GAMES, GRAPHS, AND SEQUENCES

BY ELIZABETH J. KUPIN

A dissertation submitted to the
Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Mathematics

Written under the direction of
József Beck
and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

May, 2012

ABSTRACT OF THE DISSERTATION

Games, graphs, and sequences

by Elizabeth J. Kupin

Dissertation Director: József Beck

The thesis contains four separate chapters. Chapter 2 investigates a particular class of impartial combinatorial games called subtraction–division games. These games start with a total n, and two players take turns either subtracting a number a from the total or dividing the total by b. The winner is the first player to get to number 1. We will show that the Sprague–Grundy sequence $\{SG_{1,2d}(n)\}_{n=1}$, corresponding to the game $a = 1$ and $b = 2d$, is $2d$-automatic. We can also generalize these results to show that the sequence $\{SG_{a,2d}(an)\}_{n=1}$ is $2d$-automatic, if every prime factor of a also divides $2d$.

Chapter 3 determines the cop number for a type of infinite Cayley Graphs, under the assumption that the cops and robber move simultaneously in rounds rather than alternating turns. Since it's an infinite graph, we also require that the robber choses a vertex first, and then the cops chose positions at distance at least N from the robber. We prove the following: If C is a (connected) Cayley graph on group G with generating set M, and $x^{-1}mx \in M$ for all $x \in G$, $m \in M$, then $d(C)$ cops suffice to catch the robber, regardless of how far back the cops must begin.

Chapter 4 proves a special case of a 2011 conjecture about *colorful path colorings*. A colorful path coloring is a coloring of G with $\chi(G)$ colors, such that every vertex is at the head of a colorful path: a path of length $\chi(G)$ that uses all the colors. The

main result for graphs G on $2n$ vertices of the following form: a 2-factor on n vertices, a distinct C_n, and a perfect matching connecting the two sets of n vertices, then there exists a colorful path coloring of G.

The final chapter is joint work with Debbier Yuster. Our results generalize the Goulden–Jackson cluster method for finding the generating function for words avoiding a given set of forbidden words. The modifications allow for single letter, double letter (pairwise), and triple letter probability distributions.

Acknowledgements

Without the help of numerous individuals over the past few years, this thesis could never have been written.

I have a deep appreciation for my advisor József Beck, and the time and effort he has put in to my mathematical career over the past four years. I am particularly thankful for his directing me to the variation of Cops and Robbers investigated in Chapter 3.

Doron Zeilberger has been very generous by taking extra time to meet with me. I have the greatest respect for his mathematical mind, and his advice and guidance was invaluable in developing the material in Chapters 2 and 5.

I would also like to thank my other two committee members, Michael Saks and Vladimir Gurvich, for their willingness to serve on my committee and their constructive comments. In addition, I greatly appreciate Dr. Sak's suggestion to consider Cayley graphs in the context of Cops and Robbers.

In the Spring of 2012 I was funded by NSF DMS-0902241, a joint grant of Janos Komlos and Endre Szemerédi. This grant made it possible for me to focus solely on writing my dissertation, and I'm very grateful for their generosity.

The material on Subtraction–Division games (Chapter 2) was greatly improved by Eric Rowland, who introduced me to the world of automatic sequences. His advice both on the mathematical content and feedback on the writing has been incredibly helpful, and any remaining flaws are entirely mine.

Colorful path colorings (Chapter 4) were introduced to me at the REGS program in the summer of 2011. I'd like to thank Doug West for organizing the program, and Hannah Kolb for some good discussions in the early stages of the project.

The work of Chapter 5 was joint with Debbie Yuster, and I really appreciate her help both in developing the material and preparing it for publication. These results

began as a final project in Dr. Zeilberger's Experimental Mathematics course, and Debbie and I are thankful for his advice throughout the project.

Emilie Hogan helped me by proofreading several chapters, some of them even multiple times. I owe a lot of my joy and mental health throughout grad school to her continued friendship.

Daniel Cranston has generously spent his time over the past year listening, proofreading, and overall being a true partner to me during this process. Without his technical help there would not be any figures or tables in this work, and without his loving prodding I would have given up a long time ago.

Finally, I'd like to thank my parents and sister for their immense love and support throughout my whole life, and in particular during the past 6 years.

> "Trust in the Lord with all your heart and lean not on your own understanding; in all your ways acknowledge him, and he will make your paths straight." Proverbs 3:5-6.

Table of Contents

Chapter 1

Introduction

This thesis has four distinct parts: two chapters on results in combinatorial game theory, one chapter on a problem in graph coloring, and one chapter with work on generating functions. Combinatorial game theory, however, pulls results from many different areas of combinatorics. Games are often played on graphs, and the underlying graph structure can influence who will win. Moreover games create sequences of wins and losses, and results from combinatorics on words can be used to help analyze and predict these sequences. While the four results discussed are separate, they all tie in to the study of graphs and sequences.

1.1 Subtraction–Division Games

Chapter 2 has to do with a class of combinatorial games that I call subtraction–division games. These are two player impartial combinatorial games starting with a total n. The players take turns to decrease the total by either subtracting a number $s \in S$ from the total, or dividing the total by a number $d \in D$ (and rounding up, if necessary). The winner of the game is the first player to get to 1. The main question is, given two sets S and D, to determine which values of n are first player wins and which values of n are second player wins.

The name subtraction–division games is new, but games falling into this framework have been studied before. The first game of this type was for sets $S = \{1\}$, $D = \{2\}$, and was introduced as a puzzle in 2009 in [9]. Games of this type have also been studied by Aviezri Fraenkel, under the names of Mark-t and UpMark-t [14, 15]. These names refer specifically to sets $S = \{1, 2, \ldots, t-1\}$, $D = \{t\}$, and rounding down or up, respectively.

The main result in this chapter is a closed formula for the winning positions of subtraction division games under the following conditions: $S = \{a\}$, $D = \{2d\}$, and every prime factor of a is also a prime factor of $2d$. Moreover, under these conditions I prove that the Sprague–Grundy sequence of the game (for all values of n) is $2d$-automatic. Normally the Sprague–Grundy values of a game are recursively defined, but showing that the form an automatic sequence allows for fast (non-recursive) computation.

When $a = 1$, I construct a family of recurrences that completely determines the behavior of the Sprague–Grundy sequences. For other values of a, their behavior is characterized by a phenomenon that I call "holding": the sequence eventually develops large blocks with the same Sprague–Grundy value. Because of this, we are able to understand the behavior of the whole sequence just by considering a subsequence that picks out one value from each of the blocks. I show that this subsequence is related to the sequence of a different game that has a smaller value of a but the same d. The condition on the prime factors of a allows us to understand the behavior of the game $S = \{a\}$, $D = \{2d\}$ based on the game $S = \{1\}$, $D = \{2d\}$.

1.2 Simultaneous Cops and Robbers

The second chapter is a variation of cops and robbers problem. In the original version, one cop selects an initial vertex of a finite graph, and then one robber selects a vertex. They take turns moving along edges of the graph, the cop trying to land on the same vertex of the graph as the robber and the robber trying to evade the cop indefinitely. This was also generalized to the notion of the cop number of a graph: the minimum number of cops needed to guarantee capture of the robber, assuming the cops are allowed to coordinate their movements.

We play the following variation on an infinite graph: one robber selects a vertex to begin on, and then several cops select starting positions various vertices at distance at least N from the initial position of the robber. They move in rounds simultaneously, with the cops trying to land on the same vertex as the robber and the robber is trying to escape to infinity. We are interested in the minimum number of cops needed to

guarantee capture of the robber as a function of N, and particularly in showing when the minimum number of cops needed is a constant independent of N.

Suppose that the infinite graph is the Cayley graph of an infinite group G and set of generators M (we assume that $m \in M \implies m^{-1} \in M$). If M is closed under conjugation, then $|M|$ cops suffice to capture the robber for all values of N, and with some simple conditions on the placement of the cops within the graph. This is slightly better than required, since there is no need for coordination between the cops during the game play — each one acts independently. This is a similar to a result by Frankl for finite Cayley graphs [16], but for infinite Cayley graphs this is actually best possible.

1.3 Colorful Path Colorings

The third chapter addressed the following graph coloring question: Given a graph G, we would like to (properly) color it with $\chi(G)$ colors, in such a way that every vertex is at the head of a path of length χ that uses each color exactly once. A coloring of this form is called a *colorful path* coloring. It was conjectured that colorful path colorings exist for any connected graph other than C_7 in [3], but it is only known to be true in certain special cases.

I prove that the conjecture holds for cycle permutation graphs: $2n$ vertices in the form of two cycles of length n, connected by an arbitrary matching (i.e. a permutation of the numbers 1 through n). The Petersen graph is a common example of such a graph. I can also extend the proof to *half–generalized cycle permutation graphs*: graphs on $2n$ vertices with one cycle of length n, one 2-factor (collection of smaller cycles) on n vertices, and one perfect matching connecting the two sets.

To prove that such a coloring is possible I give an algorithm for coloring any half–generalized cycle permutation graph. A half-generalized cycle permutation graph will be 3-regular, and by Brook's theorem it will have chromatic number at most 3. Since the conjecture is trivially true for any bipartite graph, I will assume that $\chi(G) = 3$.

The algorithm colors the 2-factor first, and tries to guarantee that as many vertices as possible from both the cycle and the 2-factor are already at the head of a colorful

path, just from the first half of the coloring. We then have to extend the coloring to the cycle, in such a way that any vertex not yet at the head of a colorful path is satisfied.

Each vertex on the cycle will have a list of two colors available, and so by a standard list coloring result we know that extending the coloring is possible: if every vertex in a cycle has a list of two available colors, and there are at least two vertices with different lists, then it is possible to properly color the cycle so that every vertex has a color from its list. Much of the work is in showing that we can pick a good extension of the whole coloring that will handle any leftover vertices not yet at the head of a colorful path.

1.4 Variations on the Goulden–Jackson Cluster Method

All of the work in the final chapter of the thesis was joint with Debbie Yuster. Suppose we have an alphabet A, and set B of forbidden words in that alphabet. We are interested counting the number of words of length n made up of characters in A that avoid as *factors* all forbidden words in B. For our purposes, a factor will be a string of consecutive letters starting anywhere in a word, and a word that avoids all the elements of B as factors we will call a 'good' word. We will set a_n to be the number of good words of length n, and are interested in the sequence $\{a_n\}$.

An efficient way of finding a_n is to use generating functions. We can think of a generating function as assigning each good word a symbolic weight: $\text{weight}(w) = t^{|w|}$, where $|w|$ is the length of the word w. If we already knew the complete set of good words, and added up the weight of each of them, we would have $\sum_{n=0}^{\infty} a_n t^n$. This is the taylor expansion of some function $F(t)$ called the generating function of the sequence $\{a_n\}$. Of course, doing this computation directly is not feasible for large values of n. The Goulden–Jackson cluster method, introduced in [18], is an algorithm for computing $F(t)$ directly.

We might be interested in a more detailed weight function, that can keep track of more information about the good words. For example, if $A = \{a, b, c\}$ we might use a weight function that has one variable t for the length of the good word, and three variables x_a, x_b, and x_c for the number of times each letter appears in the good word.

In this case, weight$(abca) = t^4 {x_a}^2 x_b x_c$. If we added up the weights of all the good words, we would have $F(t, x_a, x_b, x_c) = \sum a_{n,p,q,r} x_a^p x_b^q x_c^r t^n$, where $a_{n,p,q,r}$ is the number of good words of length n with exactly p occurrences of a, q occurrences of b, and r occurrences of c. Again, doing this computation directly is not feasible, and we would like to find an algorithm that computes the weight enumeration function $F(t, x_a, x_b, x_c)$ directly.

The main result of this chapter is to show that we can generalize the Goulden–Jackson cluster method to efficiently compute the weight enumeration function for good words, under various weight functions. We show that we can not only use a variation of the method for a weight function that accounts for the number of occurrences of single letters, as above, we can also generalize to account for the number of occurrences of letter pairs, and letter triples. We can also introduce a third type of variable s, that counts how many forbidden factors are contained in a word. This bridges the gap between enumerating good words and words that contain many forbidden factors.

Chapter 2

Subtraction–Division Games

2.1 Introduction

A combinatorial game is one where there is perfect information (i.e. all players have the same, complete information about the state of the game) and no randomness. Perhaps the quintessential combinatorial game is Nim, where given an initial setup of piles of counters each player has the option to remove any number of counters from a single pile, and the first player to clear the table wins. The only information the players need is the current configuration of counters and piles, which is available to both players at all times. More complicated but well known combinatorial games include chess, checkers, and Go. On the other hand, poker has both randomness and hidden information (the knowledge of each player's cards are not available to the other players), so poker is not a combinatorial game.

An impartial combinatorial game is one where both players have the same set of moves available. Nim is an example of an impartial game, whereas chess is not: only the white player can move the white pieces. For a full introduction to combinatorial games, see the classic text [10].

As laid out in [10], two player combinatorial games that are impartial can be studied by analyzing the Sprague–Grundy function. This function is recursively defined on the states of the combinatorial game; it returns 0 if the first player has no winning strategy from this state, and some value $x > 0$ if the first player has a winning strategy from this state. Equivalently, the Sprague–Grundy function will return a non-zero value for any N-position, and zero for any P-position. Usually the states can be indexed by a single parameter n, and we write this function as $SG(n)$.

We define the Sprague–Grundy function based on the function *mex*. The mex of a set, or the minimum excluded value, is the smallest non-negative integer that does not appear in the set. We define the Sprague–Grundy function as follows. Winning positions are given Sprague–Grundy value 0, and from there the value of any position of the game is given by the mex of the set of Sprague–Grundy values of the positions that can be reached in a single move.

Because the Sprague–Grundy function is recursively defined, computation of the exact value for a given state can be difficult. Many games have been shown to have periodic Sprague–Grundy functions, however, which greatly reduces the computational work required. In particular, subtraction games fit this pattern. A subtraction game starts at a value n and is a race to say the number 1. Each player, on his or her turn, may subtract one number s from the current total, where $s \in S$, a prearranged set of allowed values. As a consequence of the pigeonhole principle, for any finite set S it is known that the Sprague–Grundy function of this game is periodic. Some further results about periodicity can be found here: [10, 12].

Not all interesting combinatorial games are known to have periodic Sprague–Grundy functions, and the most famous of these is Grundy's game. In this game the two players start with a single pile of n counters, and at each move the players take one of the piles on the table and split it into two piles of unequal height. It's not known if the SG function for this game is periodic, or how to analyze it, although there has been some work on this:

Grundy's game sparked interest in a class of 2 player combinatorial games called *octal games*, that are a hybrid between Nim, where counters are removed from an initial configuration of various piles, and the Grundy's game, where players break a pile into two or more smaller piles. The rules of a particular octal game are defined by a decimal number written in base 8. If the number is $0.d_1d_2d_3d_4\ldots$, the value of d_n indicates all allowable moves that remove n counters. In particular, $d_n \in \{0,\ldots,7\}$ is the sum of the following:

- 1 if a player can remove a pile of exactly n counters, 0 otherwise.

- 2 if a player can remove n counters from a pile with more than n, 0 otherwise.
- 4 if a player can remove n counters from a pile with more than n and split the remaining counters into 2 piles, 0 otherwise.

This framework incorporates many previously understood combinatorial games, for example, Nim is the octal game with number $0.3333333\ldots$ (infinitely many repeating threes) [10]. Collecting the games in this way allows for a more general survey of results. While some of the octal games have periodic SG functions it is unclear if all of them do. A full and dynamic survey of what is known can be found online [13].

Recently, there has been interest in looking at similar games where the SG values are known to be aperiodic but still follow patterns. In an effort to shed some light on the situation for octal games, Aviezri Fraenkel looked at a generalization of subtraction games that he called MARK-t, deliberately constructed to have aperiodic SG values [14, 15]. Like the regular subtraction games, in MARK-t the players can subtract any number from 1 through $t-1$. However they also have the option to divide the total by t and round down. This was a generalization of a problem originally posed in [9], where the players have the option to either subtract 1 or divide by 2. Fraenkel showed in [15] that the winning positions of MARK-2 correspond to integers whose binary representation ends in an even number of zeros. This work was carried further by Alan Guo, who gave a nice characterization of the winning and losing positions for MARK-t for general values of t, also based on the binary representation [21].

More generally, MARK-t is an example of what we will call a subtraction–division game.

Definition 2.1. *A subtraction–division game is an impartial, two player combinatorial game with three parameters:*

- *a set S of numbers that are allowable to subtract,*
- *a set D of numbers that are allowable to divide,*
- *a starting total n.*

The players alternate turns reducing the total either by subtracting a number $s \in S$ from the current total, or dividing the current total by a number $d \in D$ (we can consider variations where we always round up or always round down)

This chapter analyzes the class of subtraction–division games where $S = \{a\}$ and $D = \{b\}$, and we round up. We will think of the parameters a and b as being fixed while n varies, and we will denote the game by $G_{a,b}(n)$, and the Sprague–Grundy value of the game by $SG_{a,b}(n)$. We are interested in sets of parameters (a, b) for which we can characterize $\{SG_{a,b}(n)\}_{n=1}$.

To characterize $\{SG_{a,b}(n)\}_{n=1}$ in general, we start by considering the special case $\{SG_{1,2d}(n)\}_{n=1}$. We build up two types of results about $\{SG_{1,2d}(n)\}_{n=1}$. First, in Section 2.2, we show when $SG_{1,2d}(n)$ will be zero and when it will be non-zero, based on the value of n mod $4d$, and in some cases more specifically the base-$2d$ representation of n.

Then from these basic results, we prove the following theorem in Section 2.3:

Theorem 2.1. *If b is even, then the sequence $\{SG_{1,b}(n)\}_{n=1}$ is b-automatic.*

Showing that this sequence is b-automatic is similar in feel to the relationship between MARK-2 and the binary expansion of n, particularly in that it allows for faster (non-recursive) computation of $SG_{1,b}(n)$. The automatic sequences ansatz, however, allows for a more general relationship between the SG value and the base-b representation of n than a simple closed formula.

We are also interested in generalizing the results about $\{SG_{1,b}(n)\}_{n=1}$ to games where we subtract values other than 1. In Section 2.2.2, we discuss the special patterns that appear in $\{SG_{a,2d}(n)\}_{n=1}$ that don't show up in $\{SG_{1,2d}(n)\}_{n=1}$. Then in Section 2.2.3, we investigate how these patterns help us generalize some of the results about $\{SG_{1,2d}(n)\}_{n=1}$ to $\{SG_{a,2d}(n)\}_{n=1}$. Ultimately, this lets us prove the following theorem at the end of Section 2.3, which is an analog to Theorem 2.1:

Theorem 2.2. *If all prime factors of a are also factors of $2d$, then the sequence $\{SG_{a,2d}(an)\}_{n=1}$ is $2d$-automatic.*

2.2 Characterization of the Game Sequences

In this section we will characterize just the zeros of some SG sequences, rather than the entire sequence. The two main theorems of this section are stated below. Theorem 2.3 is fairly straightforward to prove, but its proof introduces the *Alternating Property*: the first main pattern we notice in $\{SG_{1,2d}(n)\}_{n=1}$. Theorem 2.4 is longer, and its proof uses ideas from Sections 2.2.2 and 2.2.3. It is important primarily because it introduces *holding*, the second type of pattern we see appearing.

Theorem 2.3. *If $a = 1$ and b even, the following is a complete characterization for when $SG_{a,b}(n)$ is zero. We will write down the base $2d$ representation in the following special form: least significant digit first, and representing even digits with the symbol e and odd digits with the symbol o.*

- *If $n = e\cdots$, then $SG_{1,2d}(n) \neq 0$.*
- *If $n = oe^k o\cdots$ for some maximal $k \geq 0$, then $SG_{1,2d}(n) = 0$ if k is even, and $SG_{1,2d}(n) \neq 0$ if k is odd.*

We will prove Theorem 2.3 in Section 2.2.1 by building up a series of structural Lemmas.

Suppose that the largest prime divisor of a that is relatively prime to b is 1, and b is even. Then we also have the following more general result:

Theorem 2.4. *If the largest divisor of a that is relatively prime to b is 1, and b is even, then the following is a complete characterization of when $SG_{a,b}(n)$ is zero.*

- *For $an > ab^a$, we will see holding of length a:*
  $SG_{a,b}(an) = SG_{a,b}(an-1) = \cdots = SG_{a,b}(an-a+1)$

We write down the base b representation of n with the least significant digit first, representing even digits with e and odd digits with o.

- *If $n = e\cdots$ and $n \geq b^{a+1}$, then $SG_{a,b}(n) \neq 0$.*
- *If $n = oo\cdots$ and $n \geq b^{a+2}$, then $SG_{a,b}(n) = 0$.*

- *If $n = oe^k \cdots$ for some maximal $k > 0$, then let $n' < n$ be the number formed by successively removing the second digit from the base b expansion of n, until either the entire first block of even digits has been removed, or $n' < ab^{a+2}$.*
  - *If $SG_{a,b}(n') = 0$ then $SG_{a,b}(n) = 0$ if k even, and $SG_{a,b}(n) \neq 0$ if k odd.*
  - *f $SG_{a,b}(n') \neq 0$ then $SG_{a,b}(n) \neq 0$ if k even, and $SG_{a,b}(n) = 0$ if k odd.*

Experimental evidence suggests that the bounds in Theorem 2.4 are much larger than needed. More specific values are computed in Sections 2.2.2 and 2.2.3, as the results needed to prove Theorem 2.4 are developed.

2.2.1 Proof of Theorem 2.3

By definition, we have that $SG_{1,2d}(n) = \text{mex}\{SG_{1,2d}(n-1), SG_{1,2d}\left(\lceil \frac{n}{2d} \rceil\right)\}$. To refer to this relationship easily, we will say that $SG_{1,2d}(n)$ *depends on* $SG_{1,2d}(n-1)$, and $SG_{1,2d}\left(\lceil \frac{n}{2d} \rceil\right)$. If we want to specify which term in the definition we are referring to, we may say that one value depends on another via subtraction or via division.

We begin by noting alternation in the sequence $\{SG_{1,2d}(n)\}_{n=1}$:

Alternating Property. For any $k > 0$,

$$SG_{1,2d}(2dk) = SG_{1,2d}(2dk-2) = \cdots = SG_{1,2d}(2dk-2d+2), \text{ and}$$

$$SG_{1,2d}(2dk-1) = SG_{1,2d}(2dk-3) = \cdots = SG_{1,2d}(2dk-2d+1).$$

Proof. Consider all of the SG values listed above: $SG_{1,2d}(2dk-2d+1)$ through $SG_{1,2d}(2dk)$. These values all depend on $SG_{1,2d}(k)$. Whatever the value of $SG_{1,2d}(k)$ is, none of the values above can be the same as it.

In our game, we have at most two moves available: subtract 1 or divide by $2d$. As a result, the SG sequence only ever takes one of three values: 0, 1 and 2. Since $SG_{1,2d}(k)$ has one of those values, there is a set of only 2 available values for $SG_{1,2d}(2dk)$ through $SG_{1,2d}(2dk-2d+1)$.

Since we know that $SG_{1,2d}(2dk)$ depends on $SG_{1,2d}(2dk-1)$, they cannot have the same SG value. Similarly, $SG_{1,2d}(2dk-1)$ depends on $SG_{1,2d}(2dk-2)$, so they

cannot have the same SG value. Since there are only two possibilities, it must be that $SG_{1,2d}(2dk) = SG_{1,2d}(2dk-2)$, and all other equalities follow by a similar argument. □

We will often want to represent a game with a digraph, where vertices correspond to the current total of the game and directed edges are moves players are allowed to make to change the current total. Figure 2.1 is a digraph representing the relevant portions of the game for the discussion of the Alternating Property. The letters next to the vertices are standing in for the SG values. While we cannot say if a given position has value 0, 1 or 2, we have established the relationships represented above. Visualizing it in this form shows the alternation more clearly.

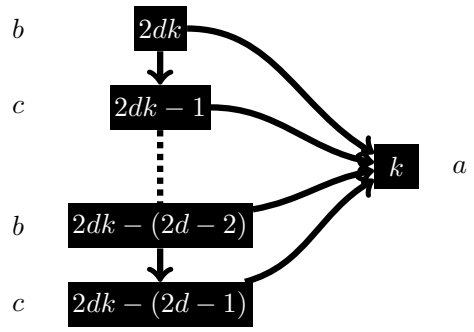


Figure 2.1: Digraph of $G_{1,2d}(n)$ exhibiting the Alternation Property.

Lemma 2.1. *Let $D = \{2d\}$, and let $n \equiv \ell \mod 4d$. If ℓ is even, $SG_{1,2d}(n) \neq 0$. If $\ell \in \{2d+1, 2d+3, \ldots, 4d-1\}$, then $SG_{1,2d}(n) = 0$.*

Proof. We will first show that if ℓ is even, $SG_{1,2d}(n) \neq 0$ by induction on n.

Base Case: $n = 2$ is clearly a winning game for the first player: he simply subtracts 1 from the total, and wins. Therefore $SG_{1,2d}(2) \neq 0$.

Given the Alternating Property, it is enough to show that if $SG_{1,2d}(2kd - 2d) \neq 0$, then $SG_{1,2d}(2kd) \neq 0$. The Alternating Property guarantees that the intervening values will be non-zero as well.

We have $SG_{1,2d}(2kd) = \text{mex}(SG_{1,2d}(2kd-1), SG_{1,2d}(k))$. If $SG_{1,2d}(k) = 0$, then it must be that $SG_{1,2d}(2kd) \neq 0$, so we assume that $SG_{1,2d}(k) \neq 0$. We know the values $SG_{1,2d}(2dk)$ through $SG_{1,2d}(2dk-2d+1)$ must alternate, and if $SG_{1,2d}(k) \neq 0$, it must be that those values alternate between zero and non-zero values.

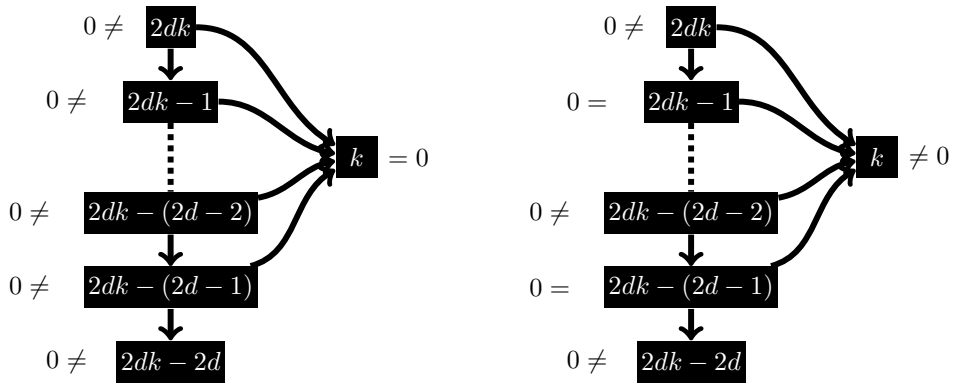


Figure 2.2: Visual representation of the proof of Lemma 2.1.

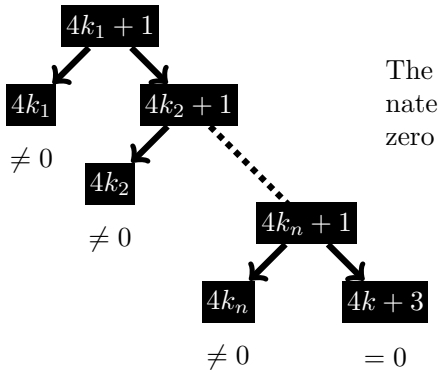
$SG_{1,2d}(2dk-2d+1) = \text{mex}(SG_{1,2d}(2dk-2d), SG_{1,2d}(k))$. Both terms in the mex we have assumed are non-zero, so $SG_{1,2d}(2dk-2d+1) = 0$. This means that the values alternate starting with 0, and so $SG_{1,2d}(2dk) \neq 0$. □

Lemma 2.2. *If $n \equiv \ell \mod 4d$ and $\ell \in \{1, 3, 5, \ldots, 2d-1\}$, the base $2d$ representation of n indicates whether or not $SG_{1,2d}(n) = 0$. Consider the block of even digits immediately following the first (least significant) odd digit of n. If that block has even length then $SG_{1,2d}(n) = 0$, and if it has odd length then $SG_{1,2d}(n) \neq 0$.*

Proof. First we notice that, given Lemma 2.1 above, $SG_{1,2d}(n-1) \neq 0$. For these values of n, whether or not $SG_{1,2d}(n) = 0$ will depend entirely on $SG_{1,2d}(\lceil \frac{n}{2d} \rceil)$: from the mex definition of $SG_{1,2d}(n)$ we have that if $SG_{1,2d}(\lceil \frac{n}{2d} \rceil) = 0$ then $SG_{1,2d}(n) \neq 0$, and if $SG_{1,2d}(\lceil \frac{n}{2d} \rceil) \neq 0$ then $SG_{1,2d}(n) = 0$.

Every time we divide n by $2d$ and round up, it has the following effect on the base $2d$ expansion of n: if $n = 1d_2d_3d_4\cdots$ (we may assume that the first digit of n is 1, by the Alternating Principle), then $\lceil \frac{n}{2d} \rceil = (d_2+1)d_3d_4\cdots$. By the Alternating Principle, we can further reduce this to $\lceil \frac{n}{2d} \rceil = 1d_3d_4\cdots$, and so the net effect of dividing by $2d$ and rounding up is simply to remove the second digit.

As illustrated in Figure 2.3, we can carry this process on until we have removed the entire block of even digits that immediately follows. At this point, we will either have a smaller value that has its first two digits odd, and we know by Lemma 2.1 that this has SG value zero, or we will have reduced it all the way to 1 which has SG value 0



The SG values along the right side will alternate between zero and non-zero, ending with zero in the lower right.

Figure 2.3: The descending sequence of odd index terms.

by definition. Since we end with a zero, if we have taken an even number of steps we must have started with a zero value, and if we have taken an odd number of steps we must have a non-zero value. The number of steps we take is exactly the length of the block of even digits. □

Lemmas 2.1 and 2.2 together prove Theorem 2.3 by giving a complete characterization of the zeros of $SG_{1,2d}(n)$.

The proof of Theorem 2.4, that develops similar results to Theorem 2.3 in the case of subtracting numbers other than 1, depends on some more patterns that arise in the SG sequence. In Section 2.2.2 we develop the concept of holding, which allows us to reduce $SG_{a,b}(n)$ to $SG_{1,2d}(n)$ for sufficiently large values of n. In Section 2.2.3 we develop the ideas which let us extend the results of Theorem 2.3, to prove that $\{SG_{a,2d}(an)\}_{n=1}$ is $2d$-automatic.

2.2.2 Holding

When we subtract values other than 1, we often have the property that adjacent groups of a certain length all (eventually) have the same SG value. For example, starting with $n = 1$, we have that $SG_{2,2}(n)$ eventually forms pairs of equal SG values:

$$SG_{2,2}(n) = 0, 2, 1, 0, 0, 2, 1, 1, 2, 2, 0, 0, 2, 2, 0, 0, 1, 1, 0, 0, 1, 1, 2, 2, \ldots$$

We call this phenomenon *holding* of length k, where k is the size of the groups. In the example above, $\{SG_{2,2}(n)\}_{n=1}$ has holding of length 2.

This section elaborates on the reasons behind the phenomenon of holding, and proves that under certain conditions we will be able to guarantee that holding of a particular length will eventually occur in the SG sequence. Once holding happens, we are really only interested in the value of each group, rather than every single index. In later sections, we will show conditions under which we can use the phenomenon of holding and the results about $SG_{1,2d}(n)$ to predict the value of $SG_{a,2d}(n)$.

Theorem 2.5. *Let a' be the largest divisor of a that is relatively prime to b. Then holding of length $s = \frac{a}{a'}$ will occur in $SG_{a,b}(n)$.*

We will prove this by first showing the persistence of holding, once it occurs, and then by showing that holding occurring once is inevitable. It is as if we are looking at a proof by induction out of order, with the inductive step first. The reason for this presentation is that, unlike a standard proof by induction, the proof of the 'base case' of the inevitability of holding is much more involved than the inductive step.

To do this, we develop the concept of subsequent blocks of indices.

Definition 2.2. *A* block *of indices is a set of numbers S where every term $SG_{a,b}(s)$ depends on the same value via division, for all $s \in S$.*

Definition 2.3. *Two blocks are* subsequent *if each value in one block depends on a value in the other block, via subtraction.*

The phenomenon of holding happens in sequences of subsequent blocks. It's easy to see that if all the SG values in one block are equal (assume they are all x), then every SG value in the subsequent block will be equal as well. Every value in the subsequent block depends on the same value under division (assume that is value y), and on a value in the initial block via subtraction. Therefore, every value in that block will be given by $\text{mex}(x, y)$, and so they must all be equal. If we have a long sequence of subsequent blocks, and the values in the first block are all equal, then that equality will persist throughout the whole sequence of blocks.

Note that because the term 'subsequent' refers to the dependence under subtraction, it does not imply adjacent. The first thing we must show is that we can cover all the

indices with sequences of subsequent blocks, and so the persistence of holding does occur.

Lemma 2.3. *If holding of length $g = \gcd(a, b)$ occurs, it will persist.*

Proof. Consider a set of terms with indices $kg, kg-1, \ldots, kg-g+1$, for some integer k. Because g divides b, this term is a block. Moreover, because g divides a, the set of terms with indices $kg-a, kg-a-1, \ldots, kg-a-g+1$ is also a block of length g. Every set of terms is part of a sequence of subsequent blocks.

Suppose we have holding occur in a block, that is for some m, $SG(mg) = SG(mg-1) = \cdots = SG(mg-g+1)$. Consider the subsequent block, $SG(mg+a), SG(mg+a-1), \ldots, SG(mg+a-g+1)$. Each term is the mex of a set of size 2. One element in the set is always identical across all terms, because they all depend on the same value under division. In this case, the value that they depend on via subtraction is identical as well, by assumption. Therefore it follows that $SG(mg+a) = SG(mg+a-1) = \cdots = SG(mg+a-g+1)$, and by induction this will occur for every block of the form $SG(mg+ka), SG(mg+ka-1), \ldots, SG(mg+ka-g+1)$. □

In Lemma 2.4, we show that holding will eventually occur in any sequence of subsequent blocks.

Lemma 2.4. *Stuttering of length $g = \gcd(a, b)$ is inevitable.*

Proof. We will create a digraph that models the behavior of subsequent blocks of length of g. Each vertex in this digraph will be a triple of SG values: (x, y, z). x and y are two values in the block, that we hope will eventually be equal. The block of course may have length larger than 2, but we only need to consider two values: if we can show that no matter what values x and y begin with that they will eventually end up equal, it will follow that a block of any length will eventually have all of its values be equal. The final term, z, is the SG value that the next block depends on via division.

For example, when $a = 4$ and $b = 2$, one triple could represent

$$(SG_{4,2}(19), SG_{4,2}(20), SG_{4,2}(12)) .$$

The subsequent block, $(23, 24)$, will depend on $(19, 20)$ via subtraction, and on 12 via division. So just from looking at one triple, we have all the information we need to tell us the SG values in the next block.

To create the digraph, we will add directed edges as shown in Figure 2.4. These represent what the values in the subsequent block must be, and allow for any possible z to accompany them.

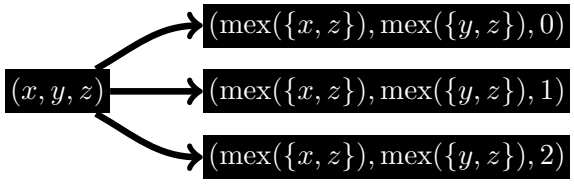


Figure 2.4: The directed edges in the digraph

The values in any sequence of subsequent blocks will correspond to a walk on this digraph. Fortunately, the digraph only has 27 vertices and 81 directed edges, so it is small enough to analyze by hand.

