}$

As mentioned previously, $\mathbf{W_1^1}$ is a bounded arithmetic language for reasoning in PSPACE. It is similar to the theory $\mathbf{V^1}$ developed by Cook and Nguyen in [8], but $\mathbf{W_1^1}$ is designed to reason over third-sorted variables, which represent sets of strings.

The symbols available for use in $\mathbf{W_1^1}$ are very similar to those available in $\mathbf{V^i}$, with a few modifications. The set of symbols is taken from the third-order language $\mathcal{L}_A^3$:

$$\{0, 1, +, \times, |\,|_2, \in_2, \in_3, \leq, =\} \tag{6.6}$$

with the additions over $\mathcal{L}_A^2$ being a $\in_3$ predicate for deciding $A \in_2 \mathcal{B}$. Although it uses a different set of symbols, $\mathbf{W_1^1}$ and $\mathbf{V^i}$ share the set of axioms **2-BASIC** described in Section 6.2.2.

The formulas contained within $\mathbf{W_1^1}$ are organized into classes in the hierarchy $\Sigma_i^{\mathcal{B}}$, whose formulas have arbitrarily many bounded first and second-order quantifiers, and exactly i alternations of third-order quantifiers. The outermost quantifier is

restricted to being an existential quantifier. For $\mathbf{W}_1^1$ we are concerned only with $i \in \{0,1\}$. **Strict** $\Sigma_i^{\mathcal{B}}$-formulas are those consisting of a $\Sigma_{0^{\mathcal{B}}}$ formula prefixed by i alternations of third-sorted quantifiers. Formula with single existential third-order quantifier followed by a formula with no third-order quantifiers. For the terms of this paper we will be concerned with a class for formulas $\forall^2\Sigma_1^{\mathcal{B}}$, consisting of single bounded universal second-order quantifier followed by a **strict** $\Sigma_1^{\mathcal{B}}$ formula.

The comprehension schemes for $\mathbf{W}_1^1$ are the following:

$$\left(\exists Y \leq t\left(\overline{x}, \overline{X}\right)\right)\left(\forall z \leq s\left(\overline{x}, \overline{X}\right)\right)\left[\phi\left(\overline{x}, \overline{X}, \overline{\mathcal{X}}, z\right) \leftrightarrow Y(z)\right] \qquad \Sigma_0^{\mathcal{B}}\text{-2COMP}$$

$$\left(\exists \mathcal{Y}\right)\left(\forall Z \leq s\left(\overline{x}, \overline{X}\right)\right)\left[\phi\left(\overline{x}, \overline{X}, \overline{\mathcal{X}}, Z\right) \leftrightarrow \mathcal{Y}(Z)\right] \qquad \Sigma_0^{\mathcal{B}}\text{-3COMP}$$

In each of these schemes $\phi \in \Sigma_0^{\mathcal{B}}$ is subject to the restriction that neither Y nor $\mathcal{Y}$ (as appropriate) occurs free in ϕ. $\mathcal{Y}(Z)$ abbreviates $Z \in_3 \mathcal{Y}$, and similarly for $Y(z)$.

Finally, for $\mathbf{W}_1^1$ we introduce the notion of segments of a string $\mathbb{X}$, which are uniform-length substrings consisting of adjacent bits of $\mathbb{X}$, where the length of each substring evenly divides the length of $\mathbb{X}$. Assuming all segments are of length n and we start with the leftmost bit of the string, each segment $\mathbb{X}^{[i]}$ of $\mathbb{X}$ begins with the bit at position $(i-1)n+1$, and ends at position $i \times n$. Equation 6.7 gives an example for a string of length 8, and segments of length 2.

$$\mathbb{X} = \overbrace{01}^{\mathbb{X}^{[1]}}\overbrace{10}^{\mathbb{X}^{[2]}}\overbrace{11}^{\mathbb{X}^{[3]}}\overbrace{00}^{\mathbb{X}^{[4]}} \tag{6.7}$$

6.4 Construction of formula $\Phi(\mathbb{X}, n, m)$

In the following sections we will restate the existence of a winning strategy for the first player of Chomp as a $\mathbf{W}_1^1$ formula, and use theorems by Skelley in [15] to show that the function computing this strategy is in PSPACE. We first give a formula $\Phi(\mathbb{X}, n, m)$ which asserts that $\mathbb{X}$ is a valid chomp game on an $n \times m$ board. Here $\mathbb{X}$ is a string of length $(n \times m)(n+m)$, consisting of $(n \times m)$ many segments (the maximum number of moves for the game, see Section 2.8, Lemma 3) of length $(n+m)$ each. Φ is the conjunction of three formulas: ϕ_{init}, ϕ_{final}, and ϕ_{move}.

