

An Algebraic Perspective on Game Theory

by

Zixuan Gao

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2024

© Zixuan Gao 2024

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Zixuan Gao was the sole author for Chapter 1, 2, and 4 which were written under the supervision of Dr Chrystopher L. Nehaniv and were not written for publication.

This thesis includes material that has been presented and published in academic venues. The exception to sole authorship of the material is as follows:

Research presented in Chapter 3:

Chapter 3 of this thesis is based on work that was presented at the 6th International Conference on Applied Mathematics, Modeling, and Computational Science (AMMCS 2023), held in Waterloo, Canada, from August 14–18, 2023. The full paper containing the content from Chapter 3 has been accepted for publication in the conference proceedings and is currently in press. Additionally, a refereed abstract of this work was published in the conference’s Book of Abstracts.

Citations:

Zixuan Gao, Attila Egri-Nagy, and Chrystopher L. Nehaniv. *Algebraic structure and complexity of games*. In D. Marc Kilgour, Herb Kunze, Roman Makarov, Roderick Melnik, and Xu Wang (Eds.), *Recent Developments in Mathematical, Statistical and Computational Sciences (AMMCS 2023)*, Springer Proceedings in Mathematics & Statistics. Springer Nature, in press.

Zixuan Gao, Attila Egri-Nagy, and Chrystopher L. Nehaniv. *Algebraic structure and complexity of games*. In *Book of Abstracts of the 6th International Conference on Applied Mathematics, Modeling and Computational Science (AMMCS 2023)*, page 170, Waterloo, Canada, 2023. Published refereed abstract.

As lead author of these publications, I was responsible for contributing to conceptualizing study design, carrying out data collection and analysis, and drafting and submitting manuscripts. My coauthors provided guidance during each step of the research and provided feedback on draft manuscripts.

Abstract

Even though the fields of game theory and algebra are well-established, limited literature analyzes games using modern algebraic theories. In 2009, Rhodes published a game complexity study titled *Complexity of Games via Applications of Automata Theory and Algebra*, which uses Krohn-Rhodes theory to analyze the best-play automata of the two-player complete-information sequential games. It defines the key terminologies, derives the automata representations from complete game trees, and proposes several theorems and conjectures. Rhodes' work heavily inspires this thesis as we expand on the established theoretical frameworks and challenge some of Rhodes' claims.

We begin by presenting the essential terminologies of the semigroup theory and Krohn-Rhodes theory. We then examine the existing methods for complexity analysis and propose novel algorithms to calculate the complexity lower bound for transformation semigroups. These algorithms underwent multiple iterations of improvements and contributed considerably to the analysis in the subsequent chapters.

A major component of the thesis is a direct expansion of Rhodes' work with respect to the two-player complete-information sequential games. We represent the gameplay machines identical to Rhodes'; however, we also consider symmetries in the structures of games to reduce the state space. We then developed practical methodologies to evaluate the algebraic structure of several simple games, such as Tic-Tac-Toe and used them to verify some of Rhodes' conjectures.

Another major component of the thesis is the algebraic analysis of extensive-form games, which encompass a much broader category of games (including the complete-information sequential games). This exploration encountered many challenges, in particular, the analysis of nondeterminism associated with mixed strategy profiles. We proposed the usage of classic powerset construction of the nondeterministic finite automata, but it turned out to be highly problematic due to state explosion and the possibility of collapsing into a single 'superset' state. To overcome these issues, we proposed an alternative construction that captures the nondeterminism without the drawbacks of the powerset construction. This construction enables us to analyze the algebraic structure and complexity of games effectively.

Overall, this thesis serves to bridge the gap between the classical game theory and the algebraic automata theory.

Acknowledgements

Completing this thesis has been a challenging task but also very rewarding. It would not have been possible without the support of many individuals.

Firstly, I thank my advisor, Professor Chrystopher L. Nehaniv, for his guidance and support throughout my research. I would also like to thank Professor Attila Egri-Nagy for his assistance and advice with the research.

Secondly, I thank Professor Martin Pei, Professor John Thistle and Professor Attila Egri-Nagy for agreeing to read and provide feedback on my thesis.

Thirdly, I also thank the hard-working people who funded my research, including my advisor, Professor Nehaniv, and the funding body of the Engineering Excellence Master's Fellowships. This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number RGPIN-2019-04669.

Fourthly, I would like to thank my parents and friends, who have provided me with emotional support in challenging times.

Thank you all for being part of my achievement.

Dedication

To mom and dad. Thanks for keeping the interest rates low on everything I owe you.
You are the best!

Table of Contents

Author’s Declaration	ii
Statement of Contributions	iii
Abstract	iv
Acknowledgements	v
Dedication	vi
List of Figures	x
List of Tables	xiii
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Organization	1
1.3 Essential Terminologies from Semigroup Theory	2
1.3.1 Automata and Transformation Semigroups	2
1.3.2 Krohn-Rhodes Theory	3

2	Algorithms to Determine a Lower Bound on Krohn-Rhodes Complexity	5
2.1	Review of Essential Dependencies	5
2.2	Implementing Essential Dependency Check	7
2.3	Naive Chain of Levels Algorithm	9
2.4	Chain of Sets Algorithm	15
2.5	Optimizations	17
2.6	Conclusion and Insights	21
3	Complexity of Simple Gameplay Automata	23
3.1	Rhodes' Gameplay Machines	23
3.2	Rhodes' Conjectures on the Complexity of Games	24
3.3	Methodology to Study Games	25
3.3.1	Representation of Tic-Tac-Toe	25
3.3.2	Representation of Nim and Subtraction Game	26
3.3.3	Representation of 3×3 Hex	26
3.3.4	Decomposition and Complexity	28
3.4	Analysis	29
3.4.1	Complexity of Tic-Tac-Toe and Hex	29
3.4.2	Complexity of Subtraction Game and Nim	31
3.5	Conclusion and Insights	32
4	Complexity of Mixed-Strategy Best-Play Automata	35
4.1	Extensive Form Games, Motivations	36
4.1.1	Simultaneous Move Games	36
4.1.2	Imperfect Information Games	40
4.2	Mixed Strategy Best Play Machine Construction	43
4.2.1	Markov Decision Process Representation	43
4.2.2	Nondeterministic Finite Automaton Representation and Powerset Construction	45

4.2.3 Automata with Context-Aware Inputs	53
4.3 Conclusion and Insights	56
References	58
APPENDICES	61
A Complexity Lower Bound Algorithm Implementation in GAP	62
A.1 Check Essential Dependency	62
A.2 Lower Bound Algorithm for Krohn-Rhodes Complexity	64
B Counterexample Code for Chain of Levels Algorithm	68

List of Figures

2.1	Example skeleton of some arbitrary semigroup. The boxes are equivalence classes; the rectangular nodes are chosen representative sets of a class. Shaded equivalence classes contain nontrivial permutation groups. The labels of the arrows denote sequences of generators that take one subset set to another. The levels are indicated on the left side. [4, p. 13]	8
2.2	(a) The DFA of Becks transformation semigroup. (b) The skeleton of the holonomy decomposition of Becks semigroup.	12
2.3	An idealized skeleton structure of the holonomy decomposition for a hypothetical semigroup S_h . The existence of S_h will prove that the chain of levels algorithm does not correctly determine the complexity lower bound.	13
2.4	(a) The DFA of Becks++ transformation semigroup. (b) The skeleton of the holonomy decomposition of Becks++ semigroup.	14
3.1	Tic-Tac-Toe Board Representation	25
3.2	Sample Best-Play Machine of Tic-Tac-Toe	27
3.3	Pair plots of complexity lower bound, upper bound, and number of states for a sample of $N = 500$ best-play automata for (a) Tic-Tac-Toe and (b) 3×3 Hex.	29
3.4	Holonomy decompositions of (a) a Tic-Tac-Toe best-play machine with complexity two and (b) a Hex best-play machine with complexity 2.	30
3.5	Lower and Upper Bounds on Complexity of Subtraction Game vs Game Size	31
3.6	Best-play automata for selected Subtraction Games ($N = 8, 12, 16, 20, 24$). Nodes show the number of chips (in opponent view); transitions are labelled by opponent moves and lead to opponent view after the best-play automaton's response.	32

3.7	(a) The unique best-play automaton for the Nim Game $1 : 2 : 3$. (b) Skeleton for the unique best-play automaton for Nim Game $1 : 2 : 3$, having two levels with nontrivial groups and an essential dependency between them showing complexity 2.	33
4.1	Game tree for a two-turn Matching Pennies, an extensive-form game. The dashed lines indicate that Player 1 and Player 2 make moves simultaneously. The grey payoffs in brackets are the expected payoffs derived from the SPNEs, consistent with the payoffs from a single-turn Matching Pennies game.	38
4.2	Game tree for a single-turn Prisoner's Dilemma, but with the unknown leniency of the prosecutor. The tree is produced after the Harsanyi transformation, where the unknown information is treated as the first player, Nature. To reiterate some important points: the tree depicts a single-turn simultaneous move game with 3 players; Player Nature has moves Harsh and Lenient; Player 1 and Player 2 have moves Cooperate (C) and Defect (D); the subgame Γ_A encapsulates Table 4.4, and subgame Γ_B encapsulates Table 4.3.	42
4.3	Partial game tree of Γ_{ex} . The opponent plays first. The (partial) game lasts up to 3 turns. The subgames that do not aid in constructing the best-play machine are collapsed and shown as solid triangles. The SPNEs and best mixed strategies have been found for my turns, so the payoffs are not shown at endgames.	44
4.4	The rectangular boxes are the states. ε indicates the initial state. The grey circles indicate actions. The transition probabilities are indicated on the arrows. The circled-times symbols are the endgame states.	46
4.5	The nondeterministic finite automaton of Γ_{ex} . ε is the starting state. A multi-state transition occurs on state $A \cdot Y$, where action A transitions to state ε and state $A \cdot Y \cdot A \cdot X$	47
4.6	Applying the powerset construction algorithm on the NDFA of Figure 4.5 results in this DFA.	48
4.7	Skeleton of the DFA from Figure 4.6 after powerset construction.	49
4.8	Two examples of how self-loop coupled with advancement causes a "fuse."	51
4.9	Two examples of another scenario that causes a "fuse."	52

4.10	ACAI DFA of Γ_{ex} . Some elements of the action space $\{A, B\} \times \{x, y, z\}$ are not shown in the transitions because they lead to the garbage state $\emptyset$	54
4.11	Skeleton of the ACAI DFA from Figure 4.10.	55

List of Tables

2.1	Comparison of Lower Bound Algorithms	21
4.1	Single-turn Rock Paper Scissors payoff table.	36
4.2	Single-turn Matching Pennies payoff table.	37
4.3	Single-turn Prisoner's Dilemma payoff table.	39
4.4	Single-turn Prisoner's Dilemma (harsh variant) payoff table.	41

Chapter 1

Introduction

1.1 Motivation

The main goal of this thesis is to develop new understandings of games by analyzing their algebraic structures and complexity. To achieve this, we develop various game automata and apply modern algebraic theories to study them. There is extensive literature on both game theory and algebra. However, a gap exists between the two fields, as limited studies exist on the analysis of games' algebraic structure. Our study aims to bridge this gap and ultimately provide deeper insights into the complexity of games.

1.2 Thesis Organization

The chapters of the thesis are organized as follows.

Chapter 1 introduces the motivation and structure of the thesis, and presents the essential terminologies and concepts related to semigroups and introduces the Krohn-Rhodes theory.

Chapter 2 proposes a new algorithm that finds the complexity lower bound of transformation semigroups. This is crucial as a significant portion of the thesis is the complexity analysis of gameplay automata.

Chapter 3 furthers Rhodes' existing work on the algebraic structure of games and studies two-player complete-information sequential games.

Chapter 4 seeks to expand the algebraic analysis to extensive-form games, where employing mixed-strategy gameplay is essential for achieving optimality. We develop representations that enable complexity analysis of nondeterministic automata.

1.3 Essential Terminologies from Semigroup Theory

1.3.1 Automata and Transformation Semigroups

We start with the key terminologies and concepts from the semigroup theory.

A **group** is a set G with a binary operation $\circ$ that satisfies:

- *Closure*: $(\forall a, b \in G) a \circ b \in G$.
- *Associativity*: $(\forall a, b, c \in G) a \circ (b \circ c) = (a \circ b) \circ c$.
- Existence of a unique *identity element*: $\exists e \in G, s.t. (\forall g \in G) g \circ e = e \circ g = g$.
- Existence of *inverse element*: $(\forall a \in G) \exists b \in G, s.t. a \circ b = e = b \circ a$, where e is the identity element.

A **semigroup** S is a set S with a binary operator $\circ$ that satisfies:

- *Closure*: $(\forall a, b \in S) a \circ b \in S$.
- *Associativity*: $(\forall a, b, c \in S) a \circ (b \circ c) = (a \circ b) \circ c$.

A group is also a semigroup. A semigroup S may contain several idempotents, where an *idempotent* of S is an element $e \in S$ such that $e^2 = e$. On the other hand, a group has exactly one idempotent, namely the identity.

A **deterministic finite automaton** (DFA) $\mathcal{A}$ is a formal model of computation defined by a tuple (Q, Σ, δ) , where Q is a finite *set of states*, Σ is a finite *input alphabet set*, and $\delta : Q \times \Sigma \rightarrow Q$ is the *state transition function*. Typically, a DFA also contains a *start state* $q_0 \in Q$ and a *set of accept states* $F \subseteq Q$, but they are not used in our algebraic analysis and therefore not defined in our context.

Each alphabet element $a \in \Sigma$ defines a transformation $a : Q \rightarrow Q$ by mapping each $q \in Q$ to $\delta(q, a)$. The set of finite words over Σ is $\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \dots = \bigcup_{k \geq 1} \Sigma^k$ and $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$ where ε is the empty string. Observe that while Σ^+ and Σ^* are infinite sets, they each contain only a finite set of equivalent transformations; Σ^+ and Σ^* are semigroups under the concatenation operator. In particular, the mapping of a string of alphabet elements to its equivalent transformation is a *homomorphism*. We refer to this set of equivalent transformations as the **characteristic semigroup** of $\mathcal{A}$, denoted $\mathcal{S}(\mathcal{A})$. The pair $(Q, \mathcal{S}(\mathcal{A}))$ is the **transformation semigroup** of the automaton $\mathcal{A}$. Oftentimes, we work with transformation semigroups $(X(S), S)$ (or sometimes for short (X, S)) where $S \subseteq \mathcal{S}(\mathcal{A})$ and $X(S) \subseteq Q$ is a subset of Q containing states of interest, and satisfies the closure property under transformations in S [25, p. 35] [18, p. 3].

A transformation semigroup $(X(G), G)$ is a **permutation group** when G is a group and $X(G) \subseteq Q$ is the set of states that G acts on and the unique idempotent e of G acts as the identity on $X = X(G)$. Because G is a group, every $g \in G$ has an inverse, and so it follows that g always permutes X , i.e., $g : X \rightarrow X$ is a bijective mapping. $(X(G), G)$ is a **permutator group** when G is a maximal subgroup of $\mathcal{S}(\mathcal{A})$ and $X(G)$ is the image of its unique idempotent.

An **aperiodic transformation semigroup** is one that does not contain any nontrivial subgroups. That is, (X, S) is aperiodic if there does not exist $X' \subseteq X, S' \subseteq S$ (where the transformation S' is reduced to act on X') such that (X', S') is a permutator group [26, Sec 12.1-19].

In summary, a DFA yields a transformation semigroup. It contains smaller transformation semigroups and possibly permutator groups. We aim to study the algebraic structure of DFA by decomposing and organizing these transformation semigroups hierarchically.

1.3.2 Krohn-Rhodes Theory

We now introduce the concept of Krohn-Rhodes Theory (See [23] or [2] for a detailed introduction). While we will not delve deeply into its intricacies, it is essential to understand that this theory allows for decomposing finite semigroups and automata. Such decomposition is achieved using elementary components such as finite aperiodic semigroups and finite simple groups (such as the modulo two counter C_2 , the alternating group A_5). These components are combined in a mathematical operation called **wreath products**, creating a cascade that emulates the original system.

We use notation $(X_2, S_2) \wr (X_1, S_1)$ to denote the wreath product (a.k.a. the **cascade combination**) of the transformation semigroup (X_1, S_1) by the transformation semigroup (X_2, S_2) . The wreath product is associative, and by definition [23, pp. 12-13],

$$(X_2, S_2) \wr (X_1, S_1) := (X_2 \times X_1, S)$$

$$\begin{aligned} S := \{ & \Pi : X_2 \times X_1 \rightarrow X_2 \times X_1 : \\ & \exists s_1 \in S_1 \text{ and } \beta_2 : X_1 \rightarrow S_2, \\ & \text{s.t. } (x_2, x_1) \cdot \Pi = (x_2 \cdot \beta_2(x_1), x_1 \cdot s_1) \} \end{aligned}$$

The **Krohn-Rhodes Theorem** [23, p. viii] [9, pp. 141-142] states that any finite transformation semigroup can be represented as a homomorphic image (a structure-preserving map between two transformation semigroups that maintains the operation) of a subsemigroup of the wreath product of a finite sequence of semigroups that are either,

- Finite permutation groups, or
- Aperiodic transformation semigroups.

The **complexity** $\theta(X, S)$ of the transformation semigroup (X, S) is the minimal number of permutation groups in such wreath product decompositions [23, p. viii]. Whether there is an algorithm to decide Krohn-Rhodes complexity is a difficult problem that has been open since the 1960s¹.

Lastly, SgpDec [27] is a package written in GAP language [8] that performs holonomy decomposition and uses a directed graph called **skeleton** to represent cascades of transformation semigroups on subsets.