Figure 2.5 is a schematic of the interesting portions of the digraph. I have omitted vertices where $x = y$, because we know that once holding begins it will persist. Our primary concern, therefore, is how the graph behaves on the vertices where $x \neq y$, which reduces the number of vertices to 18. I have also depicted the following six vertices as sinks (and drawn them in gray): $(1, 2, 1)$, $(1, 2, 2)$, $(2, 1, 1)$, $(2, 1, 2)$, $(0, 2, 0)$, and $(2, 0, 0)$. For each of these, all three out edges point to vertices where $x = y$. For the first four $x = y = 0$, and for the last two $x = y = 1$. This further reduces the number of directed edges to consider, from 54 to 36.

We should be worried, since there are directed cycles where the first two terms are not zero, and an infinite walk around one of these cycles would correspond to an infinite sequence of blocks where holding does not appear. This cannot happen in practice, however.

Observe that in Figure 2.5, above the dotted line all the vertices (other than the sinks) have $z \neq 0$, and below the dotted line all the vertices (other than the sinks) have $z = 0$. There are no arrows that go from below the dotted line to above the dotted line,

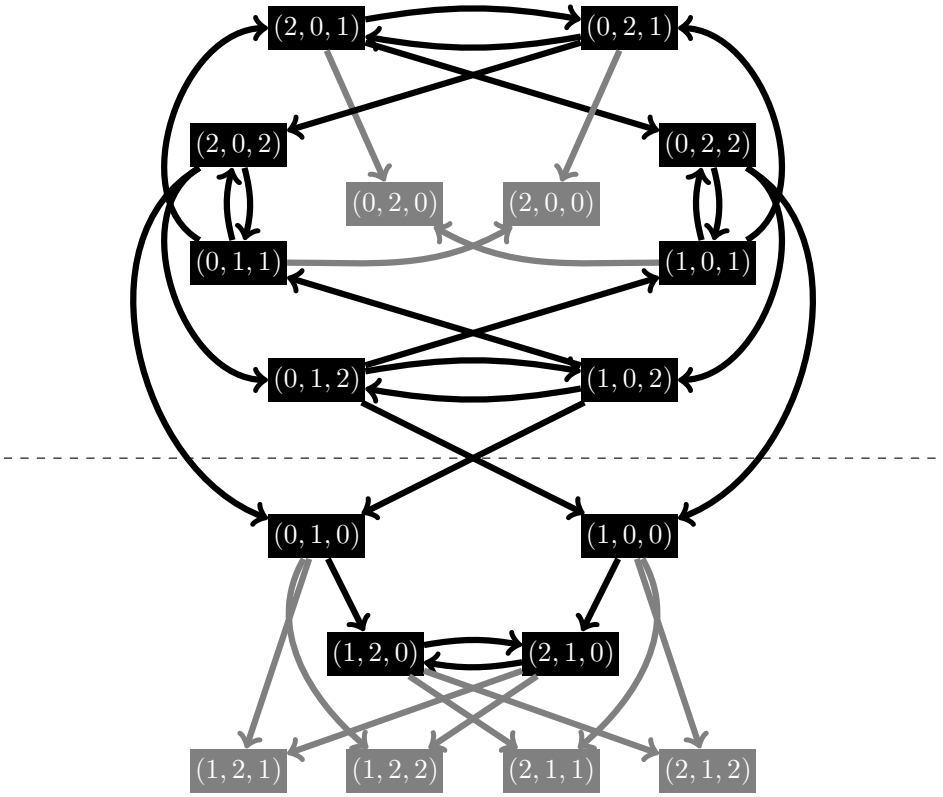


Figure 2.5: Digraph representing possible SG values in a sequence of subsequent blocks.

so any directed cycle must stay either above or below the line. Therefore, an infinite walk without holding will have an infinite stretch with $z = 0$, or an infinite stretch with $z \neq 0$.

We will show in the following lemma that z must vary between zero and non-zero values. This guarantees that wherever we begin, we will eventually pass to a sink or to a vertex below the dotted line. Moreover, once we are below the dotted line we must eventually move to a sink. Therefore, modulo proving Lemma 2.5, holding is inevitable. □

Lemma 2.5. *The value of z, above, cannot have an arbitrarily long sequence of zeros or an arbitrarily long sequence of non-zero values.*

Proof. Given a sequence of subsequent blocks, corresponding to a walk in the digraph, we will make a sequence of z values where z_i is the third term in the label of the ith vertex in the walk.

We see that this sequence depends on itself via subtraction: for every i, $z_i - a = z_{i-2d}$. The block of length g that depends on z_i via division is the block with indices $\{2dz_i$, $2dz_i - 1$, $\ldots$, $2dz_i - g + 1$, and the values that depend on $z_i - a$ via division are the block with indices $\{2dz_i - 2da$, $2dz_i - 2da - 1$, $\ldots$, $2dz_i - 2da - g + 1\}$. These two blocks are in the same sequence of subsequent blocks, and the one occurs exactly $2d$ steps before the other. Since we cannot have a zero value depending on another zero value, it's impossible to have an infinite stretch of zeros.

In theory, we could have an infinite stretch of non-zero values, but the only way this would be possible is if the value that z_i depended on under division was zero for all i. That sequence will also depend on itself via subtraction, however: $\lceil \frac{z_i}{2d} \rceil - a = \lceil \frac{z_i - 2da}{2d} \rceil = \lceil \frac{z_{i-2d}}{2d} \rceil$. Because it depends on itself via subtraction this less sparse sequence cannot be all zero, therefore the z_i cannot be all zero or non-zero. □

We have shown that holding will eventually occur in any sequence of subsequent blocks of length $g = \gcd(a, b)$. Because g divides a, these subsequent blocks of length g cover all of the indices. Therefore, holding of length g will occur throughout the entire sequence. To prove Theorem 2.5, however, we must show that we will get even longer blocks. Armed with Lemmas 2.3 and 2.4, we can now prove Theorem 2.5.

Proof of Theorem 2.5. Consider what happens in $SG_{a,b}(n)$, after holding of length g has occurred. Far enough out in the sequence, the values in each block of length g will be the same. It's natural, then, to think of the sequence not in terms of individual elements but in terms of the value of each block.

The number of blocks between two subsequent blocks is $\frac{a}{g} - 1$, so when we just look at the blocks, they are essentially behaving as if the rule of the game is now subtract $\frac{a}{g}$, rather than subtract a. The term that a block depends on under division, however, hasn't changed. Therefore, far enough out in the sequence, the blocks have the same underlying digraph as the game $G_{\frac{a}{g},b}(n)$.

The proof of the inevitability and persistence of holding tells us that if $\frac{a}{g}$ and b have any common factors, we would expect to see holding of length $\gcd(\frac{a}{g}, b)$. Since the elements are blocks, now, this would correspond to holding of length $g \cdot \gcd(\frac{a}{g}, b)$ in the

whole sequence.

This process continues for as long as the amount we subtract by continues to have common factors with b. This will end once we reach a', the largest divisor of a that is relatively prime to b. We will have holding of length $\frac{a}{a'}$, and far out in the sequence, the underlying digraph that determines the values of the blocks will be the same as the digraph of $G_{a',b}(n)$. □

It will be helpful to have a sense of how far out in the sequence we need to go before holding will occur. We get, by analyzing Lemma 2.5, that we can have at most $2d$ subsequent 0 values and at most $4d^2$ subsequent non-zero values in the sequence of z values. This implies that we can take at most $4d^2 + 2d + 2$ steps before holding occurs. Once holding of length g occurs, however, we must walk out another $2dg$ steps before we have the potential for additional holding.

Suppose we first see holding occur of length g_1, then additional holding of length $g_1 \cdot g_2$ occurs, etc., and let $g_1, g_2, g_3, \ldots, g_k$ be the whole sequence of successive lengths of holding that we see. Note that $g_i = \gcd\left(\frac{a}{g_1 g_2 \cdots g_{i-1}}, b\right)$, and so for all i, $g_i \geq g_{i+1}$. We obtain the following upper bound on the number of steps that occurs before we experience holding of length $g_1 g_2 \cdots g_k$:

$$\begin{aligned} N(a,b) =& (4d^2 + 2d + 2) \\ & \cdot \Big((2d)^k \cdot (g_1 g_2 \cdots g_k) + (2dg)^{k-1} \cdot (g_1 g_2 \cdots g_{k-1}) + \cdots + 2dg_1 + 1 \Big) \end{aligned}$$

While for any pair a and $2d$ we could compute the specific upper bound $N(a,b)$, as a rough measure we can certainly take $a(2d)^a$ as an upper bound for the number of steps. In general, however, even the more exact bound is much larger than what's needed. The same analysis yields a corresponding lower bound of $a(2d)^k$, where k is the number of times we see additional holding (this is assuming we immediately see holding, as soon as it becomes possible). Experimental evidence suggests that in practice the real value is closer to the lower bound than the upper.

2.2.3 The Misére Game and Changed Initial Values

The results from Section 2.2.2 tell us that if all the prime factors of a also appear as prime factors of $2d$ (i.e. $a' = 1$), then the $G_{a,b}(n)$ will eventually behave like $G_{1,2d}(n)$. But it's not immediately clear how the characterization of the first player losing positions in $G_{1,2d}(n)$ can be used to characterize the first player losing positions in $G_{a,2d}(n)$. It turns out that we will need a more involved understanding of the structure of the simpler game, much of the details of which are postponed until Section 2.3. However in this section we will discuss how the two games are related, what results from above can be extended directly to $SG_{a,b}(n)$, and how the upcoming results can be used to completely characterize $SG_{a,b}(n)$.

Since we are studying a sequence that has eventual holding of length a, we will start by breaking the full sequence up into subsequences based on the residue classes mod a. That is, one sequence will be the terms with indices $a, 2a, 3a$, etc., another will be the terms with indices $1, a+1, 2a+1, 3a+1$, etc., and so on. Stuttering tells us that there is a value N, such that these sequences are equal for all (subsequence) indices greater than N, so it doesn't matter which residue class we consider. For the rest of this section, we will look at only one (unspecified) residue class, and will take the view that our sequence of interest is made by taking $N-1$ fixed and completely arbitrary values from the set {0,1,2}, and then defining all subsequent terms by the usual recursive definition for $G_{1,2d}(n)$:

$$x_n = \operatorname{mex}\left(x_{n-1}, x_{\lceil \frac{n}{2d} \rceil}\right).$$

In a slight abuse of notation, we will continue to denote this function $SG_{1,2d}(n)$.

We would like to prove an analog of Lemma 2.1 for $SG_{1,2d}(n)$ when the first $N-1$ values have been set arbitrarily, that is, that even index terms will be non-zero, and terms with indices in the range $n \equiv 2d+1, 2d+3, \ldots, 4d-1 \mod 4d$ will be zero. Of course, we have no control over the first $N-1$ terms, as they may be set arbitrarily (and can perhaps be set in such a way that will influence subsequent terms), so the best we can hope for is that the results of Lemma 2.1 will hold for all indices above some threshold.

Lemma 2.6. *Let $SG_{1,2d}(n)$ denote the nth value of the SG sequence for the game $G_{1,2d}(k)$ where the first $N-1$ terms have been set arbitrarily, and take ℓ, such that $n \equiv \ell \mod 4d$. If ℓ is even and $n \geq 2dN$, then $SG_{1,2d}(n) \neq 0$. If $\ell \in \{2d+1, 2d+3, \ldots, 4d-1\}$ and $n \geq 4d^2N - 2d + 1$, then $SG_{1,2d}(n) = 0$.*

Proof. As in the proof of Lemma 2.1, we have an inductive argument for a winning strategy for even index terms. This essentially says that as long as $SG_{1,2d}(2dk + 2d)$ through $SG_{1,2d}(2dk - 2d + 1)$ are all determined by a mex rather than fixed arbitrarily (i.e. $2dk \geq N+2d-1$), and $SG_{1,2d}(2dk) \neq 0$, it must be that $SG_{1,2d}(2dk+2d) \neq 0$. From the alternating property, we get the intermediate even index terms to have non-zero SG values as well.

What we need to show is that the base case of the induction occurs at some point after $SG_{1,2d}(N-1)$ and no later than at $SG_{1,2d}(2dN)$. Suppose, then, that the SG value is zero at all even indices from $N-1$ through $2dN-2$. The value $SG_{1,2d}(2dN)$ is determined by the mex over a set that includes $SG_{1,2d}(n)$, which we have assumed to be zero. Since the mex of a set that includes zero cannot be zero, $SG_{1,2d}(2dN)$ must be non-zero. This guarantees that the base case for the induction must occur at some point after $SG_{1,2d}(N-1)$ and no later than at $SG_{1,2d}(2dN)$, and the inductive step above finishes the proof of the first part of the lemma.

The second part of the lemma depends on the first part. We know that for indices where $\ell \in \{2d+1, 2d+3, \ldots, 4d-1\}$, both of the SG values that $SG_{1,2d}(n)$ depends on will have even index. For very large values of n, the first part of the lemma will apply to both of those smaller SG values, and we will know that they are both non-zero. This implies that for large values of n, $SG_{1,2d}(n) = 0$. Based on the threshold in the first part of the lemma, we find the smallest index where this is guaranteed to happen is $n = 4d^2N - 2d + 1$: $\lceil \frac{4d^2N-2d+1}{2d} \rceil = 2dN$. □

We will now prove Theorem 2.4, by finishing the characterization of $SG_{1,2d}(n)$ for the remaining indices: $n \equiv \ell \mod 4d$, and $\ell \in \{1, 3, \ldots, 2d-1\}$:

Lemma 2.7. *If $n \equiv \ell \mod 4d$ and $\ell \in \{1, 3, \ldots, 2d-1\}$, the least significant digit in the base $2d$ representation of n is odd, and the second least significant digit is even. For*

$n \geq 4dN + 1$, let $n' \geq 2dN + 1$ be smallest number possible to be formed from n by successive removals of the second digit in the base $2d$ expansion, as long as the second digit remains even and $n' \geq 2dN + 1$. Let k be the number of even digits removed from n to form n'.

- *If $SG(n') = 0$, then $SG(n) = 0$ if k is even and $SG(n) \neq 0$ if k is odd.*
- *If $SG(n') \neq 0$, then $SG(n) \neq 0$ if k is even and $SG(n) = 0$ if k is odd.*

Proof. If we write the base $2d$ representation of n with the least significant digit occurring first, then we know (by the restrictions on ℓ) that the first digit is odd and the second digit is even. As in the proof of the analogous Lemma 2.2, we will use the fact that every time we divide n by $2d$ and round up, it has the effect of removing the second digit from the base $2d$ expansion of n. For sufficiently large n, $SG_{1,2d}(n-1)$ will be non-zero, and so whether or not $SG_{1,2d}(n) = 0$ will depend on $SG_{1,2d}(\lceil \frac{n}{2d} \rceil)$.

We would like to remove the entire first block of even digits from the base $2d$ expansion of n, but unlike in the proof of Lemma 2.2 we need to worry about what happens if in removing all the even digits leaves us with a number that is too small.

We form n' by successively removing the second digit of n, until we either run out of even digits in that first block or we get to an index n' where $4d^2N+1 > n' \geq 2dN+1$. If we drop below $2dN + 1$, we are no longer guaranteed that $SG_{1,2d}(n'-1) \neq 0$. Without this, it may not be the case that if $SG_{1,2d}(\lceil \frac{n}{2d} \rceil) = 0$ then $SG_{1,2d}(n) \neq 0$, and vice versa.

If $n' \geq 4d^2N + 1$ it must be that we have stopped because we have removed the entire first block of even digits, and so $SG_{1,2d}(n') = 0$. If $4dN + 1 > n' \geq 2dN + 1$, we must look up $SG_{1,2d}(n')$ in a pre-computed table of the first $4d^2N$ values. In either case, we will have alternated between zero and non-zero SG values every time we removed a digit, so knowing the value of $SG_{1,2d}(n')$ and the number of digits we have removed allows us to compute $SG_{1,2d}(n)$. □

Proof of Theorem 2.4. We will analyze the sequence $\{SG_{a,2d}(n)\}_{n=1}$ when the largest divisor of a that is relatively prime to b is 1, by first using holding to reduce it to the

sequence $\{SG_{a,2d}(an)\}_{n=1}$. We have shown in Section 2.5 that for these values of a and $2d$ we will see holding of length a after at most $a(2d)^a$ steps. Once this holding occurs, we only need to consider the value of each block, which will be picked up by the subsequence $\{SG_{a,2d}(an)\}_{n=1}$. The blocks will behave as if they are in $G_{1,2d}$. This means that we represent this sequence by the behavior of $SG_{1,2d}(n)$, with the first $(2d)^a$ values set arbitrarily.

Lemmas 2.6 and 2.7 together give us a complete characterization of $SG_{1,2d}(n)$ after the first $(2d)^a$ values have been set arbitrarily. □

Note, however, that this is a somewhat less satisfying characterization than we have for Theorem 2.3. If $n \equiv \ell \mod 4d$ and either ℓ even or $\ell \in \{2d+1, 2d+3, \ldots, 4d-1\}$, the characterization merely gives a threshold, after which the zeroes of $SG_{1,2d}(n)$ follow a simple pattern that doesn't involve looking up values in a pre-computed table. But if $\ell \in \{1, 3, \ldots, 2d-1\}$, there is no threshold above which we can avoid the table.

As an example, consider the misére version of $G_{1,2}(n)$, defined by keeping the same allowable moves but switching the goal of the players; in this game, the player who says 1 loses. The SG sequence for any misére game is given by defining the SG values of the final positions of the game to be 1 (rather than 0, as we do in the regular game), and then defining all subsequent SG values recursively, as usual. The misére version of $G_{1,2}(n)$ falls into this section naturally, as it has SG sequence $\{SG_{1,2}(n)\}_{n=1}$ with the initial value changed from $SG_{1,2}(1) = 0$ to $SG_{1,2}(n) = 1$.

For the misére game, $SG_{1,2}(3) \neq 0$, but all other SG values when $n \equiv 3 \mod 4$ are zero (as is the case in the regular version of the game). However, this single aberration at $n = 3$ causes an infinite number of other SG values with indices $n \equiv 1 \mod 4$ to change from zero to non-zero, or vice versa. All numbers that are one more than a power of 2, or equivalently numbers n with binary representation 10^i1 for some $i \geq 0$ will have $n' = 3$. We can only determine their correct SG value by looking up the value $SG_{1,2}(3)$ in a pre-computed table that tells us that $SG_{1,2}(3) \neq 0$.

2.3 Regularity of the Game Sequences

John Paul Allouche and Jeffry Shallit first introduced the concept of regular sequences in a pair of 1992 articles: [6] and [7]. Regular sequences are a large and reasonably well-behaved and well-understood class of sequences. Ultimately, we will show that many of the games sequences we care about are regular, so we pause here to develop some of the results we will need. This introduction draws from the book *Automatic Sequences* by Allouche and Shallit [5], and the reader is directed there for more in depth explanations of these concepts.

A sequence is said to be *k-regular* if all residue classes modulo large powers of k are linear combinations of residue classes modulo smaller powers of k. More formally, we have the following (Definition 16.1.2 in [5]):

Definition 2.4. *A sequence a_n is* k-regular *if it can be completely defined by recurrences of the following form:*

$$a_{k^m+r} = \sum_{i=1}^{L} c_i a_{k^{m_i}+r_i},$$

where $m > m_i$, $0 \leq r \leq k^m - 1$, and $0 \leq r_i \leq k^{m_i} - 1$, and the c_i can be any constants.

Note that we do not require the smaller powers of k to be equal to each other, or that we take the same linear combinations for different residue classes.

Example 2.1. We will show that the Thue–Morse sequence is 2-regular. For $n \geq 0$, let the nth term in the Thue-Morse sequence $t(n)$ be the number of 1s in the binary representation of n, modulo 2. The common recursive definition of $t(n)$ is as follows: $t(0) = 0$, $t(2n) = t(n)$, and $t(2n+1) = 1 - t(n)$. This does not fit the framework of a 2-regular sequence, but we also have the following possible definition of $t(n)$:

- $t(2n) = t(n)$,
- $t(4n+1) = t(2n+1)$,
- $t(4n+3) = t(n)$.

Because the sequence can be entirely defined by equating residue classes modulo large powers of 2 with residue classes modulo smaller powers of 2, this fits the definition of a 2-regular sequence.

Since we are interested in generalizing the idea of a closed formula, we are interested in the following concept as well (Definition 5.1.1 in [5]).

Definition 2.5. *A k-automatic sequence is one for which there is a deterministic finite automaton with output, or DFAO, that when given the digits of n in base k will return the value of a_n.*

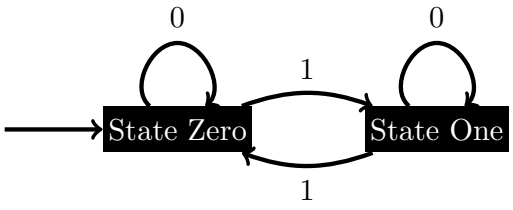
While there may not be a closed form for the nth term in an automatic sequence, the DFAO provides non-recursive (and hence generally much faster) way to compute the nth digit much in the way that a closed formula would.

It turns out that regular and automatic sequences are deeply related. We are particularly interested in the following related result (Theorem 16.1.5 in [5]):

Theorem 2.6. *If a sequence is k-regular and takes only a finite number of values, it is k-*automatic.

We refer the reader to to [5] for the proof.

Example 2.2. We have shown that the Thue–Morse sequence is 2-regular. Because it only takes values 0 or 1, it follows that it must also be 2-automatic. The following DFAO (based on Figure 5.1 in [5]) will compute $t(n)$, when fed in the binary representation of n. To see that this DFAO represents the Thus-Morse sequence, we will go back to the original understanding of the sequence as representing the parity of the number of 1's in the binary representation of n.



The binary representation of n is read in to the DFAO, starting at State Zero. With each new digit, we follow the appropriately labeled arrow from that state. When there

are no more digits in the binary representation of n, if we are in State Zero we return $t(n) = 0$, and if we are in State One we return $t(n) = 1$.

We will show that the sequence $\{SG_{1,2d}(n)\}_{n=1}$ is $2d$-regular by building up the family of recurrences that defines it. From this, we can prove Theorem 2.1: $\{SG_{1,2d}(n)\}_{n=1}$ is $2d$-automatic. Based on this result, and the results of Section 2.2.3, we will go further and prove Theorem 2.2: if all prime factors of a are also prime factors of $2d$, then $\{SG_{a,2d}(an)\}_{n=1}$ is $2d$-automatic.

2.3.1 Proof of Regularity

We start by showing the following:

Theorem 2.7. *The sequence $\{SG_{1,2d}(n)\}_{n=1}$ is $2d$-regular.*

Throughout this section we will simply write $SG(n)$, when we mean $SG_{1,2d}(n)$, since the context is clear. Let $n = R + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots$, where all the coefficients c_i are integers in $[0, 2d)$. This is equivalent to looking at the digits in the base-$2d$ representation of n, but we will write it as a polynomial so that when necessary we can borrow between terms.

We're looking for *reductions*, or equations of the form $SG(n) = SG(r)$, for $r < n$. Of course, there are only three possible SG values, so there are many equations we can write of this form. But we would like to find large classes of these relations, where by only looking at the first few terms in the polynomial of n (that is, the least significant digits in base $2d$) we can write down a formula for r in terms of n. These sets of equations will lead us to recurrences of the form $SG((2d)^k + i) = SG((2d)^m + j)$.

In particular, we're interested in reductions where to write r we have to delete digits from n, as this will give us equalities between residue classes modulo different powers of $2d$. We define the *shift* of an equation of the form $SG((2d)^k + i) = SG((2d)^m + j)$ to be $k - m$, which is equivalent to the number of digits we must delete from n to find r. Looking at the shift will be useful to help keep track of what happens when we apply multiple reductions, and helps verify that an equation of the form $SG((2d)^k + i) =$

$SG((2d)^m + j)$ is in fact a reduction. If the shift is greater than zero, it's clear that $(2d)^k + i > (2d)^m + j$. If the shift is zero, we must check that $i > j$.

To find these reductions, all we will use is Lemma 2.1, the mex definition of $SG(n)$, and the following basic facts about mex:

- If $a \in \{1, 2\}$, then $\text{mex}(0, a) = 3 - a$.
- $\text{mex}(0, \text{mex}(0, 1)) = 1$, and $\text{mex}(0, \text{mex}(0, 2)) = 2$.

To help readability, we adopt the convention that when we write out the definition $SG(n)$, the first term will always be the term corresponding to subtraction: $SG(n-1)$, and the second term will always be the term corresponding to division: $SG\left(\lceil \frac{n}{2d} \rceil\right)$. In cases where we have to work with each of these terms independently, we will refer to the first term as the Left-Hand Side (LHS) of definition, and the second term as the Right-Hand Side (RHS).

To show the pattern of these proofs, we first work through the following example, valid only for the sequence $\{SG_{1,2}(n)\}_{n=1}$:

Example 2.3. We will show that for all values of c_4, c_5, etc.,
$SG_{1,2}(0 + 16c_4 + 32c_5 + \cdots) = SG_{1,2}(0 + 4c_4 + 8c_5 + \cdots)$:

$$
\begin{aligned}
&SG_{1,2}(0 + 16c_4 + 32c_5 + \cdots) = \\
&\qquad = \text{mex}\left(SG_{1,2}(-1 + 16c_4 + 32c_5 + \cdots), SG_{1,2}(0 + 8c_4 + 16c_5 + \cdots)\right) \\
&\qquad = \text{mex}\left(0, SG_{1,2}(0 + 8c_4 + 16c_5 + \cdots\right) \qquad \text{Lemma 2.1} \\
&\qquad = \text{mex}\left(0, \text{mex}\left(SG_{1,2}(-1 + 8c_4 + 16c_5 + \ldots), SG_{1,2}(0 + 4c_4 + 8c_5 + \cdots)\right)\right) \\
&\qquad = \text{mex}\left(0, \text{mex}\left(0, SG_{1,2}(0 + 4c_4 + 8c_5 + \cdots\right)\right) \qquad \text{Lemma 2.1}
\end{aligned}
$$

Since $SG_{1,2}(0 + 4c_4 + 8c_5 + \cdots) \neq 0$, by Lemma 2.1, we can apply the 2nd principle of mex, to see that $\text{mex}\left(0, \text{mex}\left(0, SG_{1,2}(0 + 4c_4 + 8c_5 + \cdots\right)\right)\right) = SG_{1,2}(0 + 4c_4 + 8c_5 + \cdots)$. Therefore, $SG_{1,2}(0 + 16c_4 + 32c_5 + \cdots) = SG_{1,2}(0 + 4c_4 + 8c_5 + \cdots)$.

The argument above holds for any values c_4, c_5, etc., which this tells us that the subsequence $\{SG_{1,2}(16n)\}_{n=1}$ equals the subsequence $\{SG_{1,2}(4n)\}_{n=1}$, term for term. Of course, if we add any other conditions on c_4 or coefficients of higher terms, we will still

have equality. So, for example, if we restrict to $c_4 = 0$ we get that $\{SG_{1,2}(32n)\}_{n=1} = \{SG_{1,2}(8n)\}_{n=1}$, and if we restrict to $c_4 = 1$ we get that $\{SG_{1,2}(32n+16)\}_{n-1} = \{SG_{1,2}(8n+4)\}_{n=1}$. Therefore this one argument creates many reductions, all with shift 2.

We now address the general case, $\{SG_{1,2d}(n)\}_{n=1}$. The Alternating Property allows us to reduce the amount of work we need to do: we will consider only cases where $R = 0$ and $R = 1$. Moreover, motivated by Lemma 2.1 and how important the value of n mod $4d$ appears to be, we will break up our discussion of n into the following 4 cases: $R = 0$ and c_1 even, $R = 1$ and c_1 even, $R = 0$ and c_1 odd, and $R = 0$ and c_1 even.

One case is trivial:

Case 1: $R = 1$ and c_1 odd. In this case, $SG(n) = 0$, by Lemma 2.1.

Just showing that a sequence is uniformly zero is not, strictly speaking, a reduction. In fact, the reduction will be that anything in Case 1 (or anything else that is uniformly zero) will reduce to $SG(1+2d+4d^2n)$. Knowing that it is uniformly zero gives us more information, however, particularly as we use this result in examining other cases.

For each of the remaining three cases, we will build up a family of reductions. We start with the case where $R = 1$ and c_1 even, and use those reductions to build up the family for $R = 0$, c_1 odd. Those, in turn, help with the final case: $R = 0$, c_1 even.

2.3.2 Case 2: $R = 1$, c_1 even

In this case, we have the following definition of $SG(n)$:

$$SG(n) = \operatorname{mex}\left(SG(2dc_1 + 4d^2c_1 + \cdots), SG(1 + c_1 + 2dc_2 + 4d^2c_3 + \cdots)\right)$$

Since c_1 is even (by assumption) we have that the second term in the definition is equivalent to $SG(1 + 2dc_2 + 4d^2c_3 + \cdots)$, by the Alternating Property.

In the first part of this section, we will build up reductions by constraining c_2, c_3 and c_4 to be even or odd. At each step, we hope to use the conditions on c_2 through c_4 and Lemma 2.1 to determine whether or not particular terms in the mex are zero or non-zero. As long as one of these coefficients is odd, this is possible, which gives us the first four reductions in the table below.

When $c_1 \neq 0$ and c_2 through c_4 are all even, however, we get some interesting behavior, for which we define a new function: $SG^*(n) = 3 - SG(n) \mod 3$. This new function essentially keeps 0 values the same, but switches the values 1 and 2. The second part of this section develops some more properties $SG^*(n)$, and how it can be used to prove reductions. By adding constraints on c_5 and c_6 and reapplying Reductions 1 through 4, we are able to get the complete set of results recorded in Table 2.3.2

Reductions for $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots)$ when c_1 even:		
R1	$SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)$	c_2 odd
R2	$0 = SG(1 + 2d + \cdots$	c_2 even, c_3 odd
R3	$SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)$	c_2, c_3 even, c_4 odd
R4	$SG(1 + 2dc_3 + 4d^2c_4 + \cdots)$	$c_1 = 0$, c_2, c_3, c_4 even

Table 2.1: The four reductions for Case 2 that can be proved directly.

We first prove Reductions 1 through 4, directly and constructively.

Reduction 1. *When c_1 is even and c_2 is odd, $SG(1 + 2kc_1 + 4d^2c_2 + 8d^3c_3 + \cdots) = SG(c_1 + 2dc_2 + \cdots)$.*

Proof.

$$
\begin{aligned}
&SG(1 + 2kc_1 + 4d^2c_2 + 8d^3c_3 + \cdots) \\
&\quad = \operatorname{mex}\left(SG(2dc_1 + 4d^2c_2 + \cdots), SG(1 + c_1 + 2dc_2 + 4d^2c_3 + \cdots)\right) \\
&\quad = \operatorname{mex}\left(SG(2dc_1 + 4d^2c_1 + \cdots), 0\right) \qquad \text{Lemma 2.1} \\
&\quad = \operatorname{mex}\left(\operatorname{mex}\left(SG(-1 + 2dc_1 + 4d^2c_2 + \cdots), SG(c_1 + 2dc_2 + \cdots)\right), 0\right) \\
&\quad = \operatorname{mex}\left(\operatorname{mex}\left(0, SG(c_1 + 2dc_2 + \cdots)\right), 0\right) \qquad \text{Lemma 2.1}
\end{aligned}
$$

Since $SG(c_1 + 2dc_2 + \cdots)$ is never zero, we have that $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots) = SG(c_1 + 2dc_2 + \cdots)$. □

Reduction 2. *When c_1 and c_2 are even, c_3 is odd, $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots) = 0$.*

Proof.

$$\begin{aligned}
&SG(1+2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots) \\
&\qquad = \operatorname{mex}\left(SG(2dc_1 + 4d^2c_2 + \cdots), SG(1 + c_1 + 2dc_2 + 4d^2c_3 \cdots)\right) \\
&\qquad = \operatorname{mex}\Big(\operatorname{mex}\left(SG(-1 + 2dc_1 + 4d^2c_2 + \cdots), SG(c_1 + 2dc_2 + \cdots)\right), \\
&\qquad\qquad\qquad \operatorname{mex}\left(SG(c_1 + 2dc_2 + \cdots), SG(1 + c_1 + 2c_2 + 4dc_3 + \cdots)\right)\Big) \\
&\qquad = \operatorname{mex}\left(\operatorname{mex}\left(0, SG(2c_1 + 4dc_2 + \cdots)\right), \operatorname{mex}\left(SG(2c_1 + 4dc_2 + \cdots), 0\right)\right) \\
&\qquad = 0
\end{aligned}$$

Since $\operatorname{mex}\left(0, SG(2c_1 + 4dc_2 + \cdots)\right) \neq 0$ and $\operatorname{mex}\left(SG(2c_1 + 4dc_2 + \cdots), 0\right) \neq 0$, we have that the original is the mex of two non-zero elements, and so must always be zero. □

Reduction 3. *When c_1, c_2, c_3 are even, c_4 is odd, $SG(1+2dc_1+4d^2c_2+8d^3c_3+\cdots) = SG(c_1 + 2dc_2 + \cdots)$.*

Proof.

$$\begin{aligned}
&SG(1+2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) \\
&\qquad = \operatorname{mex}\left(SG(2dc_1 + 4d^2c_2 + \cdots), SG(1 + c_1 + 2dc_2 + \cdots)\right) \\
&\qquad = \operatorname{mex}\Big(\operatorname{mex}\left(SG(-1 + 2dc_1 + 4d^2c_2 + \cdots), SG(c_1 + 2dc_2 + \cdots)\right), \\
&\qquad\qquad\qquad SG(1 + c_1 + 2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)\Big) \\
&\qquad = \operatorname{mex}\left(\operatorname{mex}\left(0, SG(c_1 + 2dc_2 + \cdots\right), SG(1 + 2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)\right).
\end{aligned}$$

But by the Alternating Property and R2, we know that $SG(1 + c_1 + 2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots) = 0$, so

$$\begin{aligned}
&SG(1+2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) \\
&\qquad = \operatorname{mex}\left(\operatorname{mex}\left(0, SG(c_1 + 2dc_2 + \cdots)\right), SG(1 + 2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)\right) \\
&\qquad = \operatorname{mex}\left(\operatorname{mex}\left(0, SG(c_1 + 2dc_2 + \cdots)\right), 0\right).
\end{aligned}$$

Since $SG(c_1 + 2dc_2 + \cdots)$ is never zero (by assumption, c_1 even), we have that $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots) = SG(c_1 + 2dc_2 + \cdots)$. □

Reduction 4. *: If $c_1 = 0$, c_2, c_3, c_4 all even, $SG(1+2dc_1+4d^2c_2+8d^3c_3+16d^4c_4+\cdots) = SG(1 + 2dc_3 + 4d^2c_4 + \cdots)$.*

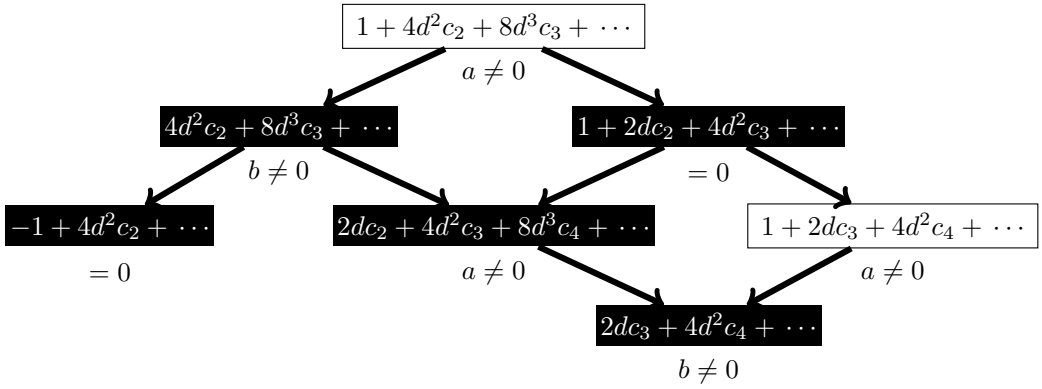
Proof.

$$
\begin{aligned}
&SG(1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) \\
&\quad = \operatorname{mex}\left(SG(4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots), SG(1 + 2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)\right) \\
&\quad = \operatorname{mex}\Big(\operatorname{mex}\left(SG(-1 + 4d^2c_2 + 8d^3c_3 + \cdots), SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)\right), \\
&\qquad\qquad \operatorname{mex}\left(SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots), SG(1 + 2dc_3 + 4d^2c_4 + \cdots)\right)\Big) \\
&\quad = \operatorname{mex}\Big(\operatorname{mex}\left(0, SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)\right), \\
&\qquad\qquad \operatorname{mex}\left(SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots), SG(1 + 2dc_3 + 4d^2c_4 + \cdots)\right)\Big)
\end{aligned}
$$

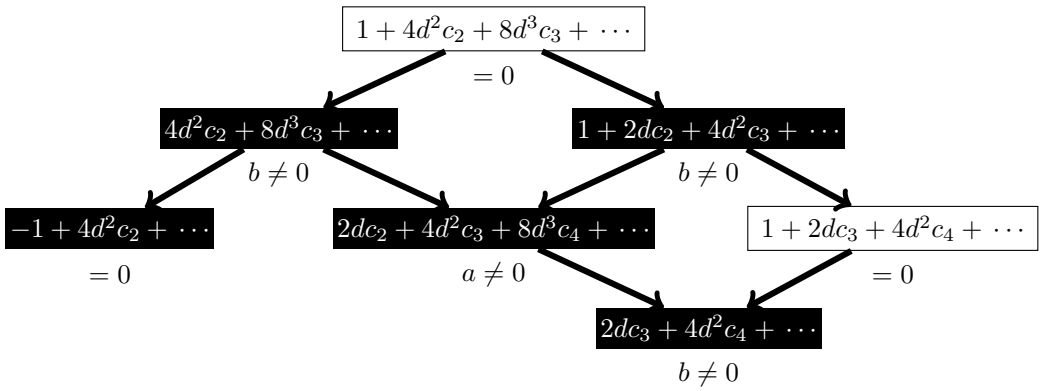
Notice that the term $SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)$ shows up in both terms of the outer mex, and can never be zero. Further notice that $SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)$ and $SG(1 + +2dc_3 + 4d^2c_4 + \cdots)$ both depend on $SG(2dc_3 + 4d^2c_4 + \cdots) \neq 0$.