1. ϕ_{init} asserts that $\mathbb{X}^{[1]}$ is the initial configuration, i.e.

$$\phi_{\text{init}}(\mathbb{X}^{[1]}, n, m) = \forall i \leq (n+m)((i > m) \rightarrow X^{[1]}(i) = 1) \tag{6.8}$$

2. ϕ_{final} asserts that $X^{[n\times m]}$ is the final configuration, i.e.

$$\phi_{\text{final}}(\mathbb{X}^{[n\times m]}, n, m) = \forall i \leq (n+m)((i > n) \rightarrow X^{[n\times m]}(i) = 0) \tag{6.9}$$

3. ϕ_{move} asserts that each segment of $\mathbb{X}$ can be obtained from one legal move on the previous segment, i.e.

$$\phi_{\text{move}}(\mathbb{X}, n, m) = \forall i < (n \times m)(X^{[i]} \text{ "yields" } X^{[i+1]}) \tag{6.10}$$

The "yields" portion of the definition of ϕ_{move} requires elaboration: how can we state that a configuration can be obtained by performing one legal move on another? In order to answer this question, we will take the angle of determining what square was played between two consecutive configurations $\mathbb{X}^{[i]}$ and $\mathbb{X}^{[i+1]}$, and ensure that the differences between the the configurations correspond to playing this square. We will use a running example beginning from the configuration in Figure 6.1 (with $\mathbb{X}^{[i]} = 01001101$) to help explain the construction and reasoning behind the formula. Note that for this construction we change the coordinate system for Chomp squares: the tuple (j, k) still corresponds to the square at row j and column k, but now the origin $(1, 1)$ is the top-left square, with (m, n) being the bottom-right square.

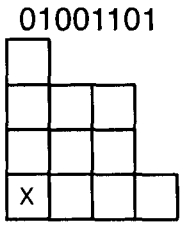

Figure 6.1: Starting configuration of an example Chomp game

The first part of our formulation involves determining what squares can be played from a given configuration. In the string representation of a Chomp configuration, a 0 occurring to the left of at least one 1 corresponds to a column of playable squares, the height of which being the number of 1s to the right of the 0. Going back to our example $\mathbb{X}^{[i]}$, the second 0 corresponds to squares $(2, 2), (3, 2)$, and $(4, 2)$, as shown by the blue highlighted squares in Figure 6.2. Thus, a square (j, k), can be played on

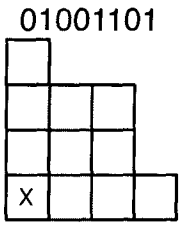

Figure 6.2: Example showing the column of squares represented by a 0 in $\mathbb{X}^{[i]}$

$\mathbb{X}^{[i]}$ if and only if in reading $\mathbb{X}^{[i]}$ from left to right, the k^{th} 0 is encountered before the j^{th} 1. From our example configuration, the square $(2,3)$ can be played, as the third 0 occurs before the second 1. The square $(1,3)$ cannot be played, however, as the third column of squares does not exist in the first row - $\mathbb{X}^{[i]}$ confirms this, as we encounter the first 1 before the third 0.

Having given the method to detect whether a square can be played on a given configuration, we now introduce two auxiliary functions F_0 and F_1 which will be used to formalize both the conditions for a playable square, and the configuration resulting from playing the square. We use the function $numones(y, \mathbb{X}^{[i]})$ given by Cook and Nguyen in [8] to aid in the definitions. This function is defined in $\mathbf{V^1}$ and thus can be used in $\mathbf{W_1^1}$, and gives the number of elements of $\mathbb{X}^{[i]}$ which are $< y$. We also introduce the notation $\mathbb{X}(a:b)$, $a < b$ and $b \leq |\mathbb{X}|$, with $\mathbb{X}(a:b)$ being shorthand for a function from strings to strings which extracts the substring of $\mathbb{X}$ beginning with $\mathbb{X}(a)$ and ending with $\mathbb{X}(b)$. Although not directly in the language of $\mathbf{W_1^1}$, it can be defined via the comprehension axioms.

Definition 18. $\mathrm{F}_0(a,b,\mathbb{X})$ *is the position in the string* $\mathbb{X}$ *where, starting from (and including) position* a*, there are* b *zeros:*

$$\mathrm{F}_0(a,b,\mathbb{X}) = c \leftrightarrow numones\,(1, \mathbb{X}(a:c)) = b$$

$F_1(a,b,\mathbb{X})$ *is defined analogously for* b *ones:*

$$\mathrm{F}_0(a,b,\mathbb{X}) = c \leftrightarrow |\mathbb{X}(a:c)| - numones\,(1, \mathbb{X}(a:c)) = b$$

Note that we define $\mathrm{F}_0(a, 0, \mathbb{X})$ and $\mathrm{F}_1(a, 0, \mathbb{X})$ to be 0 for any $a, \mathbb{X}$. As well, if we reach the end of $\mathbb{X}$ before encountering b 0s or 1s, we define $\mathrm{F}_0(a, b, \mathbb{X})$ and $\mathrm{F}_1(a, b, \mathbb{X})$ to be $|\mathbb{X}|$. These formulas allow us to restate the conditions for the existence of a playable square, which will become the first sub-formula to the overall "yields" equation. Each of these sub-formulas will have a special label ψ_i, beginning with the condition for the existence of a playable square, ψ_1:

Lemma 19. *A square (j, k) can be played on a configuration $\mathbb{X}$ if and only if*

$$\mathrm{F}_0(1, k, \mathbb{X}) < \mathrm{F}_1(1, j, \mathbb{X}) \qquad (\psi_1)$$