A significant part of this thesis is the study of complexity across various game automata, and SgpDec proves invaluable in this regard. SgpDec enables the computation of holonomy decompositions, which is a method of performing the Krohn-Rhodes decomposition [3, p. 107]. Indeed, the holonomy decomposition of a transformation semigroup immediately gives its complexity the upper bound, which is the number of levels (the hierarchical component of the skeleton that contains many subsets) that contain nontrivial permutation groups. In the following chapters, we will use SgpDec extensively and discuss the specific components of the skeleton in more detail.

¹Recent preprints by Rhodes and collaborators assert its decidability [14, 13]

Chapter 2

Algorithms to Determine a Lower Bound on Krohn-Rhodes Complexity

A crucial component in studying algebraic structures is the ability to determine their complexity. Ultimately, finding the complexity of semigroups is a highly nontrivial task. The holonomy decomposition implemented in the SgpDec package provided the upper bound of the complexity. Based on existing literature, this chapter aims to develop an algorithm for computing the maximum chain of essential dependencies for semigroups and determining their lower bound.

We will start by reviewing the literature on essential dependencies and progressively establish a correct algorithm to identify the maximum chain of essential dependencies in any semigroup. As the length of the maximum chain of essential dependencies defines the semigroup's complexity lower bound, the algorithm will enable us to analyze a variety of algebraic structures in future chapters. Additionally, this chapter sets the key notations and definitions for the rest of the thesis.

2.1 Review of Essential Dependencies

Before delving into the theorems on the essential dependencies, it is crucial to understand the stabilizer and permutator set of the semigroup and the concept of $\mathcal{R}$ -equivalence.

Given a finite automaton $\mathcal{A} := (Q, \Sigma, \delta)$ (whose semigroup is S), where Q is the set of states of $\mathcal{A}$, Σ is the input alphabet and δ is the transition function. We define the sets of stabilizer strings and permutator strings for a subset $X \subseteq Q$ as follows [18, p. 8]:

$$\begin{aligned}(X)\text{stab}_{str} &:= \{w \in \Sigma^* | X \cdot w \subseteq X\} \\ (X)\text{per}_{str} &:= \{w \in \Sigma^* | X \cdot w = X\}\end{aligned}$$

The **stabilizer** $(X)\text{stab}$ and **permutator group** $(X)\text{per}$ are the sets of equivalence classes of words (i.e., giving the same transformations) in $(X)\text{stab}_{str}$ and $(X)\text{per}_{str}$, respectively.

Next, the **reflexive transitive relation** [18, p. 9] for $X_1 \subseteq Q$ and $X_2 \subseteq Q$, denoted $X_1 \geq_{\mathcal{R}} X_2$, holds if there exists $t \in \Sigma^*$ such that $X_1 \cdot t = X_2$. We also define **$\mathcal{R}$ -equivalence**, denoted $X_1 \mathcal{R} X_2$, which holds if and only if $X_1 \geq_{\mathcal{R}} X_2$ and $X_2 \geq_{\mathcal{R}} X_1$.

An important property of $\mathcal{R}$ -equivalence is its implication of permutation group isomorphism (a bijective map between two permutation groups that preserves group operations), as stated in Theorem 1.

Theorem 1. (Thm. 2.2 [18, p. 9]) *Let $\mathcal{A} := (Q, \Sigma, \delta)$ be a finite automaton, and $X_1, X_2 \subseteq Q$. Then $X_1 \mathcal{R} X_2$ implies isomorphism of permutation groups $(X_1, (X_1)\text{per}) \cong (X_2, (X_2)\text{per})$.*

Given subgroups G_k, G_{k+1} of semigroup S with identities $e_k^2 = e_k$ and $e_{k+1}^2 = e_{k+1}$. G_k, G_{k+1} act on their respective subsets of states X_k, X_{k+1} , with $e_k > e_{k+1}$, and with the condition that $X_k \supsetneq X_{k+1} \neq \emptyset$. (We use the notation $e_1 > e_2$ if and only if $e_1 \neq e_2 \wedge e_1 e_2 = e_2 e_1 = e_2$; this $>$ is a transitive relation on the idempotents of S .) We also use the following sets for the definition of essential dependency [18, p. 12] [23, p. 49].

$$\begin{aligned}\mathcal{X}_{k+1} &= \{X_{k+1} \cdot g_k : g_k \in G_k\} \\ \mathcal{I}_{k+1} &= \{f^2 = f \in S : X_k \cdot f \in \mathcal{X}_{k+1}\}\end{aligned}$$

Let $\langle \mathcal{I}_{k+1} \rangle$ be the subsemigroup of S generated by $\mathcal{I}_{k+1}$. Then we say $X_k \supsetneq X_{k+1}$ is an **essential dependency** if and only if

$$\langle \mathcal{I}_{k+1} \rangle \cap (X_{k+1})\text{per} \supsetneq G_{k+1} \neq \{e_{k+1}\} \quad (2.1)$$

Lastly, we arrive at the **complexity lower bound** theorem.

Theorem 2. (Thm. 2.9 [18, p. 12], [23, pp. 49-50]) *Let k be the length of the longest chain of essential dependencies of semigroup S , with subsets $X_1 \supsetneq X_2 \supsetneq \dots \supsetneq X_k \neq \emptyset$ and their identities $e_1^2 = e_1 > e_2^2 = e_2 > \dots > e_k^2 = e_k$. Note that subset $X_i := X(G_i)$ for some*

subgroup G_i of S , and the identity of G_i is $e_i^2 = e_i$. Then, the number of nontrivial groups that must occur at different levels of any Krohn-Rhodes decomposition for (X, S) , i.e. the Krohn-Rhodes complexity of (X, S) is at least k .

This concludes the review of the necessary concepts from the existing literature. The following section provides an original, practical implementation of the essential dependency check.

2.2 Implementing Essential Dependency Check

We are not aware of any existing implementation of the essential dependency check. To tackle this task, we can utilize the SgpDec package, which implements the holonomy decomposition algorithm that finds a “skeleton” of the input transformation semigroup. A **skeleton tree** is a data structure that encapsulates the decomposition results using the following components [4, pp. 6-8].

- **Subsets** and their permutator groups: the subset of the states of the semigroup, as well as the group that acts on this subset. A subset is a singleton or an image of some transformation in the input semigroup S . These are the smallest units in the skeleton.
- **Equivalence classes**: the sets of subsets that are $\mathcal{R}$ -equivalent and have isomorphic permutator groups. An equivalence class contains sets of states.
- **Hierarchical levels**: the coordinate level that results from the holonomy decomposition. Level 1 is always the full state set. A level can contain various equivalence classes. Note that we may use the terms “level” and “depth” interchangeably.

An example skeleton is shown in Figure 2.1.

Suppose we have already performed the holonomy decomposition on the semigroup S . Based on the relation in (2.1), we can use the following methodology to check for essential dependencies between subsets from depth d_1 and subsets from depth d_2 .

1. Iterate through each subset X_1 at level d_1 and each subset X_2 at level d_2 .
2. If $X_1 \supsetneq X_2$, we find the permutation group (G_1, e_1) , and compute $\mathcal{X}_2$ and $\mathcal{I}_2$.

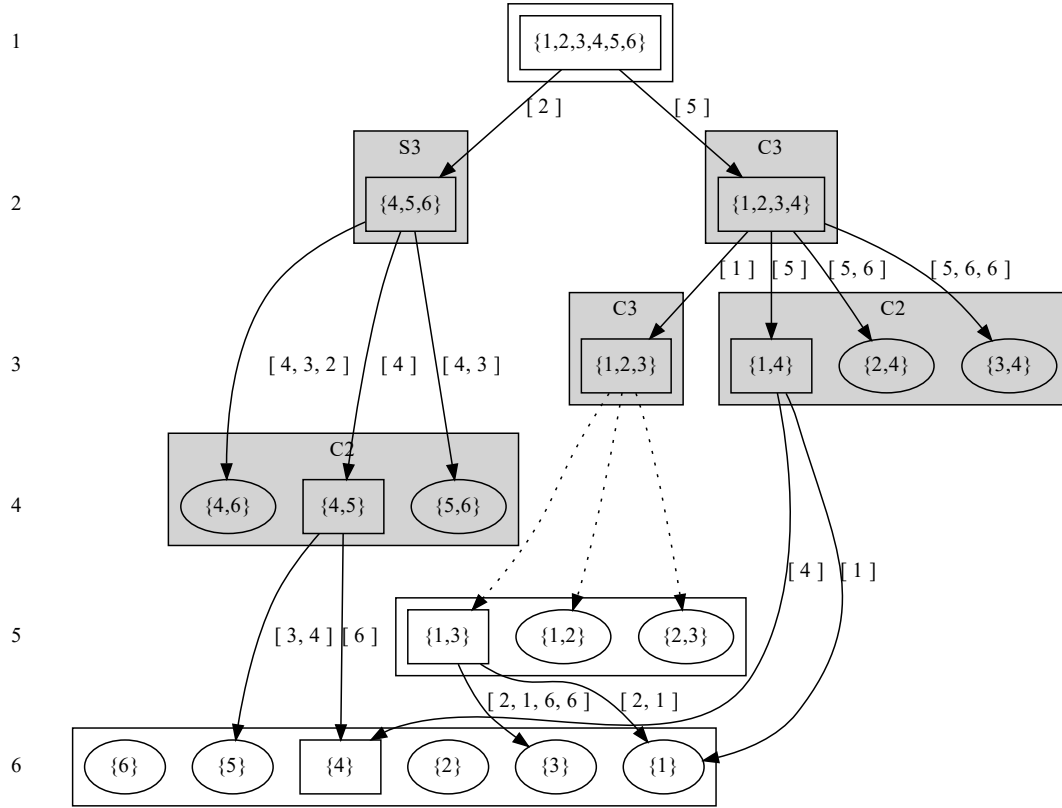


Figure 2.1: Example skeleton of some arbitrary semigroup. The boxes are equivalence classes; the rectangular nodes are chosen representative sets of a class. Shaded equivalence classes contain nontrivial permutation groups. The labels of the arrows denote sequences of generators that take one subset set to another. The levels are indicated on the left side. [4, p. 13]

3. If X_2 is in the image set of $\langle \mathcal{I}_2 \rangle$, and there is a nontrivial group acting on X_2 contained in $\langle \mathcal{I}_2 \rangle$, then we conclude that there is an essential dependency between X_1 and X_2 .
4. If, after an exhaustive search through all subsets of the two depths, we did not establish any essential dependency between any pairs of subsets X_1, X_2 , then we can conclude that there is no essential dependency between the two depths.

The essential dependency check implementation proposed in this section has now been added to the SgpDec package¹. The implementation takes two subsets from different levels as input (rather than using the levels as input) and returns any group found in step number 3. This implementation lays the foundation for lower-bound algorithms that aim to find the maximum chain of essential dependencies.

The GAP code has also been included in Appendix A.1 for reference.

2.3 Naive Chain of Levels Algorithm

Theorem 2 stated that a lower bound of the Krohn-Rhodes complexity is the maximum length of a chain of essential dependencies.

The initial variant of the lower bound algorithm attempts to find a sequence depths $d_1, d_2, \dots, d_k$ with $1 \leq d_1 < d_2 < \dots < d_k \leq n$ (n is the total number of levels) in the holonomy decomposition such that there is an essential dependency between d_1 and d_2 , d_2 and d_3 , and so on. In other words, the goal of the naive algorithm is to find a chain of levels such that each pair of adjacent levels in the chain has an essential dependency.

The general steps of the algorithm are as follows.

1. Discard levels that only contain trivial groups since only levels with nontrivial groups can form essential dependencies. Let the remaining levels be $d_1, d_2, \dots, d_k$ with $1 \leq d_1 < d_2 < \dots < d_k$.
2. If only zero or one level with a nontrivial group is left, that level (or the absence of any such level) would be the longest chain of levels, and the algorithm terminates.

¹The implementation file is found in this GitHub commit snapshot: <https://github.com/gap-packages/sgpdec/blob/e3669cf5c33dd3abcb39e8042542f603f84e79c/lib/lowerbound.gi>

3. We use a top-down iterative approach and start “chaining” from the top-most level (i.e., the lowest hierarchical level at the top of the skeleton tree). A bottom-up approach could also be correct, provided we completely reverse the direction of the two loops described here. Let $d_{current} := d_1$.
 - Check for essential dependencies between $d_{current}$ and every level above $d_{current}$ in a loop.
 - After iterating through all levels above the current (i.e., all levels $d_i < d_{current}$), choose the level d_i with the longest upstream chain of length l , such that there exists an essential dependency for some set on d_i and some set on $d_{current}$. Memorize tuple values (parent node, chain length) $:= (d_i, l + 1)$ for key $d_{current}$, so we may reconstruct the longest chain of depths at the end.
 - If the current level does not form an essential dependency with any levels above the current, then the longest upstream chain it achieves has length 1.
 - Let $d_{next} > d_{current}$ be the next level immediately below $d_{current}$ with a nontrivial group. Update $d_{current}$ to d_{next} .
 - Repeat the above steps until $d_{current} > d_k$.
4. Loop through the recorded upstream length for every level. Choose the level d_c that has the longest length.
5. Reconstruct the chain of depths, starting from d_c , by tracing the linked list of memorized depths.

The algorithm employs dynamic programming to track each level’s upstream chains. It achieves an ideal polynomial time complexity before accounting for the complexity of all the semigroup-related operations. To simplify our analysis, suppose that the essential dependency check between two sets X_1 and X_2 has a time complexity of $O(f(S))$, where $f(S)$ encapsulates the time complexity of the underlying algebraic operations involved. If there are M levels with nontrivial groups, each with at most N subsets, the time complexity of this algorithm is

$$O\left(\frac{1}{2}(M-1)MN^2 \cdot f(S)\right) = O(M^2N^2 \cdot f(S))$$

This algorithm has made significant progress towards finding the complexity lower bound and is efficient. However, it does not find the maximum chain of essential dependency directly. As a result, Assertion 1 must hold true for the algorithm to determine the complexity lower bound in general.

Assertion/Conjecture 1. *Suppose we have levels $d_1, d_2, \dots, d_k$ with $d_1 < d_2 < \dots < d_k$ for the holonomy decomposition of semigroup S . If there exists an essential dependency between a set in level d_i and a set in level d_{i+1} for all $i \in \{1, 2, \dots, k-1\}$, then there exists a chain of essential dependency of length k that spans $d_1, d_2, \dots, d_k$. For example, if some subset X_1 at level d_1 has an essential dependency on X_2 at level d_2 , and a different subset X'_2 at level d_2 has an essential dependency on X_3 at level d_3 . Then, the algorithm assumes that there exists subsets X''_1, X''_2, X''_3 at level d_1, d_2, d_3 respectively, such that $X''_1 \supsetneq X''_2 \supsetneq X''_3$ forms a chain of essential dependency.*

Proving this assertion directly is extremely challenging, and disproving it requires counterexamples. A counterexample would need to be a semigroup S with at least three holonomy decomposition levels, d_1, d_2, d_3 with $d_1 < d_2 < d_3$, each containing nontrivial groups; some subset X_1 at level d_1 has an essential dependency on X_2 at level d_2 , and a different subset X'_2 at level d_2 has an essential on X_3 at level d_3 ; however, there does not exist a chain of essential dependencies X''_1, X''_2, X''_3 at level d_1, d_2, d_3 , respectively, such that $X''_1 \supsetneq X''_2 \supsetneq X''_3$ forms a chain of essential dependency.

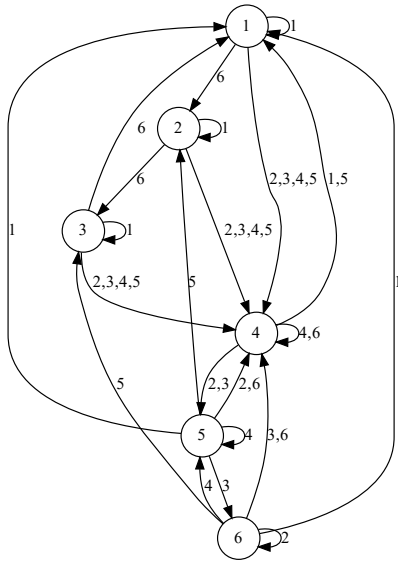
Finding a semigroup that meets specific properties is notoriously tricky, especially given our grotesque requirements. In general, there are two approaches we could employ.

1. We can use brute force to systematically construct a multitude of semigroups, proceeding in either a combinatorial or randomized sequence.
2. We can leverage existing semigroups with well-known properties and introduce deliberate and incremental changes in the hope of ultimately uncovering the desired semigroup.

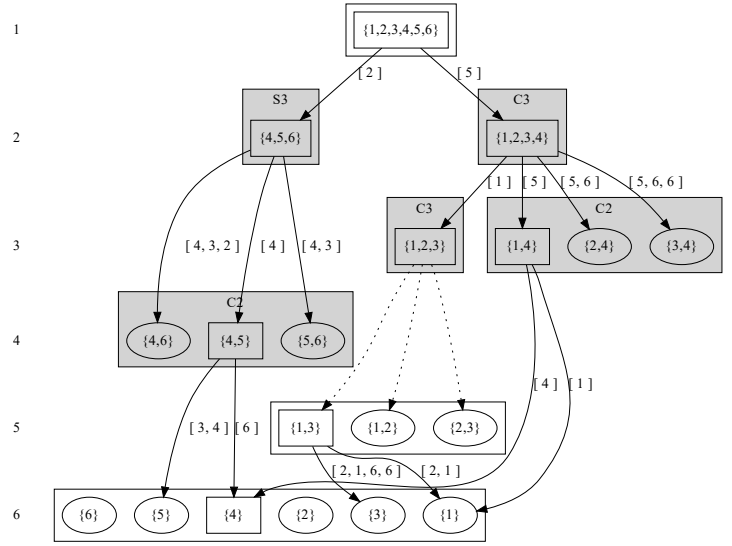
The initial round of testing via approach number 1 did not yield any counterexamples. Even a few states can produce billions of semigroups, with the vast majority having zero complexity. Additionally, the high complexity of the chaining algorithm and counterexample verification process significantly constrains the number of semigroups we can feasibly investigate.