We are using the fact that $c_1 = 0$. If $c_1 \neq 0$, then instead of $SG(2dc_2 + 4d^2c_3 + \cdots)$ showing up in both terms of the mex we would have $SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)$. This term would depend directly on $SG(1 + 2dc_3 + 4d^2c_4 + \cdots)$, rather than both of them depending on a separate value.

We have two cases to consider: $SG(1 + 2dc_3 + 4d^2c_4 + \cdots) \neq 0$, and $SG(1 + 2dc_3 + 4d^2c_4 + \cdots) = 0$. Below are diagrams of the underlying digraphs of the game, and what we can determine about the rest of the SG values given $SG(1 + 2dc_3 + 4d^2c_4 + \cdots)$ is zero or non-zero.



If $SG(1 + 2dc_3 + 4d^2c_4 + \cdots) = a \neq 0$, then $SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots) = a$, and so $SG(1 + 4d^2c_2 + \cdots) = \text{mex}\big(\text{mex}(0, a), \text{mex}(a, a)\big) = \text{mex}\big(\text{mex}(0, a), 0\big) = a$.



If $SG(1 + 2dc_3 + 4d^2c_4 + \cdots) = 0$, then both terms of the outer mex are given by $\text{mex}\big(0, SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)\big)$, and are non-zero. So the original value must be zero as well.

In both cases, the value of the original is the same as $SG(1 + 2dc_3 + 4d^2c_4 + \cdots)$, so we have $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG(1 + 2dc_3 + 4d^2c_4 + \cdots)$. □

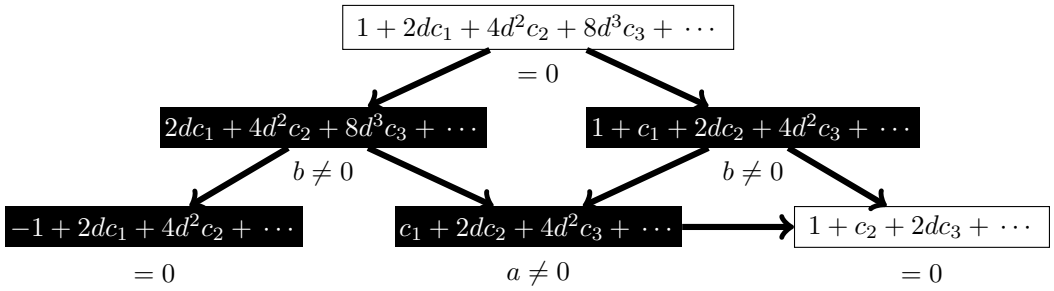
As we consider the single remaining case so far, that is, $c_1 \neq 0$, and c_2, c_3, c_4 all even, we start to notice some interesting behavior.

$$
\begin{aligned}
&SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) \\
&\quad = \text{mex}\big(SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots), SG(1 + c_1 + 2dc_2 + 4d^2c_3 + \cdots)\big) \\
&\quad = \text{mex}\big(\text{mex}\big(SG(-1 + 2dc_1 + 4d^2c_2 + \cdots), SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)\big), \\
&\qquad\qquad \text{mex}\big(SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots), SG(1 + c_2 + 2dc_3 + \cdots)\big)\big) \\
&\quad = \text{mex}\big(\text{mex}\big(0, SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)\big), \\
&\qquad\qquad \text{mex}\big(SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots), SG(1 + 2dc_3 + \cdots)\big)\big)
\end{aligned}
$$

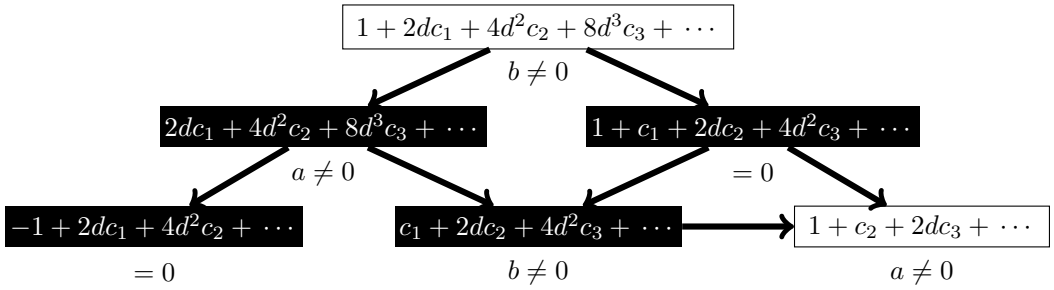
Note that the term $SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)$ can never be zero, and shows up in both terms of the outer mex. We also see that $SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)$ depends on the other term in the mex, $SG(1 + 2dc_3 + \cdots)$, rather than a third value. Here we are using the fact that $c_1 \neq 0$.

There are two cases to consider: $SG(1 + c_2 + 2dc_3 + \cdots) = 0$, and $SG(1 + c_2 +$

$2dc_3 + \cdots) \neq 0$. These determine the SG values for all of the other vertices in question. Below are the digraphs of the game, with what we know about the SG values filled out.



When $SG(1+c_2+2dc_3+4d^2c_4+\cdots) = 0$, $SG(1+2dc_1+4d^2c_2+8d^3c_3+16d^4c_4+\cdots)$ is the mex over two non-zero elements, and so must be zero as well.



When $SG(1 + c_2 + 2dc_3 + 4d^2c_4 + \cdots) = a \neq 0$, we must have that $SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots) = b \neq 0$.

We can see that there is a nice relationship between $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots)$ and $SG(1 + c_2 + 2dc_3 + 4d^2c_4 \cdots)$. To describe this behavior, we create a new function $SG^*(n)$ by assigning $SG^*(n) = (3 - SG(n)) \mod 3$. In this way, $SG^*(n)$ and $SG(n)$ are either both zero, or both non-zero and not equal. Note that $SG^{**}(n) = SG(n)$. We call this new function the starred value of $SG(n)$, or the opposite value if we are looking at a class of values n where $SG(n) \neq 0$. Using this new function, we can summarize what we have learned with the following: If $c_1 \neq 0$, $c_1, \ldots, c_4$ even,

$$SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG^*(1 + 2dc_3 + 4d^2c_4 + \cdots). \quad (2.1)$$

We will call Equation 2.1 "Rule 5". We would like to make this Reduction 5, but unfortunately it isn't a reduction: we don't get an equality with SG of some smaller index, but SG^*. Notice, however, the similarity between Reduction 4 and Rule 5: they

are both to the same index, but Reduction 4 is to the regular SG value at that index, and Rule 5 is to the SG^* value.

At various points in Cases 3 and 4 we will think of Reduction 4 and Rule 5 as going together. We will leave unspecified whether or not $c_1 = 0$, and just use the fact that we can reduce to get either $SG(1 + 2dc_3 + 4d^2c_4 + \cdots)$ or $SG^*(1 + 2dc_3 + 4d^2c_4 + \cdots)$. This is especially useful when we will first apply one of these reductions, and then apply Reduction 2 or Lemma 2.1 to show that in fact the whole sequence is zero.

To finish Case 2, however, Rule 5 is not enough. We note that the smaller index that we get after applying Rule 5 still satisfies the constraints Case 2, namely that $R = 1$ and that c_3, the coefficient of $2d$, is even. So we may use the already established Reductions 1 through 4 on the smaller index $1 + 2dc_3 + 4d^2c_4 + \cdots$, in the hope of finding further reductions.

To prove Reductions 5.2 and 5.3 we will not be able to work as directly as we were with Reductions 1 through 4. We will reduce both the LHS and the RHS with the relevant reductions from the set R1 through R4. Suppose we are able to show that $LHS = SG(r_1)$ and $RHS = SG(r_2)$. Rather than making an argument from the digraph directly, we will construct some other index r_3, that by definition satisfies $SG(r_3) = \text{mex}(SG(r_1), SG(r_2))$. This is technique is also used extensively throughout Cases 3 and 4.

One important example of the technique of reducing first and then building a slightly large r_3 value is how we can sometimes turn an equation relating an SG function to an SG^* function into an equation relating two SG functions. From our basic properties of mex, we see that if $SG^*(a) \neq 0$, then $SG^*(a) = \text{mex}(0, SG(a))$. Moreover, if $SG(a) \neq 0$ (which follows if $SG^*(a) \neq 0$), then $SG(2da) = \text{mex}(0, SG(a))$, by Lemma 2.1. Thus, if $SG^*(a) \neq 0$, $SG^*(a) = SG(2da)$. We will use this in the proof of Reduction 5.2 and at many points in Case 3.

Reduction 5.1. *If $c_1 \neq 0$, $c_1, \ldots, c_4$ are even, c_5 is odd, $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = 0$.*

Proof. After we apply Rule 5, we are in the case of Reduction 2.

$$\begin{aligned} SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) & \\ &= SG^*(1 + 2dc_3 + 4d^2c_4 + \cdots) \qquad \text{R5} \\ &= 0 \qquad \text{R2} \end{aligned}$$

□

Reduction 5.2. *If $c_1 \neq 0$, $c_1, \ldots, c_5$ are even, c_6 odd, then* $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG(2dc_3 + 4d^2c_4 + \cdots)$.

Proof. After we apply Rule 5, we are in the case of Reduction 3:

$$\begin{aligned} SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) & \\ &= SG^*(1 + 2dc_3 + 4d^2c_4 + \cdots) \qquad \text{R5} \\ &= SG^*(c_3 + 2dc_4 + 4d^2c_5 + \cdots). \qquad \text{R3} \end{aligned}$$

$SG(c_3 + 2dc_4 + 4d^2c_5 + \cdots)$ is never zero (Lemma 2.1 applies since c_3 even, by assumption), so $SG^*(c_3 + 2dc_4 + 4d^2c_5 + \cdots) = \operatorname{mex}\big(0, SG(c_3 + 2dc_4 + 4d^2c_5 + \cdots)\big)$.

From this we see that $SG(2dc_3 + 4d^2c_4 + \cdots)$ also equals $SG^*(c_3 + 2dc_4 + 4d^2c_5 + \cdots)$. Therefore, it must be that $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG(2dc_3 + 4d^2c_4 + \cdots)$. □

Reduction 5.3. *If $c_1 \neq 0$, $c_3 = 0$, c_1, c_2, c_4, c_5, c_6 all even, then* $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_5 + 16d^4c_6 + \cdots)$.

Proof. After we apply Rule 5, we are in the case of Reduction 4:

$$\begin{aligned} SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) & \\ &= SG^*(1 + 2dc_3 + 4d^2c_4 + \cdots) \qquad \text{R5} \\ &= SG^*(1 + 2dc_5 + 4d^2c_6 + \cdots) \qquad \text{R4} \end{aligned}$$

We are looking for some other value that causes us to reduce in such a way that we get equality with a starred value, and the values for c_1 and c_2 do just that. Consider taking away coefficients c_3 and c_4 and pushing forward all the other coefficients to take their place, so that we are left with $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_5 + 16d^4c_6 + \cdots)$.

Rule 5 applies (since by assumption $c_1 \neq 0$, c_2, c_5, c_6 all even), and so we must get $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_5 + 16d^4c_6 + \cdots) = SG^*(1 + 2dc_5 + 4d^2c_6 + \cdots)$.

This gives us that $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_5 + 16d^4c_6 + \cdots)$. □

Reduction 5.4. *If $c_1 \neq 0$, $c_3 \neq 0$, and c_1 through c_6 all even, then* $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG(1 + 2dc_5 + 4d^2c_6 + \cdots)$

Proof. After we apply Rule 5 once, we are back in the case of Rule 5:

$$\begin{aligned} SG(1 + 2dc_1 + 4d^2c_2 &+ 8d^3c_3 + 16d^4c_4 + \cdots) \\ &= SG^*(1 + 2dc_3 + 4d^2c_4 + \cdots) \qquad \text{R5} \\ &= SG^{**}(1 + 2dc_5 + 4d^2c_6 + \cdots) \qquad \text{R5} \\ &= SG(1 + 2dc_5 + 4d^2c_6 + \cdots) \end{aligned}$$

□

To aid with further reductions in the other cases, we again summarize the eight reductions needed for Case 2 here. The final column is the shift, or how many powers of $2d$ we lose in our reduction between the original index and the reduced index.

Reductions for $SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots)$ when c_1 even:

R1	$SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)$	c_2 odd	1
R2	$0 = SG(1 + 2d + \cdots$	c_2 even, c_3 odd	NA
R3	$SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)$	c_2, c_3 even, c_4 odd	1
R4	$SG(1 + 2dc_3 + 4d^2c_4 + \cdots)$	$c_1 = 0$, c_2, c_3, c_4 even	2
R5.1	$0 = SG(1 + 2d + \cdots$	$c_1 \neq 0$, c_2, c_3, c_4 even, c_5 odd	NA
R5.2	$SG(2dc_3 + 4d^2c_4 + \cdots)$	$c_1 \neq 0$, $c_2, \ldots, c_5$ even, c_6 odd	2
R5.3	$SG(1 + 2dc_1 + 4d^2c_2 + 8d^3c_5 + \cdots)$	$c_1 \neq 0$, $c_2, \ldots, c_6$ even, $c_3 = 0$	2
R5.4	$SG(1 + 2dc_5 + 4d^2c_6 + \cdots)$	$c_1, c_3 \neq 0$, $c_2, \ldots, c_6$ even	4

Table 2.2: The full set of reductions for Case 2.

2.3.3 Case 3: $R = 0$, c_1 odd

$$\begin{aligned} &SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) \\ &\quad = \operatorname{mex}\left(SG(-1 + 2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots), SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)\right) \\ &\quad = \operatorname{mex}\left(SG(1 + 2d(c_1 - 1) + 4d^2c_2 + 8d^3c_3 + \cdots), SG(1 + 2dc_2 + 4d^2c_3 + \cdots)\right) \end{aligned}$$

Note that instead of subtracting 1 in the standard way, we have borrowed from c_1. This is possible because c_1 odd, therefore $c_1 \geq 1$. Additionally, we have used the Alternating Property in the second term of the mex definition, to go from constant term c_1 to constant term 1. The reason for both of these changes is to put these terms into the framework of Case 2; we will use the eight reductions from Case 2 to reduce both the LHS and the RHS.

In many cases, we find that after we apply one of the eight reductions from Case 2, one of the sides of the definition is all zero. In fact, we can establish criteria for when this occurs:

Proposition 2.1. *Let c_1 be odd, and assume there is at least one other odd coefficient. Let c_i be the first odd coefficient after c_1. Iff i is odd then $LHS = 0$, and iff i is even then $RHS = 0$.*

Proof. Start by assuming i is odd, and $i \geq 3$. We can reduce the LHS, either by applying Reduction 2 directly (if $i = 3$), or by successive applications of Reduction 4 and/or Rule 5. At every step, we have shift 2. Eventually, we will get to a place where either $LHS = SG(1 + 2dc_{i-2} + 4d^2c_{i-1} + 8d^3c_i + \cdots) = 0$ by Reduction 2, or $LHS = SG^*(1 + 2dc_{i-2} + 4d^2c_{i-1} + 8d^3c_i + \cdots) = 0$ by Reduction 2.

Suppose on the other than that $LHS = 0$. We know some reduction from Case 2 applies to the LHS. It cannot be that Reductions 1 or 3 apply, as they reduce LHS to a sequence with even indices (which can never by zero, by Lemma 2.1). These rule out the case that c_2 or c_4 odd. If Reduction 2 applies directly, then it must be that c_3 is odd, which implies that i=3. If instead Reduction 4 or Rule 5 applies, it must be that c_2, c_3 and c_4 all even, so $i \geq 5$.

We will keep applying Reduction 4 and/or Rule 5, until we have the following: either $LHS = SG(1 + 2dc_k + 4d^2c_{k+1} + 8d^3c_{k+2} + \cdots)$ or $LHS = SG^*(1 + 2dc_k + 4d^2c_{k+1} + 8d^3c_{k+2} + \cdots)$, and one the set $\{c_k, c_{k+1}, c_{k+2}\}$ is odd. Note that k itself is even. If Reductions 1 or 3 apply, then $LHS \neq 0$, therefore c_k must be even and c_{k+1} must be odd. It follows that $i = k + 1$, and therefore i is odd.

The same arguments work for RHS, but since the indices in RHS are all shifted

up by one, the parity of i will be exactly opposite. □

With Proposition 2.1 we can prove the following general result:

Lemma 2.8. *Let c_1 odd, and let c_i be the smallest odd coefficient after c_1. If such a c_i exists, then there exists a reduction for $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots)$.*

Proof. We know, by Proposition 2.1, that exactly one of the terms in the mex is zero. Assume, without loss of generality, that the $RHS = 0$. We have that $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots) = \text{mex}(LHS, 0)$, where $LHS = SG(1 + 2d(c_1 - 1) + 4d^2c_2 + 8d^3c_3 + \cdots)$. LHS fits into Case 2, and so we will apply some of the reductions from Case 2.

We don't know which reductions we will apply, but we see that some of the reductions — namely R4, R5.3 and R5.4 — reduce LHS to $SG(r)$, where r still fits into the framework of Case 2. We will continue to apply reductions from Case 2 until we have $LHS = SG(r)$.

Suppose then, that we reduce LHS to $SG(r)$, where r even. Then we have $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots) = \text{mex}(0, SG(r))$. If we consider $2dr$, for r even, then $SG(2dr) = \text{mex}(SG(2dr - 1), SG(r)) = \text{mex}(0, SG(r))$, and so it must be that $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots) = SG(2dr)$.

We would like this to be a reduction, but we still must show that $2dr < 2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots$. The only potential problem is if we apply Reduction 1 or Reduction 3 to the LHS: these have shift 1, and when we multiply back by $2d$ the net effect is shift 0. Note that if we were working with RHS, this would never be a problem — anything with shift at least 1 will give us a proper reduction. If we have applied either Reduction 1 or Reduction 3 to LHS we get that $r = c_1 - 1 + 2dc_2 + 4d^2c_3 + \cdots)$, and so $2dr = 2d(c_1 - 1) + 4d^2c_2 + 8d^3c_3 + \cdots$. This is strictly less than $2dc_1 + 4d^2c_2 + \cdots$, so we do have a reduction. □

Lemma 2.8 doesn't tell us what the reductions actually are, but it does give a good algorithm for finding them. We will go through some specific results at the end of this subsection, as they will be needed for Case 4. The more important question at this point is how to handle the case when all the coefficients are even.

When all the coefficients are even, neither of the two terms in the mex go to zero, so Lemma 2.8 does not apply. Instead, we will reduce each term in the mex as much as possible, and try to construct a smaller index that by definition will equal the mex of the two reduced values. For example if we have $LHS = SG(r_1)$ and $RHS = SG(r_2)$, we're looking for a value r_3 so that by definition, $SG(r_3) = \text{mex}(SG(r_1), SG(r_2))$. If we can find such an r_3, and in addition $r_3 < 2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots$, then we have found a reduction.

Since all the coefficients are even, we must either apply Reduction 4 or Rule 5 to the LHS and to the RHS. In many way, these two are very similar rules: they both have shift 2, and in fact they both yield an equation with the same reduced index. The only difference is that Reduction 4 gives us equality with $SG(r)$, whereas Rule 5 gives us equality with $SG^*(r)$. Which rule we use is determined by whether certain coefficients are zero or non-zero.

We could keep reducing the index this way indefinitely on either side, but if we are looking for a value r_3 where by definition $SG(r_3) = \text{mex}(SG(r_1), SG(r_2))$, we want to stop at a pair of indices r_1 and r_2 where the net shift of all the reductions on the LHS (relative to the original value $2dc_1 + 4d^2c_2 + \cdots$) and the net shift for all the reductions on the RHS (relative to the original value) have difference exactly one. However, while the index of the LHS is always larger than the index of the RHS, either r_1 or r_2 may be larger.

We will show in Lemmas 2.9 and 2.10 that if we can find a pair of indices r_1 and r_2, and either both $LHS = SG(r_1)$ and $RHS = SG(r_2)$ or both $LHS = SG^*(r_1)$ and $RHS = SG^*(r_2)$), then we can construct an r_3 where $SG(r_3)$ will be a reduction of $SG(2dc_1 + 4d^2c_2 + \cdots)$.

Lemma 2.9. *Suppose c_1 is odd, all the other coefficients are even, and there are reductions of at least one of LHS and RHS so that the total difference between their shifts is exactly 1. If the reductions are both to SG values (rather than SG^* values), then there is a reduction of $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots)$.*

Proof. Since we know we will either apply Reduction 4 or Rule 5 to the LHS and

the RHS, and we have assumed that the difference between the shifts is exactly one, we know the forms r_1 and r_2 will take: for some i, one of the reduced indices will be $1 + 2dc_i + 4d^2c_{i+1} + \cdots$, and the other one will be $1 + 2dc_{i+1} + 4d^2c_{i+2} + \cdots$. Since at least one of the sides has been reduced, we know that $i \geq 2$.

Consider $SG(2d(c_i + 1) + 4d^2c_{i+1} + \cdots)$. By definition,

$$\begin{aligned} &SG(2d(c_i + 1) + 4d^2c_{i+1} + \cdots) \\ &\quad = \operatorname{mex}\left(SG(-1 + 2d(c_i + 1) + 4d^2c_{i+1} + \cdots), SG(1 + 2dc_{i+1} + 4d^2c_{i+2} + \cdots)\right) \\ &\quad = \operatorname{mex}\left(SG(1 + 2dc_i + 4d^2c_{i+1} + \cdots), SG(1 + 2dc_{i+1} + 4d^2c_{i+2} + \cdots)\right). \end{aligned}$$

We use the Alternating Property to move from the second to the third line. Since $SG(2d(c_i + 1) + 4d^2c_{i+1} + \cdots)$ and $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots)$ have the same definition, they must be equal. Moreover, because $i \geq 2$, the shift between them is non-zero, and this must be a reduction. □

Lemma 2.10. *Suppose c_1 is odd, c_2 through c_7 all even, and there are reductions of at least one of LHS and RHS so that the total difference between their shifts is exactly 1. If the reductions are both to SG^* values (rather than SG values), then there is a reduction of $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots)$.*

Proof. Since we know we will either apply Reduction 4 or Rule 5 to the LHS and the RHS, and we have assumed that the difference between the shifts is exactly one, we know the forms r_1 and r_2 will take: for some i, one of the reduced indices will be $1 + 2dc_i + 4d^2c_{i+1} + \cdots$, and the other one will be $1 + 2dc_{i+1} + 4d^2c_{i+2} + \cdots$. Since we have arrived at SG^* values, we have applied Rule 5 at least once to both sides, and so we know that $i \geq 3$.

We are looking for a value r_3, so that $SG(r_3) = \operatorname{mex}(SG^*(1 + 2dc_i + 4d^2c_{i+1} + \cdots), SG^*(1 + 2dc_{i+1} + 4d^2c_{i+2} + \cdots))$. We must consider how taking the mex over SG^* values affects the value of the mex. In general this is not clear, but we are looking at a special case, where for any index exactly one of $SG^*(1 + 2dc_i + 4d^2c_{i+1} + \cdots)$ and $SG^*(1 + 2dc_{i+1} + 4d^2c_{i+2} + \cdots)$ are zero. This follows because we know that $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + \cdots)$ is never zero by Lemma 2.1, which means that LHS

and RHS are never both non-zero. Moreover, the larger of the two depends on the smaller, so they can never both be zero. Thus, their non-zero SG values must occur at on sets of indices that are exactly complementary.

Consider mex $\big(SG^*(1+2dc_i+4d^2c_{i+1}+\cdots), SG^*(1+2dc_{i+1}+4d^2c_{i+2}+\cdots)\big)$, and how this relates to $SG(2d(c_i+1)+4d^2c_{i+1}+\cdots)$, which is the mex of the same two terms but without the * in each. One of the two terms in the mex will always be zero, the other will be opposite value of what we would get if we had equality with SG instead of SG^*. So our result will always be opposite what we would have expected, and overall we have

$$SG^*(2d(c_i+1)+4d^2c_{i+1}+\cdots)$$
$$= \text{mex}\left(SG^*(1+2dc_i+4d^2c_{i+1}+\cdots), SG^*(1+2dc_{i+1}+4d^2c_{i+2}+\cdots)\right)$$

This means we have that $SG(2dc_1+4d^2c_2+8d^3c_3+\cdots) = SG^*(2d(c_i+1)+4d^2c_{i+1}+\cdots)$, but this is still not a reduction because it is a relation to an SG^* value rather than an SG value. To fix this, we will take advantage of the following fact: If $SG^*(a) \neq 0$, then $SG^*(a) = \text{mex}(0, SG(a))$.

Since $SG^*(2d(c_i+1)+4d^2c_{i+1}+\cdots) \neq 0$, by Lemma 2.1, we have that $SG^*(2d(c_i+1)+4d^2c_{i+1}+\cdots) = \text{mex}(0, SG(2d(c_i+1)+4d^2c_{i+1}+\cdots))$. We can construct, however, an index whose SG will equal $\text{mex}(0, SG(2d(c_i+1)+4d^2c_{i+1}+\cdots))$: $SG(4d^2(c_i+1)+8d^3c_{i+1}+16d^4c_{i+2}+\cdots)$.

Therefore, we have the equation $SG(2dc_1+4d^2c_2+8d^3c_3+\cdots) = SG(4d^2(c_i+1)+8d^3c_{i+1}+\cdots)$.

Finally, we must check that this is, in fact, a reduction. We know that $i \geq 3$, however, and from this we see that the shift is greater than zero. Therefore, this is a reduction. □

We can only apply Lemmas 2.9 and 2.10 if we can find good values r_1 and r_2. Table 2.3 illustrates when this is possible (note that we are assuming c_1 odd, and all other coefficients are even). We see that in most cases, if we can find good values for r_1 and r_2, we will be able to find them without doing too many reductions. However, there is a case where this does not appear to be possible, shown in Table 2.4

	$c_1 = 1$		$c_1 \neq 1,\ c_2 \neq 0$		$c_1 \neq 1,\ c_2 = 0$ $c_3 \neq 0$			$c_1 \neq 1,\ c_2 = 0$ $c_3 = 0,\ c_4 \neq 0$	
Shift:	LHS	RHS	LHS	RHS	LHS	RHS	RHS	LHS	RHS
0	SG		SG		SG			SG	
1		$\boldsymbol{SG}$		SG		SG			SG
2	$\boldsymbol{SG}$		$\boldsymbol{SG}^*$		SG^*			SG^*	
3				$\boldsymbol{SG}^*$		$\boldsymbol{SG}$			SG
4					$\boldsymbol{SG}$			$\boldsymbol{SG}^*$	
5									$\boldsymbol{SG}^*$

Table 2.3: The cases where it is possible to apply one of Lemmas 2.9 or 2.10.

	$c_1 \neq 1,\ c_2 = 0$ $c_3 = 0,\ c_4 = 0$	
Shift:	LHS	RHS
0	SG	
1		SG
2	SG^*	
3		SG
4	SG^*	
5		SG

Table 2.4: The case where we cannot apply either of Lemmas 2.9 or 2.10.

If $c_1 \neq 1$ and $c_2 = c_3 = c_4 = 0$, then we cannot find a pair that are reduced, within one order, and both SG values or both SG^* values, so Lemmas 2.9 and 2.10 don't help us. Also note that this pattern could continue on arbitrarily long, if we have a long stretch of coefficients that are all zero. For this final case, we will use the fact that the 5th and 6th rows of the chart are the same as the 3rd and 4th rows to construct a value r_3.

We apply two reduction to each side, to get the following:

$$\begin{aligned} LHS &= SG(1 + 2d(c_1 - 1) + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + 32d^5c_5 + 64d^6c_6 + \cdots) \\ &= SG^*(1 + 2dc_3 + 4d^2c_4 + 8d^3c_5 + 16d^4c_6 + \cdots) \qquad \text{R5} \\ &= SG^*(1 + 2dc_5 + 4d^2c_6 + \cdots) \qquad \text{R4} \end{aligned}$$

$$\begin{aligned} RHS &= SG(1 + 2dc_2 + 4d^2c_3 + 8d^3c_4 + 16d^4c_5 + 32d^5c_6 + \cdots) \\ &= SG(1 + 2dc_4 + 4d^2c_5 + 8d^3c_6 + \cdots) \qquad \text{R4} \\ &= SG(1 + 2dc_6 + \cdots) \qquad \text{R4} \end{aligned}$$

We would like to find an index r_3, that when we apply the reduction rules, we get the same split into LHS term that is an SG^* value, and a RHS term that is an SG value. The first few coefficients of the original index gave us that exact split, so we will use them again by taking $r_3 = 2dc_1 + 4d^2c_2 + 8d^3c_5 + 16d^4c_6 + \cdots$.

$$\begin{aligned} &SG(2dc_1 + 4d^2c_2 + 8d^3c_5 + 16d^4c_6 + \cdots) \\ &\quad = \operatorname{mex}\big(SG(1 + 2d(c_1 - 1) + 4d^2c_2 + 8d^3c_5 + 16d^4c_6 + \cdots),\ \\ &\qquad\qquad SG(1 + 2dc_2 + 4d^2c_5 + 8d^3c_6 + \cdots)\big) \\ &\quad = \operatorname{mex}\big(SG^*(1 + 2dc_5 + 4d^2c_6 + \cdots), SG(1 + 2dc_6 + \cdots)\big) \qquad \text{R4, R5} \end{aligned}$$

So we have that

$$SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG(2dc_1 + 4d^2c_2 + 8d^3c_5 + 16d^4c_6 + \cdots).$$

This has shift 2, so it is a reduction.

Many of the results for Case 3 have been non-constructive — rather than get into the specific details as we often had to in Case 2, we have been applying results from Case 2 to show how one could construct reductions. However in Case 4, we will want to apply specific results from Case 3. Therefore we end this case with Table 2.3.3, a complete list of the reductions needed for Case 3. All of these were created using the techniques described above.

2.3.4 Case 4: $R = 0$, c_1 even

Having done so much work tabulating the results in Cases 2 and 3, Case 4 has pleasantly few cases.

$$\begin{aligned} &SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) \\ &\quad = \operatorname{mex}\big(SG(-1 + 2dc_1 + 4d^2c_2 + \cdots), SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)\big) \end{aligned}$$

Reductions of $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots)$ when c_1 odd		
$SG(2d(c_1 - 1) + 4d^2c_2 + \cdots)$	c_2 odd	0
$SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)$	c_2 even, c_3 odd	1
$SG(2d(c_1 - 1) + 4d^2c_2 + \cdots)$	c_2, c_3 even, c_4 odd	0
$SG(2dc_2 + 4d^2c_3 + 8d^3c_4 + \cdots)$	c_2, c_3, c_4 even, c_5 odd	1
$SG(4d^2c_3 + 8d^3c_4 + \cdots)$	$c_1 \neq 1$, $c_2, \ldots, c_5$ even, c_6 odd	1
$SG(2dc_3 + 4d^2c_4 + \cdots)$	$c_1 = 1$, $c_2, \ldots, c_5$ even, c_6 odd	2
$SG(4d^2c_3 + 8d^3c_4 + 16d^4c_5 + \cdots)$	$c_2 \neq 0$, $c_3, \ldots, c_6$ even, c_7 odd	1
$SG(2dc_4 + 4d^2c_5 + 8d^3c_6 + \cdots)$	$c_2 = 0$, $c_3, \ldots, c_6$ even, c_7 odd	3
	$c_2, \ldots, c_7$ all even, and:	
$SG(2d(c_2 + 1) + 4d^2c_3 + \cdots)$	$c_1 = 1$	1
$SG(4d^2(c_3 + 1) + 8d^3c_4 + \cdots)$	$c_1 \neq 1$, $c_2 \neq 0$	1
$SG(2d(c_4 + 1) + 4d^2c_5 + \cdots)$	$c_1 \neq 1$, $c_2 = 0$, $c_3 \neq 0$	3
$SG(4d^2(c_5 + 1) + 8d^3c_6 + \cdots)$	$c_1 \neq 1$, $c_2 = c_3 = 0$, $c_4 \neq 0$	3
$SG(2dc_1 + 8d^3c_5 + 16d^4c_6 + \cdots)$	$c_1 \neq 1$, $c_2 = c_3 = c_4 = 0$	2

Table 2.5: Full set of reductions for Case 3.

Assuming that there is some non-zero coefficient in the LHS (i.e. that the value is at least as big as $2d$), we can borrow from one of the terms and we will get by the Alternating Property and Lemma 2.1 that the LHS is always zero. Therefore, we have

$$SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = \operatorname{mex}\left(0, SG(c_1 + 2dc_2 + 4d^2c_3 + \cdots)\right)$$

Because c_1 is even, the RHS must either fall into Case 3, or back into Case 4. This depends on the parity of c_2, and whether or not $c_1 = 0$. If $c_1 = 0$, and c_2 even, then RHS is in Case 4, and if $c_1 = 0$ but c_2 odd RHS is in Case 3. If $c_1 \neq 0$, the Alternating Principle tells us that RHS will be in the same case as $SG(2d(c_2+1)+4d^2c_3+8d^3c_4+\cdots)$. Thus if $c_1 \neq 0$ and c_2 odd, RHS will be in Case 4, and if $c_1 \neq 0$ and c_2 even RHS will be in Case 3. We would like to address all the possibilities when RHS is in Case 3 or 4 in a unified manner, so we will change the notation of the coefficients slightly in each case.

If RHS is in Case 4, we know that either $c_1 = 0$, and c_2 even, or $c_1 \neq 0$ and c_2 odd. In this second case, the principle of alternation tells us that $RHS = SG(2d(c_2+1)+4d^2c_3+\cdots)$. It's possible that $c_2 = 2d - 1$, and so in order to keep all coefficients strictly less than $2d$ we would have to change some of the larger coefficients. To be able to address these two possibilities consistently, we will rewrite RHS as $SG(2dc_2' + 4d^2c_3' + \cdots)$,

where c_2' is even.