Proof. We will argue by contradiction. Suppose square (j, k) exists on configuration $\mathbb{X}$, but $\mathrm{F}_0(1, k, \mathbb{X}) > \mathrm{F}_1(1, j, \mathbb{X})$ - note that we cannot have $\mathrm{F}_0(1, k, \mathbb{X}) = \mathrm{F}_1(1, j, \mathbb{X})$ as a position in $\mathbb{X}^{[i]}$ can't simultaneously be 0 and 1. This means that the k^{th} 0 occurs after that j^{th} 1, which in turn means that row j terminated before column k. Thus, there are no squares beyond column $k-1$ in row j, so square (i, j) does not exist in $\mathbb{X}$, and cannot be played. □

Re-examining the previous two examples, for square $(2, 3)$ we have $\mathrm{F}_0(1, 3, 01001101) = 4$ and $\mathrm{F}_1(1, 2, 01001101) = 5$, making it a playable square, while for square $(1, 3)$ the function values are $\mathrm{F}_0(1, 3, 01001101) = 4$ and $\mathrm{F}_1(1, 1, 01001101) = 2$, indicating that it cannot be played.

Now that we can identify playable squares, we need to determine the configurations which result from playing these squares. As Figure 6.3 demonstrates, playing a square (j, k) shifts at least a single 1 behind the corresponding 0 to delimit the row being shortened or eliminated, depending on how many rows the move affects. In Figure 6.3 only a single row is affected, thus only a single 1 is shifted. If a move affects multiple rows, then multiple 1s are shifted behind the 0 corresponding to the selected square, which is shown in Figure 6.4. In figures 6.3 and 6.4, the 0 corresponding to the square being selected is highlighted in blue, while any affected 1s are marked in red. Thus, in general the substring affected by a move begins at the position of the 0, and has length equal to the number of rows affected (number of 1s) plus the largest amount of squares eliminated on the rows.

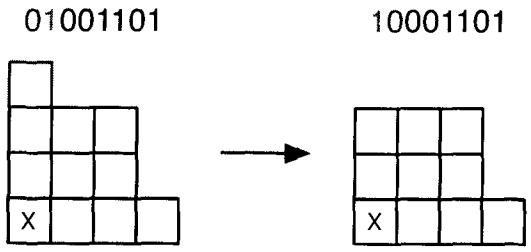

Figure 6.3: Eliminating a single square from a Chomp grid

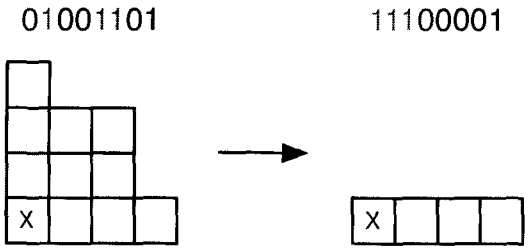

Figure 6.4: Eliminating multiple squares from a Chomp grid

We can express the starting and ending positions of the substring affected by a move on (j, k) mathematically using F_0 and F_1. We calculate two values p and q which will mark the beginning and ending of the substring (inclusive) respectively:

$$p = \mathrm{F}_0(1, k-1, \mathbb{X}^{[i]}) + 1 \qquad (\psi_2)$$

$$q = \mathrm{F}_1(p, j, \mathbb{X}^{[i]}) \qquad (\psi_3)$$

Intuitively, $(p-1)$ marks the location of the $(k-1)^{\text{th}}$ 0, so p forms the left boundary of the substring. q is the position of $\mathbb{X}^{[i]}$ in which we have seen j 1s starting from p, so it designates the right boundary. Finally, to arrive at the new configuration $\mathbb{X}^{[i+1]}$ we replace $\mathbb{X}^{[i]}(p : q)$ with $1^j 0^{(q-p-k+1)}$, i.e., the number of 1s corresponding to the number of rows affected, then the number of 0s indicating the new difference in row lengths. In order to express this new substring as a formula we break it down into several sub-formulas. First, positions that are outside the range of p and q remain

unchanged:

$$(r < p) \rightarrow (\mathbb{X}^{[i+1]}(r) = \mathbb{X}^{[i]}(r)) \tag{ψ_4}$$

$$(r > q) \rightarrow (\mathbb{X}^{[i+1]}(r) = \mathbb{X}^{[i]}(r)) \tag{ψ_5}$$

Next, for positions between p and q we assign values based on their placement relative to the 0 associated with the move. Positions before the 0 hold 1s affected by the move, while positions after contain 0s.