Intuitively, approach number 2 is more effective than approach number 1 because it ensures that we only investigate semigroups that are analogous to the existing semigroups with nonzero complexity. However, it too relies heavily on luck.

Fortunately, we stumbled upon Becks transformation semigroup [5, p. 44] shown in Figure 2.2, which appears to be a promising candidate. In particular, referring to Figure 2.2(b), we have 3 levels with nontrivial groups. Additionally, we have two separate



(a) Becks Transformation Semigroup



(b) Skeleton of Becks Semigroup

Figure 2.2: (a) The DFA of Becks transformation semigroup. (b) The skeleton of the holonomy decomposition of Becks semigroup.

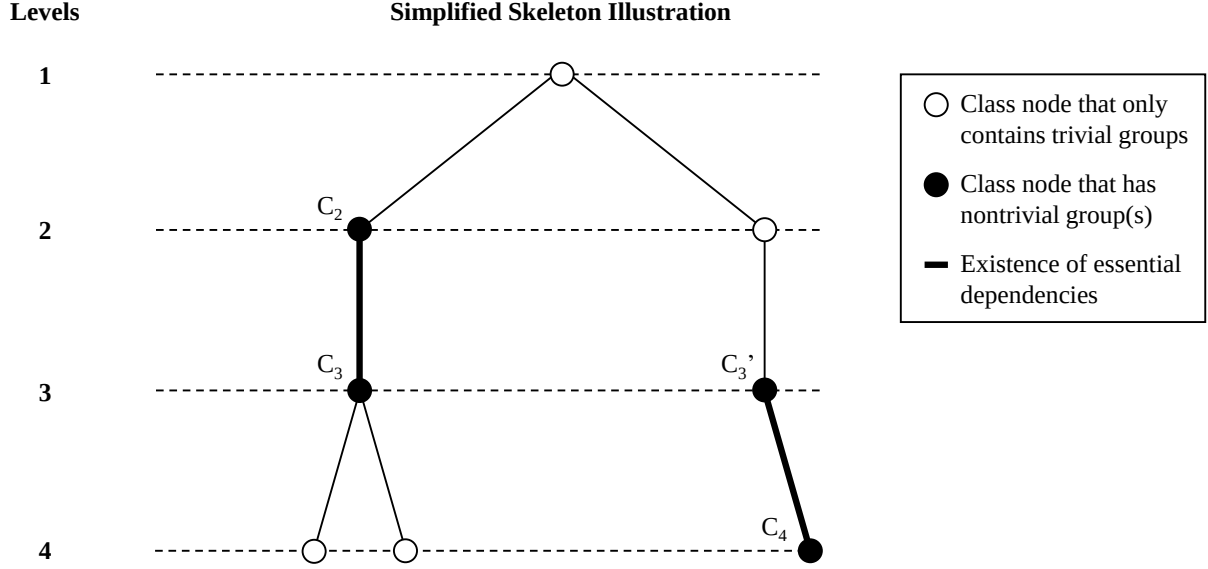
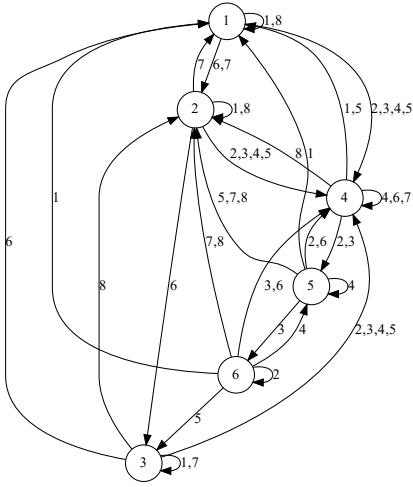


Figure 2.3: An idealized skeleton structure of the holonomy decomposition for a hypothetical semigroup S_h . The existence of S_h will prove that the chain of levels algorithm does not correctly determine the complexity lower bound.

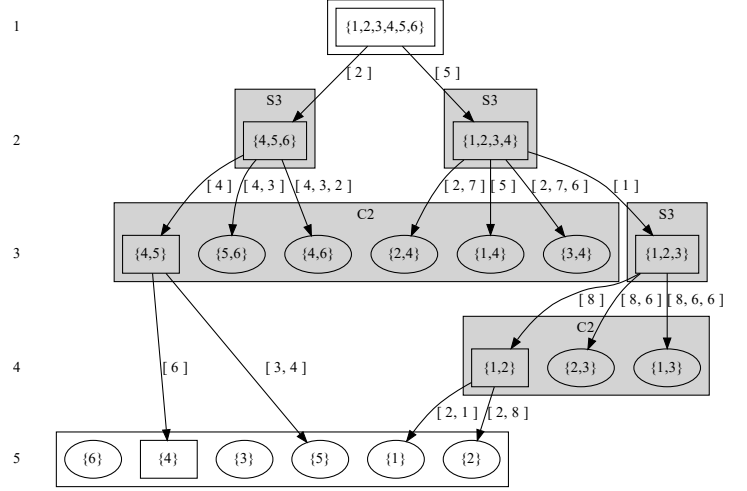
branches in the skeleton; one branch goes from a class at level 2 to a class at level 4, and another goes from a class at level 2 to a class at level 3.

If we could adjust the semigroup, e.g., add another transformation, we could potentially arrive at a counterexample. The goal of the adjustment is to retain the two-branch structure of the skeleton; on the one branch, we only have essential dependencies between sets at level 2 and sets at level 3; on the other branch, we only have essential dependencies between a set at level 3 and a set at level 4.

An idealized illustration of the holonomy decomposition for a hypothetical counterexample semigroup S_h is shown in Figure 2.3. The left branch of this skeleton sees an essential dependency between class C_2 at level 2 and class C_3 at level 3. The right branch of this skeleton sees an essential dependency between class C_3' at level 3 and class C_4 at level 4. However, none of the sets in C_3' are subsets of some set in C_2 , and none of the sets in C_4 are subsets of some set in C_3 . As such, we cannot establish a chain of essential dependency that traverses levels 2, 3, and 4. Therefore, even though the chain of levels algorithm iden-



(a) Becks++ Transformation Semigroup



(b) Skeleton of Becks++ Semigroup

Figure 2.4: (a) The DFA of Becks++ transformation semigroup. (b) The skeleton of the holonomy decomposition of Becks++ semigroup.

tifies a chain of levels 2, 3, and 4, S_h only has a maximum chain of essential dependency of length 2. All that remains is proving that such a semigroup could exist.

Luckily, we discover that by adding 2 more transformations to the Becks semigroup, we arrive at a semigroup shown in Figure 2.4(a) where the input labelled 7 and 8 are the new transformations. The new transformations have several effects; most notably, it makes the $\{\{1,3\}, \{1,2\}, \{2,3\}\}$ equivalence class contain a nontrivial C_2 group, as shown in the skeleton. Given the derivative nature of this transformation semigroup, which originates from Becks semigroup, we refer to it as “Becks++.” Figure 2.4(b) shows the skeleton of its holonomy decomposition. Using the essential dependency check algorithm, we identify an essential dependency between subset $\{1,2,3\}$ at level 3 and subset $\{1,2\}$ at level 4 for the right branch. We also identify an essential dependency between subset $\{4,5,6\}$ at level 2 and subset $\{4,5\}$ at level 3 for the left branch. All other classes’ essential dependency checks for subsets are negative. In this case, the naive algorithm identifies a chain of levels 2, 3, and 4. However, the true length of the maximum chain of essential dependency is 2. This is exactly the counterexample we are looking for. (Refer to Appendix B for the GAP code that verifies this counterexample.)

The Becks++ counterexample rejects Assertion 1, necessitating the development of a more sophisticated and accurate complexity lower bound algorithm.

2.4 Chain of Sets Algorithm

To develop an accurate lower bound algorithm, we can work directly with subsets and adhere strictly to the definition. Recall that the skeleton of the holonomy decomposition consists of three essential units: levels, classes, and subsets. A level contains classes, and a class includes subsets. The skeleton graph is a tree-like structure in which arrows indicate how certain generators facilitate the collapse of subsets in one class into subsets in another class. One could exhaustively check all proper inclusion chains of subsets in the skeleton to find a longest chain of essential dependencies and arrive at a provably correct complexity lower bound.

From the previous sections, we know how to check the essential dependency between two sets. It suffices to discover a chain of sets $X_1 \supsetneq X_2 \supsetneq \dots \supsetneq X_k$, at levels $d_1, d_2, \dots, d_k$ respectively, with $d_1 < d_2 < \dots < d_k$.

Naturally, we propose a brute-force approach:

1. Discard levels that only contain trivial groups since only levels with nontrivial groups can form essential dependencies. Let the remaining levels be $d_1, d_2, \dots, d_k$ with $1 \leq d_1 < d_2 < \dots < d_k$.
2. If only zero or one level with a nontrivial group is left, that level (or the absence of any such level) would be the longest chain of levels, and the algorithm terminates.
3. Borrowing from the last section, we use a top-down iterative approach and start “chaining” from some set $X_{current}$ of the top-most level.
 - Check for essential dependencies between $X_{current}$ and every subset in levels above $X_{current}$ ’s level in a loop.
 - After iterating through all sets at levels above the current, choose the set X_{upper} with the longest upstream chain of length l , such that there exists an essential dependency between $X_{current}$ and X_{upper} . Memorize tuple values (parent node, chain length) $:= (X_{upper}, l + 1)$ for the key $X_{current}$ so we may reconstruct the longest chain of sets at the end.

- If the current set does not form an essential dependency with any sets at levels above the current, then the longest upstream chain it achieves has length 1.
 - Update $X_{current}$ to be the next unexplored subset on the same level. If all other subsets are explored, then pick $X_{current}$ to be a set on the next level immediately below the current with a nontrivial group.
 - Repeat the above steps until all subsets with nontrivial groups have been explored.
4. Loop through the recorded upstream chain length for every explored subset. Choose the subset X_c that has the longest length.
 5. Reconstruct the chain of subsets, starting from X_c , by tracing the linked list of memorized depth.

The chain of sets algorithm is essentially a more fine-grained variant of the chain of levels algorithm. It also employs dynamic programming. However, the complexity is expected to be worse. Suppose there are M levels with nontrivial groups, each with at most N subsets, and assume the complexity of the essential dependency check to be $O(f(S))$.

- Each level d has up to N subsets.
- Each subset at level d has to check essential dependencies against up to $(d - 1) \cdot N$ subsets at lower levels.
- Each subset also has to check the memorized chain length values against up to $(d - 1) \cdot N$ subsets.
- d goes from level 2 to level M .

Then, the time complexity of this algorithm is

$$\begin{aligned}
O\left(\sum_{d=2}^M N \cdot (d - 1) \cdot (N \cdot f(S) + N)\right) &= O\left(N^2(f(S) + 1) \sum_{d=2}^M (d - 1)\right) \\
&= O\left(\frac{1}{2}M(M - 1)N^2(f(S) + 1)\right) \\
&= O(M^2N^2 \cdot f(S))
\end{aligned}$$

Therefore, the chain of sets algorithm attains the same complexity bound as the chain of levels algorithm. The main change in complexity stems from the fact we have to track the chains for each subset. An additional challenge, not captured by the time complexity, is the necessity to index every subset for dynamic programming to work efficiently. As the number of states increases, we could require exponentially large space to store all the subsets. Of course, this problem applies to SgpDec and holonomy decomposition, too.

The current implementation has severe limitations.

1. In practical applications, N could get extremely large, depending on the number of subsets per class and the branching factor of the skeleton tree.
2. An upper-level set's descendent sets likely only constitute a small fraction of all the lower-level sets. This means most of the essential dependency checks are likely between some X_{upper} and some X_{lower} , where $X_{upper} \not\supset X_{lower}$. These checks are unnecessary and increase the runtime.

Unfortunately, these limitations affect both the chain of sets algorithm and the chain of levels algorithm.

Optimizations are necessary to apply the chain of sets algorithm to more complex semigroups. Notably, the second problem can be eliminated if, for the set $X_{current}$, we only explore its descendant subsets in the skeleton. The next section proposes an improved chain of sets algorithm that addresses this issue.

2.5 Optimizations

This section will attempt to improve the chain of sets algorithm, with a particular emphasis on leveraging the structural properties of the holonomy decomposition's skeleton tree during the chaining process. Instead of exhaustively checking the essential dependencies of a subset against all higher-level subsets, we strategically compare the essential dependencies of a subset only to the subsets contained within its descendant nodes in the skeleton structure.

Using a skeleton tree to guide the chaining algorithm introduces several challenges. In particular, the following changes are made:

- Rather than looping through levels to check essential dependencies between subsets, we will perform a post-order traversal of the skeleton tree. This method enables us to chain the subsets from the bottom up, ensuring that the subsets in the descendant nodes have fully explored their downstream chains before we examine the downstream chain of the current set.
- Due to the nature of the tree traversal, a recursive implementation is decidedly more advantageous.
- We can no longer skip the levels that don't contain classes with permutator groups. This may be undesirable, as semigroups often have many levels without permutator groups in general.

As an original contribution, we propose the following algorithm that leverages the skeleton tree to efficiently search for the maximum chain of essential dependency:

1. Let $\mathcal{X}$ be the set of all subsets in the skeleton. Define an index function that maps a subset in the skeleton to an integer index $\mathcal{I} : \mathcal{X} \rightarrow \{1, 2, \dots, |\mathcal{X}|\}$. The index function is a bijection, so $\mathcal{I}^{-1}$ maps an index to the subset of interest. Our implementation uses the set's position in the *forward orbit* of the skeleton (i.e., the sequence of subsets generated by repeatedly applying transformations to the initial subset) as the index, which is deterministic at run time.
2. We employ a recursive depth-first search algorithm to traverse each subset of the skeleton tree. Additionally, we incorporate dynamic programming techniques, which prevent redundant recursive expansions on previously explored subsets. The memorization is necessary as the skeleton tree is not technically a tree structure, i.e., different branching could reach the same node. The skeleton tree is a directed graph whose "source" node is the full state set and whose "sink" nodes are the singleton subsets.
3. **Base Case No. 1:** the current subset $X_{current}$ has already been explored, and we return its memorized longest downstream chain.
4. Based on the skeleton tree, obtain the immediate child subsets of $X_{current}$ for further recursion. Discard all singleton subsets because they do not contain any nontrivial groups, and they do not have any child subsets.
5. **Base Case No. 2:** if $X_{current}$ has no child subsets, set its downstream max chain length as 1 if $X_{current}$ contains a nontrivial permutator group, and 0 otherwise.

6. **Recursion:** invoke the depth-first search function on each of $X_{current}$'s child subsets to complete the subtree traversal (i.e., all subsets reachable from $X_{current}$).
7. **Post Order Processing:** construct the maximum downstream chain for $X_{current}$, similar to the algorithms from previous sections.
 - (a) If $X_{current}$ does not contain any nontrivial permutator groups, set the maximum downstream chain length to 0 and return early. Further evaluation of essential dependencies is unnecessary, as we can be certain that $X_{current}$ will not possess essential dependencies with any other subset.
 - (b) Identify all descendent subsets of $X_{current}$, i.e., all subsets reachable from $X_{current}$ to test essential dependency against. Remove all subsets that are singleton sets or do not have nontrivial permutator groups, as they will not form an essential dependency with $X_{current}$.
 - (c) Iterate through all remaining descendent subsets and test $X_{current}$'s essential dependency against each. Choose the descendent set X_{lower} with the longest downstream chain of length l , such that there exists an essential dependency between $X_{current}$ and X_{lower} . Memorize the tuple value (parent node, chain length) $:= (X_{lower}, l + 1)$ for the key $X_{current}$ so we may reconstruct the longest chain of sets at the end.
 - (d) If the current set does not form an essential dependency with lower-level sets, then the longest downstream chain it achieves has length 1.
8. In the main body of the function (outside of the recursive function), invoke the recursion on the root of the skeleton tree, i.e., the whole set.
9. Iterate through the recorded downstream chain length for each explored subset. Choose the subset X_c that has the longest chain length.
10. Reconstruct the chain of subsets, starting from X_c , by tracing the linked list of recorded depth.

The algorithm demonstrates greater sophistication than the brute-force chain of sets approach. Its efficiency stems from a key feature: subset $X_{current}$ only checks essential dependency against its descendent subsets. This selective checking, rather than checking all subsets at levels below the current, should improve the algorithm's time complexity.

Again, suppose that there are M levels with nontrivial groups and M' levels without nontrivial groups; each level has at most N subsets, and assume the complexity of the essential dependency check is $O(f(S))$.