Computing further, we see that

$$\begin{aligned}
&SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) \\
&\quad = \operatorname{mex}\left(0, SG(2dc_2' + 4d^2c_3' + \cdots)\right) \\
&\quad = \operatorname{mex}\left(0, \operatorname{mex}\left(SG(-1 + 2dc_2' + 4d^2c_3' + \cdots), SG(c_2' + 2dc_3' + 4d^2c_4' + \cdots)\right)\right) \\
&\quad = \operatorname{mex}\left(0, \operatorname{mex}\left(0, SG(c_2' + 2dc_3' + 4d^2c_4' + \cdots)\right)\right)
\end{aligned}$$

Since $SG(c_2' + 2dc_3' + 4d^2c_4' + \cdots)$ is never zero (by Lemma 2.1), we have that

$$SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG(c_2' + 2dc_3' + 4d^2c_4' + \cdots).$$

It's possible that in moving from c_i to c_i' some of the coefficients have increased. However, because this has shift 2, we know that it is a reduction.

If instead RHS is in Case 3, we know that either $c_1 = 0$ and c_2 odd, or $c_1 > 0$ and c_2 even. In the second case, using the principles of alternation we could write the RHS as $SG(2d(c_2+1)+4d^2c_3+\cdots)$, and there is no danger of carrying since c_2 even implies that $c_2 < 2d - 1$.

From here, we reduce RHS using one of the rules of Case 3. Suppose the reduced index is r. The $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots) = SG(2d \cdot r)$.

The final to thing to verify is that $2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots > 2dr$. If we are in Case 3 because $c_1 = 0$, the inequality must be true because we haven't increased the index of the RHS — it is exactly the original index divided by $2d$. Because r is strictly less than the index of the RHS, $2dr$ must be strictly less than the original index, and we have a reduction.

If we are in Case 3 because $c_1 > 0$, c_2 odd, however, we used the Alternating Principle to put the index into the standard form of Case 3, and this involved increasing the coefficient of $2d$. It's possible that, since the index of the RHS is larger than the original index divided by $2d$, even though r is strictly less than the index of the RHS $2dr$ could still be greater than the original index.

Note that this can only happen if r comes from a reduction of the RHS with shift zero. But in fact there are only two reductions with shift zero in Case 3, and

they both yield the same value of r: if $RHS = SG(2d(c_2+1)+4d^2c_3+\cdots)$, then $r = 2dc_2 + 4d^2c_3 + \cdots$. From this we get that $2dr = 4d^2c_2 + 8d^3c_3 + \cdots$. This is strictly less than $2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots$, since this problem can only occur if $c_1 > 0$. Therefore, this is a reduction.

Shown in Table 2.6 are all the reductions for Case 4. The first two are from the first possibility, that the RHS will fall back into Case 4. The other reductions are from the possibility that the RHS will fall into Case 3, and so that part of the table is similar to Table 2.3.3.

Reductions of $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots)$ when c_1 even:		
$SG(c_2' + 2dc_3' + 4d^2c_4' + \cdots)$	$c_1 \neq 0$, c_2 odd	2
$SG(c_2 + 2dc_3 + 4d^2c_4 + \cdots)$	$c_1 = 0$, c_2 even	2

Reductions of $SG(2dc_1 + 4d^2c_2 + 8d^3c_3 + 16d^4c_4 + \cdots)$ when $c_1 = 0$, c_2' odd:		
$SG(4d^2(c_2'-1) + 8d^3c_3 + \cdots)$	c_3 odd	0
$SG(4d^2c_3 + 8d^3c_4 + 16d^4c_5 + \cdots)$	c_3 even, c_4 odd	1
$SG(4d^2(c_2'-1) + 8d^3c_3 + \cdots)$	c_3, c_4 even, c_5 odd	0
$SG(4d^2c_3 + 8d^3c_4 + 16d^4c_5 + \cdots)$	c_3, c_4, c_5 even, c_6 odd	1
$SG(8d^3c_4 + 16d^4c_5 + \cdots)$	$c_2' \neq 1$, $c_3, \ldots, c_6$ even, c_7 odd	1
$SG(4d^2c_4 + 8d^3c_5 + \cdots)$	$c_2' = 1$, $c_3, \ldots, c_6$ even, c_7 odd	2
$SG(8d^3c_4 + 16d^4c_5 + 32d^5c_6 + \cdots)$	$c_3 \neq 0$, $c_4, \ldots, c_7$ even, c_8 odd	1
$SG(4d^2c_5 + 8d^3c_6 + 16d^4c_7 + \cdots)$	$c_3 = 0$, $c_4, \ldots, c_7$ even, c_8 odd	3
	$c_1 = 0$, c_2' odd, $c_3, \ldots, c_8$ even:	
$SG(4d^2(c_3+1) + 8d^3c_4 + \cdots)$	$c_2 = 1$	1
$SG(8d^3(c_4+1) + 16d^4c_5 + \cdots)$	$c_2 \neq 1$, $c_3 \neq 0$	1
$SG(4d^2(c_5+1) + 8d^3c_6 + \cdots)$	$c_2 \neq 1$, $c_3 = 0$, $c_4 \neq 0$	3
$SG(8d^3(c_6+1) + 16d^4c_7 + \cdots)$	$c_2 \neq 1$, $c_3 = c_4 = 0$, $c_5 \neq 0$	3
$SG(4d^2c_2' + 16d^4c_6 + 32d^5c_7 + \cdots)$	$c_2 \neq 1$, $c_3 = c_4 = c_5 = 0$	2

Table 2.6: Reductions from Case 4.

Since Cases 1 through 4 cover all possible indices n, and w have found reductions for each case, we have shown that the sequence $\{SG_{1,2d}(n)\}_{n=1}$ is $2d$-regular. Theorem 2.1, that $\{SG_{1,2d}(n)\}_{n=1}$ is $2d$-automatic, follows as a corollary of this theorem and Theorem 2.6.

We will now generalize these results, to prove Theorem 2.2: all prime factors of a are also factors of $2d$, then the sequence $\{SG_{a,2d}(an)\}_{n=1}$ is $2d$-automatic.

Proof of Theorem 2.2. We will first show that any recurrence that $\{SG_{1,2d}(n)\}_{n=1}$ satisfies will also be satisfied by $\{SG_{a,2d}(an)\}_{n=1}$ for large enough values of n. The proofs for these recurrences only rely on the following: the mex definition of $SG_{1,2d}(n)$, Lemma 2.1, and some general facts about mex. For large enough values of n, we certainly have that the mex definition of $SG_{a,2d}(an)$ applies. Lemma 2.1 tells us that for (even larger) values of n certain residue classes of $SG_{a,2d}(an)$ will be zero or non-zero, and that these residue classes are the same as the zero and non-zero residue classes of $SG_{1,2d}(n)$. Therefore, for large enough values of n, every recurrence will hold for $SG_{a,2d}(an)$. This tells us that $\{SG_{a,2d}(n)\}_{n=1}$ is in essence eventually $2d$-regular.

We know that a k-regular sequence that takes only finitely many values is k-automatic. While we have not shown that $\{SG_{a,2d}(n)\}_{n=1}$ is regular, we have shown that it is regular after a certain value. Therefore, it follows that it differs in only finitely many terms from a regular sequence we will call $\{S(n)\}_{n=1}$. Note that while $\{S(n)\}_{n=1}$ and $\{SG_{1,2d}(n)\}_{n=1}$ satisfy the same recurrences, they have different initial conditions and so we would not expect them to be equal. Hence the need to introduce the separate sequence $\{S(n)\}_{n=1}$. Since $\{S(n)\}_{n=1}$ is $2d$-regular and takes only finitely many values, it must be $2d$-automatic.

Finally, we appeal to another known result about automatic sequences. Theorem 5.4.1 in [5] states that if a sequence differs from a k-automatic sequence in only finitely many terms, then it must be k-automatic as well. Since $\{SG_{a,2d}(n)\}_{n=1}$ differs from $\{S(n)\}_{n=1}$ in finitely many terms, it follows that $\{SG_{a,2d}(n)\}_{n=1}$ is $2d$-automatic. □

Chapter 3

Simultaneous Cops and Robbers

3.1 Introduction

The classic game of cops and robbers, introduced independently in [28] and [30], is a game played on a finite graph. The cop selects a vertex to begin at, and then the robber selects a vertex. They alternate turns moving along the edges, the goal of the cop being to land on the same vertex as the robber and the goal of the robber to avoid being caught be the cop. The graphs that admit a winning strategy for the cop were characterized in [28], in terms of graph products and retracts.

A natural generalization of the problem is to increase the number of cops. All cops move simultaneously, but still alternate moves with the robber. In this case, the interesting question becomes the minimum number of cops needed on a given graph. This is known as the cop number of a graph, introduced in [1], and is related to surprisingly many structural aspects of the graph. It was shown to be related to forbidden minors in [8], and to tree-width in [29].

The first person to look specifically at the cops and robbers game on Cayley Graphs was Peter Frankl, in [16, 17]. A Cayley graph is a graph that represents a group G. Every element of the group is represented by a vertex, and we put edges between two vertices represented by group elements x_1 and x_2 if $x_1 a = x_2$, for any element a in a special subset of $M \subset G$. If M is a generating set of G, then the graph will be connected. We will assume that $M = M^{-1}$, so that traveling an edge in either direction always corresponds to multiplication by an element of M.

Frankl makes the following definition, which we will find helpful:

Definition 3.1. *A Cayley graph is said to be* full *if M is closed under conjugation, or*

equivalently, for all $m \in M$, $g \in G$, $g^{-1}mg \in M$.

Frankl's work in [17] establishes upper bounds on the cop number of full Cayley graphs. This chapter is an extension of Frankl's work, establishing exactly how many cops are needed on a full Cayley graph, with two important distinctions: we consider infinite Cayley graphs, and simultaneous (rather than alternating) movements of cops and robbers.

When we move from a finite graph to an infinite graph, the initial placement of the cop(s) and robber becomes slightly more problematic. In the finite version, the cops place themselves on the graph first, then the robber choses a position, and then play begins with the cops moving. On an infinite graph, this arrangement gives the robber too much freedom.

Consider the case of the infinite 2-dimensional integer lattice, which is the Cayley graph of the free abelian group on two generators a and b, with $M = \{a, b, a^{-1}, b^{-1}\}$. Suppose we have a finite number of cops, placed so that the largest number of bs in their position is y. The robber can simply chose to start at a position where the umber of bs is strictly more than y, and always move from position p to position pb. Since the cops and the robber move at the same speed, the robber can never be caught. Therefore, with this initial method of placement, no finite number of cops will ever be able to catch the robber.

To get around this difficulty we will instead place the robber first, and then let the cops choose their positions. We will specify a value N, however, which is the minimum distance that the cops are allowed to be from the robber. It's clear in the case of the two dimensional integer lattice that if at any point the cops are able to form a solid rectangle around the robber, the robber will be caught. But to do this requires roughly $2N$ cops. We are interested in showing that, if the Cayley graph is full, a finite number of cops have a strategy for catching the robber, independent of the value N that dictates how far back they must begin.

Under the above conditions, the main result of the chapter is as follows:

Theorem 3.1. *Let G be the a full Cayley graph with set M (including inverses), $|M| =$*

m. For all N there exist placements of m cops starting at least at distance N from the robber, such that the cops can capture the robber.

3.2 Toy Example

We will start with a concrete example, to illustrate the main idea used in the winning strategy. Some of the concepts we will use below, particularly the idea of assigning directions to edges, do not have analogs for arbitrary graphs. How we expand these ideas in the general result for Cayley Graphs is described in Section 3.3.

Imagine the game where one robber stands at a point on the two-dimensional integer lattice with some number of cops at distance at least n away (for some fixed $n > 0$). The cops and the robber can each move one step to any of the 8 nearest points. We will think of each of these moves as corresponding to one of the 8 compass directions: N, S, E, W, and NW, NE, SW, SE. These are shown on the integer lattice in Figure 3.1.

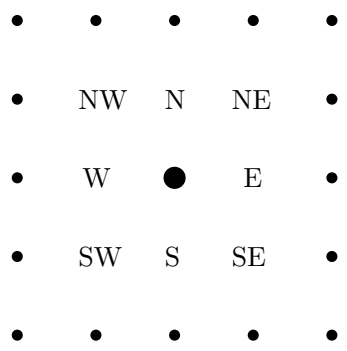


Figure 3.1: The 8 possible movements in the toy example.

Note that this is the Cayley graph of the free abelian group on two generators a and b, with $M = \{a, b, a^{-1}, b^{-1}, ab, a^{-1}b, ab^{-1}, a^{-1}b^{-1}\}$. Since the group is abelian M is trivially closed under conjugation, and this fits the conditions of the theorem. The theorem states that since $|M| = 8$, there should be ways of placing the 8 cops at any distance back, so that they can still catch the robber. The goal in this section is not to prove this particular case of the theorem, but to build up intuition for why the argument makes sense.

First we need to consider how the cops can capture the robber in this game. Without knowledge of where the robber will move, their only option is to control all the spaces at distance one from where the robber was at the end of the previous round. So the essential elements of a guaranteed capture are as follows: if the robber is at a position at the end of round k, the cops must occupy all the adjacent vertices at the end of round $k+1$. It follows that at the end of round $k+1$ the robber must be at one of the vertices occupied by a cop, and therefore captured. It's clear from this that we will need at least 8 (or in general $|M|$) cops, so the upper bound of the theorem in fact matches the trivial lower bound.

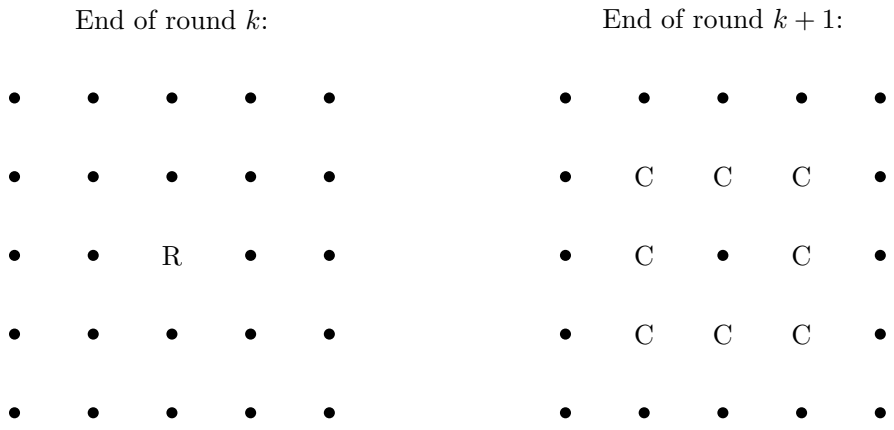


Figure 3.2: Necessary conditions of a capture.

We cannot control or predict the moves of the robber, but we do know that each move will belong to a particular direction. We will try to break down the whole game into smaller, easier pieces, by making one cop responsible for each potential direction the robber could move in. First we will show what that simpler game will look like, and then we will show how we can put together all those smaller games to have a complete strategy.

Consider a game played between one cop and a predictably moving target. In round 1 the cop takes one step (in a direction of his choosing). Then play proceeds in rounds as follows: the target takes a step in one (fixed) direction, and then the cop takes a step (in any direction of his choosing). The cop wins if he lands on the same point as the target at the end of a round. It's clear that whether or not the cop has a winning

strategy depends only on the relative position between him and the target. Moreover, we can say exactly which points on the lattice will be winners: all the points exactly one step from the initial location of the target, all the points exactly two steps from where the target is after one step, all the points exactly three steps from where the target is after two steps, etc. Since the target's moves are set in advance (the target only moves in one, predetermined direction), this set of points is well-defined. We call this game the sub-game for a particular direction.

We will think of the whole game as 8 copies of the moving target sub-game, each with a different cop and a different fixed direction. The target for each game, however, will not be the robber himself, but the point that is one step away from the robber in that particular direction. For example, there will be one moving target game where the target always moves north, and the cop will be placed so that he has a winning strategy to land on the spot that is one step north of the robber. This shift between the target being the robber to a neighboring square comes directly from our sense of what a capture looks like.

In the moving target game we also had the cop and the robber alternate turns. To put this into the context of simultaneous movement, we will have the cop's response and the robber's next move happen simultaneously. This means that in every round after the first, the cop will be moving in response to the robber's move in the previous round. It also means that when we refer to the relative position between the cop and the robber, we are really interested in measuring this between the cop's position at the end of round k and the robber's position at the beginning of round k.

3.2.1 Sample Game Play

To explain more fully how the sub-games fit together to form a winning strategy for the cops, we will work through an example game. For our initial conditions, we require that every cop is placed so that he has a winning strategy in the sub-game for his direction. Note that this doesn't put any limits on how far back the cops can be from the robber. Below is an example of such an initial placement of cops in the case $n = 4$, with a winning strategy for each of their games drawn out. Two of the arrows point toward

the target (which is not the robber himself), corresponding to moves that the cops will make. One points away from the target, corresponding to the one move that the robber makes in each game.

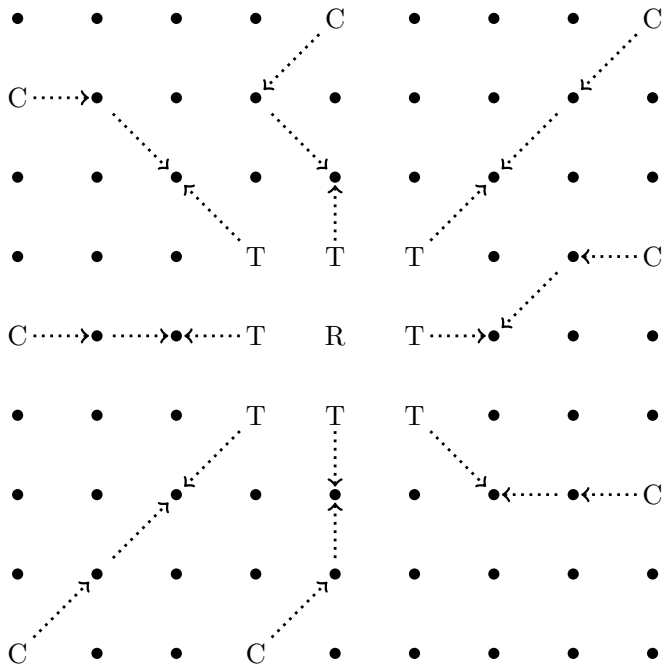


Figure 3.3: Initial positions, and the initial strategies of the cops.

Throughout the game, play for each cop proceeds as follows. In round 1, every cop takes the first step in the winning strategy of his game. Simultaneously the robber moves, but of course the cops don't know where the robber has moved until the end of the round. For all the subsequent rounds, the cops will be basing their moves on the move that the robber made in the previous round. If in the previous round the robber moved in the direction that their target moves in their game, they will take the next step in their winning strategy. If in the previous round the robber moved in a different direction, they will copy the move of the robber, and thus preserve the relative position between their target at the beginning of the round and their position at the end of the round.

Figure 3.4 is an example of what a sample game would look like, from the initial placement in Figure 3.3.The first moves of each of the players are shown in bold arrows, while the proposed moves of the cops are shown in dotted arrows.

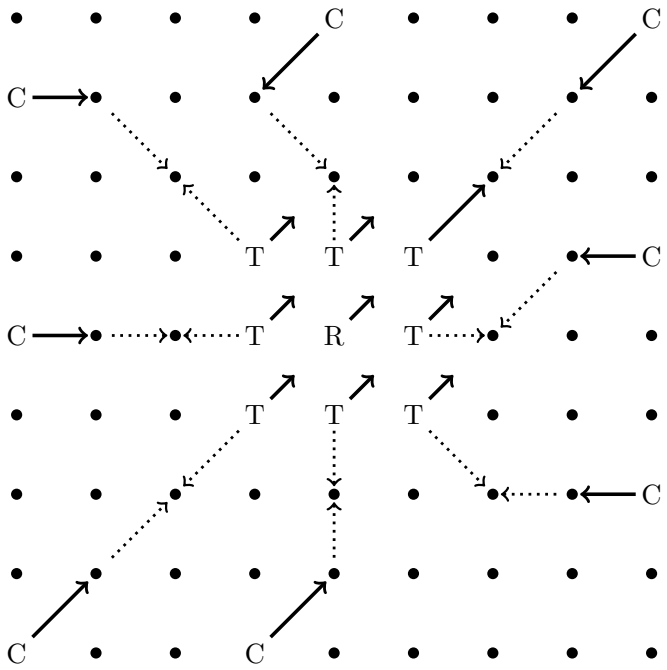


Figure 3.4: Round 1

In the first round, all the cops take the first step in their respective winning strategies. They have no information about where the robber will move, so their first step must be predetermined or random. The robber moves arbitrarily, in this case one step northeast. All the targets, which are relative to the robber, therefore also move one step northeast. Since the target of the northeastern cop has moved northeast, his sub-game has now been advanced, while the rest of the sub-games have not.

In round 2, the northeastern cop is able to take the final step of his winning strategy and win his sub-game. Note, however, that this doesn't correspond to immediately capturing the robber. All other cops have not advanced in their games, and so instead move to maintain their standing in their own sub-games. Their targets have moved, along with the robber, and so their moves are to return to the same relative position to their targets as they were in the previous round. This will let them preserve their winning strategies for future rounds.

The robber moves arbitrarily and simultaneously, in this case one step west. In the next round, the western cop will be able to advance (and in fact win) his sub-game.

For this game, it's not clear that simply preserving the relative position and the

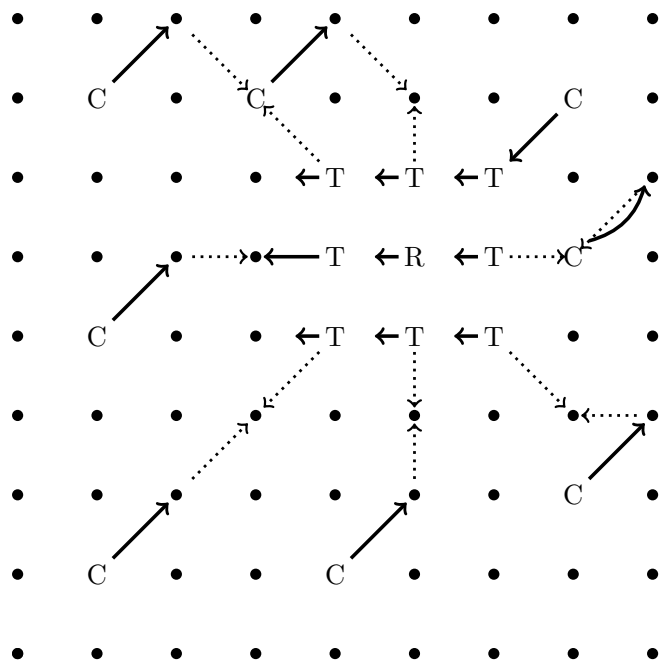


Figure 3.5: Round 2

existing winning strategies is the fastest way of winning. For example, the eastern cop can also win his game at this round, and it would seem obvious that it would be better to capitalize on this advantage now. In fact it will not hurt to take these opportunities as they arise, but we cannot build a more general argument based on this. If you consider the game on the integer lattice where no diagonal moves are allowed (for the robber or the cops), in some cases the only way to ensure that the cops will recover any winning strategy at all is to move so as to restore the relative position between him and his target. Therefore, we do not build our general decomposition around these potential improvements.

In this third and final round, the western cop is able to take the final step of his winning strategy and win his game. The robber moves arbitrarily, in this case one step in the northeast direction, where he is caught by the northeastern cop. In fact, once any cop has won his sub-game, any further moves that the robber makes in that direction will result in his capture.

It's clear that this will be a winning strategy for the cops. Each cop begins with a finite length strategy to win his sub-game, and every move that the robber makes

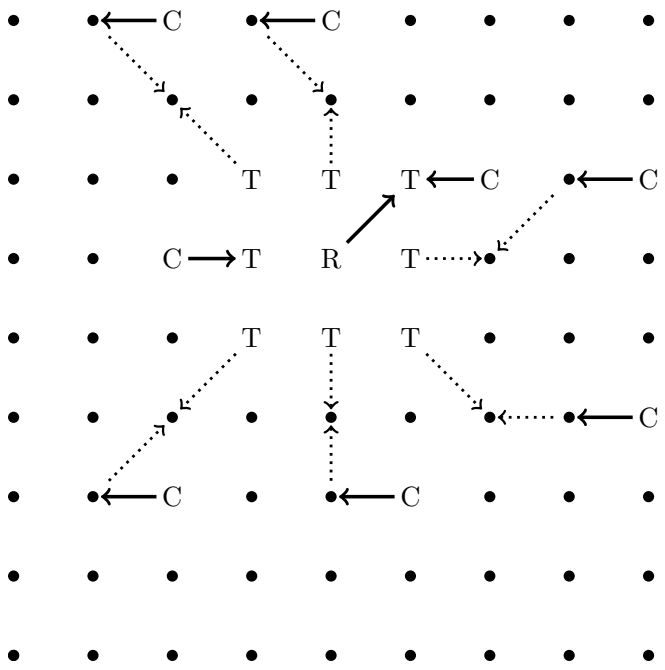


Figure 3.6: Round 3

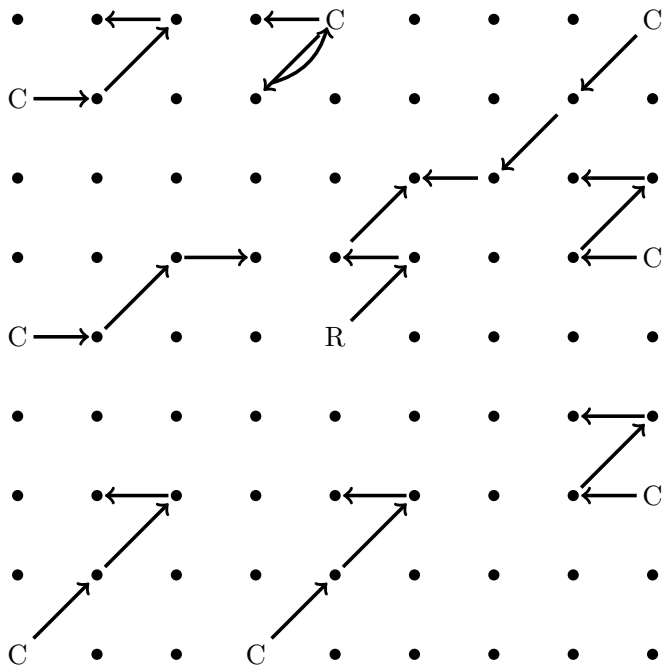


Figure 3.7: All movements of cops and robber throughout the sample game.

advances the games of one of the cops. If the robber evades capture for long enough then eventually, all cops will have won their sub-games and capture is guaranteed. For general graphs, if we could consistently break down the steps of the robber and the policemen into directions, we might be able find a wining strategy using these same ideas. To formally examine winning strategies, we will move into the context of Cayley Graphs, where the idea of direction can be made more precise.

3.3 Cayley Graphs

3.3.1 Free Abelian Groups

Let G be the free abelian group on k elements, and let C be a Cayley graph of G with generating set M of size m (that includes inverses). We will show the following special case of Theorem 3.1:

Theorem 3.2. *For any natural number N, there is a placement of m cops on C (as above) where each cop is at distance at least N from the robber, from which the cops have a winning strategy to catch the robber.*

Note that since G is abelian, the requirement of Theorem 3.1 that the Cayley graph be full (i.e. that the generating set be closed under conjugation) is trivially satisfied. We will develop that further when we examine non-abelian groups.

Proof. Fix an origin somewhere, so that the position of a cop after round k corresponds to an element in the group p_k, and the robber's position after round k corresponds to r_k. The relative position between any two elements a and b is the following element of the group: $a^{-1}b$. We are most interested in measuring $r_k^{-1}p_{k+1}$, the relative position between the robber's position at the end of round k, and the cop's position at the end of round $k+1$.

A *winning word* with respect to $a \in M$ is a string of elements of M where strictly more than half of the characters are a, and the total length of the string is odd (the parity issue is discussed more in depth in Section 3.4). We say that a cop is in a *winning position* relative to a if $r_k^{-1}p_{k+1}$ can be written as a winning word with respect to a.

There may be multiple winning words that describe a winning relative position, but based on the initial placement of the cops we will fix one particular winning word that will correspond to the winning strategy of the cop. Note that this definition of winning position does correspond to the sample game in Section 3.2, where the path is required to be even length but the target is exactly one step in direction a from the robber.

We will show the following two lemmas:

Lemma 3.1. *A cop in winning position with respect to $x \in M$ at the end of round k can maintain his winning position at the end of round $k+1$, regardless of the move the robber makes.*

Lemma 3.2. *If the robber takes a step in the direction $x \in M$ in round k, a cop in winning position with respect to x at the end of round k can move in round $k+1$ to decrease by two the length of the winning word describing the winning position between him and the robber, while still maintaining a winning position with respect to x.*

Once we have these lemmas, our winning strategy for the cops will be as follows. We will select the initial placements of the cops so that after the first round there will be one cop in a winning position relative to each element of the generating set (and relative to the original position of the robber). It's clear that winning positions exist, regardless of how far back the cops must begin: one example of such a winning word with respect to a generator a is a^N if N odd and a^{N+1} if N even. These will correspond to winning positions at distance at least N from the robber. However, there are many possible winning positions and winning words.

For each cop's initial winning position we will fix a corresponding winning word w. By Lemmas 3.1 and 3.2, throughout the game, each cop will either maintain relative position $r_k^{-1}p_{k+1} = w$ (and hence winning position) exactly, or move so that his new relative position is a subword of w that is still a winning word itself. Because subsequent positions of the cop will correspond to subwords of the fixed initial winning word, this gives us a way to measure the progress of each cop towards catching the robber. However, the subword of w may not be the minimal length winning word for any particular winning position.

Since the robber must move in some direction in every round, in every round one of the cops will be able to decrease the length of his winning word by two. Suppose that one cop has reduced the length of his winning word to one. If the robber ever takes a step in that direction again, he will be captured. This is because for any generator a, there is a single winning word of length 1 with respect to a: the word a. If the robber was at position r at the end of round k, then at the end of round $k+1$ the cop will have moved to maintain his winning word of length 1, so he will be at position ra. If the robber has also taken step a, he will be capture at position ra.

Since every cop begins the game with a finite length winning word, and at every round (after the first one) some cop's winning word reduces by two, the robber cannot escape capture indefinitely. In fact, half the sum of the lengths of the winning words of all the cops is a bound on the number of rounds that the robber can go before being captured. □

Proof of Lemma 3.1. If the robber takes step x, the cop can always counter by taking step x as well. Their relative position not only remains a winning position, by commutativity it actually stays identical:

$$r_k^{-1}p_{k+1} = (r_{k-1}x)^{-1}(p_kx) = x^{-1}r_{k-1}^{-1}p_kx = r_{k-1}^{-1}p_k. \qquad \square$$

Proof of Lemma 3.2. If the robber takes step x, and the cop was in a winning position with respect to step x, then the cop has the ability to shorten the length of winning word that describes their relative position (while maintaining it as a win) by exactly two.

Suppose $r_{k-1}^{-1}p_k = x^n k_1 k_2 \cdots k_{n-1}$. Then

$$r_k^{-1}p_{k+1} = (r_{k-1}x)^{-1}(p_ky) = x^{-1}x^n k_1 k_2 \cdots k_{n-1}y = x^{n-1}k_1 k_2 \cdots k_{n-1}y$$

The cop choses his step y to be k_i^{-1}, so that there are still at least $n-1$ elements x in the relative position. Since the number of characters is still odd, and strictly more than half of the characters are still x, this is still a winning word. □

Note that in the previous section, the grid we were looking at was Cayley graph of the free abelian group on two generators a and b, with generating set $\{a, b, ab, ab^{-1}\}$.

As a corollary to Theorem 3.2, we have that 8 cops suffice. The result also holds for the integer lattice in arbitrary dimensions.

3.3.2 Non-Abelian Groups

For a non-abelian group, we need a slightly more stringent definition of what constitutes a winning word. We will say that a word on the elements of the generating set is a *winning word* relative to an element $a \in M$ if it has length length $2k+1$, and the *first* $k+1$ elements are a. As before, we are interested in $r_k^{-1}p_{k+1}$, the relative position between the cop at element p and the robber at element r to be $r^{-1}p$. The cop will be in a winning position if $r_k^{-1}p_{k+1}$ can be described with a winning word.

Our goal is to show the same two general properties: regardless of the move that the robber makes, all cops can maintain their winning positions and one cop can improve his winning position. As Lemma 3.1 is worded currently, however it's not clear that we need commutativity. In general, all we need is to do is move to another winning position with a winning word of the same length, but it doesn't necessarily have to be an identical winning position. However there is only one winning position with respect to a of length 1, corresponding to the word a, so in that case we do require that the new relative position is identical to the old one.

Suppose the relative position $r_{k-1}^{-1}p_k = a$. Suppose that the robber moves a step x in round k, bringing him to position rx. To maintain the relative position a, the cop will take some step y in round $k+1$, bringing him to position ray. Their new relative position will be $(rx)^{-1}ray = x^{-1}r^{-1}ray = x^{-1}ay$, and in order to show that the cop can maintain a winning position of length 1, we need to show that there is a y in the generating set, such that $x^{-1}ay = a$ for any $x, a \in M$.

Essentially, this is the same as showing that $a^{-1}xa \in M$, for all $x, a \in M$. If we assume this, then in fact we get the following for free: $g^{-1}xg \in M$, for all $g \in G$, $x \in M$. This is exactly the condition that the Cayley graph be full. If we assume that the Cayley graph is full, we can prove the following non-abelian analogs of Lemma 3.1 and Lemma 3.2.

Lemma 3.3. *In a full Cayley graph, regardless of the move the robber makes a cop can always maintain his relative position.*

Proof. We will show that, regardless of the move the robber makes, the cop is able to maintain the relative position $r_{k-1}^{-1}p_k$ after round $k+1$. Assume the robber makes a move x, and the cop makes a move y. Their new relative position is $r_k^{-1}p_{k+1} = x^{-1}r_{k-1}^{-1}p_k y$. We want to show that the cop can chose a direction y, so that $x^{-1}r_{k-1}^{-1}p_k y = r_{k-1}^{-1}p_k$. This is equivalent to showing that $(x^{-1}r_{k-1}^{-1}p_k)^{-1}r_{k-1}^{-1}p_k = y \in M$.

$$(x^{-1}r_{k-1}^{-1}p_k)^{-1}r_{k-1}^{-1}p_k = p_k^{-1}r_{k-1}xr_{k-1}p_k$$

If we let $q = p_k^{-1}r_{k-1}$, then $q^{-1} = r_{k-1}p_k$, and $y = qxq^{-1}$. Because of the assumption that the Cayley graph is full, we know that $qxq^{-1} \in M$. Therefore, an appropriate move will always be available to the cop. □

Lemma 3.4. *In a full Cayley graph, a cop can improve a winning relative position with respect to element a, if the robber takes step a.*

Proof. Suppose $r_{k-1}^{-1}p_k = a^n b_1 b_2 \cdots b_{n-1}$. If the robber takes a step a in round k, the cop will respond with a move b_{n-1}^{-1}. Their new relative position is given by the following:

$$\begin{aligned} r_k^{-1}p_{k+1} &= (r_{k-1}a)^{-1}p_k b_{n-1}^{-1} = a^{-1}(r_{k-1}^{-1}p_k)b_{n-1}^{-1} \\ &= a^{-1}a^n b_1 b_2 \cdots b_{n-1}b_{n-1}^{-1} = a^{n-1}b_1 b_2 \cdots b_{n-2} \end{aligned}$$

Their new relative position becomes shorter by 2. Since the length of the winning word is still odd, and $n-1$ is still strictly greater than the length of their relative position, the cop is still in a winning position. □

We can now prove the main theorem of this chapter:

Theorem 3.1. *Let G be a full Cayley graph with set M (including the inverses), $|M| = m$. For all N there are placements of m cops starting at least at distance N from the robber, such that the cops can capture the robber.*

Proof of Theorem 3.1. Lemmas 3.3 and 3.4 show that if we can find initial placements for all the cops so that at the end of round 1 there is a cop in a winning position (with a fixed winning word) with respect to each element of M, and the cops are all at distance at least N from the robber, then at every round one cop will reduce the length of his winning word by exactly two, and all other cops will maintain their winning words exactly. Eventually, this leads to capture of the robber. Therefore, it only remains to show that for any element of M, there exist winning positions distance at least N from the robber.