$$(r < p + j) \rightarrow (\mathbb{X}^{[i+1]}(r) = 1) \tag{ψ_6}$$

$$(r \geq p + j) \rightarrow (\mathbb{X}^{[i+1]}(r) = 0) \tag{ψ_7}$$

$$(p \leq r \leq q) \rightarrow (\psi_6 \wedge \psi_7) \tag{ψ_8}$$

Finally, we put formulas ψ_4, ψ_5 and ψ_8 together to create one formula describing the changes between configurations $\mathbb{X}^{[i]}$ and $\mathbb{X}^{[i+1]}$:

$$(\forall r \leq |\mathbb{X}|)(\psi_4 \wedge \psi_5 \wedge \psi_8) \tag{ψ_9}$$

Thus, if we take the formula ψ_9 for describing legal changes between configurations and combine it with formula ψ_1 for the existence of a playable square, as well as ψ_2 and ψ_3 for the existence of values for p and q, we have the following formula for "yields":

$$(\exists j \leq NumOnes)(\exists k \leq NumZeros)\left[\psi_1 \wedge (\exists p < |\mathbb{X}^{[i]}|)(\exists q \leq |\mathbb{X}^{[i]}|)(\psi_2 \wedge \psi_3) \wedge \psi_9\right] \tag{6.11}$$

where

$$NumOnes = |\mathbb{X}^{[i]}| - numones(1, \mathbb{X}^{[i]}) \tag{6.12}$$

$$NumZeroes = numones(1, \mathbb{X}^{[i]}) \tag{6.13}$$

Having completed the formula $\Phi(\mathbb{X}, n, m)$ we can now restate the existence of a winning strategy for the first player in $\mathbf{W_1^1}$.

6.5 Existence of a Winning Strategy for the First Player, Revisited

As stated in Section 3, a winning strategy is a function f from configurations to configurations. In this section as well as section 6.6, will show that using $\mathbf{W_1^1}$, we

have the function $\mathbb{S} : \mathbb{X} \rightarrow \mathbb{Y}$, which takes configurations in our string representation, so that whoever plays by $\mathbb{S}$ wins. Recall that in Section 3.5.1 we showed that the first player has a winning strategy from any initial configuration of Chomp. We can restate this property using our representation of configurations as strings and segments: $\forall \mathbb{Y}$, where $\mathbb{Y}$ has $\left(\frac{n \times m}{2}\right)$ many segments of length $(n + m)$ each, there exists an $\mathbb{X}$ with $(n \times m)$ many segments of length $(n + m)$ each with the following properties:

1. Every other segment of $\mathbb{X}$ is equal to a corresponding segment from $\mathbb{Y}$, i.e. for i odd, we have $\mathbb{X}^{[i]} = \mathbb{Y}^{[\lceil i/2 \rceil]}$. This also implies that $\mathbb{Y}^{[1]}$ is the initial configuration for the game.

2. For i even, $\mathbb{X}^{[i]}$ is obtained by an application of the strategy function on the previous configuration, i.e. $\mathbb{X}^{[i]} = \mathbb{S}(\mathbb{X}^{[i-1]})$.

3. $\mathbb{X}$ gives a legal game of Chomp, i.e. $\Phi(\mathbb{X}, n, m)$ is true.

4. The first time the final configuration occurs, it is in an even segment of $\mathbb{X}$:

$$\exists i \leq (n \times m) \left[(i \bmod 2 = 0) \wedge (\mathbb{X}^{[i]} = 1^n 0^m) \wedge (i < j \leq (n \times m) \rightarrow \mathbb{X}^{[j]} = \mathbb{X}^{[i]}) \right]$$

Here $\mathbb{X}$ has the usual meaning of listing the configurations of a single Chomp game, while $\mathbb{Y}$ is intended to contain only the configurations encountered by the second player (hence its reduced size). The first property links the configurations of $\mathbb{Y}$ to the Chomp game given by $\mathbb{X}$, and explicitly states that $\mathbb{Y}$ contains configurations encountered by the second player. The second property states that player 1 consults (and follows) the strategy function's advice for each of his / her moves. Property 3 states that the game given by $\mathbb{X}$ is a legal Chomp game, while the final property states that the first player wins, as if the final configuration (no squares remaining) occurs on an even segment of $\mathbb{X}$, then player 2 must have played the poisoned square on their turn.

Finally, note that the order or quantification is reversed from previous constructions. Previously, we stated that there exists a move of the first player such that for every move of the second player, etc., the first player wins. Now our quantification states that for any legal sequence of moves by the second player, there exists a Chomp game in which the first player can follow a strategy function to force player 2 to lose.

6.6 Winning Strategies From Any Configuration

Having shown that we can express the existence of a winning strategy for the first player in $\mathbf{W}_1^1$, we now change our focus to expressing the existence of a winning strategy from any configuration C:

$$\forall C \exists \mathbb{S} \left[\mathrm{Win}_{P_1}(\mathbb{S}, C) \vee \mathrm{Win}_{P_2}(\mathbb{S}, C) \right] \tag{6.14}$$

Here $i \in \{0, 1\}$, and $\mathrm{Win}_{P_i}(\mathbb{S}, C)$ asserts that if player i plays by strategy $\mathbb{S}$ starting at configuration C, then they will win. We can define $\mathrm{Win}_{P_i}(\mathbb{S}, C)$ as follows: $\forall \mathbb{Y}$, where $\mathbb{Y}$ is a sequence of moves of player $\bar{i}$ (where $\bar{i} = \{0, 1\} - \{i\}$), if it is player i's turn on configuration C and their move is to $C' = \mathbb{S}(C)$, then player $\bar{i}$ will eventually be left with only the poisoned square. Notice here that $\mathrm{Win}_{P_i}(\mathbb{S}, C)$ is a $\Sigma_0^{\mathbb{B}}$ formula, while equation 6.14 is a $\Sigma_1^{\mathbb{B}}$ formula. In both cases we omit the bounds on the quantified variables for clarity; in the definition of $\mathrm{Win}_{P_i}(\mathbb{S}, C)$ $\mathbb{Y}$ is less than $\left(\frac{n \times m}{2}\right)$, while in equation 6.14 C is bounded by the dimensions of the Chomp grid $(n + m)$.