- Suppose the max in-degree for vertices in the skeleton (i.e., subsets) in the skeleton is $\alpha \in \mathbb{N} \setminus \{0\}$, with $\alpha \ll (M + M') \cdot N$.
- Even though we are dealing with a recursion algorithm on a directed graph, each vertex is visited at most α times. Therefore, the time complexity for visiting every vertex is $O(\alpha(M + M') \cdot N)$. Additionally, due to dynamic programming, the essential dependency check between pairs of subsets with nontrivial groups occurs exactly once.
- Each subset has to check up to $\frac{1}{\beta}(d - 1) \cdot N$, where $\frac{1}{\beta}$ is the maximum proportion of descendant subset count to count of all subsets at higher levels, with $\beta \geq 1$. For simplicity, the root of the skeleton (i.e., the full set) is excluded in the calculation of β , even if it contains a nontrivial group.
- Each subset also has to check the memorized chain length values against up to $\frac{1}{\beta}(d - 1) \cdot N$ subsets.
- This happens for all levels with nontrivial groups, i.e., d goes from level 2 to level M .

The time complexity of this algorithm is

$$\begin{aligned}
& O \left(\alpha(M + M') \cdot N + \sum_{d=2}^M N \cdot \frac{1}{\beta}(d - 1) \cdot (N \cdot f(S) + N) \right) \\
& = O \left(\alpha(M + M') \cdot N + \frac{1}{\beta} \sum_{d=2}^M N \cdot (d - 1) \cdot (N \cdot f(S) + N) \right) \\
& = O \left(\alpha(M + M') \cdot N + \frac{1}{\beta} M^2 N^2 \cdot f(S) \right)
\end{aligned}$$

It is crucial to acknowledge that this complexity analysis is highly simplified. On a high level, when $\beta \gg 1$, the recursion will run much faster than the brute-force chain of sets algorithm. On the other hand, when $\beta = 1$, the algorithm's complexity is

$$O(\alpha(M + M') \cdot N + M^2 N^2 \cdot f(S))$$

which appears worse than the complexity of the brute-force chain of sets algorithm, which is $O(M^2 N^2 \cdot f(S))$. This is true. In the case of a very dense graph, e.g., where for each $d < M + M'$, and for each subset X_d at level d , X_d connects to every subset X_{d+1} at

level $d + 1$, the recursion performs with the same complexity as the brute-force chain of sets algorithm. In particular, the recursion will perform worse when $M' \gg M$ since the recursion explores sets at levels that don't contain nontrivial groups. Fortunately, semigroups with a maximally dense skeleton tree are exceedingly rare, as we have yet to encounter one in our study. Considering we are dealing with smaller semigroups with relatively few states and transformations, even if $M' \gg M$, it will not make a noticeable impact in the runtime.

Overall, we consider the tree-search approach the most optimized chain of sets algorithm by far. The algorithm discussed in this section is implemented, proposed to SgpDec, and released on Version 1.0.0 [27].

The implementation of the improved chain of sets algorithm is also included in Appendix A.2.

2.6 Conclusion and Insights

This chapter examines the background of essential dependencies and the complexity lower bound of semigroups. By identifying the maximum chain of essential dependencies, the complexity lower bound is the length of the chain.

We proposed three algorithms to identify the complexity lower bound of a semigroup. The characteristics of these algorithms are summarized in Table 2.1.

Algorithm	Time Complexity	Correct
Chain of Levels	$O(M^2 N^2 \cdot f(S))$	No
Chain of Sets	$O(M^2 N^2 \cdot f(S))$	Yes
Improved Chain of Sets	$O\left(\alpha(M + M') \cdot N + \frac{1}{\beta} M^2 N^2 \cdot f(S)\right)$	Yes

Table 2.1: Comparison of Lower Bound Algorithms

Despite our efforts of optimization. The chaining algorithm's time complexity still significantly limits its general applicability. For the scope of the thesis, we will only study smaller semigroups with relatively few states and transitions.

For future work, one can explore the possibility of a chain of classes algorithm that could potentially further reduce the time complexity. This will likely involve the following.

- Prove a chain of classes implies a chain of essential dependencies of the same length.

- Develop an index function that maps a class to an index.
- Rather than checking essential dependencies of all sets between two classes, utilize a fixed class representative.

Developing the chain of classes algorithm presents considerable challenges. Due to time constraints and its deviation from the scope of this thesis, we will not pursue further optimizations. However, exploring the chain of classes algorithm might be helpful for those needing to analyze a large semigroup with numerous levels in the skeleton.

Chapter 3

Complexity of Simple Gameplay Automata

This chapter is based on work that was presented at the 6th International Conference on Applied Mathematics, Modeling and Computational Science (AMMCS 2023), held in Waterloo, Canada, from August 14–18, 2023¹.

Scholars have delved into game theory and explored game strategies for centuries. However, research on the underlying structures of gameplay remains scarce, particularly on the algebraic structure of gameplay automata. This gap in the literature is somewhat addressed in Rhodes’ game complexity study [23, Part IV], which employs modern algebraic theory. This chapter draws inspiration from Rhodes’ work to explore these concepts further and studies simple two-player, complete-information sequential games.

3.1 Rhodes’ Gameplay Machines

Reusing Rhodes’ definition [23, Defn. 6.106], a **gameplay machine** is a tuple $C = (A, B, Q, \lambda, \delta)$ where A is the set of moves that the opponent can make, B is the set of moves that I, the machine, can make, the state set Q is the set of possible opponent views (i.e., the board positions at the opponent’s turn), and two transition functions

¹The full paper has been accepted for publication in the proceedings and is currently in press [7]. Additionally, a refereed abstract of this work was published in the conference’s Book of Abstracts [6].

- $\lambda : Q \times A \rightarrow Q$ with $(q, a) \mapsto (q \cdot a) \cdot b$ and $b := \delta(q, a)$. Note that q is the opponent's view, and a is the opponent's move. Essentially, λ maps a current opponent view (and move) to a new opponent view after the move is made and I responded.
- $\delta : Q \times A \rightarrow B$ provides my move given my view, i.e. $q \cdot a$, after the opponent's move. In short, δ describes my strategy.

A few additional notes regarding our construction:

1. λ has a special case as Q does not include endgame positions; it loops back to the starting position q_0 and thus resets the game. The game may be over after an opponent move (i.e., $(q \cdot a)$) or after my response (i.e., $(q \cdot a) \cdot b$).
2. Initial state $q_0 \in Q$. This implies that the opponent always goes first regardless of who won the preceding game.
3. Making an illegal move results in a permanent “garbage” state denoted $\emptyset$. That is, for every $q \in Q, a \in A$ such that $q \cdot a$ is illegal, we have $q \cdot a = \emptyset$. Additionally, $(\forall a \in A) \emptyset \cdot a = \emptyset$. While Rhodes discusses several possible approaches to illegal moves, this particular rule is enforced here to eliminate any effect illegal moves may have on the algebraic structure of the game.

A **best-play** machine is a gameplay machine that guarantees the best outcome, i.e., the highest utility for myself. Furthermore, we define **game complexity** $\theta(G)$ as the minimum complexity of any best-play automaton that plays G .

3.2 Rhodes' Conjectures on the Complexity of Games

For our study, we aim to analyze the algebraic structure of several two-player perfect-information games. These games, namely Nim, Subtraction Game, Tic-Tac-Toe, and 3×3 Hex games, have small state space, enabling total minimax tree searches. Firstly, we shall find suitable state-space formulations, considering board symmetry. Secondly, we will construct the best-play automata. We will define optimality (utility function), perform minimax, and sample best strategies. Thirdly, we will utilize the computer algebra GAP package SgpDec [27, 3] to decompose the best-play automata and perform complexity analysis. We hope to obtain some results on the complexity of particular games and verify a few conjectured assertions made by Rhodes [23, Part IV, pp. 247-248 & p. 255], in particular, related to his theorem:

O		X
X	O	
	X	O

0	1	2
3	4	5
6	7	8

Figure 3.1: Tic-Tac-Toe Board Representation

Theorem 3 (Rhodes (Thm. 6.108 [23])). *If the game G has an $n \times m$ board where pieces continue to fill the board, then $\theta(G) \leq nm/2$.*

Assertion/Conjecture 2. *Tic-Tac-Toe has a complexity of 1.*

Assertion/Conjecture 3. *Nim has a complexity of 1.*

Assertion/Conjecture 4. *7×7 Hex has complexity ≤ 24 and is approximately 18.*

The theorem gives complexity upper bound of 4 for Tic-Tac-Toe and 24 for 7×7 Hex, but no proof is provided for further reduction of these bounds.

3.3 Methodology to Study Games

We should develop suitable state-space representations for each game to begin the experiment.

3.3.1 Representation of Tic-Tac-Toe

Consider a machine $C = (A, B, Q, \lambda, \delta)$ that plays Tic-Tac-Toe. Let $A = B = \{0, 1, \dots, 8\}$ where action $x \in \{0, 1, \dots, 8\}$ represents placing a piece at position x . (See Figure 3.1.) Placing a piece in a position that is already occupied is illegal.

Q is the set of reachable board positions where we take symmetry into account. Given any position, rotating and mirroring the board will yield an equivalent position. Therefore,

we pick a fixed *representative* for each set of equivalent positions when constructing the gameplay machine. This will reduce the total number of states by up to a constant factor.

In a sample best-play automaton construction shown in Figure 3.2, each state is a board position in the opponent’s view, and each arrow with a label (0-8) represents a possible move from the opponent. The arrow leads to a new state where our player has already responded optimally to the opponent’s move. An illegal move goes to a garbage state permanently.

3.3.2 Representation of Nim and Subtraction Game

Nim and Subtraction Games are both impartial games. For Nim, we set up the game with several piles of chips; players take turns removing any amount of chips from one of the piles until all of the chips run out. The player who removes the last chip wins. In our model, state Q describes the piles of chips. We use notation

$$x_1 : x_2 : \cdots : x_k$$

with $x_i \in \mathbb{N}$ to represent k piles of chips where pile i has x_i chips. Permuting the piles yields an equivalent state, so we choose sorted piles as the symmetry representative in this case. For example, pile $3 : 1 : 2$ has many equivalent states, and pile $1 : 2 : 3$ is the symmetry representative of all the equivalent states; if a move results in $3 : 1 : 2$, we will automatically replace it with $1 : 2 : 3$ before making the next move. For player action, we write $i \leftarrow x$ to indicate the removal of x chips in pile index i . Removing more chips than are present in a pile is illegal.

Subtraction Game is a slightly lesser-known game. It is set up with a single pile of chips, and two players successively remove 1, 2 or 3 chips until the pile is emptied. The player who removes the last chip wins. In this case, the state is simply an $x \in \mathbb{N}$, i.e., the number of chips in the pile; and each player’s action set is $\{1, 2, 3\}$. Removing more chips than are present in the pile is illegal.

3.3.3 Representation of 3×3 Hex

Hex is a 2-player perfect information game where the *red* and *blue* players place coloured pieces on a hexagonally tessellated Rhombus grid, one at a time. The first player that connects the two sides (e.g., top to bottom or side to side) that they are assigned wins the game. The states are board positions similar to Tic-Tac-Toe. However, the symmetry here

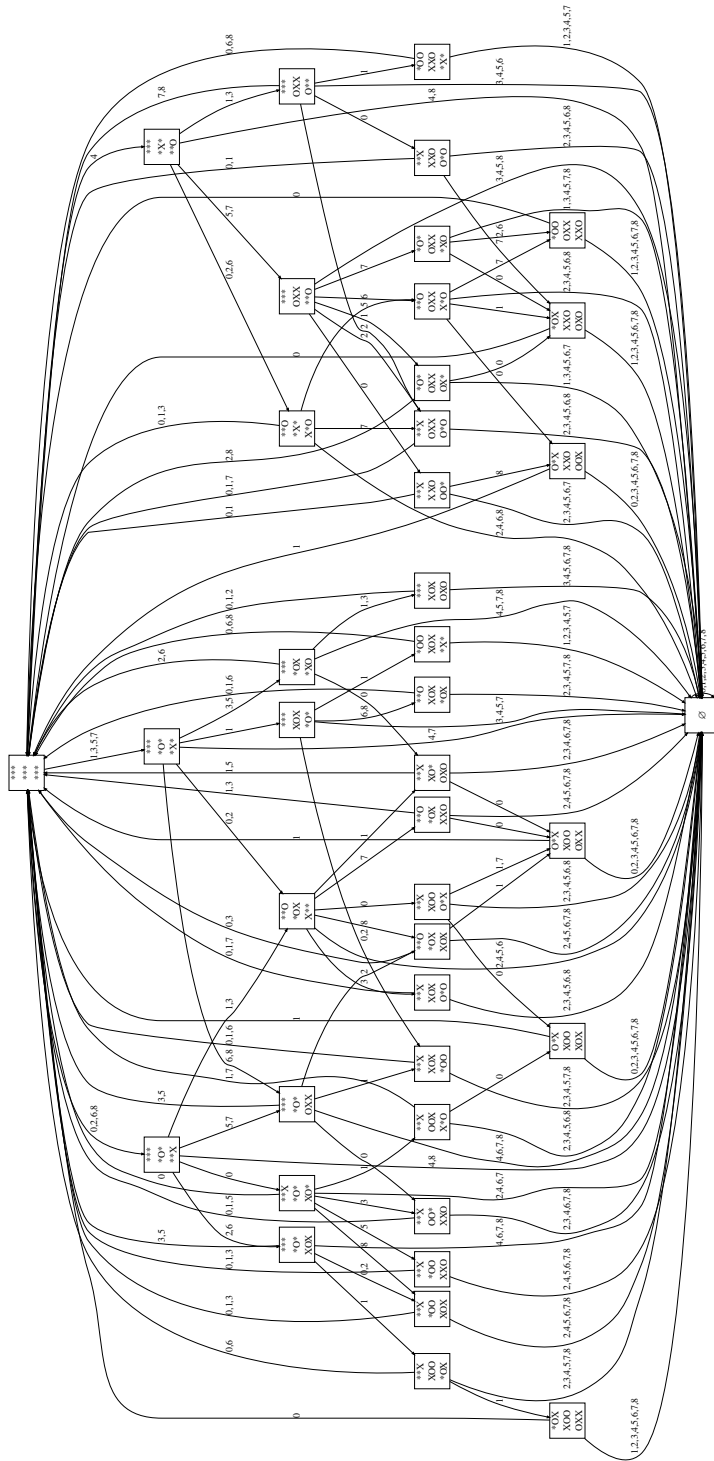


Figure 3.2: Sample Best-Play Machine of Tic-Tac-Toe

is more restrictive, as only rotating 180 degrees yields an equivalent state since the sides matter, and the rhombus shape prohibits 90-degree rotation. We use the same notation system as Tic-Tac-Toe for the actions.

3.3.4 Decomposition and Complexity

We may construct a best-play automaton with appropriate state-space formulation and convert it to a transformation semigroup. The utility function dictates the best-play automata. By only caring about winning (utility of 1), losing (utility of -1), or getting a draw (utility of 0), we can obtain the complete set of best-play automata. However, this leads to an enormous number of automata, even in the most straightforward games like Tic-Tac-Toe, and absurd situations where a player purposefully drags the game on even though they could win in one move. Consequently, it is wise to take the duration of the game into account when calculating utility. For example, when I win, the utility will be $1/(\text{number of turns})$. Similarly, if I am losing a game, I will try to survive for as long as possible, hoping my opponent will make a mistake. In other words, a victory with fewer turns should get a higher utility, and a loss with more turns should get a higher utility, too. As Nim and Subtraction Games are impartial games, we can use the Sprague-Grundy theorem to find the *nimber* of the game [24, pp. 180-181] and therefore determine winning moves directly. We would only need to apply minimax for losing games.

Ideally, we would like to construct all of the best-play automata by picking our optimal strategies one at a time since the complexity of the game is minimal complexity of all of these. However, this is unrealistic due to the sheer number of automata generated by all possible optimal strategies. The process is also highly nontrivial. One needs to recursively perform breadth-first search game tree construction, resulting in trillions of trees for even a simple game like Tic-Tac-Toe. Therefore, we only attempt to obtain the tightest possible complexity upper bounds of the game by collecting only a small sample of the best-play automata. We apply a compound uniform random sample (with possible replacement). We will convert each best-play automaton in the sample to a transformation semigroup to study its complexity.

The SgpDec package written in GAP [8] provides the **holonomy decomposition**, which provides a complexity upper bound for a system. We additionally implement a function to verify **maximum chain of essential dependency**, which guarantees a lower bound [18, pp. 11-12] as discussed in the last chapter. We store each automaton sample's complexity bounds and the maximum chain of essential dependency in a database.

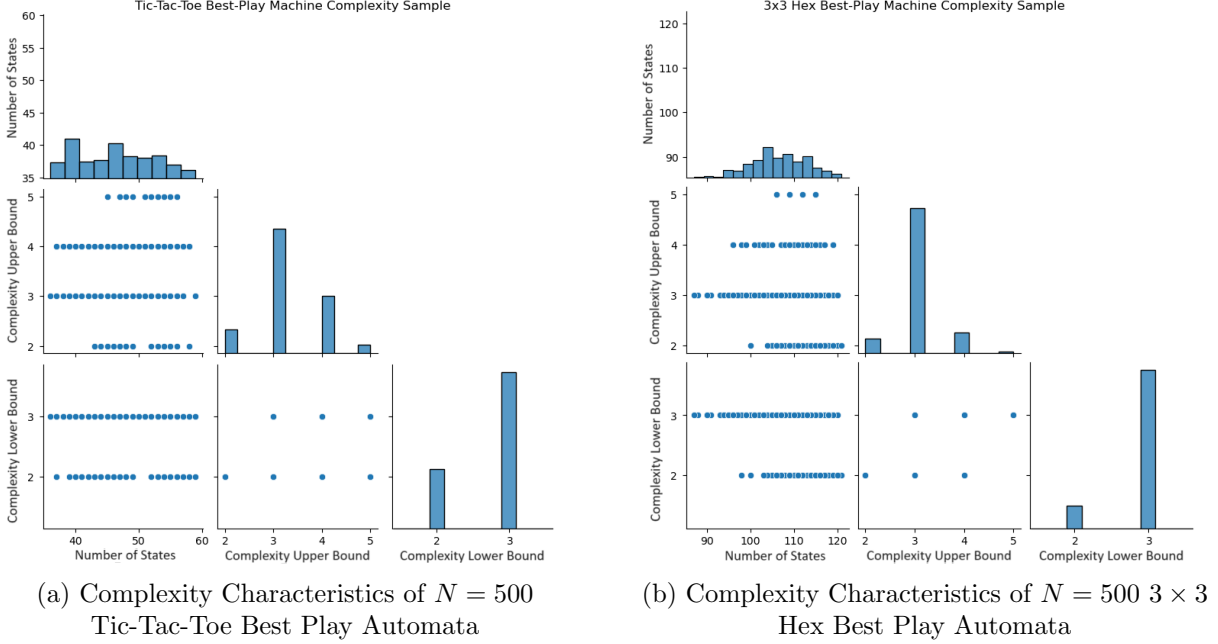


Figure 3.3: Pair plots of complexity lower bound, upper bound, and number of states for a sample of $N = 500$ best-play automata for (a) Tic-Tac-Toe and (b) 3×3 Hex.