If the generator a has infinite order, we can find at least one winning position relative to a: a^N if N odd, and a^{N+1} if N even. This must be at distance at least N from the robber, as no relations can reduce this, and certainly has odd parity and satisfies that at least the first half of the elements in the word are a.

Suppose we have a generator b of finite order n. For large enough values of N, it appears that there can be no winning positions at distance N from the robber, because the reduced form of the word describing their relative positions cannot have more than $n-1$ copies of b. In this case, however, we can take any position of distance at least N from the robber, and pad the the relative position with copies of b^n as an initial prefix until it becomes a winning position. This isn't necessarily the most efficient solution, but it is the easiest way to fit this case directly into the decomposition framework described above.

Since there are winning positions for all elements of M at distance at least N from the robber for all values of N, there are initial placements of m cops that will catch the robber regardless of how far back the cops must begin. □

3.4 Parity Concerns

There is an underlying question of parity, that is somewhat bothersome. If the parity of a winning word is not correct, it can happen that in one round a cop will move from vertex p to vertex q, and in the same round the robber will move from q to p. Because they haven't ever landed on the same vertex, this doesn't officially count as a capture

under the original rules.

One way to overcome the parity issue is to give the cops the freedom not to move in a given round. With this extension, if the cop and the robber are at distance exactly 1, the cop will chose not to move. This will either end the round in a capture, or have the effect of switching the parity of their relative position and getting the cop back to a winning position.

If we keep the game symmetric, and also grant the freedom not to move to the robber, this causes an interesting change on the number of cops needed to capture him. In this scenario, play goes as described above on the rounds where the robber moves. If however, the robber chooses not to move in a given round, all the cops who have not yet won their games will take the next step in their winning strategies. This will eventually result in either the capture of the robber (by making too many moves of one type), or it will result with all the cops having won their games and occupying all of the vertices at distance one from the robber. If the robber moves in any direction, he will be caught. However, if the robber choses not to move, the game ends with a stalemate: the robber is surrounded and subdued, but not captured. Under these rules, an extra cop is needed to end the stalemate.

Another way to overcome the parity issue is to extend the notion of capture beyond simply landing on a vertex to either having a cop land on the same vertex or "bump into" the robber. If the robber and a cop are on adjacent vertices at the end of one round, and during the next round the robber moves to the vertex the cop was occupying and the cop moves to the vertex the robber was occupying, we consider this bumping into each other, and can count this as a capture. This eliminates the need for an extra cop to end a stalemate game: if all cops are at distance one from the robber, on the next round they all move to the robbers current position. Either the robber remains there and is captured the traditional way, or the robber attempts to move but (necessarily) bumps into one of the cops and so is captured.

Each of these ways of overcoming the parity concern have their advantages and disadvantages, but they are all compatible with the game decomposition method described here. We will define an even length winning word relative to an element $a \in M$ to be

a word in M of length $2k$, where the first $k+1$ elements are a. A winning position will still be an element that can be written as a winning word. There is also still no restriction on the length of a winning position, so given any distance N behind which the cops must begin we can always find a placement of the cops from which they will be able to capture the robber.

Chapter 4

Colorful Path Coloring

4.1 Introduction

Given a proper coloring of a graph with k colors, we define a *colorful path* to be a path with k vertices where each vertex has been assigned a different color. A proper coloring of a graph where every vertex is at the head of a colorful path is a *colorful path coloring* of the graph. The following was conjectured in [2], and remains open:

Conjecture 4.1. *Every connected graph G, except for C_7, has a colorful path coloring with $\chi(G)$ colors.*

Finding a colorful path coloring of a graph is somewhat similar in feel to the well known problem of finding an *equitable coloring*: a proper coloring where every pair of color classes differ in size by at most one vertex. Both conditions are attempts to find colorings where the colors are evenly distributed. In an equitable coloring the colors are used evenly over the whole graph, however any particular vertex may be at a very large distance from some of the color classes. A colorful path coloring may not be an equitable coloring (in general it will not be), but every vertex will be at distance at most $\chi(G)$ from each of the color classes.

If G is bipartite then $\chi(G) = 2$, and every proper coloring is a colorful path coloring. For the special case of graphs where the clique number $\omega(G)$ equals $\chi(G)$, Conjecture 4.1 was proven in [3]. In [4], Conjecture 4.1 was expanded to include graphs where the circular chromatic number $\chi_c(G)$ equals $\chi(G)$. The main result of this chapter is that Conjecture 4.1 is true in the special case of *half–generalized cycle permutation graphs* (defined below).

Definition 4.1. *A* cycle permutation graph *is a graph on $2n$ vertices made up of two*

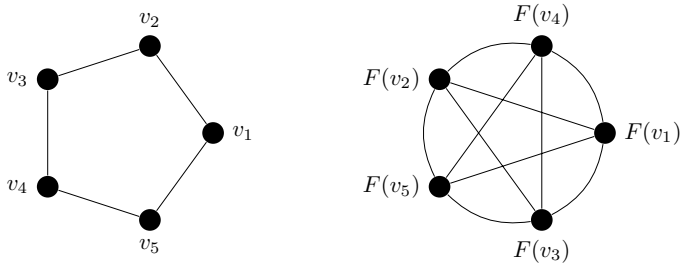
distinct copies of C_n, and a perfect matching connecting vertices in one copy of C_n to vertices in the other copy.

Definition 4.2. *We define a* half–generalized cycle permutation graph *to be a graph on $2n$ vertices made of an arbitrary 2-factor (collection of cycles) on n vertices, a distinct copy of C_n, and a perfect matching connecting the two sets of n vertices.*

We will now briefly discus some of the properties of the circular chromatic number, with the intent of showing that the main result of this chapter is not superseded by the results of [4]. The circular chromatic number $\chi_c(G)$ of a graph G is defined as follows: let $F : V(G) \to C$ be an embedding of the vertices of G onto a circle with circumference 1, and let $d(x, y)$ be the Euclidean distance around the circle between two points on the circle (we will always take the shorter path, and so we assume that $d(x, y) \leq \frac{1}{2}$).

$$\chi_c(G) = \inf_F \max_{\{x,y\} \in E(G)} \left(\frac{1}{d(F(x), F(y))} \right).$$

We always have that $\chi_c(G) \leq \chi(G)$, and a graph for which they are not equal is C_5: $\chi(C_5) = 3$, but $\chi_c(C_5) \leq 2.5$. We see this by considering the following embedding:



In this case, the minimum distance between the endpoints of an edge is $\frac{2}{5}$. Therefore $\chi_c(C_5) \leq \frac{5}{2} = 2.5$. In fact this is the embedding that achieves the smallest value, and we have $\chi_c(C_5) = 2.5$. The set of graphs where $\chi_c(G) = \chi(G)$ includes all graphs where $\chi(G) = \omega(G)$, although a full characterization of which graphs satisfy $\chi_c(G) = \chi(G)$ remains a difficult open problem. See [32] for a more complete treatment of the circular chromatic number.

While some half–generalized cycle permutation graphs satisfy $\chi_c(G) = \chi(G)$, there are infinite families that do not. An example of one such graph is shown in Figure 4.1.

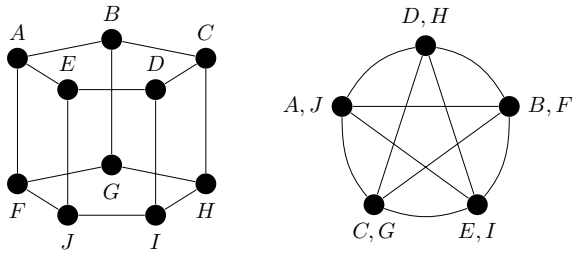


Figure 4.1: A cycle permutation graph Γ, and the embedding into a circle with circumference 1 that shows $\chi_c(\Gamma) \leq 2.5 < 3 = \chi(\Gamma)$.

4.2 Cycle Permutation graphs, 0 mod 3

The general methods in this chapter are constructive. Since they can be somewhat complicated, however, we start with what happens to be the easiest possible example: a cycle permutation graph on $2n$ vertices, where $n \equiv 0 \mod 3$. For this case we will construct the first half of the coloring as follows: Starting at an arbitrary vertex on one of the cycles (we will consider this the outer cycle), color the first vertex with color 1, the second with color 2, the third with color 3, and the fourth with color 1, etc., around the entire cycle of length n. Since 3 divides n, we will use each color exactly $\frac{n}{3}$ times on the outer cycle.

Note that every vertex on the outer cycle is at the head of a colorful path on the outer cycle. Furthermore, in every extension of this partial coloring to a proper coloring of G, every vertex on the inner cycle will be at the head of a colorful path as well. This path will start on the inner cycle with color a, and then step in to the outer cycle onto a vertex of color b (necessarily $a \neq b$, since this is a proper coloring of G). The two vertices available to finish the path on the outer cycle are guaranteed to have distinct colors, and (since it's a proper coloring) to not be the same as color b. Therefore, no matter what color a was we can find a third vertex of color c and complete the path.

All that remains to be shown is that there exists a way to extend this coloring to a coloring of all of G. For the inner cycle, we know that at least two colors are available at each vertex. Furthermore, while we have no information about how the inner cycle is paired with the outer cycle, we do know that vertices of each color were used on the outer cycle. This gives us that there are vertices on the inner cycle with distinct lists of

colors available. From the following simple argument, which is a standard part of the folklore of list coloring, we will show that we can extend the coloring.

Theorem 4.1. *Suppose there is an assignment of lists of size 2 to a cycle C_n, such that not all the lists are identical. Then it is possible to properly color the vertices of C_n, so that each vertex receives a color from its list.*

Proof. Consider the list of colors available at each vertex. Since they are not all identical, there is some edge where the lists at the endpoints are not identical, say $\{a, b\}$ at one endpoint, and $\{b, c\}$ at the other endpoint. Color the first endpoint with color a, and then color greedily around the cycle, away from the edge and the second endpoint. At every step there were initially two colors available in the list and at most one has been used already. This is even true at the final vertex colored, by the special choice for the color of the first vertex. Therefore, we can complete the coloring of the cycle. □

The idea of first coloring one half of the graph, so that many of the vertices are already guaranteed to be at the head of a colorful path, and then extending the coloring to the other half, will generalize to arbitrary cycle permutation graphs. In fact, it also can be generalized to graphs where an arbitrary 2-factor is connected to a large cycle with a perfect matching. In both cases, however, the coloring of the first half leaves a small number of vertices left over, that are not guaranteed to be at the head of a colorful path. In the remaining sections, we examine how it is possible to both extend the coloring to a valid coloring of the whole graph, and at the same time take care of any leftover vertices.

4.3 More General Graphs

Motivated by the discussion of cycle permutation graphs where $n \equiv 0 \mod 3$, we will now work with half–generalized cycle permutation graphs, as defined in Definition 4.2. We see that if the 2-factor in a half-generalized cycle permutation graph happens to be a single C_n, then this will be a cycle permutation graph.

The general outline of the construction of the colorful path coloring is as follows.

We start by coloring the 2-factor, in such a way that most (although not all) of the vertices in the cycle and the 2-factor will be at the head of a colorful path. We will then extend the coloring to the cycle, in such a way that satisfies any vertices not yet at the head of a colorful path. Extending the coloring is always possible by Theorem 4.1, but showing that we can also satisfy the remaining vertices will require an in depth examination of the types of problems that occur. A more detailed plan for how we can do satisfy the remaining vertices is given at the end of this section.

The first step is to color the 2-factor. Each small cycle in the 2-factor will be colored based on its length mod 3. If the length of the cycle is a multiple of 3, we color the vertices 1, 2, 3, etc., and end with color 3. Every vertex in this cycle is at the head of a colorful path. Moreover, every vertex connected to this in the large cycle will already be at the head of a colorful path in any valid large coloring. This is the same as in the case of a cycle permutation graph with length $n \equiv 0 \mod 3$, above.

If the length of the cycle is equivalent to 1 mod 3, we begin by coloring the vertices 1, 2, 3 etc., but we end with a single vertex left over. We give this vertex color 2. Note that this final vertex that is given color 2 is not at the head of a colorful path. In addition, the two vertices on the large cycle that are matched to the neighbors of the final vertex colored on the smaller cycle are not guaranteed to be at the head of a colorful path.

If the length of the cycle is equivalent to 2 mod 3, we color groups of three vertices with colors 1, 2, and 3, and we are left with 2 uncolored vertices. These receive colors 1 and 2. Every vertex in the cycle is at the head of a colorful path, however the two vertices on the large cycle that are matched with the final two vertices colored are not guaranteed to be at the head of a colorful path.

Figure 4.2 shows how small cycles with lengths 12, 10 and 8 respectively, would be colored. Vertices represented with squares instead of circles are not guaranteed to be at the head of a colorful path after the colors of the 2-factor have been assigned.

The goal is to extend the coloring to include the cycle, in such a way that every vertex will be at the head of a colorful path. We know, by Theorem 4.1, that there are many possible extensions to a proper coloring. Each vertex not yet at the head of

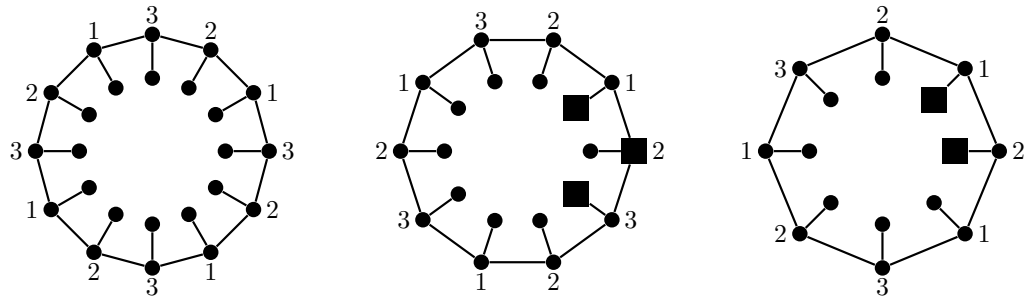


Figure 4.2: Coloring of a 2-factor with cycles of length 12, 10, and 8, respectively.

a colorful path places a restriction on which extension we choose: the extension must include a colorful path for that vertex. To aid in keeping track of when we can do this, we will formally define two types of restrictions, that will both formally occur on the 2-factor (regardless of the location of the vertex is that is not yet guaranteed to be at the head of a colorful path).

Definition 4.3. *A vertex v in the 2-factor that is not already at the head of a colorful path we will call an* outer restriction.

Note that outer restrictions occur exactly when the vertices at distance 2 from v in the 2-factor have the same color as v. We have chosen the coloring of the 2-factor so that this only occurs when one of the cycles in the 2-factor has length 1 mod 3, and that when this happens the vertex must have color 2. To emphasize that when we have an outer restriction we know what color it must have, we will often refer to them as 2-outer restrictions.

Definition 4.4. *A vertex w in the 2-factor whose corresponding vertex in the cycle is not guaranteed to be at the head of a colorful path we will call an* inner restriction.

Note that an inner restriction exists at a vertex w in the 2-factor, ever though it is the neighbor of w in the cycle that is not guaranteed to be at the head of a colorful path. An inner restriction happens exactly when the two neighbors of w in the 2-factor have the same color. This can occur for vertices w of any color, and to distinguish we will refer to them as 1-inner, 2-inner, and 3-inner restrictions as appropriate. Although

we don't have special notation for it, we will often also be interested in the color of the neighbors of w in the 2-factor that cause the restriction.

We will often make distinctions between the neighbor(s) of a vertex that are in the 2-factor, and the neighbor(s) of a vertex that are on the cycle. To help distinguish, we will define the following function π:

Definition 4.5. *If v is a vertex in the 2-factor and x is the neighbor of v in the cycle, then we define $\pi(v) = x$, and $\pi^{-1}(x) = v$.*

The choice of the symbol π is consistent with the idea that if the 2-factor is a single cycle then, the graph is a representation of a permutation. Using this language, v is an outer restriction if v is not guaranteed to be at the head of a colorful path, and v is an inner restriction if $\pi(v)$ is not guaranteed to be at the head of a colorful path. Note that, given how we have colored the 2-factor, no vertex v is both an outer restriction and an inner restriction.

Our method of extending the coloring will be similar to that of Theorem 4.1: we will sweep around the large cycle, coloring as we go. Based on that direction we may refer to vertices in the cycle as being a certain distance 'downstream' (i.e. in the same direction that we will sweep around) or 'upstream' (i.e. in the opposite direction that we will sweep around) of each other. In diagrams the direction of the coloring will usually be marked, but will always be from left to right.

Since the extension of the coloring is affected by the restrictions (which occur on the 2-factor) as well as the order of the vertices on the cycle, we are very interested in the order that the cycle imposes on the vertices of the 2-factor. In particular, we define the following special relationships between vertices of the 2-factor:

Definition 4.6. *Let x and y be two vertices in the 2-factor.*

- *x and y are at* imposed order distance *k (we will write* io-distance *k) if $\pi(x)$ and $\pi(y)$ are at distance k in the cycle.*
- *x and y are* imposed order neighbors *(we will write* io-neighbors*) if $\pi(x)$ and $\pi(y)$ are neighbors in the cycle.*

If further specification is necessary, we may refer to x as an upstream or downstream io-neighbor of v, depending on whether $\pi(x)$ is upstream or downstream of $\pi(v)$.

We can now further elaborate on the algorithm to construct a colorful path coloring. We have colored the 2-factor so that there are only 4 types of restrictions. The next step in the algorithm is to fix a direction along the cycle to be downstream. In Section 4.3.1 we will develop sufficient conditions to satisfy any individual restriction, and in Section 4.3.2 we will develop sufficient conditions to mutually satisfy any group of restrictions. Finally, we will show in Section 4.3.3 that there exist places to start the coloring. Putting these together, we will be able to argue as in Theorem 4.1: we will be able to extend the coloring by sweeping around the cycle, and satisfy every restriction in the process.

4.3.1 Properties of Restrictions

The goal of this section is to show that any restriction v can be satisfied in an extension of the coloring analogous to Theorem 4.1, as long as there are enough vertices downstream of $\pi(v)$ that have not yet been colored. In this section, we will show sufficient conditions to meet each restriction individually. In Section 4.3.2, we will investigate combinations of restrictions, where the conditions to meet each restriction individually are not met.

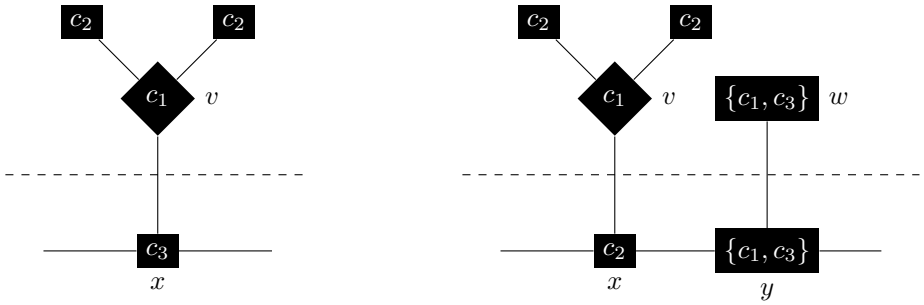
For consistency, in all of the diagrams we will represent a vertex with no restrictions as a rectangular node, a 2-outer restriction with a circular node, and an inner restriction with a diamond shaped node. We will also usually represent the vertices in the cycle along the bottom, with the coloring sweeping from left to right. Above, and separated with a dashed line, will be the vertices of the 2-factor, in the order that they appear relative to the cycle.

We first note that under certain circumstances, inner and outer restrictions will be trivially satisfied by any valid coloring.

Neighboring Colors Lemma. *If we have a restriction at v, it will be trivially satisfied unless the following is about the colors of the io-neighbors of v:*

- *If v is a 1-inner restriction, its io-neighbors must both have color 2.*
- *If v is a 2-inner restriction, its io-neighbors must both have color 1.*
- *If v is a 3-inner restriction, its io-neighbors must both have color 2.*
- *If v is a 2-outer restriction, its io-neighbors must both have colors in the set $\{1, 3\}$.*

Proof. We can prove the first three points from the following general argument about inner restrictions.

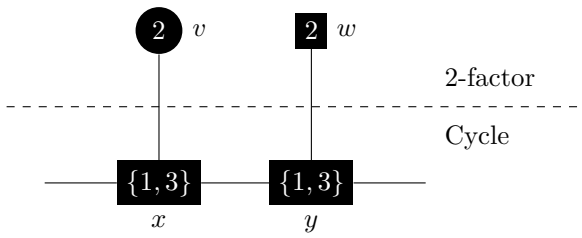


We first note that if $x = \pi(v)$ is assigned the color c_3 we are already done: any path beginning at x and then moving to the outer cycle will have colors c_3, c_1, c_2.

Suppose then that x was assigned the color c_2. If w has color c_3 the only color available at $y = \pi(w)$ is c_1, and if w has color c_1 then the only color available at y is c_3. Either way, the path x, y, w is a colorful path.

Based on how we color the 2-factor, we know that for 1-inner restriction $c_1 = 1$, $c_2 = 2$, and $c_3 = 3$. Therefore the only allowed color for the io-neighbors of a 1-inner restriction is color 2. Also based on the coloring of the 2-factor, we know that for a 2-inner restriction $c_1 = 2$, $c_2 = 1$, and $c_3 = 3$. So the only available color for the io-neighbors of a 2-inner restriction is color 1. Finally, we know that for a 3-inner restriction $c_1 = 3$, $c_2 = 2$, and $c_3 = 1$. So the only available color for the io-neighbors of a 2-inner restriction is color 2.

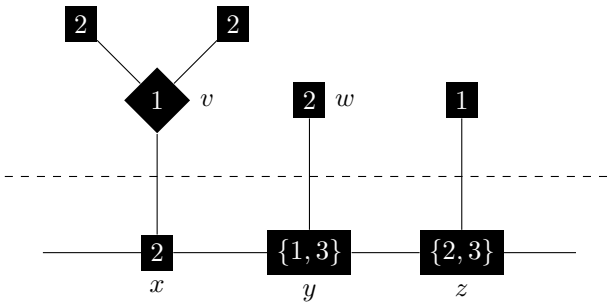
The final point follows from a simple observation about 2-outer restrictions. In any valid extension of the coloring, one of $x = \pi(v)$ and $y = \pi(w)$ will have color 1, and the other will have color 3. In either case the path v, x, y will be a colorful path



with v at the head, and so the restriction will be satisfied. Therefore, the allowable colors for any io-neighbor of a 2-outer restriction are colors 1 and 3. □

When restrictions will be met in any extension of the coloring because they do not fall into the conditions of the Neighboring Colors Lemma, we will not consider them restrictions at all. Therefore for the rest of the chapter when we refer to restrictions we will refer back to this idea. This will be especially helpful in later sections when we look at combinations of restrictions, and when we discuss valid starting points. For now, we will use it to help analyze how we might be able to satisfy a single inner restriction and a single outer restriction.

Example 4.1. Consider what can happen when we try to find an extension of the coloring of the 2-factor while simultaneously satisfying an inner restriction.



If we assume that x has already been assigned color 2, then x is not the head of a colorful path headed up into the 2-factor, and x may not be at the head of a colorful path around the cycle headed upstream. To guarantee that x will be at the head of a colorful path, we must guarantee that x, y, z is a colorful path. Naively, it appears that we have a free choice between color 1 and color 3 for the color of y. But in fact if we want to make x, y, z a colorful path we don't have the option to give y color 3

because color 1 is not available at z. Instead we have to look at the vertices y and z together, and assign their colors as a unit.

This situation is typical for both inner and outer restrictions. We will call the vertices that have to be chosen together the *footprint* of a restriction. In particular, we will make the following two definitions:

- If v is an inner restriction, then the *footprint* of v is the set of the two vertices in the cycle at distance 1 and 2 downstream of $\pi(v)$.
- If v is an outer restriction, then the *footprint* of v is the set with $\pi(v)$ and the vertex in the cycle at distance 1 downstream of $\pi(v)$.

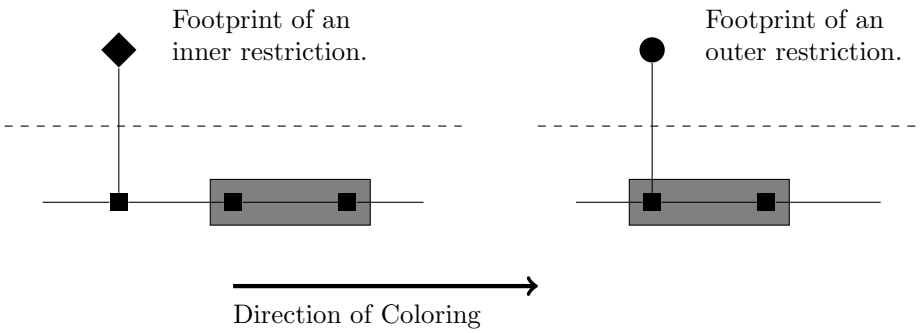


Figure 4.3: Footprints of inner restrictions and outer restrictions.

Note that while the restrictions occur in the 2-factor, the footprints occur in the cycle.

We say that we *have control over* the footprint of a restriction if none of the vertices in the footprint are part of any other footprint, or are the first vertex to be colored. The main result of this section is that having control over the footprint of a restriction is a sufficient condition to satisfy that restriction.

Footprints Lemma. *If we have control over the footprint of an inner restriction or an outer restriction, we will be able to find an extension of the coloring of the 2-factor that satisfies that restriction.*

Proof. We start by considering inner restrictions. Let v be a c_1-inner restriction with neighbors in the 2-factor with color c_2, let $x = \pi(v)$, and let y and z be the vertices

immediately downstream of x. The footprint of v is the set $\{y, z\}$. This is shown in Figure 4.4.

The colors available at x are c_2 and c_3. Note that the statement of the lemma allows for the color of x to already have been assigned, since x is not part of the footprint. If x has been assigned color c_3, however, then x will already be at the head of a colorful path. Therefore we assume that x has been assigned color c_2.

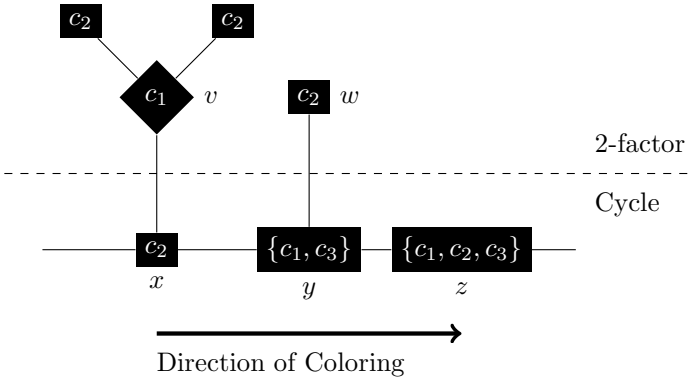


Figure 4.4: An inner restriction v and its footprint $\{y, z\}$.

We know by the Neighboring Colors Lemma that $w = \pi^{-1}(y)$ must have color c_2, otherwise v would not actually constitute an inner restriction, and so the colors available at y are c_1 and c_3. Since x has been assigned color c_2, both remain valid choices for y. At least one color from the set $\{c_1, c_3\}$ must be available at z. By looking ahead, we can select one of those colors for z, and then take the other color for y. Therefore, we can force x, y, z to be a colorful path.

We now consider 2-outer restrictions. Let v be a 2-outer restriction, let $x = \pi(v)$, y the vertex immediately downstream of x. The footprint of v is the set $\{x, y\}$, and since we are assuming that we have control over the footprint we know that x is not the first vertex colored. Let q be the vertex immediately upstream of x. This configuration is shown in Figure 4.5. We may assume that q has been assigned a color since x is not the starting point of the coloring. We will return to this idea and relax this condition slightly when we discuss finding a good starting point, in Section 4.3.3.

First consider that if q has been given color 1 or color 3, the color of x is determined and we have that v, x, q is a colorful path. Therefore, we assume that q has color 2.

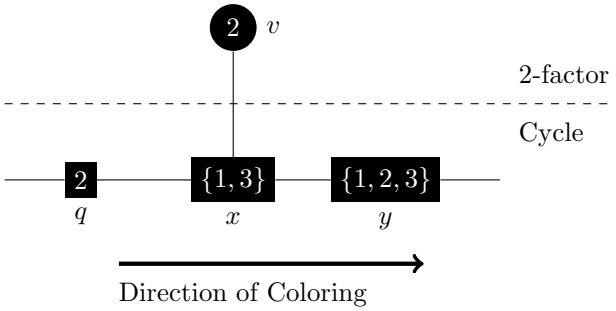


Figure 4.5: An outer restriction v and its footprint $\{x, y\}$.

If this is true, then the color for x is not determined. At least one color from the set $\{1, 3\}$ must be available to y, so by looking ahead we can pick the assignments of x and y so that v, x, y is a colorful path. □

In the following sections we will investigate what happens when we don't have control over the footprint of a restriction. In Section 4.3.2 we will consider what happens when two (or more) restrictions have overlapping footprints, and so we don't have complete control over the footprint of either restriction. Later in Section 4.3.3 we consider how to choose a good starting point for the coloring, so that even if we lose control over the footprints of some restrictions we will still be able to satisfy them.

4.3.2 Combinations of Restrictions

In this section we will consider what happens when restrictions have intersecting or *overlapping* footprints, and so we don't have complete control over the footprint of either restriction. In most cases we will be able to mutually satisfy all the restrictions. There is one case, however, handled in Lemma 4.2, where we cannot do this and we must make a separate argument to show that we can avoid this configuration.

The different types of footprints for inner and outer restrictions are shown in Figure 4.6. We will consider all possible pairs of restrictions, to determine when we could have overlapping footprints. We break down the cases first by looking at whether the restrictions are inner or outer restrictions, and then using the Neighboring Colors Lemma to determine what colors are possible for the inner restrictions.

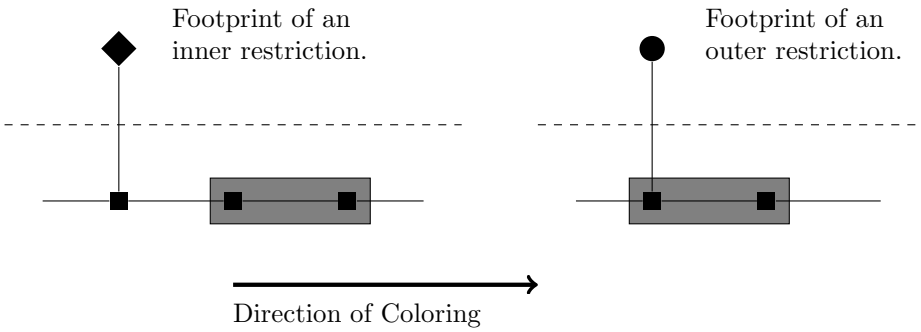


Figure 4.6: The footprints of inner and outer restrictions.

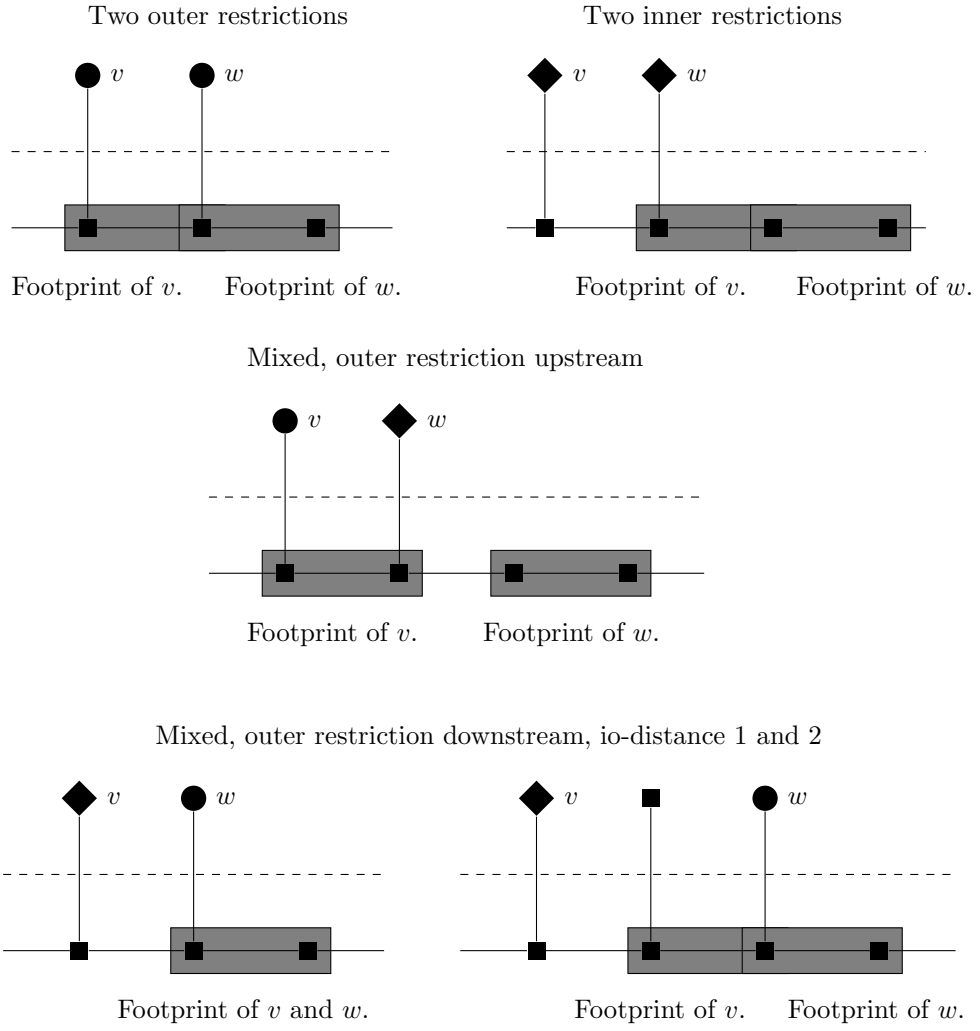


Figure 4.7: Placements of restrictions with possibly overlapping footprints.

Two outer restrictions can only have overlapping footprints if they are io-neighbors. But outer restrictions must have color 2, and we know by the Neighboring Colors Lemma that a 2-outer restriction with a io-neighbor that has color 2 will be trivially satisfied. Therefore we don't have to do any extra work to check this case — while we don't have control over the footprints of the restrictions, they will always be satisfied.

Two inner restrictions can only have overlapping footprints if they are io-neighbors. The Neighboring Colors Lemma restricts the colors of io-neighbors of inner restrictions. From this lemma it follows that one of the inner restrictions must have color 1, and the other must have color 2. In fact we have to consider more than just a pair that have overlapping footprints: we have the potential to have a long string of inner restrictions that are io-neighbors, and whose colors alternate between color 1 and color 2. We will refer to a string of inner restrictions with overlapping footprints as an alternating block or*A-block*. Lemma 4.3 shows that we can mutually satisfy all of the restrictions in an A-block, but in doing so places an extra barrier on where we can start the coloring.

A mixed pair, with both an inner restriction and an outer restriction, cannot have overlapping footprints if the outer restriction is upstream. If the outer restriction is downstream, however, a mixed pair can have overlapping footprints in two ways: if they are io-neighbors, or if they are at io-distance 2.

An outer restriction always has color 2. We can use the Neighboring Colors Lemma to determine what colors are possible for the inner restrictions. If the inner restriction and the outer restriction are io-neighbors, the Neighboring Colors Lemma requires that it must be a 1-inner restriction or a 3-inner restriction. We will show that these restrictions are mutually satisfiable in Lemma 4.1.

If the inner restriction and the outer restriction are at io-distance 2, then there is another vertex that is an io-neighbor of both them. To be an io-neighbor of the outer restriction it must have color 1 or color 3, which precludes the possibility of having 1-inner restriction or a 3-inner restriction at io-distance 2. However it could be a 2-inner restriction, followed by a vertex with color 1, followed by a 2-outer restriction. It turns out that a 2-outer restriction and a 2-inner restriction cannot always be mutually satisfied. In Lemma 4.2 we will use Hall's theorem to show that, by cycling the coloring

on the small cycles in the 2-factor, we can avoid this case.