We now move to proving that equation 6.14 can be expressed in $\mathbf{W}_1^1$:

Theorem 20. $\mathbf{W}_1^1 \vdash \forall C \exists \mathbb{S} \left[\mathit{Win}_{P_1}(\mathbb{S}, C) \vee \mathit{Win}_{P_2}(\mathbb{S}, C) \right]$

Proof. We will use an inductive argument on $|C|$, the number of squares present in configuration C. $|C| = n \times m$ for initial configurations, and 1 for the configuration with only the poisoned square. Our base case is said configuration with only the poisoned square, in which $|C| = 1$. Here $\mathrm{Win}_{P_2}(\mathbb{S}, C)$ is true, as the first player is forced to select the poisoned square, so the $\mathbb{S}$ function for player 2 can be anything.

For the induction step, we assume that the induction hypothesis holds for all configurations C such that $|C| \leq e$, and examine what happens if we have a C' with $|C'| = e + 1$. In this case we need to show that

$$\exists \mathbb{S} \left[\mathrm{Win}_{P_1}(\mathbb{S}, C') \vee \mathrm{Win}_{P_2}(\mathbb{S}, C') \right] \tag{6.15}$$

Thus, suppose that it is the first player's turn on C', and they make a move to configuration C'', with $|C''| < |C'|$. Notice here that as $|C''| < |C'| < e + 1$, we can apply the induction hypothesis, which results in

$$\exists \mathbb{S} \left[\mathrm{Win}_{P_1}(\mathbb{S}, C'') \vee \mathrm{Win}_{P_2}(\mathbb{S}, C'') \right] \tag{6.16}$$

and thus

$$\exists \mathbb{S}\,[\mathrm{Win}_{P_1}(\mathbb{S}, C'')] \vee \exists \mathbb{S}\,[\mathrm{Win}_{P_2}(\mathbb{S}, C'')] \tag{6.17}$$

If the first player's move always leads to $\exists \mathbb{S}\,[\mathrm{Win}_{P_2}(\mathbb{S}, C'')]$ being true, then we can conclude $\exists \mathbb{S}\,[\mathrm{Win}_{P_2}(\mathbb{S}, C')]$. This signifies that no matter what player 1 does, the second player can win. If, on the other hand, some move of player 1 leads to $\exists \mathbb{S}\,[\mathrm{Win}_{P_1}(\mathbb{S}, C'')]$ being true, then we define $\mathbb{S}'$ (using comprehension) to be the same as $\mathbb{S}$, except that $\mathbb{S}'(C') = C''$ (i.e., the strategy that guarantees victory for player 1). This allows us to conclude $\mathrm{Win}_{P_1}(\mathbb{S}', C')$, and further $\exists \mathbb{S}\,[\mathrm{Win}_{P_1}(\mathbb{S}, C')]$. Thus, no matter the outcome of player 1's move, equation 6.15 is true.

The second part of the proof involves showing that both players cannot have winning strategies:

$$\neg \exists \mathbb{S}\,[\mathrm{Win}_{P_1}(\mathbb{S}, C') \wedge \mathrm{Win}_{P_2}(\mathbb{S}, C')] \tag{6.18}$$

To do this we must show that both players cannot avoid selecting the poisoned square. As the rules of Chomp require players to select at least one square on their turn, eventually when only the poisoned square remains (as each player will continue selecting other squares as long as they are available) one player will be forced to select it, and thus lose the game. □

Having shown that $\mathbf{W}_1^1$ proves theorem 20, we can now show that $\mathbf{W}_1^1$ proves the existence of a winning strategy for the first player from any initial configuration.

Theorem 21. $\mathbf{W}_1^1 \vdash (C = 0^m 1^n) \rightarrow \exists \mathbb{S}\,[\mathit{Win}_{P_1}(\mathbb{S}, C)]$

Proof. We apply the same strategy-stealing argument found in Section 3.5.1 - having shown that either player has a winning strategy, we assume the second player has a winning strategy $\mathbb{S}$ from the initial configuration C, and that player 1's first move is the top-right square of the grid, putting the game into configuration C'. Then player 2 makes a move following the advice of $\mathbb{S}$ which places the game into a configuration C'' from which the next player loses. We then observe that any such move of player 2 could have been played by the first player initially, as it must contain the top-right square of the grid (the supremum of the poset). Thus we can modify $\mathbb{S}$ into $\mathbb{S}'$ by using comprehension, with the only change being $\mathbb{S}'(C) = C''$, otherwise $\mathbb{S}' = \mathbb{S}$. We now have a winning strategy for the first player, which contradicts our original assumption. □