3.4 Analysis

3.4.1 Complexity of Tic-Tac-Toe and Hex

As described earlier, we sample 500 random best-play automata for Tic-Tac-Toe and Hex and perform the complexity analysis on each one. The 500-sample size was chosen because it is large enough for an exploratory analysis and can be conducted within a reasonable time frame. Their complexity characteristics are summarized as pair plots in Figure 3.3.

The pair plots comprise three scatter plots and three histograms for the complexity lower bound, upper bound, and the number of states for the sampled best-play automata; they provide many insights into their complexity characteristics. Firstly, there are no correlations between the complexity and the number of states in the automata. In both cases, the sampled best-play machines with the fewest states do not have the lowest complexity (of 2). Secondly, the histograms show that only a tiny proportion of sampled best-play automata achieved a complexity of 2 (for both the lower and the upper bounds). This

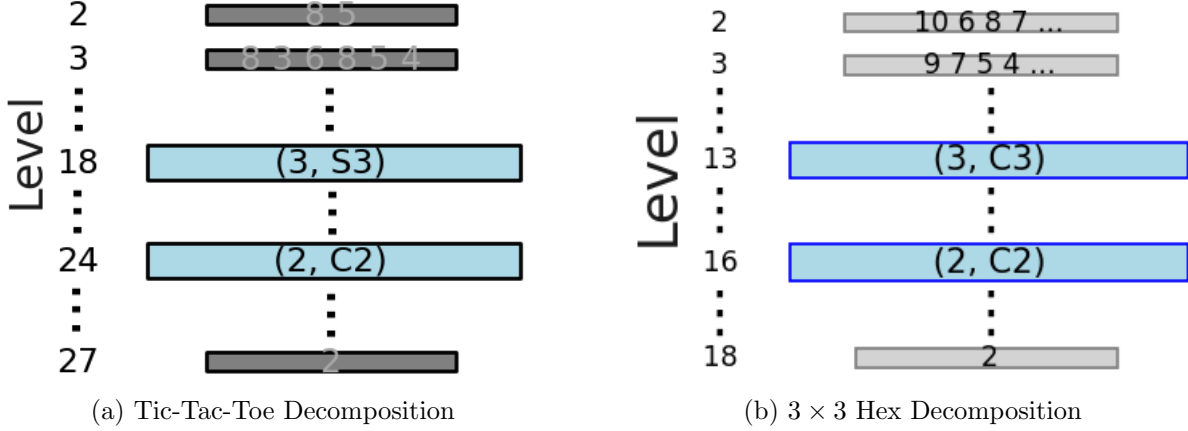


Figure 3.4: Holonomy decompositions of (a) a Tic-Tac-Toe best-play machine with complexity two and (b) a Hex best-play machine with complexity 2.

indicates that we will need a considerably larger sample size to potentially discover an automaton with a lower complexity. Thirdly, in both of the complexity lower bound vs. upper bound scatter plots, there exist automata whose complexity upper bound and lower bound are both 2, implying that some machines have an exact complexity of 2.

For Tic-Tac-Toe, one automaton with an exact complexity of 2 has the holonomy decomposition shown in Figure 3.4(a); this particular holonomy decomposition shows 27 total levels, but only 2 of them (levels 18 and 24) have nontrivial groups. There is also an essential dependency between these two levels with C_2 structure. Since we only narrow the game complexity upper bound to 2, we cannot confirm Conjecture 2 that Tic-Tac-Toe has a complexity of 1 yet.

For the 3×3 Hex game, one automaton with an exact complexity of 2 has the holonomy decomposition shown in Figure 3.4(b); we see level 13 with C_3 and level 16 with C_2 . Exploring a regular Hex game is currently unrealistic. Nonetheless, it does not contradict Theorem 3, as $\theta(G) \leq 2 \leq \frac{3.3}{2}$.

In both cases, we used a sample of 500 automata to test the complexity characteristics. In all 500 samples, we attain complexity ≥ 2 . Let X be the random variable representing the number of times where complexity < 2 . Define $p_0 := P(\text{complexity} < 2)$. The binomial distribution states that

$$P(X = 0) = \binom{500}{0} p_0^0 (1 - p_0)^{500} = (1 - p_0)^{500}$$

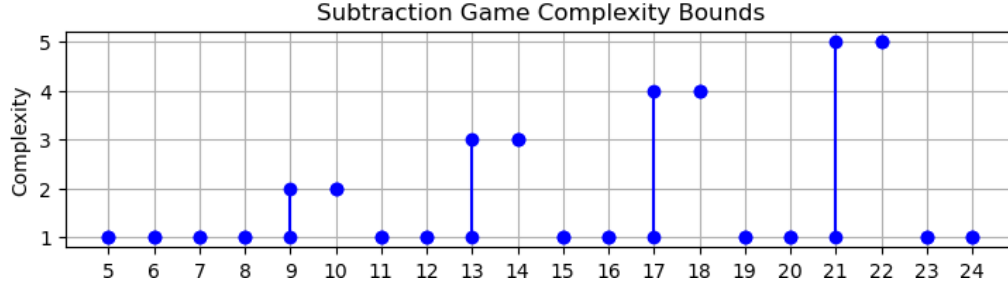


Figure 3.5: Lower and Upper Bounds on Complexity of Subtraction Game vs Game Size

For example, $p_0 = 0.00001$ would imply $P(X = 0) \approx 0.995$, i.e., if 1 out of 100000 samples has a complexity less than 2, then out of 200 trials, we expect to see 1 trial that proves the game complexity is less than 2. Regardless, $p_0 = 0.00001$ is inconsequential considering the population size is likely in the 10^{21+} s.

3.4.2 Complexity of Subtraction Game and Nim

We start with a subtraction game since we can search all of the best-play machines for a game small enough, e.g., five chips only. Searching among game sizes of 5 up to 24 yields the following minimal bounds:

From Figure 3.5, we obtain Proposition 1. Note that when $N \bmod 4 \equiv 2$, we do not know the exact complexity of the game, only that it is between 1 and $\lfloor N/4 \rfloor$.

Proposition 1. *Let N be the size of the Subtraction Game, i.e., the number of chips in a pile at the start of the game. For $N \in \{5, 6, \dots, 24\}$:*

If $N \bmod 4 \equiv 0$ or $N \bmod 4 \equiv 3$, then $\theta(G) = 1$.

If $N \bmod 4 \equiv 1$, then $1 \leq \theta(G) \leq \lfloor N/4 \rfloor$.

If $N \bmod 4 \equiv 2$, then $\theta(G) = \lfloor N/4 \rfloor$.

We would conjecture that this finding extends to any $N \geq 5$, and we can easily prove the following,

Theorem 4. *For any Subtraction Game G with size $N \bmod 4 \equiv 0$, $\theta(G) = 1$.*

Proof. Observe that when $N \bmod 4 \equiv 0$, the game is in a losing state (for the opponent), so whenever the opponent removes $x \in \{1, 2, 3\}$ chips, our gameplay machine has no choice but to remove $4 - x$ chips, and the resulting best-play machine always have the cyclic group

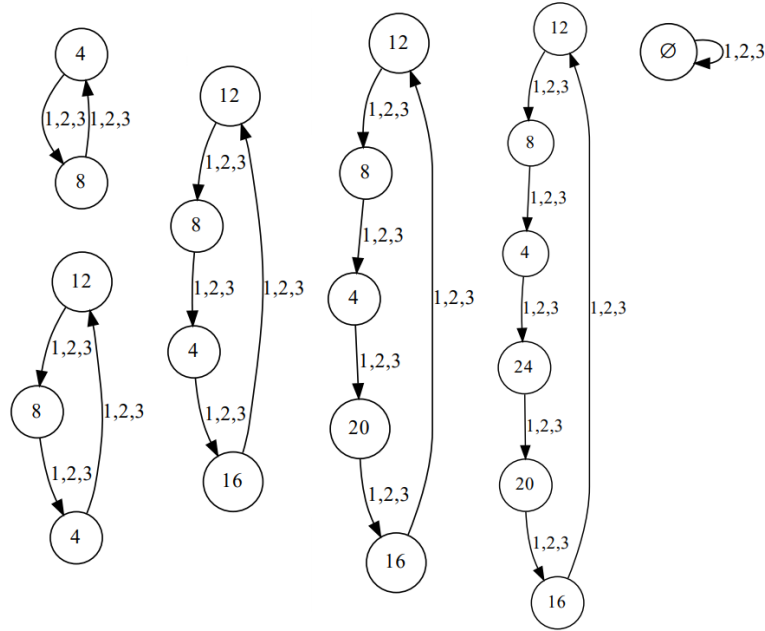


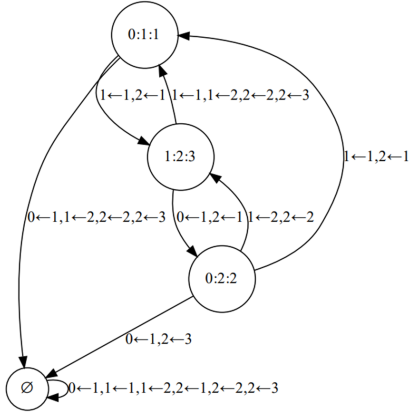
Figure 3.6: Best-play automata for selected Subtraction Games ($N = 8, 12, 16, 20, 24$). Nodes show the number of chips (in opponent view); transitions are labelled by opponent moves and lead to opponent view after the best-play automaton's response.

structure C_i where $i = N/4$, as illustrated in Figure 3.6. This implies a Krohn-Rhodes complexity of 1. $\square$

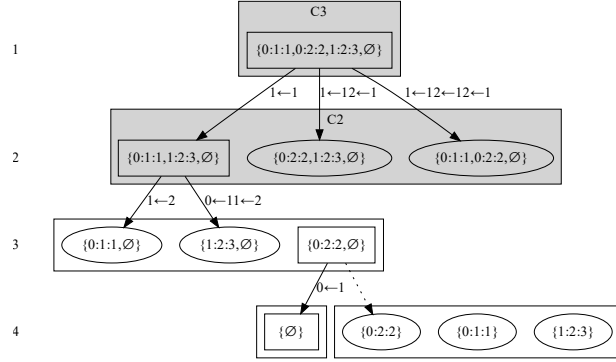
As for Nim, the best-play machine can vary wildly depending on how the game is set up. For the simple configuration, $2 : 2 : 3$, the full minimax tree search yields 12 best-play automata with an unknown game complexity that can be 1 or 2. However, for the simple configuration $1 : 2 : 3$, there is only one unique best-play machine, and it has exact complexity 2; its holonomy decomposition has two levels with nontrivial groups and a maximum chain of essential dependency of length 2 (Figure 3.7). Therefore, we disproved the Conjecture 3 that Nim has a complexity of 1.

3.5 Conclusion and Insights

In this chapter, we have undertaken a detailed analysis of the algebraic structures underlying several two-player games with perfect information, explicitly focusing on Nim,



(a) 1 : 2 : 3 Best-Play Automaton



(b) 1 : 2 : 3 Holonomy Decomposition

Figure 3.7: (a) The unique best-play automaton for the Nim Game 1 : 2 : 3. (b) Skeleton for the unique best-play automaton for Nim Game 1 : 2 : 3, having two levels with nontrivial groups and an essential dependency between them showing complexity 2.

Subtraction Game, Tic-Tac-Toe, and the 3×3 Hex game. These games' relatively small state spaces allowed for comprehensive minimax tree searches. Our approach involved crafting precise and compact state-space formulations for each game, considering factors such as board symmetry. Then, we backtracked our minimax algorithm to generate all possible best-play automata for smaller Nim and Subtraction Games, where we performed holonomy decomposition and found the maximum chain of essential dependency to confirm each one's exact complexity. However, due to the practical limit and the absurd number of best-play automata for Tic-Tac-Toe and 3×3 Hex, we only sampled 500 best-play automata to estimate the tightest complexity bound (Figure 3.3). We do not know the exact number of best-play automata for these games. However, extensive computational analysis conducted over several days indicates that the number is at least in the trillions and potentially several orders of magnitude larger. Our sample constitutes much less than 0.00000005% of all the best-play automata for Tic-Tac-Toe, so in the future, it may be more effective to attempt to directly construct a best-play automaton of complexity one.

To summarize, we have made several significant strides that further our understanding of the algebraic structure of games. Firstly, none of our findings contradicted Theorem 3, reaffirming its broad applicability. Secondly, a notable discovery was a Nim game setup with a complexity of 2, which refutes Conjecture 3. Thirdly, our analysis of Tic-Tac-Toe gives a complexity upper bound of 2. However, the possibility of a best-play machine with

an exact complexity of 1 remains open.

Chapter 4

Complexity of Mixed-Strategy Best-Play Automata

In previous chapters, we studied two-player complete information sequential games in which moves from either player have deterministic transition functions. Their optimal strategy is pure (i.e., non-probabilistic) and can be found through a complete minimax tree search. This chapter explores games in which moves from either player may result in multiple possible states. In this case, the optimality is achieved when a probabilistic choice of moves gives the highest expected utility among all the probabilistic state transitions.

Simultaneous move games are games where two players make moves simultaneously for an arbitrary number of turns. An example would be Rock Paper Scissors: Best Two out of Three. The state transition of the player's move depends on the opponent's move and is, therefore, not deterministic from the player's perspective. We can determine the optimal mixed strategy for simultaneous move games utilizing results from the classical game theory; the same methodology also applies to other imperfect information games with nondeterministic transition functions, such as those involving randomness or incomplete information. These games facilitate the need for mixed-strategy best-play automata and the study of their algebraic structures and properties, which will be examined in greater detail.

4.1 Extensive Form Games, Motivations

4.1.1 Simultaneous Move Games

In the context of a single-turn, two-player strategic game, the players make a move simultaneously, and the game ends. Each pair of moves results in a utility for the player and a different utility for its opponent, which is collected into a **payoff table**. Table 4.1 displays the payoff table for the single-turn Rock Paper Scissors game. The table's rows correspond to my moves as Player 1, and the columns correspond to the opponent's moves as Player 2. Each cell in the table contains a 2-tuple, indicating my utility followed by the opponent's utility. For example, the cell corresponding to the row "Rock" and the column "Paper" is $(-1, 1)$. This indicates that if I play "Rock" and my opponent plays "Paper," I lose, and therefore, I receive a utility of -1 , whereas my opponent gets a utility of 1 .

		Player 2 (Opponent)		
		Rock	Paper	Scissors
Player 1 (Me)	Rock	(0, 0)	(-1, 1)	(1, -1)
	Paper	(1, -1)	(0, 0)	(-1, 1)
	Scissors	(-1, 1)	(1, -1)	(0, 0)

Table 4.1: Single-turn Rock Paper Scissors payoff table.

None of my moves are optimal by themselves, as my opponent can adjust their strategy to ensure I lose and receive a utility of -1 . Instead, we say the player's strategy is *optimal* at the **Nash equilibrium** where the opponent has no incentive to switch strategies [15, p. 258]. There are two types of strategies. A **pure strategy** specifies a move in every possible situation, whereas a **mixed strategy** has a probabilistic move at each state. There may not always be a pure-strategy Nash equilibrium. In the single-turn Rock Paper Scissors game, the mixed-strategy Nash Equilibrium is to play each of the three moves with an equal probability, giving an expected utility of 0 for me and my opponent.

A single-turn two-player strategic game can be defined as tuple $\Gamma = (S_1, S_2, u_1, u_2)$ where S_1 is the set of moves that Player 1 can make, S_2 is the set of moves that Player 2 can make; $u_1 : S_1 \times S_2 \rightarrow \mathbb{R}$ defines the payoff (utility) of Player 1 and $u_2 : S_1 \times S_2 \rightarrow \mathbb{R}$ defines the payoff of Player 2. A (mixed) **strategy profile** $\mathcal{P}$ is the tuple of the probability distribution of moves for each player, i.e., in our case,

$$\mathcal{P} := (p_1 : S_1 \rightarrow (0, 1), p_2 : S_2 \rightarrow (0, 1)) \quad \sum_{s_1 \in S_1} p_1(s_1) = 1 = \sum_{s_2 \in S_2} p_2(s_2)$$

Nash’s Theorem states every strategic game with a finite number of strategies has a (mixed-strategy) Nash equilibrium [17, pp. 286-295]. Typically, finding the Nash equilibrium requires two steps [12, pp. 1-3]:

1. For each player, find the **best response** function, which produces the set of payoff-maximizing moves given the other player’s move.
2. Find the set of all strategy profiles compatible with the best response functions for both players. These are the mixed-strategy Nash Equilibria.