We have seen that we only have three possibilities to consider where we could have overlapping footprints and the restrictions are not trivially satisfied:

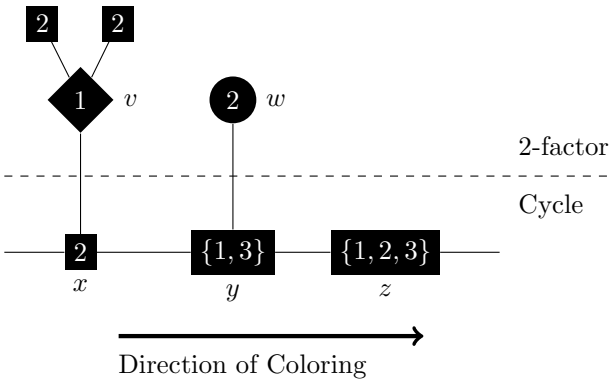
- A 1- or 3-inner restriction with a 2-outer restriction as its downstream io-neighbor.
- A 2-inner restriction with a 2-outer restriction downstream at io-distance 2.
- An A-block.

We will work through the cases in this order.

We say that we *have control over the union* of a set of footprints if no vertex in the union is the first vertex to be colored, or belongs to any footprint not in the union. Note that this is not the same as having control over each of the footprints, individually.

Lemma 4.1. *Let w be a 2-outer restriction, v a 1- or 3-inner restriction, and w the downstream io-neighbor of v. If we have control over the union of the footprints of v and w, we will be able to mutually satisfy both w and v.*

Proof. If we assume that $x = \pi(v)$ has color 3, then we can satisfy the 2-inner restriction at w just by picking color 1 for $y = \pi(w)$, and then both restrictions are satisfied (note that this also works if v is 3-inner restriction, and we pick color 3 for y). So we will assume that x has been assigned color 2.

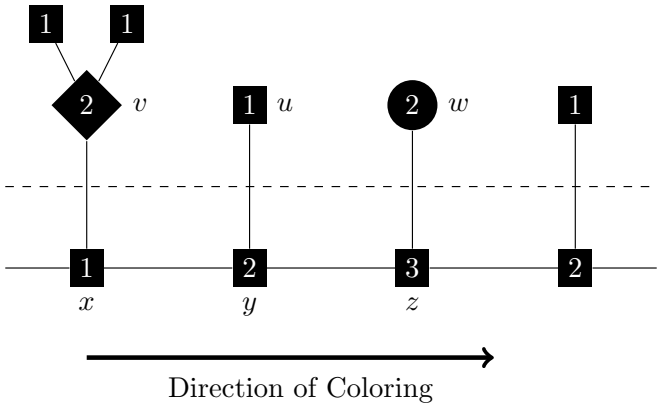


We're now in a situation where x and w both have color 2. Even if x is already at the head of a colorful path pointed upstream, we will chose to put x at the head of a colorful path headed downstream. In doing this, w will automatically also be at

the head of a colorful path headed downstream, so these constraints can be mutually satisfied. □

In the case where we have a 2-inner restriction, and then at io-distance 2 downstream of that a 2-outer restriction. First, we show that in certain conditions we cannot mutually satisfy these restrictions.

Example 4.2. Let v be a 2-inner restriction, u its downstream io-neighbor, and w, the downstream io-neighbor of u a 2-outer restriction. We know, by the Neighboring Colors Lemma, that the color of u is 1. Assume that $\pi(v) = x$ has been given color 1, so that we are forced to look for a colorful path headed downstream for x.



The only way to satisfy the restriction at v is to assign color 2 to y and color 3 to z. If color 1 is not available at the downstream neighbor of z, then we may not be able to satisfy the restriction at w.

Lemma 4.2. *We can select the coloring of the 2-factor so that we never have an outer restriction v and an inner restriction w, such that $\pi(v)$ is at io-distance exactly two downstream from $\pi(w)$.*

Proof. The main idea behind this lemma is that 2-outer restrictions can only come from cycles in the 2-factor that have length 1 mod 3, and 2-inner restrictions can only come from cycles in the 2-factor that have length 2 mod 3. By rotating the coloring on each of these small cycles, we change which vertices on the cycle have the 2-inner and/or the 2-outer restrictions as neighbors. Therefore any particular instance of a

2-inner restriction and a 2-outer restriction with overlapping footprints can be avoided by rotating the coloring on the small cycles.

Of course, the main problem is in showing that we can not only do this individually, but that we can find colorings of the small cycles where we avoid all the instances simultaneously. To do this, we will construct a special bipartite graph based on how the 2-factor is paired with the cycle. A perfect matching in this bipartite graph will be a good choice for where to start the coloring on each small cycle, so that we meet the conditions of the lemma.

We are concerned about all possible places on the cycle where the overlapping footprints of a 2-inner and a 2-outer restriction could occur. Formally, a *site* will be an ordered pair of vertices on the cycle, where the first vertex in the pair is at distance exactly two upstream from the second vertex. We are especially interested in sites $\{v_1, v_2\}$ where $\pi^{-1}(v_1)$ is part of a cycle of length 2 mod 3 (and so $\pi^{-1}(v_1)$ is a potential 2-inner restriction), and $\pi^{-1}(v_2)$ is part of a cycle of length 1 mod 3 (and so $\pi^{-1}(v_2)$ is a potential 2-outer restriction). We want to make sure that when we color the cycles that contain $\pi^{-1}(v_1)$ and $\pi^{-1}(v_2)$, the restrictions don't have neighbors in the same site.

To quantify the situation more formally, we will construct a bipartite multi-graph. The vertices $V = A \cup B$ will be of different types. The set A will have exactly one vertex for every cycle in the 2-factor that has length either 1 mod 3 or 2 mod 3, i.e. each cycle that contributes a 2-outer or 2-inner restriction.

It would easy if we could make B the set of all sites. The problem is that we haven't addressed the issue of cycles of length 4. These cycles actually contribute two 2-outer restrictions, and so whenever we color them we create potential problems with two different sites. To correct for this, we will identify two sites $\{v_1, v_2\}$ and $\{x_1, x_2\}$ if $\pi^{-1}(v_2)$ and $\pi^{-1}(x_2)$ are at distance exactly 2 in a cycle of length 4. This way, if we color the 4-cycle so that v_2 is the neighbor of a 2-outer restriction, we will automatically account for the fact that x_2 must also be the neighbor of a 2-outer restriction.

Therefore the vertices of B will either be sites or pairs of sites, identified because of a C_4, as in the previous paragraph. Edges between elements A and B will be as follows.

For every site $\{v_1, v_2\}$ that is (or is part of) a vertex of B, we will put an edge between that element of B and the cycle that contains $\pi^{-1}(v_1)$ if that cycle has length 2 mod 3, and an edge between that element of B and the cycle that contains $\pi^{-1}(v_2)$ if that cycle has length 1 mod 3. If we have identified a pair of sites to form an element of B, we will put in multiple edges as needed.

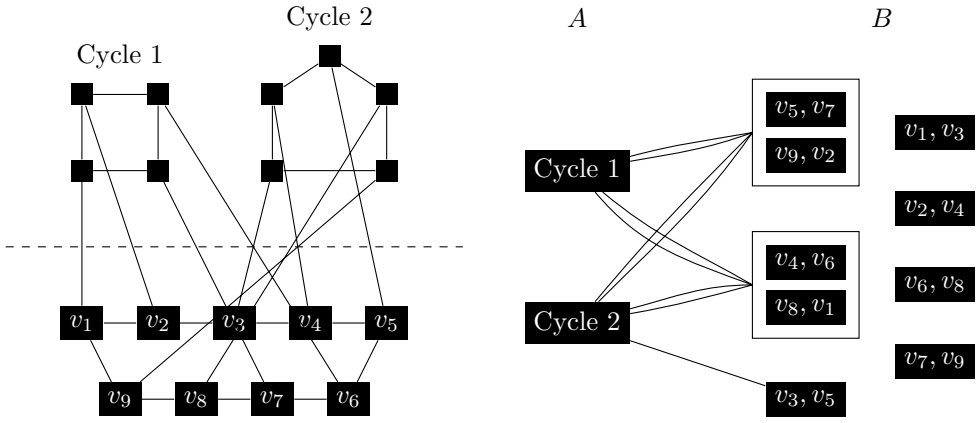


Figure 4.8: Example of a graph, and the bipartite multi-graph constructed from it.

Selecting an edge from $a \in A$ to $b \in B$ will correspond to making a decision about where to start the coloring of cycle a, so that the restriction (or restrictions) will fall on the appropriate vertices of the sites in b. We will ultimately have to select a set of such edges, one from each element of A. If each element of B is incident with at most one selected edge, we have at most one restriction at any of the sites in an element of B, and we have avoided the case where a 2-inner restriction and a 2-outer restriction have overlapping footprints at that site. Thus, what we would like to show is that we can always find an A-perfect matching in the bipartite multi-graph we have constructed.

We can prove that such a perfect matching exists by verifying Hall's condition. Each site has max degree 2. Because we never identify more than two sites to form an element of B, we will have max degree 4 for all $b \in B$. However, every cycle in A has min length 4, and since every vertex of the cycle contributes an edge, we have min degree 4 for all $a \in A$. Therefore, the neighborhood of any subset of A must be at least as large as the original subset, and so such a matching exists. □

Lemma 4.3. *Suppose we have an A-block made up of inner restrictions at vertices*

$v_1, v_2, \ldots, v_k$ ($k \geq 2$). We can mutually satisfy these restrictions as long as we have control over the union of their footprints, and if in addition $\pi(v_1)$ is not the first vertex colored.

Proof. We will assume that v_1 is a 1-inner restriction. Ultimately we will color most of the A-block in groups of three vertices, broken up based on which vertices receive color 3. This idea lets us break the proof into three parts: first will look at how we color the middle of the A-block, in groups of three. Then we consider the beginning of the string up to the first vertex to receive color 3 which could be a partial group, and finally we consider the partial group of 3 vertices at the end of the A-block.

The easiest part of the proof, and so the part we will handle first, is to show that vertices in the middle of the A-block in groups of three will always be at the head of a colorful path. Fig 4.9 is a diagram showing two complete groups of three. The main idea is that vertices will either be assigned the color 3 (which removes their constraint), or they have been forced to use another color and we must guarantee that they are at the head of a path along the cycle.

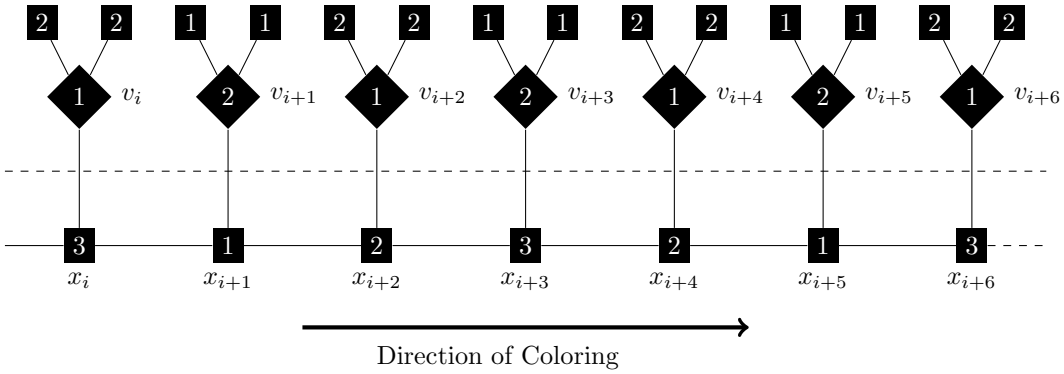


Figure 4.9: Coloring of the middle of an A-block in groups of 3.

The first restriction v_i is satisfied, but x_{i+1} is forced to take color 1. To satisfy x_{i+1}, we must color x_{i+2} and x_{i+3} both with colors from the set $\{2, 3\}$, and furthermore it must be that x_{i+2} gets color 2 and x_{i+3} gets color 3 (based on what's available at x_{i+2}). x_{i+3} is now also head of a colorful path (that heads into the 2-factor), so v_{i+4} ceases to be a restriction. Moreover, x_{i+2} is also at the head of a colorful path counting

backwards: x_{i+2}, x_{i+1}, x_i. This is one group of three, and in fact x_i through x_{i+3} are satisfied.

Looking at v_{i+3}, we see that it is in the same condition as v_i, except that now it is a 2-inner restriction instead of a 1-inner restriction. As with vertex x_{i+1}, x_{i+4} cannot be colored with color 3. In this case, x_{i+4} is forced to take color 2. We can only satisfy the 1-inner restriction at v_{i+4} by coloring x_{i+5} color 1 and x_{i+6} with color 3. Vertex x_{i+6} has color 3, so it is at the head of a colorful path .For x_{i+5} we can again find a colorful path by moving upstream: x_{i+5}, x_{i+4}, x_{i+3}. In this way the coloring proceeds in groups of 3 vertices, broken up based on which vertices receive color 3, and any vertex that is part of a block will be at the head of a colorful path.

We see that for long A-blocks, anything that is in a group of three will be at the head of a colorful path if we continue with this coloring. All that remains to be seen is what happens with the partial blocks of three at the beginning and the end of the A-block.

To examine the first partial block we know that w, the upstream io-neighbor of v_1, has color 2 (this is guaranteed by the Neighboring Colors Lemma). The conditions of this lemma require that the color of $\pi(w)$ has been determined, and as a result we have the following color possibilities:

- $\pi(w)$ has color 1 and $\pi(v_1)$ has color 3,
- $\pi(w)$ has color 1 and $\pi(v_1)$ has color 2, and
- $\pi(w)$ has color 3 and $\pi(v_1)$ has color 2.

The first and third cases are easy to handle. If $\pi(v_1)$ has color 3, then there essentially is no initial partial group. The restriction at v_1 is satisfied, and because $\pi(v_1)$ has color 3 we will begin a group of three with $\pi(v_1)$. In addition, if $\pi(w)$ has color 3 and $\pi(v_1)$ has color 2, we will give $\pi(v_2)$ color 1, and the vertex immediately downstream from $\pi(v_2)$ color 3. We know that the io-neighbor of v_2 has color 1, so 3 must be available. This means that $\pi(v_2)$ is at the head of a colorful path headed upstream, and $\pi(v_1)$ is at the head of a colorful path headed downstream. We are either finished,

or we have started another block.

We will always choose to avoid the second case, if possible. If $\pi(w)$ has color 1 then we are not immediately forced to give $\pi(v_1)$ color 2. If we are in the second case, something particular must have happened to keep us from choosing the color of $\pi(v_1)$ to be 3. We know that $\pi(v_1)$ cannot be the first vertex colored, so it must be that $\pi(v_1)$ is part of the footprint of some other restriction. If so, the other restriction is either a 2-outer restriction at w, or a 1-inner or 3-inner restriction at the upstream io-neighbor of w. In each of these cases, however, we see that to satisfy the initial restriction we will always be forced into case 1, never case 2.

The following cases are illustrated in Figure 4.10. If w is a 2-outer restriction, then to satisfy w, we will want to be in case 1, not case 2. If y, the upstream io-neighbor of w, is a 1-inner restriction, then we will also want to be in case 1, not case 2. If y is a 3-inner restriction, we will want to be in case 1, not case 2.

Note that if v_1 were a 2-inner restriction, rather than a 1-inner restriction, the only possibility would be a 2-inner restriction at the upstream io-neighbor of w. This case would force us into the analog of case 1, where $\pi(w)$ would receive color 2 and $\pi(v_1)$ would receive color 3. This is illustrated in Figure 4.11. Therefore, we will never be forced into case 2, and so we will always be able to satisfy the restrictions that appear before the first block.

Finally, we must also consider partial block at the end of the string. But if we end on a vertex that has color 3, it's clear that we've managed to simultaneously satisfy all of our constraints. If we have one vertex left over, we simply use the fact that we have control over its footprint to force it to be at the head of a (downstream) colorful path. Since that is our only remaining constraint (all of the more upstream vertices are already at the heads of colorful paths) it's clearly possible to satisfy it.

We also have to consider the case where we have two vertices left over, shown in Figure 4.12. Since we have lemmas restricting the color of the io-neighbors of inner restrictions we know exactly what color z must have, based on the color of v_k. We will force x_{k-1} to be at the head of a colorful path headed downstream. If we do that, we are forced to color x_k with color 1, making x_k already at the head of a colorful path:

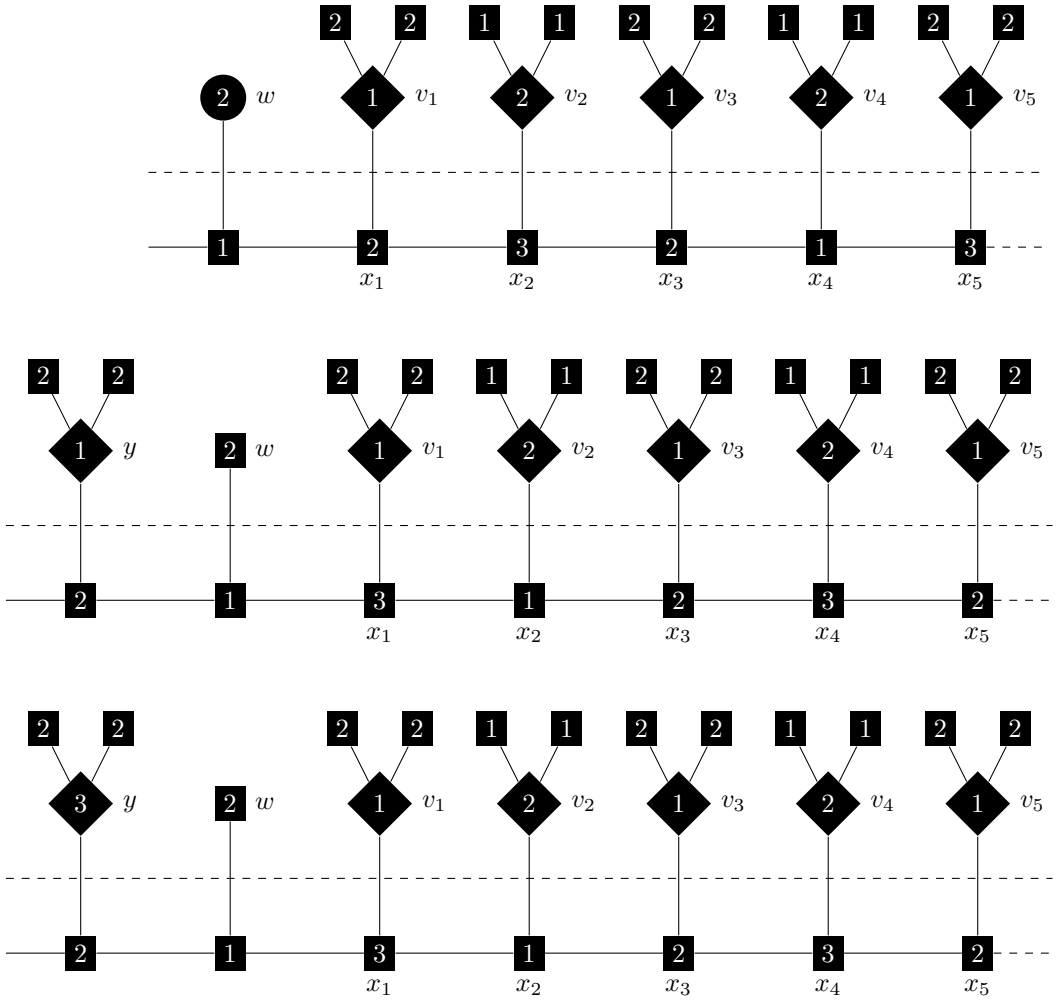


Figure 4.10: The three possibilities for restrictions that could influence the color of x_1 when v_1 has color 1.

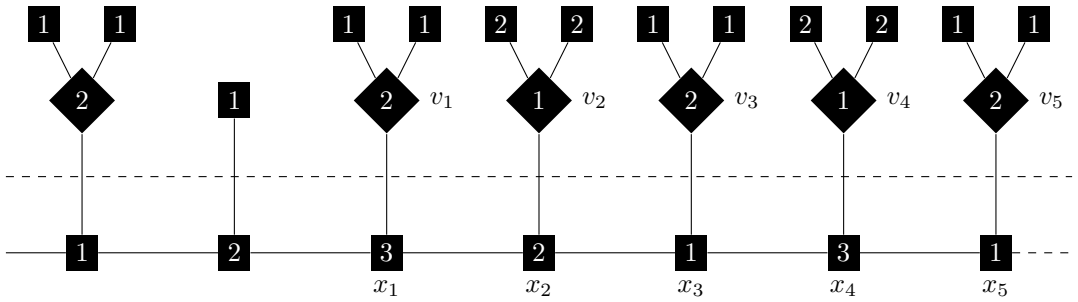


Figure 4.11: The only possibility for a restriction that could influence the color of x_1 when v_1 has color 2.

x_k, x_{k-1}, x_{k-2}.

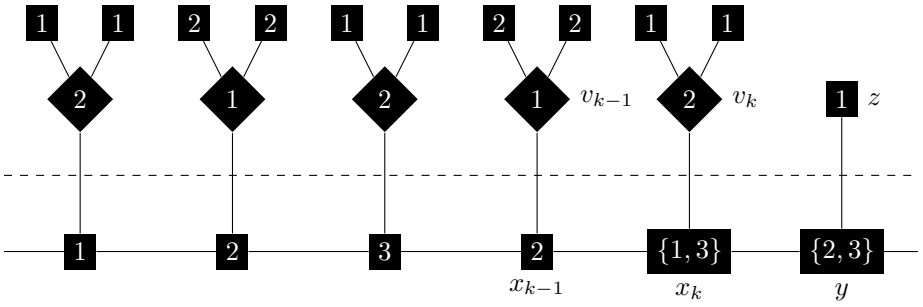


Figure 4.12: The coloring of a final partial group that has length 2.

We might worry about overlapping footprints with other restrictions at the end of the string. It's impossible to have an overlapping footprint with some other inner restriction: if the final restriction in the A-block is v_k, the only way to have an overlapping footprint with an inner restriction would be if it occurs at the downstream io-neighbor of v_k, at which point this would really be a continuation of the A-block.

It is in theory possible to have a 2-outer restriction at z, the downstream io-neighbor of v_k if v_k is a 1-inner restriction. This would mean that v_{k-1} is a 2-inner restriction, however, and by Lemma 4.2 we have shown that we can avoid having any 2-inner restrictions at io-distance 2 upstream of a 2-outer restriction. Similarly, by Lemma 4.2 we can also avoid the case where v_k is a 2-inner restriction, and the downstream io-neighbor of z is a 2-outer restriction. □

We see even more from this proof than sufficient conditions to meet the restrictions in an A-block. We also see that the coloring of the entire A-block is determined by the color of the first restriction (1 or 2), and a residue class mod 3 that indicates which vertices receive color 3. If the $\pi(v_1)$ receives color 3, we say that this corresponds to indices 0 mod 3 (since the first restriction is not technically part of the union of the footprints). If the $\pi(v_2)$ receives color 3 this corresponds to indices 1 mod 3, and if $\pi(v_3)$ receives color 3, indices 2 mod 3. We will use the idea of the indices of the coloring of an A-block in the following section as we try to find special cases where we can relax the conditions of this lemma.

4.3.3 Finding a Good Starting Point

In this section we are looking for a place where we can start the coloring of the cycle. As in Theorem 4.1, if x is the first vertex in the cycle to be colored and y the last, the colors of $\pi^{-1}(x)$ and $\pi^{-1}(y)$ must be different. Note that x will be assigned a particular color, in fact the same color as $\pi^{-1}(y)$, to guarantee that the three neighbors of y will use at most two colors among them. Therefore, there will be at least one color available for y. Because we have assumed that the colors of $\pi^{-1}(x)$ and $\pi^{-1}(y)$ are different, this color is available at x.

Unlike in Theorem 4.1, we are further limited in our choice of a first vertex to be colored. Because the first vertex to be colored has its color assigned, we have defined 'control over a footprint' to exclude the case that any of the vertices in a footprint are the first vertex colored. If we start the coloring at a vertex that is in the footprint of a restriction, we may not be able to satisfy that restriction. Moreover we have an extra restriction from Lemma 4.3: if we have an A-block starting with v_1, we may not be able to satisfy all of the restrictions in the A-block if $\pi(v_1)$ is the first vertex to be colored.

The condition that we have control over the footprint of a restriction in order to be able to satisfy that restriction is sufficient but not necessary. In special cases, we may be able to start the coloring at one of the vertices of the footprint and still meet the restriction. In particular, throughout this section we will use the following two propositions about satisfying a 2-outer restrictions and A-blocks even if the first vertex to be colored makes it impossible to apply the Footprints Lemma or Lemma 4.3.

Proposition 4.1. *If the colors of the upstream io-neighbor and the downstream io-neighbor of a 2-outer restriction are the same, we can start the coloring directly upstream of the 2-outer restriction.*

Proof. This situation is shown in Figure 4.13. The Neighboring Colors Lemma guarantees the first condition of being a good starting point: there are different colors available at the first and last vertices to be colored. It remains to show that any restrictions whose footprints might include x_1 can be satisfied.

First we observe that the color of v_0 determines what color x_1 will take. As long as v_0

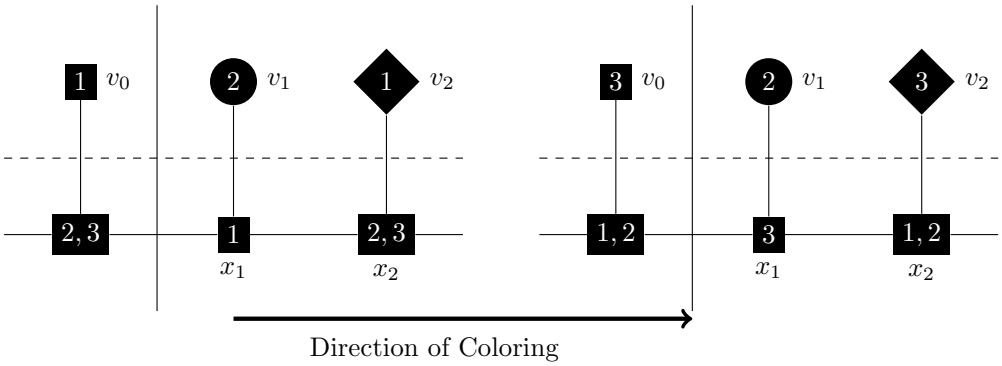


Figure 4.13: Two possibilities for the upstream and downstream io-neighbors of a 2-outer restriction

and v_2 have the same color, we will have the right set of colors available at x_2 to satisfy the 2-outer restriction. Moreover, there cannot be any other restrictions downstream of the split that include x_1, so all the downstream footprints are satisfied.

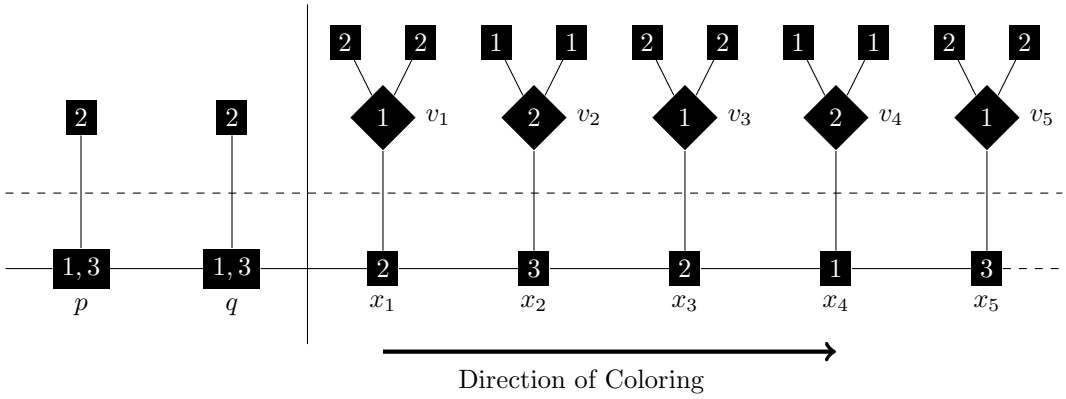
Upstream, we actually have a limited set of possible restrictions whose footprints could include x_1. Another 2-outer would have to be distance two upstream (by the Neighboring Colors Lemma it cannot be io-neighbors with another 2-outer restriction), and if it is at io-distance upstream at least two it's footprint is not long enough to include x_1. A 2-inner restriction would have to be at io-distance at least two upstream as well, and Lemma 4.2 forbids it from being at distance exactly two, so its footprint can't contain x_1 either.

The only potential problem would be if we had a 1-inner or 3-inner restriction at v_0. But in fact, in either case these will be trivially satisfied by not receiving color 2, or they will receive color 2 and be at the head of a colorful path headed downstream (over the split). □

Proposition 4.2. *Let v_1 through v_k be restrictions in an A-block. If the upstream io-neighbor of v_1 and the vertex that is upstream at io-distance 2 from v_1 both have the same color, then we can begin the coloring at $\pi(v_1) = x_1$ and still satisfy all the restrictions in the A-block.*

Proof. We work through the case where v_1 has color 1 and the upstream io-nieghbor of v_1 has color 2, but a similar argument holds for when v_1 has color 2 and the upstream

io-neighbor has color 1. The Neighboring Colors Lemma guarantees that these are the only two cases.



If we start the coloring at x_1, we're forced to use color 2 at x_1. Given that choice, we cannot put both x_1 and x_2 at the heads of colorful paths headed downstream. If we try to put x_1 at the head of a colorful path headed downstream we would give color 1 to x_2 and color 3 to x_3. To finish the colorful path for x_2 we would need to give color 2 to x_4, which is a contradiction.

We need to pick at least one of these vertices to be at the head of a colorful path headed upstream rather than down. Normally we couldn't guarantee this without fixing some colors upstream, which would invalidate the idea that we start the coloring at x_1 and have free choice for the colors of p and q. However in this special case we know that both p and q will take colors from the set $\{1, 3\}$, and that they must each take different colors. Therefore, since x_1 is already forced to have color 2, x_1 will have to be at the head of a colorful path headed upstream (no matter how the colors of p and q are assigned). We will waive the restriction that x_1 should be at the head of a colorful path headed downstream, assign color 3 to x_2, and from there we can color the rest of the A-block. □

Armed with these two propositions, we will make three separate arguments — one fairly general and two special cases — where we find a good starting point explicitly. Generally, these arguments require first narrowing down the situation by excluding cases where it's easy to find a good starting point. What's left then has so much structure

that we can hope to find additional special cases where we can relax the requirement that no vertex in the footprint be the first vertex colored.

We start by making an argument for the case where the 2-factor contains at least one cycle of length 1 mod 3, but the 2-factor is not entirely composed of 4-cycles. This is the most general case, and in many ways the most difficult. If we have no 2-outer restrictions (i.e. all cycles in the 2-factor have lengths either 0 or 2 mod 3), a much simpler argument suffices to find a good starting point. On the other hand if we have all 4-cycles, the 2-factor has so much structure that we can make a very specific argument to handle that case. Both of these are handled after the most general case.

Lemma 4.4. *Suppose the 2-factor has at least one cycle of length 1 mod 3, and has at least one cycle that is not a C_4. Then there is a pair of vertices x_1 and x_2, adjacent on the cycle with x_1 upstream of x_2, such that the following two conditions hold:*

- *$\pi^{-1}(x_1)$ and $\pi^{-1}(x_2)$ have different colors.*
- *any restrictions whose footprints contain x_2 can be met, even if x_2 is the first vertex to be colored.*

Proof. We will be able to find x_1 and x_2 by considering the vertices on the cycle whose neighbors in the 2-factor have color 2. We will think of the neighbors of the 2-outer restrictions as breaking the cycle into *regions*, and consider the neighbors of other vertices with color 2 based on which region they fall into. In particular, it's necessary to place the above conditions on the cycles of the 2-factor so that we may assume that the 2-factor contributes at least one 2-outer restriction, and at least one other 2. We see that there will be at least as many unrestricted 2s in the 2-factor as 2-inner restrictions, and in particular this means that there is (at least) one region with at least as many unrestricted 2s as 2-inners (and at least one unrestricted 2). We will call such a region *non-empty*.

We start by excluding several cases, which allow us to narrow down what a non-empty region will look like. First we will show that if a non-empty region ever has io-neighbors v and w (assume w is downstream of v in the imposed order) where both

v and w have colors in the set $\{1, 3\}$, and if the downstream io-neighbor of w is not a 2-outer restriction, we can find a good starting place. The Neighboring Colors Lemma guarantees that neither w nor v can be restrictions. If we start the coloring after w, no restrictions from upstream can be a problem. The only restrictions from downstream could be the start of an A-block.

In fact, though, if v and w both have color 1, Proposition 4.2 allows us to start the coloring after w even if there is an A-block immediately after. We also see that if w has color 3, then its downstream io-neighor cannot be an inner restriction (the Neighboring Colors Lemma), so the only remaining case to consider is if the color of v is 3, the color of w is 1, and the downstream io-neighbor of w is a 2-inner restriction that begins an A-block. But if the color of v is 3, then we can't have an inner restriction as the upstream io-neighbor of v, either, and we can start the coloring at $\pi(w)$ Even if we have a 2-outer directly upstream of v, $\pi(w)$ will not be in the footprint of a restriction either upstream or downstream of w.

From this it follows that either we can find a good starting point in a non-empty region, or after the first 2-outer restriction, the colors of the vertices in the 2-factor must alternate between 2 and something from the set $\{1, 3\}$ until all the 2s in that region except for the final 2-outer restriction have appeared. But we can also eliminate the case where we have a string of io-neighbors with colors 2, 3, 2, and show that in fact the colors must alternate between 1 and 2 until all the 2s in that region have appeared.

If we ever have a string of three io-neighbors u, v and w whose colors are 2, 3, 2 respectively, we can find a good starting point either directly upstream or directly downstream of v. By the Neighboring Colors Lemma, neither u nor w can be 2-inner restrictions, and if v is an unrestricted 3 we can certainly start the coloring at $\pi(w)$. Therefore, we will only consider the case where v is a 3-inner restriction, shown in Figure 4.14.

Let q be the upstream io-neighbor of u. If q is not an inner restriction (and of course it cannot be a 2-outer restriction, by the Neighboring Colors Lemma), then we can start directly upstream of the 3 and we're fine. If q is an inner restriction, it could either be a 1-inner or a 3-inner. But if q is a 3-inner restriction, it must also be between

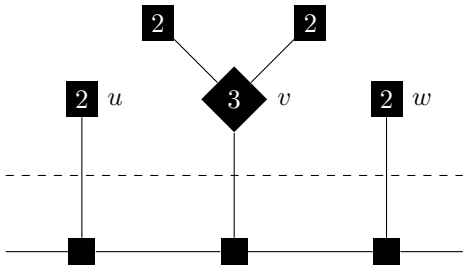


Figure 4.14: A 3-inner restriction inside a region.

two 2s (in the imposed order). So we will assume that v is the most upstream 3 in the region, and therefore q must be a 1-inner restriction.

If q is not part of an A-block, we can win: we simply require that no matter what, q is at the head of a colorful path headed downstream. We can do this, as we have complete control over the footprint of q. This colorful path will end at $\pi(v)$, and that will satisfy the restriction at v.

If q is the last restriction of an A-block, we just need to check the three possible cases for the indices of the coloring of the A-block.

Therefore we know that after the first 2-outer restriction, the colors in this region must alternate between 1 and 2 until the final 2. Essentially, we will color the whole thing like a giant A-block starting at the 2-outer restriction, even though not all of the 1s and 2s are necessarily inner restrictions. Every vertex that isn't an inner restriction can provide some freedom in our coloring.

The coloring of an A-block is completely determined by the indices of the coloring, that is, which residue class mod 3 contains the vertices which receive color 3. We will try to use the fact that, unlike an A-block, not all of our 1s and 2s are restrictions to change that residue class, in the middle of the A-block. If there is a 3 that appears immediately upstream of x_i, but v_i is not an inner restriction, we have the option to change the indices of the coloring:

Thus every unrestricted element at io-distance k from the 2-outer restriction has the ability to change the residue class of the coloring from $k-1$ to $k+1$ mod 3.