6.7 Showing the winning strategy is in PSPACE

Now that we have our formula Φ and strategy function $\mathbb{S}$ we can use the **Witnessing Theorem for $\mathbf{W}_1^1$** [15] to show that this function to compute the winning strategy is in PSPACE:

Theorem 22. *Suppose $\mathbf{W}_1^1 \vdash \exists Y \phi(X,Y)$, for $\phi(X,Y) \in \sum_1^B$ with all free variables displayed. Then there exists a function $f \in$ PSPACE of polynomial growth rate such that for every string X, $\phi(X, f(X))$ is true.*

Thus, with $\exists \mathbb{S}\,[\mathrm{Win}_{P_1}(\mathbb{S}, C)]$, we have a proof that constructing the winning strategy $\mathbb{S}$ is in PSPACE. This also means that any proof of $\mathbb{S}$ being in PSPACE will require a construction of $\mathbb{S}$, giving the strategy itself. If it is discovered that the existence of a winning strategy for the first player can be proven in a weaker theory than $\mathbf{W}_1^1$ ($\mathbf{V}^1$, for example), then the proof of this fact will construct a strategy in a lower complexity class ($\mathbf{P}$, if the theory is $\mathbf{V}^1$).

Conclusions and Future Work

In this thesis we have shown three things. First, a new method for showing poset games in PSPACE, via a polynomial-time (logarithmic-space) reduction to the game of Geography. Second, a reformulation of the existence of a winning strategy for the first player in Chomp, using the Bounded Arithmetic theory $\mathbf{W_1^1}$ (which characterizes PSPACE), which uses the witnessing theorem of Bounded Arithmetic to show Chomp $\in$ PSPACE. Finally, a definition of the concept of Game Completeness, and a translation from tree-like poset games to a modified version of Chomp. As future work, it would be interesting to establish whether or not poset games are PSPACE-complete (it is the opinion of the author that they are not), and we hope that formulating Chomp using Bounded Arithmetic may eventually yield a feasible algorithm for computing the winning strategy for the first player.

Appendix A

Appendix

This appendix presents the beginnings of an alternate algorithm for translating general poset games to Chomp. It operates in two stages. The first stage assigns a tuple (r_i, c_i) to every node i of the source tree T, where r_i and c_i are the number of rows and columns in a Chomp grid required for the subtree rooted at i. The second stage uses these tuples to assign x and y coordinates in the Labeled Chomp grid to each node i, thus mapping each node to a cell in the grid. The following sections detail the two stages, although as mentioned the algorithm is not complete.

A.1 Stage One: Space Requirements for Subtrees

The first step in our translation is to associate with each node i of the source tree T an additional label (r_i, c_i) which indicates the number of rows and columns required in the Chomp grid for the subtree rooted at i. To do this we define a function $Dim : \mathcal{T} \times \mathbb{N} \longrightarrow \mathbb{N} \times \mathbb{N}$, where $\mathcal{T}$ is the set of all finite, rooted trees labeled using a breadth-first labeling method. Given a tree $T \in \mathcal{T}$ and a node i in the tree (i.e., a valid label of T), $Dim(T, i)$ returns a tuple (r_i, c_i) with the number of rows r_i and columns c_i in which the subtree rooted at i will be inscribed. We say that the size of i's subtree, or simply the size of i is (r_i, c_i). We define the function recursively, first giving two base cases which detail how to inscribe trees consisting of a node with a single leaf, and a 2-level tree with a single root and n leaves. Note that we use o_degree_i to indicate the out-degree of node i.

Algorithm 23. *Recursive procedure to calculate the number of rows r and columns c to inscribe a subtree of T rooted at node i:*

Base Cases:

1. A tree consisting of a single node i occupies 1 row and 1 column.
2. For a tree consisting of a node i with one leaf j, we place j diagonally above i in the Chomp grid. This leaves us with a tree which occupies 2 rows and 2 columns, as in Figure A.1.

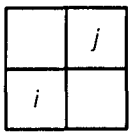

Figure A.1: A node i with a single leaf j in Chomp form

3. For a tree of height 2 with n leaves, this occupies n rows and n columns in a Chomp grid, with the leaves along the n^{th} diagonal. Figures A.2 and A.3 illustrates this.

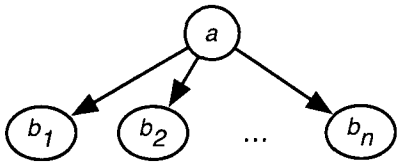

Figure A.2: A tree of height 2 of with n leaves

Recursion Steps:

1. Obtain o_degree_i.
   (a) $o\_degree_i = 0$: Node i is a leaf of the tree. Set the size of i to $(1, 1)$, and return.
   (b) $o\_degree_i = 1$: Node i has one child, thus set the size of i to $(1, 1)$ and continue.