This process is generally nontrivial. Thankfully, open-source tools, such as Nashpy [11] and Gambit [22], enable us to programmatically solve mixed-strategy Nash Equilibria.

The **extensive form games** is a generalized type of finite strategic game where players make simultaneous moves over multiple turns. This forms a game tree, each node representing an independent strategic game. The payoffs are only known at the terminal node. For our purpose, we will only consider two-player simultaneous move games.

Consider Matching Pennies where Player 1 and 2 simultaneously choose “Heads” or “Tails” of their penny. Player 1 gains a point if the two pennies match and loses a point otherwise; Player 2 gains a point if the two pennies do not match and loses a point otherwise. This gives the payoff table in Table 4.2.

		Player 2 (Opponent)	
		Heads	Tails
Player 1 (Me)	Heads	(1, -1)	(-1, 1)
	Tails	(-1, 1)	(1, -1)

Table 4.2: Single-turn Matching Pennies payoff table.

Figure 4.1 illustrates the game tree of a two-turn Matching Pennies, an extensive-form game. The payoffs at the terminal nodes are the final scores. This chapter aims to find an appropriate algebraic representation of mixed-strategy best-play machines for extensive-form games.

To determine the optimal mixed strategy at a given game state in an extensive-form game, we need to identify the **subgame perfect Nash equilibrium (SPNE)**. A **subgame** is a node in the game tree, which is a game that starts at some history. The SPNE is the Nash equilibrium in each subgame of the game. It is based on the expected payoffs of the child nodes, assuming optimal strategy profiles for both players in the subsequent

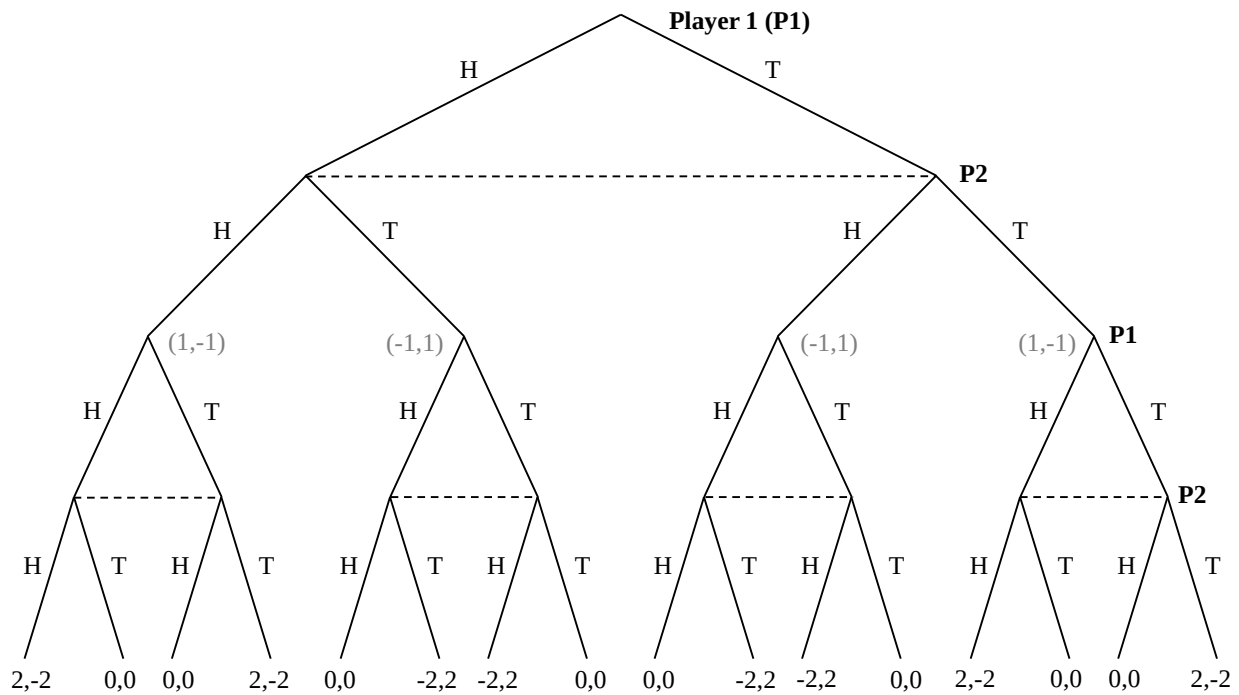


Figure 4.1: Game tree for a two-turn Matching Pennies, an extensive-form game. The dashed lines indicate that Player 1 and Player 2 make moves simultaneously. The grey payoffs in brackets are the expected payoffs derived from the SPNEs, consistent with the payoffs from a single-turn Matching Pennies game.

turns. For a game with a finite number of turns, we can use **backward induction** to find the SPNE [12, pp. 7-8]:

1. Given the full game tree, start from nodes that are the last turns before endgame terminal nodes, i.e., the immediate parent nodes of leaf nodes.
2. Determine the mixed-strategy Nash equilibria and expected payoffs for the parent nodes of the leaves.
3. Replace these nodes with the expected payoffs at Nash equilibrium.
4. In a bottom-up manner, reduce the game tree iteratively by replacing each unvisited bottom-most node with the expected payoffs of its subgame. This process continues until the Nash equilibrium of the root node is found.

In general, finding the SPNE for an infinite-turn extensive form game is very challenging. However, for an infinite-turn repeated two-player strategic game, where each turn produces payoffs based on the same payoff table, verifying SPNEs for specific strategies is possible. One such game is the infinite-turn Prisoner's Dilemma. For context, a single-turn Prisoner's Dilemma gives the payoff table in Table 4.3; the specific payoff values can be changed depending on the game definition. However, the gist is that a player is always better off playing Defect.

		Player 2 (Opponent)	
		Cooperate	Defect
Player 1 (Me)	Cooperate	(5, 5)	(0, 6)
	Defect	(6, 0)	(1, 1)

Table 4.3: Single-turn Prisoner's Dilemma payoff table.

For an infinite-turn repeated two-player strategic game, the expected payoff at any node of the game is its **present value (PV)**. It is calculated based on a discount factor $\delta \in [0, 1)$ and the payoff stream $\pi = (\pi_0, \pi_1, \dots, \pi_t, \dots)$ where π_0 is the payoff on the current turn, and the rest are the payoffs for each of the future turns that extends into infinity.

$$PV(\pi, \delta) = \sum_{t=0}^{\infty} \delta^t \pi_t = \pi_0 + \delta \pi_1 + \dots + \delta^t \pi_t + \dots$$

The typical steps to analytically verify SPNEs would then be the following [20, pp. 207-217].

1. Construct candidate strategies. In the example of the infinite-turn Prisoner's Dilemma, some common candidate strategies include:
 - **Grim**: Always play Cooperate in the first turn. Continue to play Cooperate if the opponent has always played Cooperate in the past; play Defect if The opponent ever played Defect.
 - **Naive**: Always play Cooperate.
 - **Tic-for-Tat**: Play Cooperate in the first turn, and then copy whatever the opponent played in the last turn.
2. Reduce the game to **augmented stage game**, which reduces the infinite-turn sub-game into expected payoffs based on the present value [20, p. 209].
3. Apply the **single-deviation principle** to check for SPNE. The single-deviation principle states that a strategy profile in an infinite-turn repeated game is subgame-perfect if it always yields an SPNE in the augmented stage game [19, pp. 178-180] [20, pp. 209-210].
4. Apply **folk theorem** which determines the range of δ and necessary conditions for the SPNE [20, pp. 216-221].

The details and proofs for these steps are highly nontrivial and out of scope for our study. Additionally, these steps verify whether the proposed candidate strategies are SPNEs; they do not capture all the SPNEs of interest. Unfortunately, these limitations mean we can only explore the algebraic structure of best-play machines for only a limited set of SPNE candidate strategies regarding infinite-turn repeated strategic games.

4.1.2 Imperfect Information Games

It is worth noting that extensive form games encompass not only simultaneous-move games but also sequential games and games with imperfect information, including elements like randomness and incomplete information. We typically use the **Harsanyi transformation** to solve randomness and incomplete information [29].

1. Introduce “Nature” as the first player whose moves determine the incompleteness of the game state, i.e., produce effects identical to the randomness or incomplete information.
2. Each player chooses their strategy, completing the turn.
3. The subsequent turns should not involve Nature; the players choose their strategies until the game ends.
4. Fill in the endgame payoffs according to the payoff tables and calculate the SPNEs through backward induction.

This process converts the imperfect information game into an extensive form game [29] [28, p. 251]. To demonstrate this process, we provide the following example.

Suppose the Prisoner’s Dilemma’s payoffs are tied to the leniency of the prosecutor, which is unknown to either player. The harsh prosecutor A will produce game Γ_A with payoff table shown in Table 4.4, whereas the lenient prosecutor B produces game Γ_B with payoff table shown in Table 4.3. Playing the game for a single turn will produce the game tree in Figure 4.2. In this game, the randomness is the prosecutor’s leniency, which can be either Lenient or Harsh. Player Nature represents this randomness. Observe that all three “players” make moves simultaneously, as the randomness is unknown to either Player 1 or 2 until all moves are made. In general, the simultaneity depends on the game setup. For example, if the players make moves after some dice are rolled, and the randomness becomes known to both players, then Player Nature’s move is made before the actual players in a sequential manner.

		Player 2 (Opponent)	
		Cooperate	Defect
Player 1 (Me)	Cooperate	(9, 9)	(0, 2)
	Defect	(2, 0)	(1, 1)

Table 4.4: Single-turn Prisoner’s Dilemma (harsh variant) payoff table.

In the specific game scenario where the exact probability distribution of Player Nature’s moves is known, we can still determine SPNEs using backward induction. However, for the SPNE of the first round (i.e., nodes at the top of the game tree), we need to identify the set of strategy profiles that align with the best response functions for all players, given the known strategy of Player Nature. One can use the Gambit package [22] to find the approximate Mixed-Strategy Nash equilibrium for this.

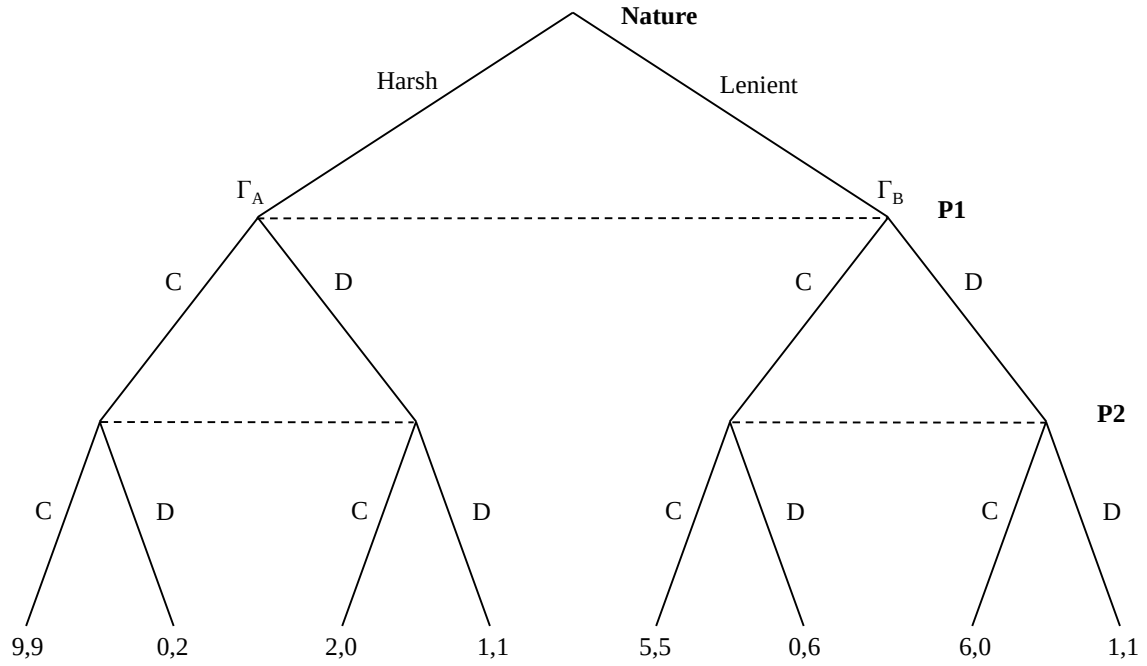


Figure 4.2: Game tree for a single-turn Prisoner's Dilemma, but with the unknown leniency of the prosecutor. The tree is produced after the Harsanyi transformation, where the unknown information is treated as the first player, Nature. To reiterate some important points: the tree depicts a single-turn simultaneous move game with 3 players; Player Nature has moves Harsh and Lenient; Player 1 and Player 2 have moves Cooperate (C) and Defect (D); the subgame Γ_A encapsulates Table 4.4, and subgame Γ_B encapsulates Table 4.3.

Overall, the extensive-form game represents a broad category of games, and tools from strategic game theory, such as SPNE, could be employed to solve for the optimal mixed strategies. These optimal strategies enabled the construction of a best-play machine for extensive games. This section concludes the background and motivation for mixed-strategy best-play machines and their potential use cases.

4.2 Mixed Strategy Best Play Machine Construction

This section aims to develop an algebraic construction to model mixed-strategy best-play machines for extensive form games.

4.2.1 Markov Decision Process Representation

As we recall from Chapter 3, one of the challenges for modelling complete-information two-player sequential games as finite automata is keeping the set of opponent moves constant. The constant set of opponent moves is then used as the alphabet for the transition function of the finite automata. For most of the games, the set of moves changes as the game progresses; e.g., Tic-Tac-Toe, where the set of valid moves keeps reducing, or a strange two-turn game Γ_{mix} whose first round is Prisoner’s Dilemma and the second round is Rock Paper Scissors.

The workaround for the inconsistent set of legal moves, as described in Chapter 3, is to introduce a permanent “garbage” state in the best-play automaton that captures all the irrelevant or illegal moves. Once the game enters the garbage state, it will remain there permanently. In the case of an extensive form game, we could take all possible opponent moves into a single set of moves $\tilde{A}$ that serves as the alphabet for the automata. The strange game Γ_{mix} would then have $\tilde{A} := \{\text{Cooperate, Defect, Rock, Paper, Scissor}\}$; if the opponent played Rock in the first turn when they were supposed to play the Prisoner’s Dilemma, the game would enter the garbage state. This is not a very efficient workaround, as actions that are only legal for a couple of rare occasions will increase the state space. Therefore, for simplicity and ease of progression, this chapter only considers games where the set of legal actions is consistent in all the opponent’s turns.

Consider a multi-turn extensive form game Γ_{ex} whose partial game tree is shown in Figure 4.3. The partial tree describes a simultaneous move, an extensive form game that lasts up to 3 turns. To be consistent with Chapter 3, the opponent makes a move first. The set of opponent moves is $S_{opp.} = \{A, B\}$, and the set of my moves is $S_{me} = \{X, Y, Z\}$.

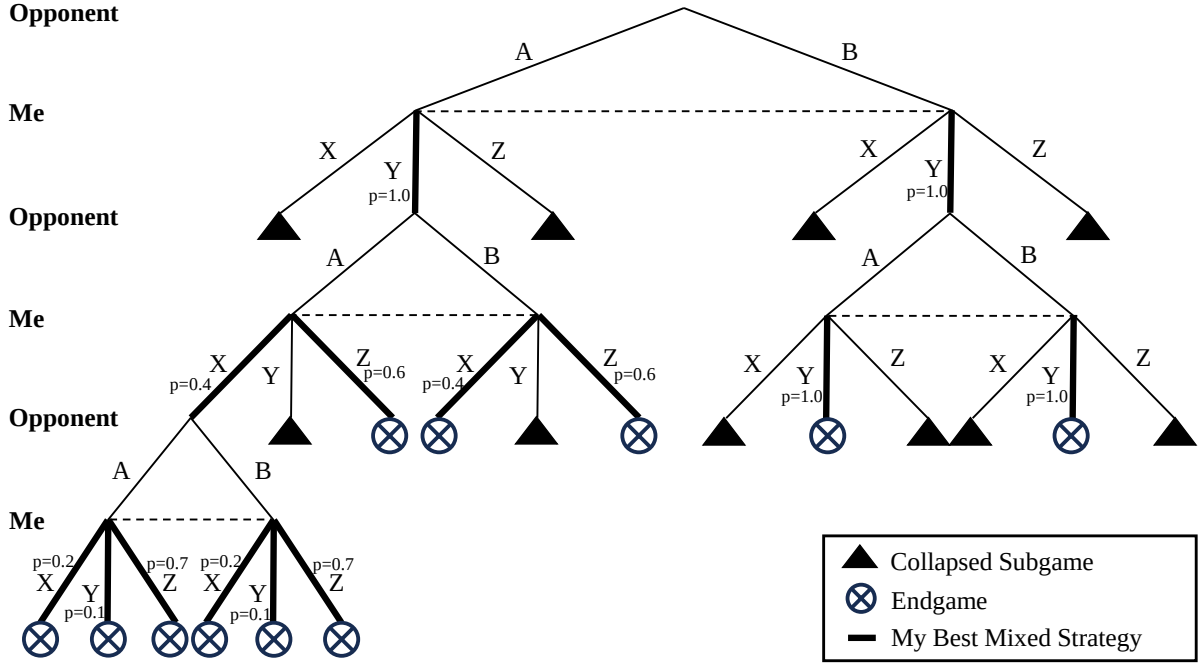


Figure 4.3: Partial game tree of Γ_{ex} . The opponent plays first. The (partial) game lasts up to 3 turns. The subgames that do not aid in constructing the best-play machine are collapsed and shown as solid triangles. The SPNEs and best mixed strategies have been found for my turns, so the payoffs are not shown at endgames.