We will use some facts about the beginning of the region. We know that the color of the downstream io-neighbor of the 2-outer restriction is 1. If the color of the upstream

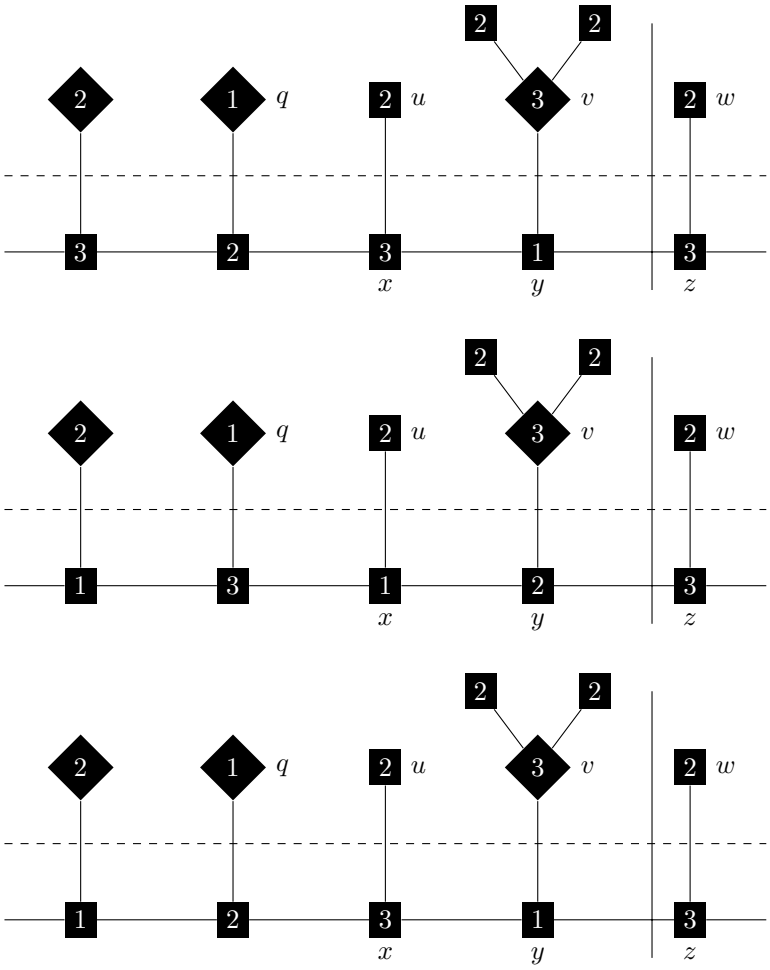


Figure 4.15: Possible colorings of an A-block, and how this interacts with a 3-inner restriction at io-distance 2 downstream.

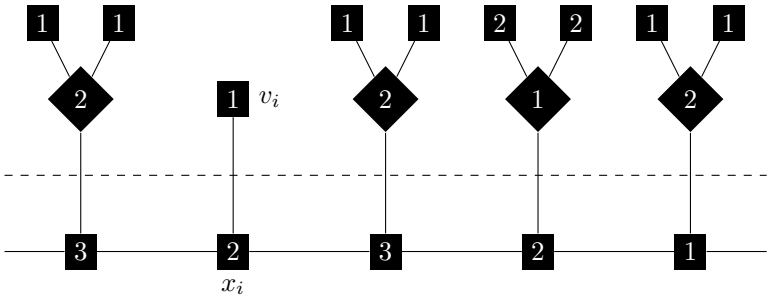
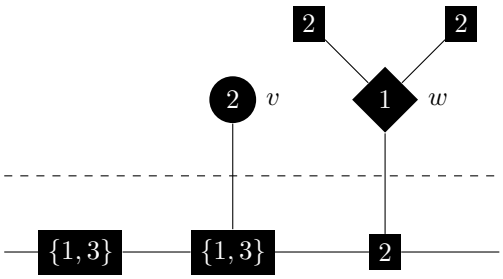


Figure 4.16: Example of the change in the residue class of the coloring of an A-block that an unrestricted vertex v_i allows us to make.

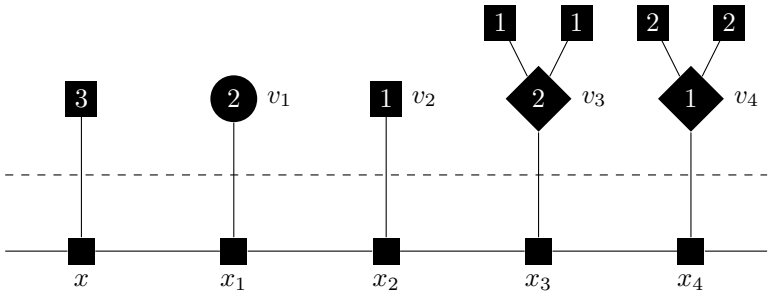
io-neighbor of the initial 2-outer restriction is also 1, we can find a good starting point, by Lemma 4.1. Therefore, we assume that it has color 3. We will also show that the downstream io-neighbor of the initial 2-outer restriction isn't a 1-inner: even if it was, its restriction will be trivially satisfied in any coloring that that satisfies the 2-outer restriction.

Proposition 4.3. *If a 2-outer restriction v is the upstream io-neighbor of a 1-inner or 3-inner restriction w, then the restriction at w will be met by any extension of the coloring that satisfies v.*



Proof. Suppose that $\pi(w)$ is forced to take color 2. This can only have happened if v has been satisfied with a path upstream, that x and y have colors 1 and 3. Therefore, $\pi(w)$ is already at the head of a colorful path headed upstream. □

Putting all this information together, we can assume we are in the following general case:



If the 2-outer restriction at v_1 is satisfied, we know that one of x, x_1 and x_2 has color 3. Since it cannot be at x, we start out knowing that we have a 3 either at x_1 or x_2, that is, we are in residue class 0 or 1, but not 2.

If we have an unrestricted 2 at io-distance 2 mod 6 downstream from v_1 and we are in residue class 1, we can toggle so that we are in residue class $1 + 2 = 3 \equiv 0 \mod 3$. This lets us start our coloring anywhere after that unrestricted 2, as long as we start it consistent with coloring the entire A-block with residue class 0. No matter how we begin the A-block, we can finish coloring the cycle consistently with residue class zero.

Additionally, if we have an unrestricted 2 at io-distance 0 mod 6 downstream (but at least 6) from v_1, we can first change residue class 0 to be residue class 2 (based on the fact that the first 1 is not an inner restriction), and then at the unrestricted 2 we can change residue class 2 to residue class 1. Therefore, whether we initially start with residue class 1 or 0, we can guarantee that we will end the coloring of the A-block with residue class 1. This lets us start our coloring anywhere after the unrestricted 2, as long as we start the coloring consistent with coloring the entire A-block with residue class 2.

Both of these results for when we have an unrestricted 2 at io-distance 0 or 2 mod 6, aren't quite the same as finding a good starting point. In particular, the first vertex to be colored is colored consistently with coloring the whole A-block according to a particular residue class, instead of being colored to allow for two colors at the final vertex. However, what we have shown guarantees that we will end in the same residue class, so we will be able to meet all the restrictions of the A-block.

The only potential problem is if we have all our unrestricted 2s at io-distance 4 mod 6 from v_1. Using the unrestricted 2 lets us change from residue class 0 to residue class 2 but doesn't affect residue class 1, and so we cannot get down to a single residue class for this A-block. For a large number of 2s, however, this cannot happen: in general, only having unrestricted 2s at io-distance 4 mod 6 will take roughly twice as many 2-inner restrictions as unrestricted 2s. We know that we have at least as many unrestricted 2s, and so the only time this can happen is if we have exactly two 2s in a region - a 2-inner at io-distance 2 from v_1, and then an unrestricted 2 at io-distance 4.

If we only have two 2s, however, we know that after the unrestricted 2 we cannot have an A-block. In that case, we can still start the coloring just after the unrestricted 2. In both residue class zero and residue class 1, the colorful path of the final 1-inner

restriction does not extend past the unrestricted 2: it's only in residue class 2 that we would need to have control over the final vertex in the footprint of the A-block. This is shown in more detail in Figure 4.17. Therefore, there are no problems upstream with starting the coloring just after the final unrestricted 2.

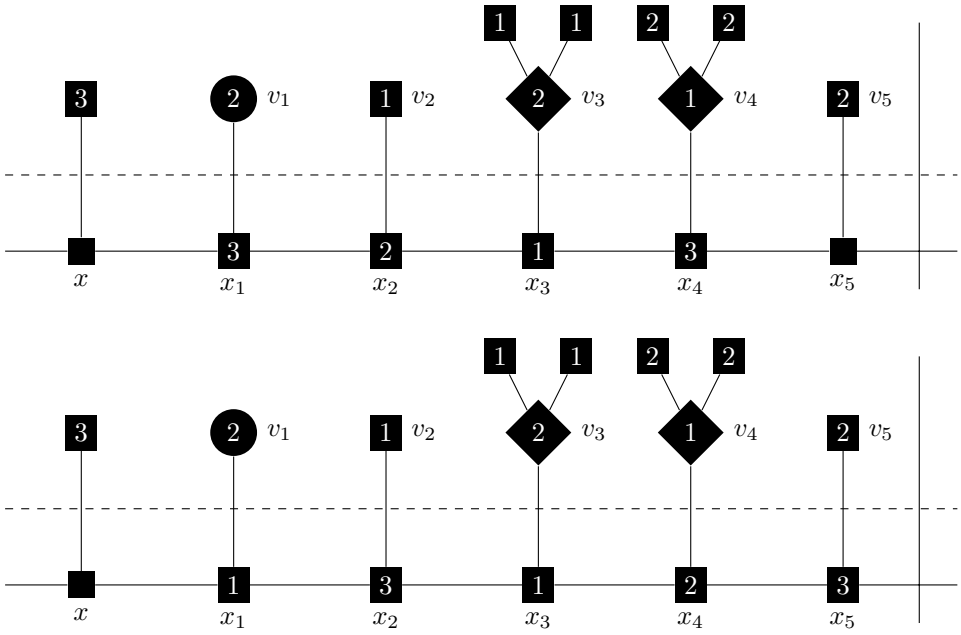


Figure 4.17: The two possible residue classes for the A-block do not extend past x_5.

Downstream, the only problem to starting the coloring here would be if the downstream io-neighbor of the unrestricted 2 was a 2-outer (which by the Neighboring Colors Lemma is forbidden), or if it's a 1-inner that is part of another A-block. By assumption there are no more 2s in this region, however, so we cannot begin an A-block here. □

There are a few remaining special cases to consider. In Lemma 4.4 we used the fact that we started with a 2-outer restriction, and this allowed us to drop one of the inner restrictions. So we must make a special argument for when we don't have any 2-outer restrictions. In fact, however, this case is much simpler.

Lemma 4.5. *If the 2-factor has no cycles that have length 1 mod 3, then there is a pair of vertices x_1 and x_2, adjacent on the inner cycle with x_1 upstream of x_2, such that the following two conditions hold:*

- *$\pi^{-1}(x_1)$ and $\pi^{-1}(x_2)$ have different colors,*

- *any restrictions whose footprints contain x_2 can be met even if x_2 is the first vertex to be colored.*

Proof. If we have no cycles that have length 1 mod 3 we have no 2-outer restrictions or 3-inner restrictions anywhere in the graph. Consider a pair of vertices x_1 and x_2 adjacent in the cycle with x_1 upstream, so that $\pi^{-1}(x_1)$ has color 3 and $\pi^{-1}(x_2)$ does not have color 3. We will show that we can start the coloring at x_2 without losing control over any of the footprints of any of the restrictions.

By the Neighboring Colors Lemma, no io-neighbor of a vertex with color 3 can be a 1-inner or a 2-inner restriction (and in this case, these are the only restrictions we have). Therefore, $\pi^{-1}(x_2)$ cannot be an inner restriction, and in particular it cannot be the start of an A-block of 1-inner and 2-inner restrictions. Moreover, the upstream io-neighbor of $\pi^{-1}(x_1)$ cannot be an inner restriction either, and the footprints of any restrictions further upstream would not include x_2. Therefore, we may start the coloring at x_2. □

Finally, we must consider what happens if we have entirely 4-cycles. We will not necessarily have any unrestricted 2s or unrestricted 3s, so the arguments above don't generalize to this case. But our assumptions about the 2-factor provide so much structure that we can make a very specific argument to handle this case.

Lemma 4.6. *If the 2-factor is entirely composed of 4-cycles, then there is a pair of vertices x_1 and x_2, adjacent on the inner cycle such that the following two conditions hold:*

- *$\pi^{-1}(x_1)$ and $\pi^{-1}(x_2)$ have different colors,*
- *any restrictions whose footprints contain x_2 can be met even if x_2 is the first vertex to be colored.*

Proof. We have no 2-inner restrictions, so if we ever have unrestricted 2s, we will be able to find a good starting point. This could happen if two 2-outer restrictions are adjacent (the constraint on both of them will be dropped). Therefore we will assume that the 2-outer restrictions are never io-neighbors.

Since each 4-cycle contributes two 2-outer restrictions, one 1-inner restriction and one 3-inner restriction, exactly half of the vertices are 2-outer restrictions. To avoid having any two 2-outer restrictions being io-neighbors, they must alternate strictly. This means that every 1-inner restriction and every 3-inner restriction is immediately downstream of a 2-outer restriction. By Proposition 4.3, we can drop the inner restrictions: they will automatically be satisfied if the 2-outer restrictions are satisfied.

However, every vertex in the cycle is in the footprint of a 2-outer restriction, so there is no place to start the coloring that does not interfere with having control over some footprint. If there is ever a case where the upstream io-neighbor and the downstream io-neighbor of a 2-outer restriction have the same color, then Proposition 4.1 applies, and we can start the coloring there.

If Proposition 4.1 does not apply, it follows that the graph is highly structured: it must be of the following form: $2, 1, 2, 3, 2, 1, 2, 3, \ldots$ etc., and it must have length a multiple of 4. In this case, we may start the coloring with the first vertex of any 2-outer restriction.

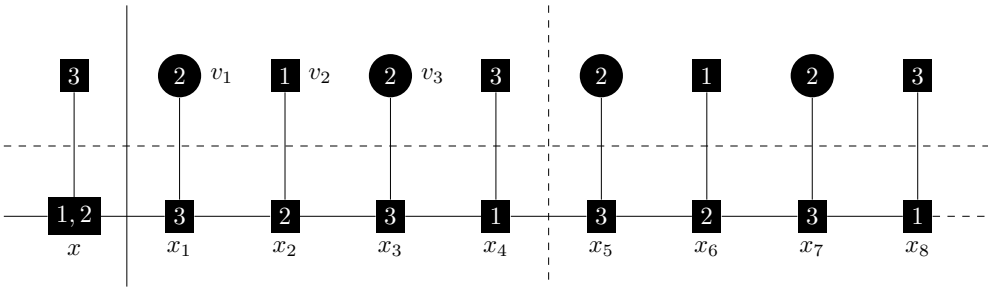


Figure 4.18: The highly structured case when the 2-factor is entirely made up of 4-cycles.

The color at x_1 is fixed, because the first vertex colored must leave two colors available for the final vertex colored. Fixing color 3 at x_1 determines color 2 at x_2. To meet the restriction at v_3 we must give x_3 color 3 and x_4 color 1. This forces x_5 to have color 3, and we can see that the whole graph breaks down into groups of 4 where the colors repeat in the order 3, 2, 3, 1. Because we have this repeating pattern, we know that we will end with the final vertex x receiving color 1, and so the restrictions at v_1

and v_2 will be satisfied. □

We end this chapter with the full coloring algorithm:

Theorem 4.2. *There exists a colorful path coloring for all generalized cycle permutation graphs.*

Proof. If the graph has chromatic number 2, then this is trivially true, so we assume that the graph has chromatic number 3.

Step 1: Color the 2-factor with three colors, so that the only types of restrictions are the four discussed: 2-outer, 1-inner, 2-inner, and 3-inner restrictions.

Step 2: Fix a direction around the cycle to be 'downstream' (this can be chosen arbitrarily).

Step 3: Cycle the coloring of the 2-factor so that no 2-inner restriction and 2-outer restriction have overlapping footprints (by Lemma 4.2).

Step 4: By the Lemmas 4.4, 4.5, and 4.6, find a good starting point for the coloring.

Step 5: Color the first vertex in such a way that the last vertex still has 2 colors available.

Step 6: Sweep around the cycle, coloring footprints and groups of overlapping footprints as needed to (mutually) satisfy their corresponding restrictions. At every point we will have at least one color available.

Step 6 needs to be verified. We know that by the Footprint Lemma, we will always be able to satisfy individual restrictions, and by Lemmas 4.1 and 4.3 we will be able to satisfy any occurring groups of restrictions with overlapping footprints (we have already excluded a 2-inner at io-distance exactly two upstream from a 2-outer, by Lemma 4.2). When we chose a starting point for the coloring, we required that if we lost control over the footprint of any restriction we would still be able to meet it. Therefore, we will in fact be able to mutually satisfy all the restrictions. □

Chapter 5

Generalizations of the Goulden–Jackson Cluster Method

5.1 Introduction

Suppose we are given a finite alphabet, and a finite set of forbidden words in this alphabet.[1] We would like to know how many n-letter words in our alphabet avoid the forbidden words as subwords or *factors*, i.e. strings of consecutive letters. In order to do this we will find the generating function for the number of such words. In Section 5.2 we describe a straightforward, recursive approach to solving this using ordinary generating functions. The remainder of this article deals with the Goulden–Jackson cluster method, a powerful method that considers overlap between forbidden factors in computing generating functions. The Goulden–Jackson cluster method was introduced in [18] and [19] and described very clearly and concisely in [27]. For earlier work see [20], and further extensions can be found in [25]. Applications of the Goulden–Jackson cluster method to genomics can be found in [23, 31, 22, 24]. In Section 5.3 we review the classical Goulden–Jackson cluster method. We then describe some modifications to the original Goulden–Jackson problem as follows: In Section 5.4 we modify the Goulden–Jackson cluster method to take single letter weights into account. In Section 5.5 we include double letter (i.e. pairwise) weights, and in Section 5.6 we consider triple letter weights. Further generalizations are described in Section 5.7.

All the methods discussed have been implemented in a Maple package. The Maple package, which includes documentation, can be found at the website for these results [2].

[1] This entire chapter was joint work with Debbie Yuster.

[2] The website for these results is located at `http://www.math.rutgers.edu/~ekupin/GJ.html`.

5.2 The Straightforward Recursive Approach

Given a finite alphabet A and a set of forbidden or 'bad' words B, we would like to find $a(n)$, the number of n-letter words in the alphabet A that do not contain any members of B as factors. Rather than find $a(n)$ directly, we will find the generating function

$$f(t) = \sum_{n=0}^{\infty} a(n)t^n. \tag{5.1}$$

In order to find this generating function, we need not use the Goulden–Jackson cluster method, which will be described in Section 5.3. We can use a straightforward recursive approach, though as we will demonstrate, this method will not be as efficient as Goulden–Jackson. The approach contained in this section was described by Dr. Doron Zeilberger in his Spring 2008 Experimental Math class at Rutgers University.

We first illustrate the approach with an example. Suppose $A = \{a, b\}$ and $B = \{abb, ba\}$. We will start by decomposing the set of allowed words according to their first letter. Denote by $[B, a_i, \{w_1, ...w_n\}]$ the set of words avoiding members of B, starting with a_i, and avoiding any word in $\{w_1, ...w_n\}$ as an initial subword. We can express the allowable words with the decomposition

$$[B, *] \leftrightarrow \{\text{empty\_word}\} \cup [B, a, \{\}] \cup [B, b, \{\}], \tag{5.2}$$

where $[B, *, \{\}]$ denotes the set of all words in alphabet A avoiding the words in B (with no additional restrictions). Consider the set of allowed words beginning with a. Any such word is either a itself, or consists of a followed by a smaller word starting with either a or b. What are the restrictions on the smaller word, following the initial letter a? If it starts with a, it must still avoid abb and ba, but there are no additional restrictions. However, if the word following the initial letter a begins with b, it must avoid abb and ba, but in addition it must not begin with bb, so as to avoid the forbidden word abb. This gives a correspondence among different sets of words. Translating the above example into this notation, we have:

$$[B, a, \{\}] \leftrightarrow \{a\} \cup [B, a, \{\}] \cup [B, b, \{bb\}], \tag{5.3}$$

where the latter two terms on the right hand side describe the allowable subwords following the initial letter a.

Now let us consider the set of allowable words beginning with b. Either the word is b itself, or consists of b followed by a smaller word starting with either a or b. Of course the letter following the initial b cannot be a, since this would form the forbidden word ba, but rather than exclude this *a priori*, we will instead say that any word following the initial b and starting with a must not begin with the word a. Of course no words satisfy this condition so the corresponding set will be empty. Any word following the initial b that starts with b must avoid the forbidden words in B but has no additional restrictions. To summarize, we have

$$[B, b, \{\}] \leftrightarrow \{b\} \cup [B, a, \{a\}] \cup [B, b, \{\}]. \tag{5.4}$$

Eventually we will turn these correspondence relations into equations, by taking a weighted count of the set elements. First, in order to solve explicitly for the latter two sets in the above relation, we must decompose the remaining sets on the right hand sides of (5.3) and (5.4), as well as any new sets arising from those relations. This leads to the following:

$$\begin{aligned} [B, b, \{bb\}] &\leftrightarrow \{b\} \cup [B, a, \{a\}] \cup [B, b, \{b\}] \\ [B, a, \{a\}] &\leftrightarrow \emptyset \\ [B, b, \{b\}] &\leftrightarrow \emptyset. \end{aligned}$$

If one simply wants to know the number of allowable n-letter words, a generating function such as the one given in equation (5.1) can be found. However, it is possible to find variant generating functions which give more information about the allowable words. These variants will be described in later sections. In order to find the various generating functions, we will make use of a weighted counting system, performing a weighted count of the words in the sets above. Taking weights in (5.5) gives the desired generating function. The particular 'weight' used varies based on the method (to be described in the following sections) so we postpone further calculations. It is worth pointing out, however, that one must be careful not to merely sum the weights of the sets on the right hand sides of the correspondence relations. Rather, it is necessary to account for the weights of the truncated initial letters, as well as any transition weights that may arise. See Example 5.1 for further details.

The Maple code for this recursive method can be found in our accompanying Maple package under the function names `RecursiveSingle`, `RecursiveDouble`, and `RecursiveProbDouble`. These functions implement the straightforward recursive analogues of the cluster method generalizations to be described in Sections 5.4, 5.5, and 5.7.1, respectively.

5.3 Basic Goulden–Jackson Cluster Method

We borrow from [27] in briefly reviewing the basic Goulden–Jackson cluster method, and encourage the reader to consult this source for a more detailed exposition.

In order to find the generating function given in equation (5.1), we will do a weighted count of *marked words*. A *marked word* is a pair $(w; S)$, where w is a word in the alphabet A, and S is an arbitrary multiset whose entries are members of $Bad(w)$, the forbidden words of B contained as factors in w. We allow repetition in S since a word may contain several copies of a given forbidden word. If no subset S is specified, we assume it is the empty set. We define the weight of a marked word as $\text{weight}(w; S) = (-1)^{|S|}t^{|w|}$, where $|S|$ is the cardinality of S and $|w|$ is the length of w. The weight of a set of marked words is obtained by summing the weights of the marked words in the set. The generating function from equation (5.1) now becomes

$$f(t) = \sum_{w \in A^*} \sum_{S \subset Bad(w)} (-1)^{|S|} t^{|w|}, \tag{5.5}$$

where A^* is the set of all words in the alphabet A. In order to see why this is valid, consider an arbitrary word w in our alphabet. This word will appear as the first argument in 2^k marked words, where k is the number of forbidden subwords contained in w. In equation (5.5), we sum over all possible subsets of $Bad(w)$. Thus if w contains no forbidden subwords, it will be counted exactly once in the above sum. If w contains k forbidden subwords, $k > 0$, the number of times it will be counted in the above sum is:

$$\sum_{i=0}^{k} (-1)^i \binom{k}{i} = (1 + (-1))^k = 0.$$

Thus every allowable word is counted once, while words containing forbidden words

are not counted. We have verified the equivalence of equation (5.5) and our original generating function, given in equation (5.1).

We call a marked word $(w; S)$ a *cluster* if neighboring factors in S overlap (i.e. are not disjoint) in w, and the forbidden words of S span all of w. For example, if our alphabet A is $\{a, b, c\}$ and the set of forbidden words B is $\{ba, aca\}$, then the marked word $(bacaca; \{ba, aca, aca\})$ is a cluster. The marked word $(bacaca; \{aca, aca\})$ is not a cluster because the factors in S do not span all of w (the first b is not part of a factor in S), and the marked word $(acaba; \{aca, ba\})$ is not a cluster because the factors in S do not overlap. We will denote the set of all (nonempty) clusters by $\mathcal{C}$.

We now decompose $\mathcal{M}$, the set of marked words, into three groups: the empty word, marked words beginning with a letter that is not part of any cluster, and marked words beginning with a cluster. We thus obtain the decomposition

$$\mathcal{M} = \{\text{empty\_word}\} \cup A\mathcal{M} \cup \mathcal{C}\mathcal{M},$$

where an element of $A\mathcal{M}$ consists of a single letter of the alphabet A prepended to a marked word and an element of $\mathcal{C}\mathcal{M}$ consists of a cluster prepended to a marked word. Let m be the number of letters in A. By taking weights on both sides of the preceding equation, we obtain

$$\text{weight}(\mathcal{M}) = 1 + mt \cdot \text{weight}(\mathcal{M}) + \text{weight}(\mathcal{C})\text{weight}(\mathcal{M}).$$

But $\text{weight}(\mathcal{M})$ equals $f(t)$, as shown in equation (5.5), so by substituting and solving for $f(t)$ we get

$$f(t) = \frac{1}{1 - mt - \text{weight}(\mathcal{C})} \tag{5.6}$$

and it remains to solve for $\text{weight}(\mathcal{C})$, which we will call the *cluster generating function*.

In order to find $\text{weight}(\mathcal{C})$, we partition the set of clusters $\mathcal{C}$ according to the first forbidden word of the cluster. Let $\mathcal{C}[v]$ denote the set of clusters starting with forbidden word v. Then $\mathcal{C} = \bigcup_{v \in B} \mathcal{C}[v]$, and $\text{weight}(\mathcal{C}) = \sum_{v \in B} \text{weight}(\mathcal{C}[v])$.

In order to find $\text{weight}(\mathcal{C}[v])$, we will further decompose $\mathcal{C}[v]$ as follows: consider a cluster in $\mathcal{C}[v]$. Either it consists of v alone, or we can remove v from the list of forbidden words in our marked cluster, and what remains will contain a smaller cluster,

beginning with some bad word u such that some initial subword of u coincides with some final subword of v. For example, consider the cluster $(bacaca; \{ba, aca, aca\})$, where the alphabet and forbidden word set are as above. This cluster is in $\mathcal{C}[ba]$. Removing the initial forbidden word ba leaves a new cluster $(acaca; \{aca, aca\})$ in $\mathcal{C}[aca]$.

Let $O(v, u)$ be the set of possible 'overlaps' of v and u, that is, all possible nonempty intersections of final subwords of v with initial subwords of u. Each of these overlaps corresponds to a way that our cluster can have its first two forbidden words be v and u, respectively. To create the smaller cluster we will peel off exactly the part of v that does not overlap with u. For any word v and a final subword r of v, $v\backslash r$ will denote the word obtained by chopping r from the end of v. For example, $abcb\backslash cb = ab$. This leads to the decomposition

$$\mathcal{C}[v] \leftrightarrow \{(v; \{v\})\} \cup \bigcup_{u \in B} \bigcup_{r \in O(v,u)} \Big(\{v\backslash r\} \cdot \Big),$$

where $W_1 \cdot W_2$ is the concatenation of W_1 with W_2. Taking weights, we obtain the following linear equations. Note that the sum over $u \in B$ is negative (i.e. multiplied by -1) in order to compensate for having reduced the number of bad words in our cluster by one (because we are calculating weights of clusters containing one fewer bad word than the clusters in $\mathcal{C}[v]$).

$$\text{weight}(\mathcal{C}[v]) = \text{weight}((v; \{v\})) - \sum_{u \in B} \Big(\text{weight}(\mathcal{C}[u]) \cdot \sum_{r \in O(v,u)} \text{weight}(v\backslash r) \Big)$$

We can explicitly calculate $O(v, u)$, and so by writing this equation for $\mathcal{C}[v]$ for all $v \in B$, we obtain a sparse system of $|B|$ linear equations in $|B|$ unknowns. Solving for the weight$(\mathcal{C}[v])$ and summing them gives us weight$(\mathcal{C})$, which can then be substituted into equation (5.6), giving the desired generating function.

Variations of the Basic Cluster Method

By changing how the weight of a word is defined, we can alter the interpretation of the resulting generating function. In the following sections we present several such variations. All the variations keep track of how many words of each length avoid the set

of forbidden words. The first variation, described in Section 5.4, also takes into account how many times each letter appears in any given 'good' word, by adding extra variables into the weight function. One possible use of this is to substitute probabilities for these variables, thus giving a generating function which takes into account a probability distribution on the alphabet. Several other variations mentioned later also take into account double letter, or pairwise weights. These are variables corresponding to each ordered pair of letters in the alphabet. This allows tracking of which consecutive letter combinations occur, and can also allow for double letter probabilities to be filled in. Similarly, Section 5.6 has variables in the weight function corresponding to ordered triples of letters. In the sections that follow, we describe these modifications to the basic Goulden-Jackson cluster method in detail.

5.4 Single Letter Weights

In order to keep track not only of how many words of a certain length avoid certain subwords, but also which letters these words contain, we will redo the basic cluster method, using a different weight enumerator. The variation discussed in this section was initially described in [27]. The weight of a marked word $(w; S)$ (where $w = w_1 w_2 \cdots w_k$) will be

$$(-1)^{|S|} t^{|w|} x_{w_1} x_{w_2} \ldots x_{w_k}.$$

For example, $\text{weight}(abcab; \{\}) = t^5 (x_a)^2 (x_b)^2 x_c$.

As in the original application of the Goulden–Jackson method, we use the decomposition $\mathcal{M} = \{\text{empty\_word}\} \cup A\mathcal{M} \cup \mathcal{C}\mathcal{M}$. This leads to the following recursive formula for $\text{weight}(\mathcal{M})$:

$$\text{weight}(\mathcal{M}) = 1 + t \sum_{a \in A} x_a \text{weight}(\mathcal{M}) + \text{weight}(\mathcal{C})\text{weight}(\mathcal{M}).$$

Note that, since we are keeping track of letter weights, the second term records not just how many letters are in A, but exactly which ones appear. Simplifying, we get

$$\text{weight}(\mathcal{M}) = \frac{1}{1 - t \sum_{a \in A} x_a - \text{weight}(\mathcal{C})}. \tag{5.7}$$

All that remains is to solve for the cluster generating functions, weight($\mathcal{C}$). We do this exactly as in the original Goulden–Jackson cluster method, with the same decomposition: weight($\mathcal{C}$) $= \sum_{v\in B}$ weight($\mathcal{C}[v]$). We will write the same system of linear equations as before, except that the weight function is different. In particular, we still have

$$\text{weight}(\mathcal{C}[v]) = \text{weight}((v;\{v\})) - \sum_{u\in B}\Big(\text{weight}(\mathcal{C}[u])\cdot \sum_{r\in O(v,u)} \text{weight}(v\backslash r)\Big)$$

for all $v \in B$, which becomes

$$\text{weight}(\mathcal{C}[v]) = -t^{|v|}x_{v_1}\dots x_{v_{|v|}} - \sum_{u\in B}\Big(\text{weight}(\mathcal{C}[u])\cdot \sum_{r\in O(v,u)} t^{|v\backslash r|}x_{v_1}\dots x_{v_{|v\backslash r|}}\Big).$$

Solving for weight($\mathcal{C}[v]$) for each forbidden word v and substituting back into equation (5.7) yields the desired generating function.

Example 5.1. Find the generating function of all words in the alphabet $\{a, b\}$ avoiding the forbidden words abb and ba.
We will find the generating function in two ways: (1) using the cluster method described in this section, and (2) using the straightforward recursive approach from Section 5.2.

1. We have:

$$\begin{aligned}\text{weight}(\mathcal{C}[abb]) &= -t^3x_ax_b^2 - \text{weight}(\mathcal{C}[ba])t^2x_ax_b\\ \text{weight}(\mathcal{C}[ba]) &= -t^2x_bx_a - \text{weight}(\mathcal{C}[abb])tx_b\end{aligned}$$

   from which

$$\begin{aligned}\text{weight}(\mathcal{C}[abb]) &= \frac{-t^3x_ax_b^2 + t^4x_a^2x_b^2}{1-t^3x_ax_b^2}\\ \text{weight}(\mathcal{C}[ba]) &= -t^2x_bx_a + \frac{t^4x_ax_b^3 - t^5x_a^2x_b^3}{1-t^3x_ax_b^2},\end{aligned}$$

   and therefore weight($\mathcal{C}$) $= \dfrac{-t^2x_ax_b - t^3x_ax_b^2 + t^4x_a^2x_b^2 + t^4x_ax_b^3}{1-t^3x_ax_b^2}$. Substituting this into equation (5.7) yields the desired generating function:

$$\text{weight}(\mathcal{M}) = \frac{1-t^3x_ax_b^2}{1-tx_a-tx_b+t^2x_ax_b}.$$

Taking the first few terms of the Taylor expansion of this generating function yields:

$$1 + (x_a + x_b)t + (x_a x_b + {x_a}^2 + {x_b}^2)t^2 + ({x_a}^2 x_b + {x_a}^3 + {x_b}^3)t^3 + ({x_a}^3 x_b + {x_a}^4 + {x_b}^4)t^4 + O(t^5)$$

The constant term 1 corresponds to the empty word. The coefficients of powers of t correspond to the allowable words. For example, the coefficient of t^3 corresponds to the permissible 3-letter words aab, aaa, and bbb.

2. Returning to the notation of Section 5.2, we need to take the weight of $[B, *, \{\}]$. Recall the following set decompositions:

$$\begin{aligned}
[B, *, \{\}] &\leftrightarrow \{\text{empty_word}\} \cup [B, a, \{\}] \cup [B, b, \{\}] \\
[B, a, \{\}] &\leftrightarrow \{a\} \cup [B, a, \{\}] \cup [B, b, \{bb\}] \\
[B, b, \{\}] &\leftrightarrow \{b\} \cup [B, a, \{a\}] \cup [B, b, \{\}] \\
[B, b, \{bb\}] &\leftrightarrow \{b\} \cup [B, a, \{a\}] \cup [B, b, \{b\}] \\
[B, a, \{a\}] &\leftrightarrow \emptyset \\
[B, b, \{b\}] &\leftrightarrow \emptyset.
\end{aligned}$$

It remains to take weights of all the sets listed, from the bottom up, and solve for the unknown weights. We must be careful, however, to distinguish between identical sets on the left hand sides and right hand sides of the correspondence relations. For example, consider the correspondence

$$[B, a, \{\}] \leftrightarrow \{a\} \cup [B, a, \{\}] \cup [B, b, \{bb\}].$$

The weight of the left hand side is simply $\text{weight}([B, a, \{\}])$, while the latter two sets on the right hand side are assumed to have had their initial letter a removed. Thus, the total weight of the right hand side is $\text{weight}(a) + \text{weight}(a) \cdot \text{weight}([B, a, \{\}]) + \text{weight}(a) \cdot \text{weight}([B, b, \{bb\}])$. Solving from the bottom up,

we find:

$$\begin{aligned}
\text{weight}([B,b,\{b\}]) &= 0\\
\text{weight}([B,a,\{a\}]) &= 0\\
\text{weight}([B,b,\{bb\}]) &= \text{weight}(b)\\
\text{weight}([B,b,\{\}]) &= \text{weight}(b)+\text{weight}(b)\text{weight}([B,b,\{\}])\\
\text{weight}([B,a,\{\}]) &= \text{weight}(a)+\text{weight}(a)\text{weight}([B,a,\{\}])\\
&\quad +\text{weight}(a)\text{weight}([B,b,\{bb\}])\\
\text{weight}([B,*,\{\}]) &= \text{weight}(\text{empty_word})+\text{weight}([B,a,\{\}])\\
&\quad +\text{weight}([B,b,\{\}])
\end{aligned}$$

Solving for each left hand side quantity and substituting into the last equation, which is the equation for weight($\mathcal{M}$), we find:

$$\begin{aligned}
\text{weight}(\mathcal{M}) &= 1+\frac{tx_a+t^2x_ax_b}{1-tx_a}+\frac{tx_b}{1-tx_b}\\
&= \frac{1-t^3x_ax_b^2}{1-tx_a-tx_b+t^2x_ax_b}.
\end{aligned}$$

We have implemented this modification of the original Goulden–Jackson cluster method, and the code is available in our accompanying Maple package under the function name `SingleGJ`.