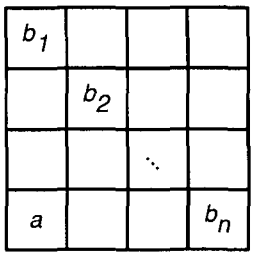

Figure A.3: A tree of height 2 with n leaves in Chomp form

(c) $o\_degree_i > 1$: Initialize the size of i to $(0, 0)$ and continue.

2. For each child ch_i of i, calculate $Dim(T, ch_i)$ and add the result to the initial size of i.

3. Once the dimensions of all of i's children have been added to the initial size of i, return.

Now that we have the labels (r_i, c_i) for each node i, we can use them to position each i into the Chomp grid. Section A.2 details the method for doing this.

A.2 Positioning Nodes into Chomp Squares

After obtaining the dimensions of all nodes in our forest, we go back through the tree(s) and use this information to place exactly where each node will reside on the Chomp grid. As we have the depth d_i from the root to a node i, we can classify the nodes into **levels** $l_k = \{i \in T | d_i = k\}$. Our algorithm then goes one level at a time, examining only the nodes on the current level to place them on the Chomp grid. So, the formula $f(i, k)$ for calculating the x and y coordinates on the Chomp grid for a node i on level k is the following. Recall that by convention $\sum_{i \in \emptyset} = 0$.

Algorithm 24. *To calculate the position (x_i, y_i) of a node i of T, do the following:*

$$f(i,k) = \begin{cases} \left(\left(k + \sum_{j=(i+1)}^{|l_k|} r_j\right), \left(k + \sum_{j=1}^{(i-1)} c_j\right)\right) & i \text{ is not leaf} \\ \left(\left(2 + \sum_{j=(i+1)}^{|l_k|} r_j\right), \left(2 + \sum_{j=1}^{(i-1)} c_j\right)\right) & i \text{ is a leaf} \end{cases}$$

Thus, to determine the x coordinate for node n_i, we look at the combined x-dimensions of all the subtrees rooted by nodes with indices $< n_i$. This adheres to the conventions described earlier, where if a node has a single leaf the leaf is placed above the parent in Chomp. If a parent has multiple children, then each successive child is placed to the right of the preceding one. Similar to this, in order to obtain the y coordinate for n_i, we look at the combined y-dimensions of all the subtrees rooted by nodes with indices $> n_i$. Going back to our conventions, if a parent has multiple children then each successive child will be placed lower than the previous ones, but must also be placed above those ahead of it.

A.3 Other Observations

There are a few other observations we can make from this translation. The first is that the Chomp grid for any tree-like poset with n leaves will have n rows and n columns. This makes sense intuitively, as the number of leaves represents the maximum number of incomparable elements in the graph.

A.4 Non-Treelike Posets

If our poset does not have a treelike form, our algorithm must be slightly modified. We will refer to non-treelike posets as **Layered Posets**, and make the distinction between the two main forms, **type 1** and **type 2**. The next two sections describe the main features of each type.

A.4.1 Type 1 Layered Posets

Figure A.4 gives an example of a type 1 layered poset. The key feature of a type 1 poset is "multiple inheritance", meaning that if a, b, c are members of the poset, we can have $a \preceq c$ and $b \preceq c$ directly. In terms of graphs, nodes in a type 1 layered posets can have in-degree > 1. From this we also note that Treelike Posets are a subset of Type 1 Layered Posets.

In order to handle the case where a node is comparable to two "parents", we

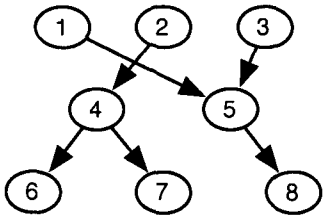

Figure A.4: An example of a Type 1 Layered Poset

modify the algorithm given in section A.1 to also calculate the total in-degree p_i of node i, as follows:

1. Set $p_i = 0$.
2. For all nodes j in the graph such that there exists a path from j to i, add p_j to p_i.
3. Add the actual in-degree of i to p_i.

We can then use p_i as an offset to position the subtree rooted at i according to how many other nodes / subtrees are comparable to it. To see this in better detail, examine the Chomp grid given in Figure A.5. Normally, squares 5 and 8 would be

	8		
	5		
1			
	3		
		6	
		4	7
X		2	

Figure A.5: The Chomp representation of the poset in Figure A.4

placed directly above square 1. However, these squares are also comparable to square 3 in the source graph. To represent this in the grid, we shift them over one square to the right, so that they are above square 3. This also explains why, in the calculation

for p_i we propagate the total in-degree values from the parents down through the subtrees. If this was not done, node 8 would have $p_8 = 1$, and would not be shifted over, making it incomparable to node 3. In terms of the in-degree p_i, square 5 has $p_5 = 2$, so the shift factor is $p_i - 1$. Incorporating this into the equations for node positions from section A.2, we have:

$$f(i,k) = \begin{cases} \left(\left(1 + \sum_{j=(i+1)}^{N_k} d_j(1)\right), p_i\right) & i = \min(l_k) \\ \left(\left(1 + \sum_{j=(i+1)}^{N_k} d_j(1)\right), \left(p_i + \sum_{j=1}^{(i-1)} d_j(2)\right)\right) & \min(l_k) < i < \max(l_k) \\ \left(1, \left(p_i + \sum_{j=1}^{(i-1)} d_j(2)\right)\right) & i = \max(l_k) \end{cases} \tag{A.1}$$