$S_{opp.}$ and S_{me} contain the only legal moves in each turn. This game tree is constructed for demonstration purposes. The goal is to find a suitable best-play machine that captures the mixed-strategy nature of optimal gameplay so that we can study its algebraic properties.

In the Γ_{ex} game tree, we can assume that backward induction has already been used to find each state's subgame perfect Nash equilibrium (SPNE). Therefore, the payoffs are omitted, and the endgames are marked with the circled times symbol. The SPNE gives the optimal mixed strategy for my turns, and moves in my optimal mixed strategy are shown as bold branches along with their probability. For moves not in my optimal mixed strategy, they will never be chosen during the best-play machine construction; therefore, their subtrees are collapsed and shown as solid triangles.

Recall in Chapter 3 the best-play machine responded to opponent moves with my best (pure) strategy. In this chapter, we consider that my best strategy might be mixed.

One candidate for the modelling of mixed strategy best play machine is the simplified

Markov Decision Process (MDP), a 3-tuple (Q, S, δ) [16] perfectly models the state transition for a mixed-strategy best-play machine. Here, Q is the states, i.e., the set of all reachable board positions in the opponent’s view, and S is the actions, i.e. the set of moves the opponent can make; $\delta : Q \times S \rightarrow \Pi(Q)$ is the state transition function that takes the opponent view and action as input, adds my probabilistic response and returns the probability distribution over the new states.

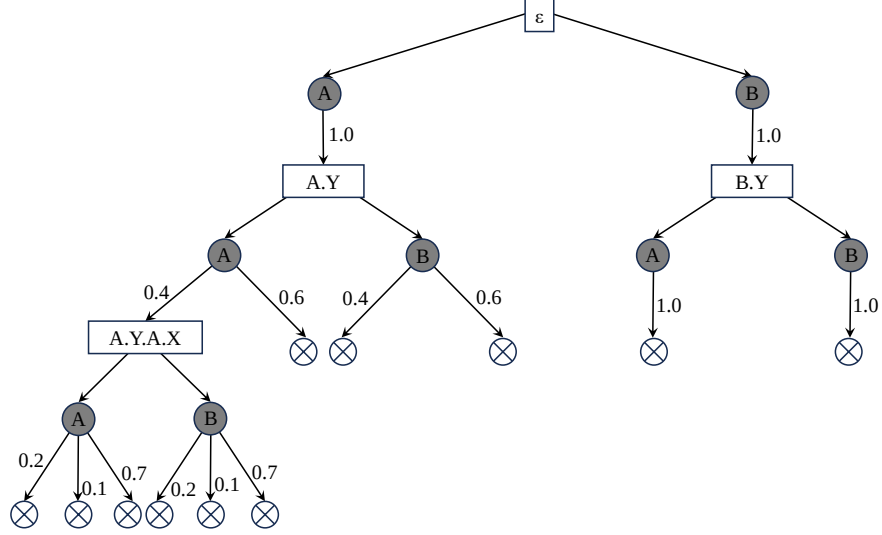
Using Γ_{ex} for demonstration, we can transform the game tree from Figure 4.3 into an MDP process, as illustrated in Figure 4.4. In this MDP, each state transition is initiated by the opponent’s move, and the new states represent the opponent’s perspectives after I have responded using optimal mixed strategies. The probabilities of my moves are also depicted in the MDP diagram. In Figure 4.4(a), the non-endgame states are rectangular boxes, and the endgame states are indicated with circled-times symbols. Recall in Chapter 3, we require the endgames to loop back to the initial state to reset the game; we can do so by replacing all the endgame states with ε . Consequently, the same action has multiple probabilistic transitions to different endgame states; they are merged into a single transition, yielding Figure 4.4(b). The probabilistic transition is highlighted in red.

4.2.2 Nondeterministic Finite Automaton Representation and Powerset Construction

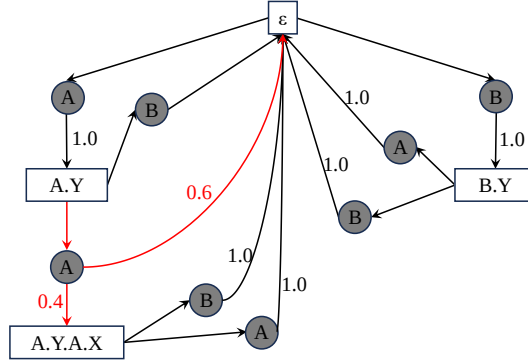
Obtaining the MDP of the best-play is a step closer to representing the best-play machine in deterministic finite automaton (DFA) form. As the ultimate goal is to study the algebraic structure of games, the specific probabilities of each transition are not of interest. Naturally, the next candidate is the **nondeterministic finite automaton (NDFA)**, which disregards the specific probabilities of the transitions.

An NDFA is very similar to a DFA. The main difference is that the NDFA machine can transition to multiple states or none of all for a given input symbol. For instance, if an input symbol causes transitions to two different states, s_1 and s_2 , the state machine essentially “splits” into two copies in parallel, one with s_1 and the other with s_2 . An NDFA also has an ε transition that can occur without consuming any input symbol.

An MDP can be directly converted into an NDFA [1, p. 12]. This conversion involves copying the states from the MDP to the NDFA and defining the NDFA’s transition function to match the MDP’s transitions with a nonzero probability. The MDP of Γ_{ex} from Figure 4.4(b) can be converted into the NDFA shown in Figure 4.5 using precisely this process.



(a) Conversion of Γ_{ex} 's game tree into MDP. Treating endgames as separate states.



(b) Compact MDP representation. All endgame states loop back to initial state.

Figure 4.4: The rectangular boxes are the states. ε indicates the initial state. The grey circles indicate actions. The transition probabilities are indicated on the arrows. The circled-times symbols are the endgame states.

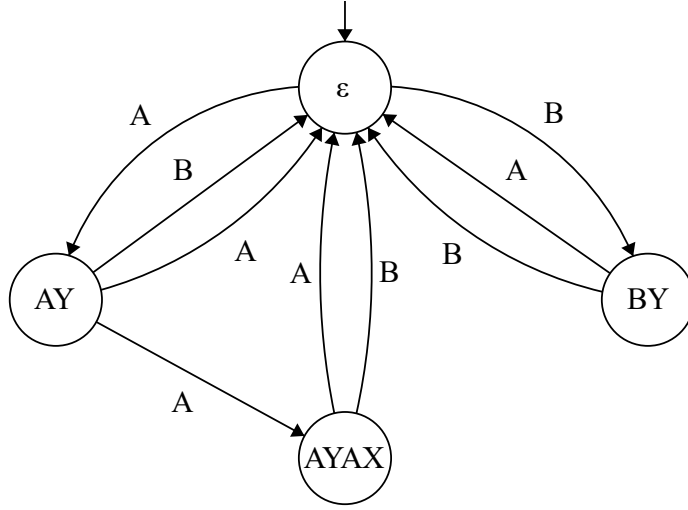


Figure 4.5: The nondeterministic finite automaton of Γ_{ex} . ε is the starting state. A multi-state transition occurs on state $A \cdot Y$, where action A transitions to state ε and state $A \cdot Y \cdot A \cdot X$.

While there is extensive literature and software tools to analyze the algebraic structure of DFAs based on the semigroup theories, there are no comparable resources to do the same analysis on NDFAs directly. Therefore, it is sensible first to convert the N DFA to DFA using the **powerset construction** and then perform the analysis using SgpDec.

The powerset construction is the standard technique to convert an N DFA to an equivalent DFA. While the full powerset construction algorithm has some nuances, we will not discuss those details here due to the simplified properties of our N DFA. Specifically, our N DFA is guaranteed to have only one starting state and no accepting states. Given these constraints, the general steps to convert our N DFA to a DFA are as follows:

1. Determine the initial state of the DFA, which is a set containing the initial state of the N DFA.
2. For each state in the DFA (which is a set of N DFA states), compute the set of states reachable for each input symbol. If the resulting set of states is not already in the DFA, add it as a new state. This gives the new transition function.
3. Exhaust the creation of all the new states by completing the loops in the last step. This gives the resulting DFA.

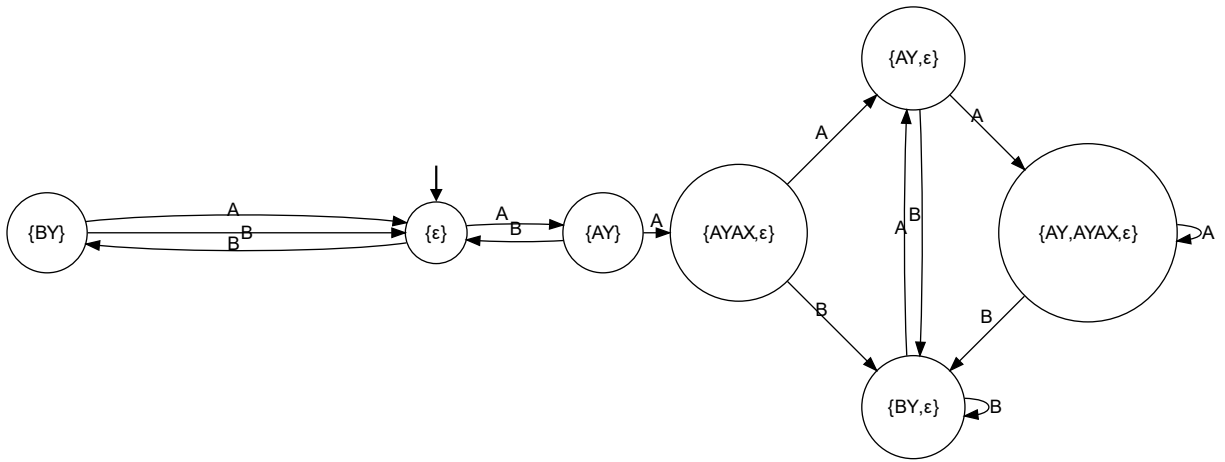


Figure 4.6: Applying the powerset construction algorithm on the NFA of Figure 4.5 results in this DFA.

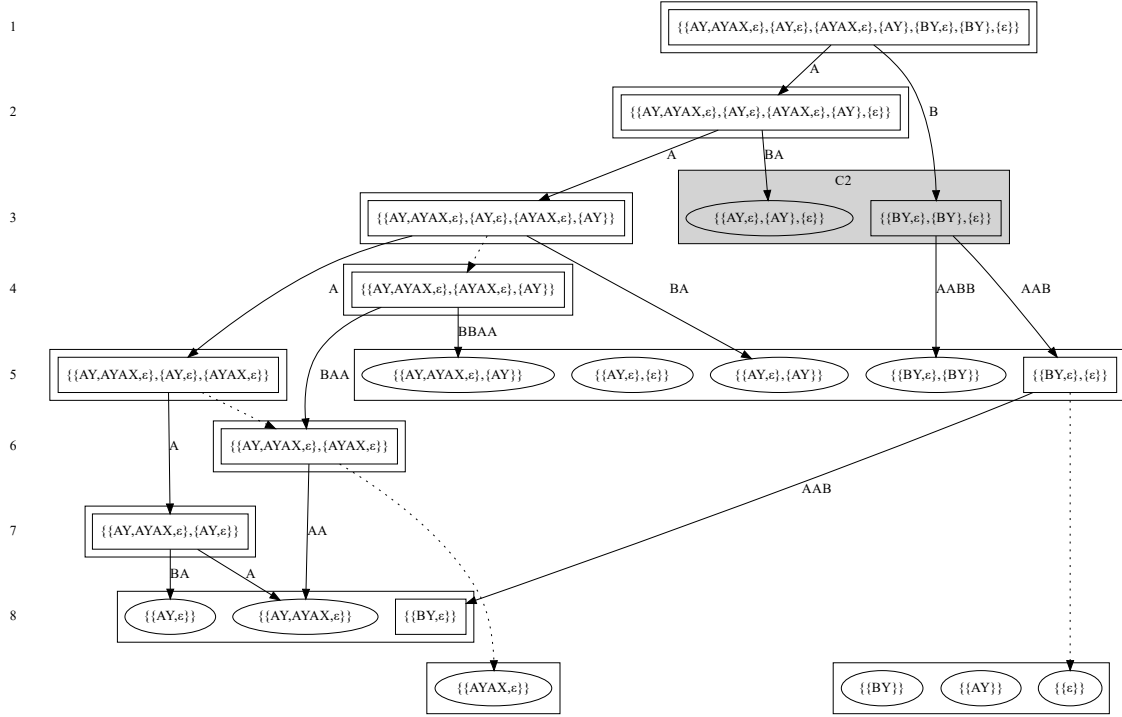


Figure 4.7: Skeleton of the DFA from Figure 4.6 after powerset construction.

Performing Holonomy Decomposition on the Figure 4.6 yields the skeleton shown in Figure 4.7. Only level 3 contains a nontrivial C_2 group. Since no other level contains nontrivial groups, the complexity of the Γ_{ex} DFA is 1 and no essential dependency check is required.

Γ_{ex} demonstrates a method to analyze the algebraic structure of mixed-strategy best-play automata. The first step involves representing the automata as nondeterministic finite automata (NDFAs). Subsequently, the NDFAs are transformed into deterministic finite automata (DFAs) using powerset construction. Finally, the Krohn-Rhodes decomposition is employed on the resulting DFAs. Unfortunately, this method has several significant drawbacks.

One obvious drawback is the state explosion issue. If the NFA has N states, the powerset construction creates a DFA with up to 2^N states [25, p. 55]. This exponential increase can make the DFA impractically large.

There are ways to mitigate the state explosion.

- The powerset construction method we proposed is already a lazy construction algorithm, i.e., it only produces states that are reachable from the initial state. Therefore, we do not need to prune any unreachable states explicitly.
- We can apply DFA minimization algorithms, such as Hopcroft’s algorithm [10, pp. 189-196], after the powerset construction. In this case, if a transition arrives at two different states that behave identically, they will be merged into a single state. Note that this does not inherently solve the state explosion issue immediately after powerset construction.

Despite these optimizations, the state explosion remains a significant challenge, further limiting the size of the games we can practically study.

Another drawback of the powerset construction is the possibility of creating a “fuse” semigroup, i.e. a finite aperiodic semigroup where every state collapses into a single final state (i.e., the set of all states in the NDFA). In this case, the fuse will not contain any permutator groups, and its complexity is guaranteed to be 0. This is highly undesirable because if the powerset construction produces a fuse, we lose the information on the complexity of the original NDFA.

There are a few ways that a “fuse” can occur after the powerset construction.

Firstly, the powerset construction can produce a fuse if the NDFA has an input symbol that creates both a self-loop and an advancement on a state. As seen in Figure 4.8, performing powerset construction on (b) yields a fuse (c). This is highly undesirable because (b) “contains” the C_2 group (a), and we expect to see a nonzero complexity. Similarly, example (d) “contains” a C_3 group, but the powerset construction yields a fuse (e).

This behaviour could be avoided if we modify the powerset construction to eliminate all unnecessary self-loops. A self-loop is *unnecessary* if the input symbol that causes the self-loop also causes an advancement in the state. In essence, unless a move from an opponent permanently traps the game in a state, we will automatically skip all turns that do not progress the game.

Secondly, if the sequence of states has multiple nodes that both advance the sequence and loop back to a fixed earlier state following the same input, a fuse could occur during the powerset construction. Two examples are shown in Figure 4.9; we can derive pure-strategy

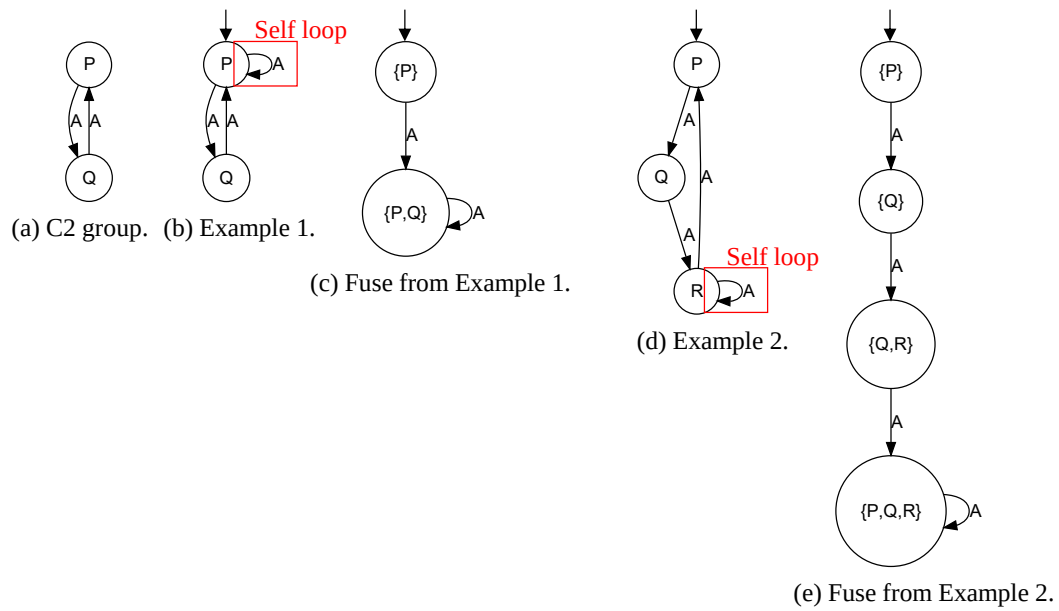


Figure 4.8: Two examples of how self-loop coupled with advancement causes a “fuse.”