5.5 Double Letter Weights

Sometimes we would like to keep track not just of how many times each letter appears in a word, but also which consecutive letter pairs appear. This could be relevant, for example, if studying English words, when the pair 'QU' is many times more likely to appear than the pair 'QB'. In order to keep track of such data, we introduce *double letter weights*, that is, variables which represent the occurrence of consecutive letter pairs.

To include double letter weights, the weight of a marked word $(w;S)$, where $w = w_1w_2w_3\cdots w_k$, will now be

$$(-1)^{|S|}t^k(x_{w_1}\dots x_{w_k})(x_{w_1,w_2}x_{w_2,w_3}\dots x_{w_{k-1},w_k}).$$

We will denote this new weight function $W((w;S))$. For example, $W((cat;\{\})) = t^3(x_cx_ax_t)(x_{c,a}x_{a,t})$. This new weight function does not have all of the nice properties of weight functions we have seen in the earlier methods. In particular, concatenation of words no longer corresponds to a simple multiplication of weights. To see why this is true, consider the word *abab* as the result of concatenating *ab* with itself. In this case, $W((uu;\{\}))$ does not equal $W((u;\{\}))^2$. $W((ab;\{\})) = t^2(x_ax_b)(x_{a,b})$ and so $W((ab;\{\}))^2 = t^4(x_a)^2(x_b)^2(x_{a,b})^2$, while $W((abab;\{\})) = t^4(x_a)^2(x_b)^2(x_{a,b})^2x_{b,a}$.

In general, whenever we concatenate two strings we need to account for the double letter weight that crosses from one string to the next. We call this the extra factor the *transition weight*. The original cluster method involves decomposing $\mathcal{M}$, then using the fact that a disjoint union of sets corresponds to addition of weight functions, and concatenation corresponds to multiplication. We can still use this basic principle, but we must be more careful with concatenation. In particular, whenever we concatenate strings we will need to know the last letter of the first string and the first letter of the second string, in order to be able to multiply by the appropriate transition weight. This forces us to change how $\mathcal{M}$ is decomposed.

In the original method, we used the decomposition

$$\mathcal{M} = \{\text{empty\_word}\} \cup A\mathcal{M} \cup \mathcal{C}\mathcal{M}.$$

This involves concatenation in two places: in the second term we concatenate an arbitrary marked word to a single letter, and in the third term we concatenate an arbitrary marked word to a cluster. To incorporate the transition weights we will need to know the first letter of an arbitrary marked word, as well as the last letter of an arbitrary cluster.

We start by splitting up the set of marked words according to their first letter. Let $\mathcal{M}_a$ be the set of marked words that start with a, $\mathcal{M}_b$ be the set of marked words that

start with b, and so on. We have

$$\mathcal{M} = \{\text{empty\_word}\} \cup \left(\bigcup_{a \in A} \mathcal{M}_a \right).$$

To find weight$(\mathcal{M}_a)$, we examine the different types of marked words that can begin with the letter a. Such a word may be a itself, or we can peel off the initial a to get a shorter marked word (assuming the initial a is not part of a cluster), or the word begins with a cluster that begins with a. Let B_a be the set of forbidden words beginning with a. We have the decomposition

$$\mathcal{M}_a = a \cup \left(\bigcup_{b \in A} a\mathcal{M}_b \right) \cup \left(\bigcup_{v \in B_a} \mathcal{C}[v]\mathcal{M} \right).$$

Accounting for the fact that the entire marked word may be a cluster, we get

$$\mathcal{M}_a = a \cup \left(\bigcup_{b \in A} a\mathcal{M}_b \right) \cup \left(\bigcup_{v \in B_a} \bigcup_{b \in A} \mathcal{C}[v]\mathcal{M}_b \right) \cup \left(\bigcup_{v \in B_a} \mathcal{C}[v] \right). \tag{5.8}$$

In this manner we keep track of the first letter of each marked word. It remains to address concatenation in the cluster generating functions.

Cluster Generating Functions

The decomposition of $\mathcal{C}[v]$ in the basic cluster method is based on the idea that if we have a cluster beginning with a bad word v, the cluster is either just that word, or we can peel the first word off and get a smaller cluster beginning with a bad word u that has some non-trivial overlap with v. Thus we have

$$\mathcal{C}[v] = v \cup \left(\bigcup_{u \in B} \bigcup_{r \in O(v,u)} (v \backslash r)\mathcal{C}[u] \right).$$

Since we are computing this for a specific v, we know what the last letter of $v \backslash r$ will be. Moreover, we know what the first letter of u will be, so the transition weight is easy to write down. Taking weights on both sides, we get

$$W(\mathcal{C}[v]) = W((v; \{v\})) - \left(\sum_{u \in B} \sum_{r \in O(v,u)} W(v \backslash r) \underbrace{x_{v_{|v \backslash r|}, v_{|v \backslash r|+1}}}_{\text{transition weight}} W(\mathcal{C}[u]) \right).$$

The cluster method is based on computing weights of individual letters and clusters, then computing weights of marked words in terms of the letters and clusters they contain. However, computing weights of such concatenations requires the inclusion of transition weights, since we are incorporating double letter weights into our weight-enumerators. In order to do this, it becomes necessary to keep track of the last letter of each cluster. When computing cluster weights, we successively remove leading forbidden words from a cluster, until we are left with a cluster consisting of only one forbidden word. By keeping track of its last letter, we are keeping track of the last letter of the original cluster. Thus, it suffices to record the last letter of single-word clusters only. We do this by adding a dummy variable to the end of each one-word cluster. This dummy variable records the last letter of a one-word cluster:

$$W(\mathcal{C}[v]) := W((v;\{v\}))\underbrace{\mathrm{End}_{v_k}}_{\text{dummy}} - \left(\sum_{u\in B}\sum_{r\in O(v,u)} W(v\backslash r)x_{v_{|v\backslash r|},v_{|v\backslash r|+1}}W(\mathcal{C}[u])\right).$$

Now we have a system of linear equations with variables $W(\mathcal{C}[v])$, for $v \in B$, and we can solve for each of these in terms of the dummy variables End_a, where $a \in A$. However, we don't want our final equation in terms of these variables. When we prepend a cluster to an arbitrary word beginning with a, and wish to take the resulting weight, we must first replace all occurrences of End_b (for any letter b) with the transition weight $x_{b,a}$. This gives us a way to calculate the transition weight directly from the cluster generating function, allowing us to use equation (5.8). We cannot write down exactly what the transition weight will be in general, since it will depend on which cluster we are looking at, but if we let $T_{\mathcal{C}}$ denote the transition weight calculated for a specific cluster, we have

$$W(\mathcal{M}_a) = tx_a + \sum_{b\in A} tx_a x_{a,b} W(\mathcal{M}_b) + \left(\sum_{f\in B_a}\sum_{c\in A} W(\mathcal{C}[f])\, T_{\mathcal{C}}\, W(\mathcal{M}_c)\right) + \sum_{f\in B_a} W(\mathcal{C}[f]).$$

Now that we can calculate the $W(\mathcal{M}_a)$, we can put them together to find $W(\mathcal{M})$. The Maple code for the cluster method, implementing the weight enumerator described

in this section, can be found in our accompanying Maple package under the function name `DoubleGJ`.

Example 5.2. We return to the setup from Example 5.1, redoing the example with double letter weights. Recall the problem: find the generating function of all words in the alphabet $\{a, b\}$ avoiding the forbidden words abb and ba.

Running `DoubleGJ` to get the generating function and doing a Taylor expansion, we see the first several terms are:

$$1 + (x_b + x_a)t + (x_a x_{a,b} x_b + x_a^2 x_{a,a} + x_{b,b} x_b^2)t^2 + (x_a^2 x_{a,a} x_{a,b} x_b + x_a^3 x_{a,a}^2 + x_{b,b}^2 x_b^3)t^3$$
$$+ (x_a^3 x_{a,a}^2 x_{a,b} x_b + x_a^4 x_{a,a}^3 + x_{b,b}^3 x_b^4)t^4 + (x_a^4 x_{a,a}^3 x_{a,b} x_b + x_a^5 x_{a,a}^4 + x_{b,b}^4 x_b^5)t^5 + O(t^6).$$

The coefficient of t^4 has terms corresponding to the allowable four-letter words $aaab$, $aaaa$, and $bbbb$.

Remark 5.1 (Comparison of straightforward recursive approach and Goulden–Jackson cluster method)**.** The straightforward recursive approach will usually require a system of at least $|A|+|B|$ equations and unknowns, often more (where A is the alphabet and B is the set of forbidden words). The Goulden–Jackson method with double letter weights first solves a system of size $|B|$ (the cluster generating functions), and then a system of size $|A|$. Even if the number of equations is roughly the same, by breaking things apart a bit we would still expect the Goulden–Jackson method to be slightly faster. In practice, however, the differences seem to be small for small examples. In general, the best approach depends on the situation. If there are many forbidden words with a lot of overlap, then Goulden–Jackson may take longer to compute the cluster generating functions, making it slower. However, if there are fewer, but longer, forbidden words, the straightforward recursive approach may require many more than $|A|+|B|$ equations and therefore take longer.

5.6 Triple Letter Weights

As a further generalization of the original cluster method, in this section we will keep track of the occurrences of each letter in a word, the occurrences of consecutive letter

pairs, and the occurrences of consecutive letter triples. We will use a new weight function, W', that accounts for all single letters, letter pairs and letter triples in a word. For example,

$$W'((abcabc;\{\})) = t^6(x_a^2x_b^2x_c^2)(x_{a,b}^2x_{b,c}^2x_{c,a})(x_{a,b,c}^2x_{b,c,a}x_{c,a,b}).$$

Note that, as in Section 5.5, we do not have $W'((uv;\{\})) = W'((u;\{\})) \cdot W'((v;\{\}))$. In fact in the example above we can see that $W'((abcabc;\{\}))$ has three extra terms that do not appear in $W'((abc;\{\}))^2$. The term $x_{c,a}$ is from the double letter transition weight, as described in Section 5.5. The other two extra terms, $x_{b,c,a}$ and $x_{c,a,b}$, correspond to the triples that cross between the two factors. These are also considered transition weights, and our main problem in this section will be modifying our methods so that it is possible to figure out exactly what these transition weights will be.

For considering double letter weights but not triple letter weights, it is necessary to know the last letter of the first string and the first letter of the second string when concatenating, in order to write down the appropriate transition weight. Now that we are also considering letter triples, we need to know the last two letters of the first string and the first two letters of the last string in order to get both triple transition weights. For example, to concatenate $abcd$ and $klmn$, the transition weights will come from the strings cdk, dkl, and dk, so in order to find these weights we need the final two letters of $abcd$, as well as the first two letters of $klmn$.

Recall that in our original setup, we decomposed the set of marked words $\mathcal{M}$ as follows:

$$\mathcal{M} = \{\text{empty\_word}\} \cup A\mathcal{M} \cup \mathcal{C}\mathcal{M}.$$

In this decomposition, we append marked words to individual letters, and marked words to clusters. In order to include triple letter weights we will need to know the first two letters of an arbitrary marked word. We will also need the last two letters of an arbitrary cluster.

In order to keep track of the first two letters of an arbitrary marked word, let $\mathcal{M}_{ab}$

be the set of marked words beginning ab, and decompose $\mathcal{M}$ as follows:

$$\mathcal{M} = \{\text{empty\_word}\} \cup S \cup \left(\bigcup_{a,b \in A} \mathcal{M}_{ab} \right).$$

Here S is the set of one-letter marked words. Taking weights gives us

$$W'(\mathcal{M}) = 1 + \sum_{a \in A} W'((a;\{\})) + \sum_{a \in A} \sum_{b \in A} W'(\mathcal{M}_{ab}). \tag{5.9}$$

To solve for $W'(\mathcal{M}_{ab})$, we decompose $\mathcal{M}_{ab}$ further. A marked word in M_{ab} could be the two-letter marked word $(ab;\{\})$, or the single letter a followed by an arbitrary marked word beginning with b, or it could consist of a cluster beginning with ab, followed by an arbitrary marked word. Let $B_{ab} \subset B$ be the set of all forbidden words that begin with ab. We may assume that there are no one-letter forbidden words (in that case we would simply remove that letter from the alphabet). Thus B is completely partitioned into the B_{ab}. We have

$$\mathcal{M}_{ab} = \{ab\} \cup \left(\bigcup_{c \in A} a\mathcal{M}_{bc} \right) \cup \left(\bigcup_{v \in B_{ab}} \mathcal{C}[v]\mathcal{M} \right).$$

By taking weights on both sides, using the substitution given in equation (5.9), and adding in the appropriate transition weights when we can, we get

$$\begin{aligned} W'(\mathcal{M}_{ab}) = W'(ab) + \sum_{c \in a} W'(a) \underbrace{x_{a,b}x_{a,b,c}}_{\text{transition}} W'(\mathcal{M}_{bc}) + \sum_{v \in B_{ab}} W'(\mathcal{C}[v]) \\ + \sum_{v \in B_{ab}} \sum_{c \in A} W'(\mathcal{C}[v])\, T_{\mathcal{C}}\, W'(c) + \sum_{v \in B_{ab}} \sum_{c,d \in A} W'(\mathcal{C}[v])\, T_{\mathcal{C}}\, W'(\mathcal{M}_{cd}) \end{aligned} \tag{5.10}$$

We aren't yet able to fill in the transition weights $T_{\mathcal{C}}$ when they occur at the end of a cluster. This is the same problem that occurs in the double letter case, and the solution is the same. Into the cluster generating function we will put the dummy variables End_{ab}, for all $a, b \in A$. They will go in the same place End_a went, and record the last two letters of a cluster. Then we can solve for the $W'(\mathcal{M}_{ab})$ in terms of the dummy variables and substitute for them when needed.

We have implemented the cluster method incorporating triple letter weights in the function `TripleGJ`. The code can be found in our accompanying Maple package.

Remark 5.2. While it is possible to further generalize this method to include weights for four-letter strings or higher, we can see even from triple letter weights why this might be problematic. For one thing, efficiency suffers greatly. Using only double letter weights leads to a system of $|A|$ linear equations, while including triple letter weights requires $|A|^2$ equations. In general, including weights of strings up to length k leads to a system of $|A|^{k-1}$ linear equations.

Moreover, small forbidden words become increasingly problematic. In the triple letter case, we relied on the fact that having a one-letter forbidden word is equivalent to looking at a smaller alphabet. If we wanted to weight four-letter strings, we would partition the forbidden words based on their first three letters, so two-letter forbidden words would be a problem. As we include longer and longer subwords in our weight function, we get more and more short forbidden word exceptions.

5.7 Further Generalizations

In this section we introduce further variations of the original Goulden–Jackson cluster method. These variations have been implemented in our accompanying Maple package, and we refer to the variations according to their corresponding function names in our software.

5.7.1 ProbDoubleGJ, ProbTripleGJ

`ProbDoubleGJ` and `ProbTripleGJ` are variants of `DoubleGJ` and `TripleGJ` (respectively), that are specifically designed for applications with an underlying Markov chain structure. We can think of the set of states in a Markov chain as an alphabet, and a word in that alphabet will correspond to a history of steps in the Markov process. We move to the next state (or equivalently, add the next letter to our word) with some probability that depends only on the current state.

`ProbDoubleGJ` returns a generating function for subword avoidance in which each word is weighted with all of the letter pairs it contains, as well as the single letter weight for the initial single letter only. For example, $\text{weight}((cat;\{\})) = t^3(x_c)(x_{c,a}x_{a,t})$. We can interpret the double letter weight as a conditional probability: $x_{a,b}$ is the probability we will move to state b, given that we are currently at state a. The initial single letter probability represents the probability of starting in a given state.

`ProbTripleGJ` yields a generating function for subword avoidance where each word is weighted according to the triple letter strings it contains, as well as its initial double letter pair. The weight of a word of length one is the single letter weight. For example,

$$\text{weight}((abcabc;\{\})) = t^6(x_{a,b})(x^2_{a,b,c}x_{b,c,a}x_{c,a,b}).$$

We can interpret the triple letter weight $x_{a,b,c}$ as the conditional probability of seeing c given that the letters immediately preceding it are ab, and the initial double letter weight as the probability of satisfying a two-state initial condition.

Note that these programs aren't contained in the original double letter weight and triple letter weight programs, in the sense that we can't get all the results here by a clever choice of the weights in those programs (described in Sections 5.5 and 5.6). Our goal here is to selectively weight only the single or double letters that show up at the beginning of a word, whereas in the other programs, we assigned a weight to every letter in a word, regardless of where it appeared. Nevertheless, with some simple modifications to the original programs, we can get the desired results.

In order to modify our original double letter weight method (implemented in the function `DoubleGJ`), we need only change the weight enumerator used. The new weight enumerator will consist of the initial single letter weight multiplied by $V(w;S)$, which we define to be the product of all consecutive letter pair weights. Define $V(a;\{\}) = t$ for all single letter words a. For example, $V(abba;\{\}) = x_{a,b}x_{b,b}x_{b,a}$. We first solve for the $V(\mathcal{M}_a)$ as in Section 5.5, yielding generating functions that enumerate all words beginning with a, and weighted only by their double letter components. It remains to multiply by initial letter weights and put things back together:

$$f(t) = 1 + \sum_{a\in A} x_a V(\mathcal{M}_a).$$

The situation is similar in adapting the triple letter weight method. We define $V'(w, S)$, which counts triple letter weights, i.e. $V'(abab; \{\}) = x_{a,b,a}x_{b,a,b}$, $V'(ab; \{\}) = t^2$, $V'(a; \{\}) = t$. By solving for the $V'(\mathcal{M}_a b)$ as in Section 5.6, we get generating functions for marked words, weighted only by their triple letter weights. Multiplying by the initial double letter weight and putting everything together yields:

$$f(t) = 1 + \sum_{a \in A} x_a t + \sum_{a,b \in A} x_{a,b} V'(\mathcal{M}_{ab}).$$

Example 5.3. Suppose we are given all initial letter probabilities and pairwise transition probabilities. For example, let's start with the alphabet $\{a, b\}$, initial letter probabilities $x_a = 0.75$, $x_b = 0.25$, and double letter probabilities $x_{a,a} = 0.5$, $x_{a,b} = 0.5$, $x_{b,a} = 0.7$, $x_{b,b} = 0.3$.

Suppose we would like to find the probability of avoiding the forbidden words *bbb* and *ab*. Running `ProbDoubleGJ` produces the generating function

$$f(t) = -\frac{1}{100} \cdot \frac{100t + 200 + 3t^3 + 25t^2}{t - 2}.$$

The first few terms of the Taylor expansion are

$$1 + t + \frac{5}{8}t^2 + \frac{131}{400}t^3 + \frac{131}{800}t^4 + O(t^5).$$

The coefficient of t^2 is the probability of seeing one of the three allowable two letter words (*aa*, *ba*, or *bb*). We can calculate this probability by subtracting the probability of *ab* from 1. The probability of *ab* is the initial probability of *a*, 0.75, multiplied by the digraph probability of *ab*, 0.5. Given a two letter word, we see that the probability of seeing the forbidden word *ab* is 3/8, thus the probability of an allowable two letter word is 5/8.

Example 5.4 (Modeling the English language)**.** We will consider a passage of written English as a long string over an alphabet of 27 characters: the 26 lowercase letters of the alphabet, and a character SP that corresponds to a space. To keep things simple, we ignore punctuation and capitalization. In this example we will use the words *character* and *string* when referring to the model and including the character SP, and *letter* and *word* when referring to English.

To use `ProbDoubleGJ`, we need to know information about the frequencies of single characters and character pairs. Where this information comes from can have a huge effect on the accuracy of this model. For details on how we obtained the letter frequencies used in this example, we refer the reader to Appendix A.

In addition to the table of frequencies, we will need a set of forbidden strings. For this example, we will use the forbidden string "SP, t, h, e". This corresponds to typing a word beginning with "*the*", a common word beginning.

Running `ProbDoubleGJ` outputs a rational function with degree 29 polynomials in both the numerator and the denominator. We will call this function $F(x)$, and we list the first several terms of its Taylor expansion:

$$\begin{aligned} F(x) = 1 + x + x^2 + x^3 + 0.9992162308 \cdot x^4 + 0.9992162308 \cdot x^5 + 0.9991288963 \cdot x^6 \\ + 0.9990341113 \cdot x^7 + 0.9989403663 \cdot x^8 + 0.9988466147 \cdot x^9 + 0.9987529643 \cdot x^{10} \\ + 0.9986592740 \cdot x^{11} + 0.9985656098 \cdot x^{12} + 0.9984719485 \cdot x^{13} + 0.9983782981 \cdot x^{14} \\ + 0.9982846557 \cdot x^{15} + 0.9981910224 \cdot x^{16} + 0.9980973977 \cdot x^{17} + 0.9980037819 \cdot x^{18} \\ + 0.9979101748 \cdot x^{19} + 0.9978165765 \cdot x^{20} + 0.9977229870 \cdot x^{21} + 0.9976294063 \cdot x^{22} \\ + 0.9975358344 \cdot x^{23} + 0.9974422712 \cdot x^{24} + 0.9973487168 \cdot x^{25} + O(x^{26}) \end{aligned}$$

The coefficient of x^n corresponds to the probability of avoiding "SP, t, h, e" in a length n string, according to the probability distributions given. It makes sense that the coefficients of x, x^2, and x^3 are all one, as the forbidden string is four characters long. Looking forward in the series, the coefficient of x^{100} is 0.9903570875, the coefficient of x^{200} is 0.9811110978, the coefficient of x^{300} is 0.9719514288, and the coefficient of x^{400} is 0.9628772746.

This means that if a monkey is banging keys on a typewriter according to the probability distributions given, even after 100 keystrokes (including the spacebar) the monkey only has a 1% chance of typing a word beginning "the...". This probability climbs to just under 4% after typing 400 keystrokes.

To compare, and to answer the age old question about monkeys being able to type Hamlet, according to our model, the probability that a monkey could type "to be or

not to be" after 300 keystrokes is $5.861724357 \cdot 10^{-21}$. More examples are available on the website, as well as some of the functions used to analyze these long strings of data, and the dictionary used in this example.

5.7.2 DoubleGJIF

The program `DoubleGJIF` is a generalization of `ProbDoubleGJ`. This function returns the generating function for subword avoidance where each word is weighted by all of its digraph weights as well as the initial single letter and final single letter weights (the `IF` stands for Initial-Final). The implementation allows for setting different values for the weight of a single letter depending on whether it is the first or the last letter in a word. If we set all of the final letter probabilities to 1, the program reduces to `ProbDoubleGJ`.

There is a way to modify `ProbDoubleGJ` to obtain `DoubleGJIF` - simply multiply by the final letter weights as they occur. Of course, locating the end of a word in the decomposition is a bit more complicated, but it turns out that there are only a few places where we need to add terms.

Recall that in `ProbDoubleGJ` we have the equation

$$f(t) = 1 + \sum_{a \in A} x_a V(\mathcal{M}_a),$$

and adapted from `DoubleGJ` we have

$$V(\mathcal{M}_a) = t + \sum_{b \in A} t x_{a,b} V(\mathcal{M}_b) + \left(\sum_{f \in B_a} \sum_{c \in A} V(\mathcal{C}[f])\, T_{\mathcal{C}}\, V(\mathcal{M}_c) \right) + \sum_{f \in B_a} V(\mathcal{C}[f]),$$

where V is the weight enumerator that weights words only with the letter pairs they contain.

Suppose a marked word ends with a single letter (as opposed to a cluster). We decompose the word by peeling off letters or clusters from the beginning of the word. Once we arrive at the final letter it will seem as if we are looking at a marked word consisting of a single letter. Therefore, if we multiply the term in the equation above that corresponds to a single letter with the appropriate final letter weight, we will have successfully modified the end of every marked word that ends in a single letter.

Similarly, in dealing with the marked words that end in a cluster, we need only change the term in the above sum that corresponds to a marked word made of one single cluster. In order to implement `DoubleGJ` we had to locate the end of the clusters, and so they are now flagged with the dummy variables $\text{End}_a, a \in A$. Normally, when we concatenate the empty word to a cluster (effectively ending a word with a cluster), we substitute 1 for the terms End_a, for all $a \in A$. Instead, we can substitute the final letter probabilities for End_a, and we will have successfully modified all the marked words that end in a cluster. Since every nonempty marked word must end with a single letter or a cluster, we have added in the final letter probability to every marked word.

5.7.3 DoubleGJst

All of the programs we have discussed so far return a generating function in terms of t, but in fact any of the programs can be modified so that they return a generating function in two variables, s and t:

$$f(s,t) = \sum_{n=0}^{\infty} \sum_{i=0}^{n} a(i,n) s^i t^n,$$

where $a(i,n)$ is the number (or weight, depending on the application) of words of length n that contain exactly i forbidden subwords (counted with multiplicity). For example, if $A = \{a, b\}$ and $B = \{abb, ba\}$, then three out of the sixteen four-letter words contain no subwords in B: *aaaa*, *aaab*, and *bbbb*. Ten of the words contain exactly one forbidden subword: *aaba*, *aabb*, *abaa*, *abab*, *abbb*, *baab*, *baaa*, *bbaa*, *bbab*, and *bbba*. Finally, three of the words contain exactly two forbidden subwords each: *baba*, *babb*, *abba*. Therefore, we have $a(0,4) = 3$, $a(1,4) = 10$, and $a(2,4) = 3$. Since this accounts for all 16 of the subwords of length 4, we must have that $a(3,4) = a(4,4) = 0$. Therefore, the coefficient of t^4 in $f(s,t)$ will be $3 + 10s + 3s^2$. In more complicated programs, $a(i,n)$ would give the weight of these words, not just the number of them. Not surprisingly, we can get this generatin! g function by modifying the weight enumerator in any of the above programs.

Suppose we have a word w that contains k forbidden subwords. The word w will be the first argument in 2^k marked words – one for each subset of the k forbidden

subwords. In the original Goulden–Jackson method, we multiplied the weight of the letter in w by $(-1)^{|S|}$, so that the number of times a word with k forbidden subwords will be be counted is

$$\sum_{i=0}^{k}(-1)^i\binom{k}{i} = (1+(-1))^k = 0^k,$$

which is 0 if $k > 0$ and 1 if $k = 0$. If we would like to keep track of the number of forbidden subwords a word contains, we can simply replace the (-1) in the weight function with $(s-1)$. Thus the weight of a marked word $(w; S)$, where $w = w_1w_2w_3\cdots w_k$, will be

$$(s-1)^{|S|}t^k(x_{w_1}\dots x_{w_k})(x_{w_1,w_2}x_{w_2,w_3}\dots x_{w_{k-1},w_k}).$$

Under this new weight function, the number of times a word containing k subwords will be counted is

$$\sum_{i=0}^{k}(s-1)^i\binom{k}{i} = (1+(s-1))^k = s^k,$$

as desired. The notion of including an extra variable to count the number of forbidden word occurrences was introduced in [27]. The functions `SingleGJst`, `DoubleGJst`, and `ProbDoubleGJst` in our accompanying Maple package are the respective analogues of `SingleGJ`, `DoubleGJ`, and `ProbDoubleGJ` incorporating the variable s.

Appendix A

Data for Example 5.4

In order to obtain frequencies for single letter occurrences as well as pairwise letter frequencies, we analyzed a list of $20,422$ distinct English words that we will refer to as the dictionary. We created a large transition probability matrix by taking the frequency of a character pair and then dividing by the number of occurrences of the initial character. For example, the Table A.1 corresponds to the probabilities that any of the 27 characters should follow a:

pr(a,a)=0.00037487,	pr(a,b)= 0.044235,	pr(a,c)= 0.059454,
pr(a,d)= 0.042885,	pr(a,e)= 0.0030739,	pr(a,f)= 0.010046
pr(a,g)= 0.033138,	pr(a,h)= 0.0039736,	pr(a,i)= 0.029090
pr(a,j)= 0.00067476,	pr(a,k)= 0.011771,	pr(a,l)= 0.11553
pr(a,m)= 0.039061,	pr(a,n)=0.14342,	pr(a,o)= 0.00074974
pr(a,p)= 0.034638,	pr(a,q)= 0.00089969,	pr(a,r)=0.12003
pr(a,s)= 0.052856,	pr(a,t)=0.14530,	pr(a,u)= 0.019793
pr(a,v)= 0.014020,	pr(a,w)= 0.0099715,	pr(a,x)= 0.0044235
pr(a,y)= 0.015070,	pr(a,z)= 0.0044235,	pr(a,SP)= 0.041086

Table A.1: The double letter probabilities for pairs beginning with a.

The last value, pr(a, SP), was computed not from a letter pair in English, but from the frequency of words in English that end with a. The overall sum is 1, since every occurrence of the letter a is either followed by another letter, or occurs at the end of a word.

Relatively speaking, the probabilities for the character a are fairly evenly distributed. To compare, the corresponding values for q are shown in Table A.2, and look quite different.

Of the 342 occurrences of the letter q in our dictionary, 341 of them are followed by the letter u, and exactly one of them is at the end of the word. Scrabble fanatics

pr(q,a)=0.0,	pr(q,b)= 0.0,	pr(q,c)= 0.0
pr(q,d)= 0.0,	pr(q,e)= 0.0,	pr(q,f)= 0.0
pr(q,g)= 0.0,	pr(q,h)= 0.0,	pr(q,i)= 0.0
pr(q,j)= 0.0,	pr(q,k)= 0.0,	pr(q,l)= 0.0
pr(q,m)= 0.0,	pr(q,n)=0.0,	pr(q,o)= 0.0
pr(q,p)= 0.0,	pr(q,q)= 0.0,	pr(q,r)=0.0
pr(q,s)= 0.0,	pr(q,t)=0.0,	pr(q,u)= 0.99708
pr(q,v)= 0.0,	pr(q,w)= 0.0,	pr(q,x)= 0.0
pr(q,y)= 0.0,	pr(q,z)= 0.0,	pr(q,SP)= 0.0029240

Table A.2: The double letter probabilities for pairs beginning with q.

will no doubt appreciate that our dictionary is incomplete. We further remark that our model ignores context, and the fact that some words are more common than others in written English.

The last row of the probability matrix will be the probalities pr(SP, a), pr(SP, b), etc. These are the initial letter probabilities, in other words, the probability a word begins with a particular letter. As a default, we set pr$(SP, SP) = 0$, ensuring that between two words there will only be one space. Calculating the single character frequencies is straightforward for the characters that are letters. We set the frequency of SP according to the number of words in the dictionary, with the idea that between each word there must be exactly one space.

References

[1] M. Aigner, M. Fromme. *A game of cops and robbers.* Discrete Appl. Math. 8 no. 1, 1–11, 1984.

[2] S. Akbari, F. Khaghanpoor, S. Moazzeni. *Colorful paths in vertex coloring of graphs.*

[3] S. Akbari, V. Liaghat, A. Nikzad. *Colorful paths in vertex coloring of graphs.* Electron. J. Combin. 18 (2011).

[4] M. Alishahi, A. Taherkhani, C. Thomassen. *Rainbow paths with prescribed ends.* Electron. J. Combin. 18 (2011).

[5] J. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations.* Cambridge University Press, 2003.

[6] J. Allouche and J. Shallit. *The ring of k-regular sequences.* Theoretical Computer Science Volume 98, 1992, Pages 163–197.

[7] J. Allouche and J. Shallit. *The ring of k-regular sequences, II.* Theoretical Computer Science Volume 307, 2003, Pages 3-29.

[8] T. Andreae. *On a pursuit game played on graphs for which a minor is excluded.*Journal of Combinatorial Theory, Series B 41, 1986, pp. 37–47

[9] E. Berlekamp and J. P. Buhler. *Puzzles Column, Emissary – MSRI Gazette.* p. 6, Fall 2009.

[10] E. Berlekamp, J. H. Conway, R. Guy. *Winning Ways for your mathematical plays.* Academic Press, London. 1982.

[11] E. Chiniforooshan. *A better bound for the cop number of general graphs.* J. Graph Theory 58 (2008), no. 1, 45–48.

[12] T.S. Ferguson. *On sums of graph games with last player losing.* International Journal of Game Theory, 3 (1974), 159–167.

[13] A. Flammenkamp. `http://wwwhomes.uni-bielefeld.de/achim/octal.html`

[14] A.S. Fraenkel *The vile, dopey, evil and odious game players.* Discrete Math. 312 (2012) 42–46, special volume in honor of the 80th birthday of Gert Sabidussi.

[15] A.S. Fraenkel. *Aperiodic subtraction games.* Electronic J. Combinatorics vol. 18(2) P19 12pp., 2011.

[16] P. Frankl. *Cops and robbers in graphs with large girth and Cayley graphs.* Discrete Applied Mathematics. 17 (3), 301–305, 1987.

[17] P. Frankl. *On a pursuit game on Cayley graphs.* Combinatorica. 7 (1), 1987, 67–70.

[18] I.P. Goulden and D.M. Jackson. *An inversion theorem for cluster decompositions of sequences with distinguished subsequences.* J. London Math. Soc. 20(2), 1979, pp. 567–576.

[19] I.P. Goulden and D.M. Jackson. *Combinatorial Enumeration,* with a foreword by Gian-Carlo Rota, Wiley-Interscience Series in Discrete Mathematics. Wiley, New York, 1983.

[20] L. J. Guibas and A. M. Odlyzko. *String overlaps, pattern matching, and nontransitive games.* J. Combin. Theory Ser. A 30, 1981, pp. 183–208.

[21] A. Guo. *Winning strategies for aperiodic subtraction games.* Available on the ArXiV: `http://arxiv.org/abs/1108.1239`

[22] B. Hao, H. Xie, Z. Yu and G. Chen. *Avoided Strings in Bacterial Complete Genomes and a Related Combinatorial Problem.* Annals of Combinatorics Conference on Combinatorics and Physics (Los Alamos, NM, 1998) 4, 2000 247-255.

[23] B. Hao. *Fractals from genomes exact solutions of a biology-inspired problem.* Physica A: Statistical Mechanics and its Applications 282, 2000, pp. 225–246.

[24] B. Hao, H. Xie, Z. Yu, and G. Chen. *Factorizable language: from dynamics to bacterial complete genomes.* Physica A: Statistical Mechanics and its Applications 288, 2000, pp. 10–20.

[25] Y. Kong. *Extension of GouldenJackson cluster method on pattern occurrences in random sequences and comparison with RgnierSzpankowski method.* Journal of Difference Equations and Applications Volume 11, 2005, pp. 1265–1271.

[26] M. Maamoun, H. Meyniel. *On a game of policemen and robber.* Discrete Applied Math. 17 (1987), 307–309.

[27] J. Noonan and D. Zeilberger. *The Goulden–Jackson cluster method: extensions, applications and implementations.* J. Differ. Equations Appl. 5, 1999, pp. 355–377.

[28] R. Nowakowski, P. Winkler. *Vertex-to-vertex pursuit in a graph.* Discrete Math. 43 (1983), no. 2-3, 235–239.

[29] P. D. Seymour, R. Thomas. *Graph searching, and a min-max theorem for tree-width.* J. Combin. Theory Series B. 58 (1993), 22–33.

[30] A. Quilliot. *A short note about pursuit games played on a graph with a given genus.* Journal of Combinatorial Theory Series B. 38 (1985), 89–92.

[31] H. Xie, and B. Hao. *Visualization of K-tuple distribution in procaryote complete genomes and their randomized counterparts,* in *Bioinformatics Conference*, Internation IEEE Computer Society, Los Alamitos, CA, 2002, pp. 31–42

[32] X. Zhu. *Circular chromatic number, a survey.* Discrete Mathematics, Vol. 229 (1-3) (2001), 371–410.