Although shifting subtrees over seems to solve the problem of multiple inheritance, a closer look at Figure A.5 reveals a subtle but very important point. Notice that along the "main diagonal", the ordering of the nodes, starting from the top-left is $1, 3, 2$, as opposed to the expected $1, 2, 3$. The switch in order between 2 and 3 is done to avoid making nodes 5 and 8 comparable to node 2. If we had left the squares in the usual order but shifted 5 and 8 over so they were comparable to 3, this would result in the grid given by Figure A.6. In this case squares 5 and 8 are now comparable to

			8
			5
1			
	6		
	4	7	
	2		
X			3

Figure A.6: The Chomp grid of Figure A.5, without nodes 2 and 3 switched.

all of the elements in node 2's subtree. Thus, we must find a way to sort subtrees to ensure that no elements become comparable in our Chomp translation.

A.4.2 Type 2 Layered Posets

Figure A.7 gives an example of a type 2 layered poset. As in type 1 nodes may have an in-degree > 1, however in type 2 posets we also allow comparability between nodes on the same level of the graph (see section A.2 for a definition of levels).

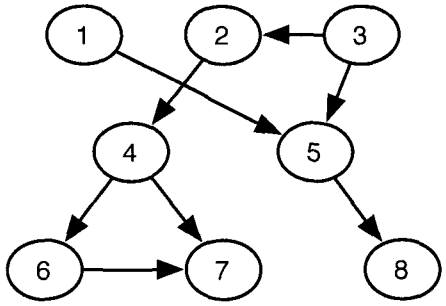

Figure A.7: An example of a Type 2 Layered Poset

A.5 Input Format

As this algorithm will potentially require expensive calculations, such as parsing a DAG to determine the space requirements of all subtrees, selecting an input format which minimizes these costs is an important consideration. Presently three types of structures have been identified - the adjacency matrix, adjacency list, and incidence matrix. As an initial comparison between the first two structures, consider the cost of identifying the "roots" of a DAG with n nodes; that is, identifying the nodes which have no incoming edges. For an adjacency matrix, this is $O(n^2)$, as we must traverse all n columns for n rows. For an adjacency list the cost is also $O(n)$, however there is a subtlety hidden - in an adjacency list only actual connections are listed, whereas an adjacency matrix lists connections even if they do not exist. Thus, an adjacency list representation may be significantly shorter if the DAG is sparse.

Bibliography

[1] Roland Backhouse and Diethard Michaelis. Fixed-Point Characterisation of Winning Strategies in Impartial Games. In Relational and Kleene-Algebraic Methods in Computer Science, volume 7, pages 34–47. Springer, 2004.

[2] Elwyn Berlekamp, John Conway, and Richard Guy. Winning Ways for Your Mathematical Plays, volume 1. A K Peters, Ltd., second edition, 2001.

[3] Charles L. Bouton. Nim, A Game with a Complete Mathematical Theory. The Annals of Mathematics, vol. 3(1/4):35–39, 1902.

[4] Andries Brouwer. Chomp, 2005. http://www.win.tue.nl/~aeb/games/chomp.html. Accessed October 9, 2007.

[5] Samuel R. Buss. Bounded Arithmetic. Bibliopolis, Naples, Italy, 1986.

[6] Steven Byrnes. Poset Game Periodicity. Integers: Electronic Journal of Combinatorial Number Theory, 3(G03), November 2003.

[7] John Conway. On Numbers and Games. A K Peters, Ltd., second edition, 2001.

[8] Stephen Cook and Phuong Nguyen. Foundations of Proof Complexity: Bounded Arithmetic and Propositional Translations. http://www.cs.toronto.edu/~sacook/csc2429h/book. Last accessed March 29, 2008.

[9] Aviezri S. Fraenkel. Scenic Trails Ascending from Sea-Level Nim to Alpine Chess. In Games of No Chance, volume 29. MSRI, 1996.

[10] Eric J. Friedman and Adam Scott Landsberg. Nonlinear dynamics in combinatorial games: Renormalizing chomp. Chaos: An Interdisciplinary Journal of Nonlinear Science, 17(2):023117, 2007.

[11] David Gale. A Curious Nim-Type Game. The American Mathematical Monthly, 81(8):876–879, October 1974.

[12] Grzegorz Herman. Private communication, 2006.

[13] Christos H. Papadimitriou. Computational Complexity. Addison Wesley Longman, 1995.

[14] Michael Sipser. Introduction to the Theory of Computation. Thomson Course Technology, second edition, 2006.

[15] Alan Skelley. A Third-Order Bounded Arithmetic Theory for PSPACE. In CSL, pages 340–354, 2004.

[16] Xinyu Sun. Improvements on chomp. Integers: Electronic Journal of Combinatorial Number Theory, 2(G01):1–8, 2002.

[17] Doron Zeilberger. Three - Rowed CHOMP. Advanced Applied Math, 26:168–179, 2001.