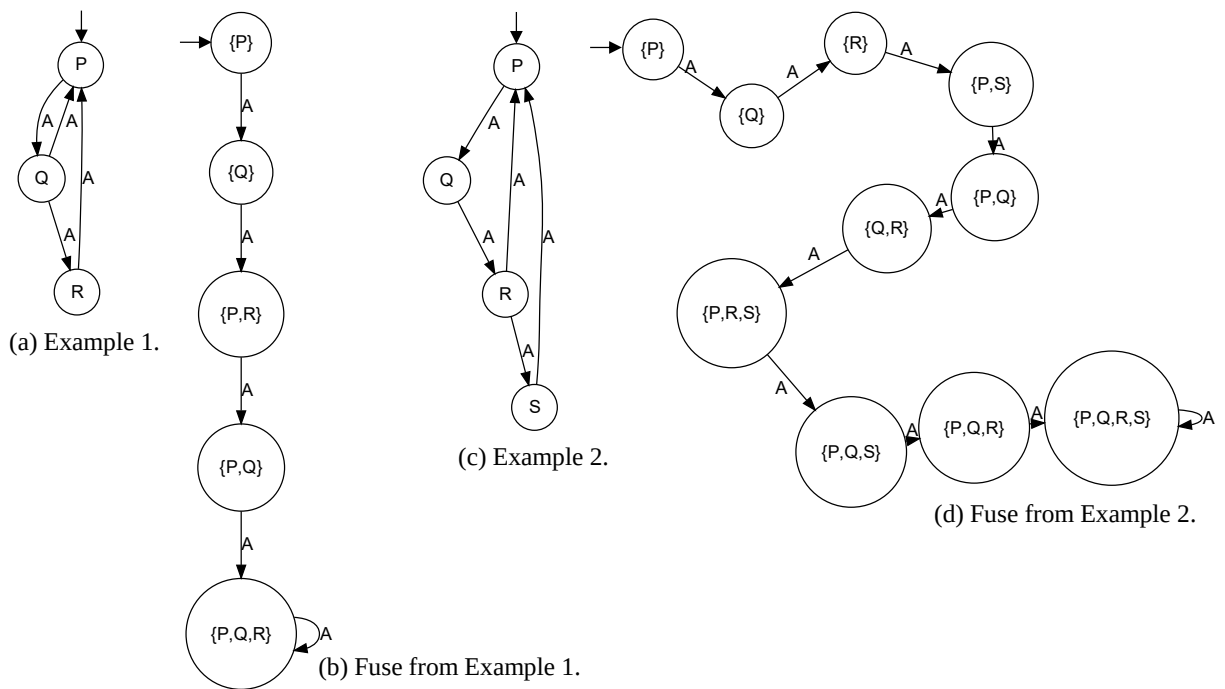


Figure 4.9: Two examples of another scenario that causes a “fuse.”

DFA from the NDFA in (a) that is a C_2 group or a C_3 group; however, the powerset construction produces a fuse of complexity 0. This is again highly undesirable.

We do not have a comprehensive list of scenarios where the powerset construction leads to a fuse. Moreover, there are no general solutions to “defuse” these situations. This, combined with the exploding state issue, highlights the limited applicability of NDFA to represent mixed-strategy best-play machines.

4.2.3 Automata with Context-Aware Inputs

This subsection proposes an alternative representation of mixed-strategy best-play machines called **automata with context-aware inputs (ACAI)**. ACAI is a DFA $\mathcal{M} := (Q, \Sigma, \mathcal{C}, \delta, q_0)$ where the input symbol is augmented with context to transition to another state deterministically.

- Q is the finite set of states, i.e., all board positions at the opponent’s turns and the garbage state.
- Σ is the finite set of input symbols, i.e., the set of opponent moves.
- $\mathcal{C}$ is the finite set of **context**; a context is a tuple of objects that encapsulates sufficient information to identify the state at the start of the next turn deterministically.
- $\delta : Q \times \Sigma \times \mathcal{C} \rightarrow Q$ is the transition function. If the context does not fit the input symbol or the specific state provided, the transition will go to the garbage state.
- q_0 is the start state.

This construction allows us to maintain the same number of states as the NDFA, eliminating the state explosion issue. The trade-off is an enlarged action space, but only by a constant factor. Another advantage is that the fuse problem that arises from powerset construction is no longer an issue.

To differentiate from the term **input symbol**, we use the term *action space* to refer to $\Sigma \times \mathcal{C}$. The elements of the action space dictate the transitions of states deterministically.

Back to the example of Γ_{ex} , we have $Q = \{\varepsilon, A.Y, B.Y, A.Y.A.X\}$, $\Sigma = \{A, B\}$. To deterministically identify the state at the start of the next turn, we could let context be the set of my moves, but to distinguish the set of my moves with $\mathcal{C}$, we will use lowercase letters, i.e., $\mathcal{C} = \{x, y, z\}$. This will lead to the DFA shown in Figure 4.10, which is very

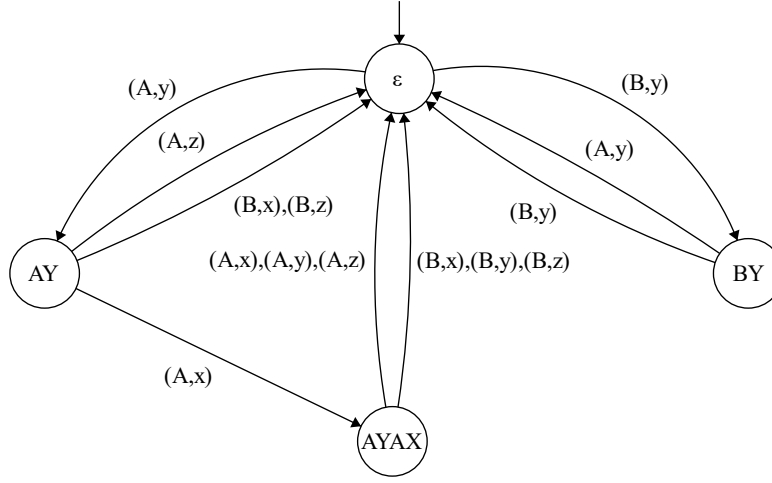


Figure 4.10: ACAI DFA of Γ_{ex} . Some elements of the action space $\{A, B\} \times \{x, y, z\}$ are not shown in the transitions because they lead to the garbage state $\emptyset$.

similar to the NDFA from Figure 4.5. An immediate observation is the much larger action space.

Performing Holonomy Decomposition on the Figure 4.10 yields the skeleton shown in Figure 4.11. Only level 3 contains a nontrivial C_2 group. Since no other level contains nontrivial groups, the complexity of the Γ_{ex} ACAI DFA is one and no essential dependency check is required.

Using the set of my moves as context could work for all deterministic, complete-information simultaneous move games. For imperfect information games, the context could be the moves from all other players, including Nature (from the Harsanyi transformation).

There are also drawbacks to the ACAI representation of mixed-strategy best-play machines.

- There is no equivalence between ACAI DFA and the NDFA representing the same game, unlike the DFA obtained through powerset construction. Therefore, conclusions obtained from the ACAI DFA do not apply to the NDFA.
- The context augmentation of the input symbol may confuse classical game theorists. The action space of ACAI DFA may make it seem as if the player gets to decide the exact trajectory of the game, even though the game is nondeterministic. It is

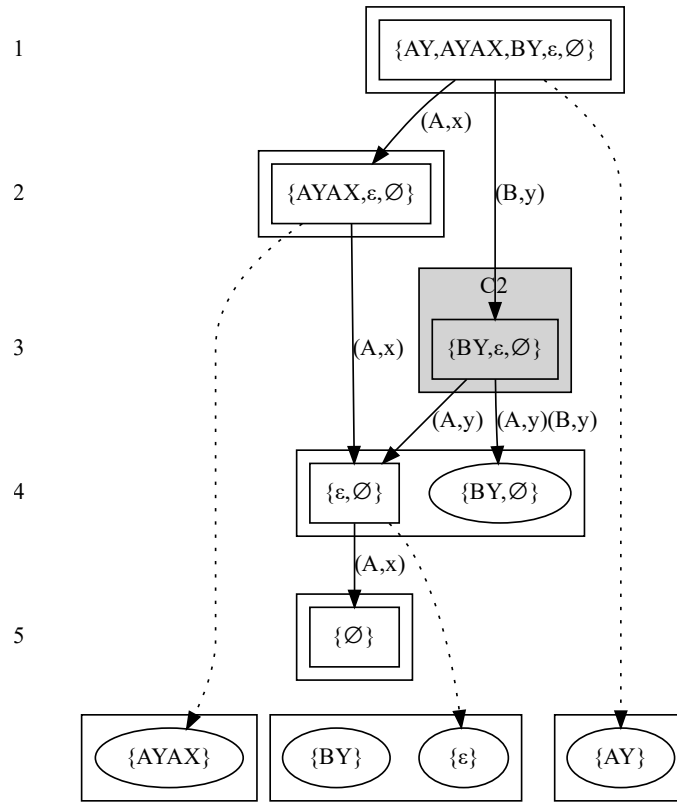


Figure 4.11: Skeleton of the ACAI DFA from Figure 4.10.

important to recognize that the primary purpose of ACAI is to study the algebraic structure of extensive form games rather than to simulate realistic gameplay.

- The action space is enlarged by a constant factor due to the context augmentation. This could be mitigated if we eliminate all “garbage” contexts that transition to the garbage state with every combination of state and input symbol.

Overall, we propose the ACAI technique to convert the game tree of the extensive form game into a DFA. It avoids the drawbacks of NDFA and the powerset construction at the cost of a large action space.

4.3 Conclusion and Insights

This chapter proposes two approaches to representing mixed-strategy best-play machines for extensive form games.

1. Represent the mixed-strategy gameplay, an MDP, using NDFA first and then using powerset construction to convert it to a DFA.
2. Represent the game tree of the extensive form game directly as an ACAI DFA. Use the same set of states and input symbols as the NDFA, but augment an input symbol with context to form an expanded action space.

Using an example game Γ_{ex} , we demonstrated that both approaches can represent the algebraic structure of mixed-strategy best-play machines, similar to Chapter 3. Unlike Chapter 3, an implicit benefit of both constructions is that each game yields exactly one best-play machine. There is no longer a need to sample thousands of best-play automata for a tighter complexity bound.

The NDFA powerset construction approach faces significant challenges. The exploding state issue means that the powerset construction can yield an exponentially more significant number of states in the DFA. Despite our proposed optimization methods, this approach can only be practically applied to games with an extremely limited number of states. Additionally, the approach faces a “fuse” problem, in which a reasonably complex NDFA representation of the game can collapse to a zero-complexity fuse during the powerset construction.

The ACAI representation is proposed as an alternative. It does not face the same level of challenges as the powerset construction. Even though the context augmentation of input symbols may be controversial to classical game theorists, ACAI perfectly replicates all the game paths from the game tree. Therefore, it is suitable for the study of algebraic structure.

Overall, some challenges from Chapter 3 carry over here. We are still limited in exploring games with moderately small state spaces. Nonetheless, this chapter lays the foundation for studying the algebraic structure and complexity of a much broader category of games, namely, extensive form games.

References

- [1] Alessandro Abate, David Parker, Cesare Tinelli, and Michael Whalen. A Survey on Applications of Model Checking to Software Configuration. 2019. <https://arxiv.org/pdf/1902.01042>.
- [2] Pál Dömösi and Chrystopher L Nehaniv. *Algebraic Theory of Automata Networks*, volume 11 of *Monographs in Discrete Mathematics*. SIAM, 2005.
- [3] A. Egri-Nagy, J.D. Mitchell, and C.L. Nehaniv. SgpDec: Cascade (de)compositions of finite transformation semigroups and permutation groups. In *Mathematical Software (ICMS 2014)*, pages 75–82. Springer, 2015.
- [4] A. Egri-Nagy and C.L. Nehaniv. Computational holonomy decomposition of transformation semigroups, 2015. <https://doi.org/10.48550/arXiv.1508.06345>.
- [5] Attila Egri-Nagy and Chrystopher L. Nehaniv. A bestiary of transformation semigroups for the holonomy decomposition. In D. Marc Kilgour, Herb Kunze, Roman Makarov, Roderick Melnik, and Xu Wang, editors, *Recent Developments in Mathematical, Statistical and Computational Sciences*, pages 37–46. Springer, 2021.
- [6] Zixuan Gao, Attila Egri-Nagy, and Chrystopher L. Nehaniv. Algebraic structure and complexity of games. In *Book of Abstracts of the 6th International Conference on Applied Mathematics, Modeling and Computational Science (AMMCS 2023)*, page 170, Waterloo, Canada, 2023. Published refereed abstract.
- [7] Zixuan Gao, Attila Egri-Nagy, and Chrystopher L. Nehaniv. Algebraic structure and complexity of games. In D. Marc Kilgour, Herb Kunze, Roman Makarov, Roderick Melnik, and Xu Wang, editors, *Recent Developments in Mathematical, Statistical and Computational Sciences (AMMCS 2023)*, Springer Proceedings in Mathematics & Statistics. Springer Nature, In press.

- [8] The GAP Group. GAP - Groups, Algorithms, and Programming. <http://www.gap-system.org>.
- [9] W. M. L. Holcombe. *Algebraic Automata Theory*. Cambridge University Press, 1982.
- [10] John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations (Proc. Internat. Sympos., Technion, Haifa, 1971)*. Academic Press, New York, 1971. See also preliminary version, Technical Report STAN-CS-71-190, Stanford University, Computer Science Department, January 1971.
- [11] Vincent Knight. Nashpy. <https://github.com/drvinceknight/Nashpy>.
- [12] Akos Lada. Section 11 - Normal-Form and Extensive-Form Games, 2012. Harvard University, <https://scholar.harvard.edu/files/alada/files/section11.pdf>.
- [13] Stuart Margolis and John Rhodes. Aperiodic flows on finite semigroups: Foundations and first examples. *arXiv preprint*, October 2024. <https://arxiv.org/abs/2410.06668>.
- [14] Stuart Margolis, John Rhodes, and Anne Schilling. Decidability of krohn-rhodes complexity for all finite semigroups and automata. *arXiv preprint*, June 2024. <https://arxiv.org/abs/2406.18477>.
- [15] R. Preston McAfee. *Introduction to Economic Analysis*. self published, 2006.
- [16] Roger B. Myerson. Mixed strategies. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*. Elsevier, 2001.
- [17] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.
- [18] C L Nehaniv, J Rhodes, A Egri-Nagy, P Dini, E Rothstein Morris, G Horváth, F Karimi, D Schreckling, and M Schilstra. Symmetry structure in discrete models of biochemical systems: Natural subsystems and the weak control hierarchy in a new model of computation driven by interactions. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 373, 2015.
- [19] MIT OpenCourseWare. Economic applications of game theory: Chapter 11 - subgame-perfect nash equilibrium, 2012. https://ocw.mit.edu/courses/14-12-economic-applications-of-game-theory-fall-2012/114453d67d6fd1a1d72ecedb29bd88ef_MIT14_12F12_chapter11.pdf.

- [20] MIT OpenCourseWare. Economic applications of game theory: Chapter 12 - repeated games, 2012. https://ocw.mit.edu/courses/14-12-economic-applications-of-game-theory-fall-2012/54f153ef7ed7800f367ce2b4021a18db_MIT14_12F12_chapter12.pdf.
- [21] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [22] Gambit Project. Gambit: The package for computation in game theory. <https://github.com/gambitproject/gambit>.
- [23] John Rhodes. *Applications of Automata Theory and Algebra: Via the Mathematical Theory of Complexity to Biology, Physics, Psychology, Philosophy, and Games*. Chrystopher L. Nehaniv (ed.), Foreword by Morris W. Hirsch World Scientific Press, 2009.
- [24] A. N. Siegel. *Combinatorial Game Theory*. American Mathematical Society, Providence, RI, 2013.
- [25] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3 edition, 2013.
- [26] The GAP Team. Semigroups - Chapter 12, 2024. <https://docs.gap-system.org/pkg/semigroups/doc/chap12.html>.
- [27] A. Egri-Nagy, J.D. Mitchell, C.L. Nehaniv. SgpDec - software package for hierarchical coordinatization of groups and semigroups, implemented in the GAP computer algebra system. <https://github.com/gap-packages/sgpdec>.
- [28] H. Zhang. Game theory. In H. Zhang, editor, *Models and Methods for Management Science*, Management for Professionals. Springer, Singapore, 2022.
- [29] Wei Zhang, Ming Li, and Jian Wang. Harsanyi-transformation oriented default risk prediction based on fa-xgboost. *Journal of Physics: Conference Series*, 1325(1), 2019.

APPENDICES

Appendix A

Complexity Lower Bound Algorithm Implementation in GAP

A.1 Check Essential Dependency

Recall that relation (2.1) determines the existence of essential dependency between sets X_1 and X_2 . The function `EssentialDependencyGroup` returns the group G_2 found,

$$G_2 \subseteq \langle \mathcal{I}_2 \rangle \cap (X_2)_{\text{per}}$$

If such a nontrivial group exists, an essential dependency exists between X_1 and X_2 ; otherwise, there is no such essential dependency.

Additionally, we provide a helper function `IdempotentsForSubset` that identifies a list of idempotents, given a skeleton of the original semigroup and a subset of interest.

```
InstallGlobalFunction(EssentialDependencyGroup, function(sk, x1,
    x2)
    local G1, CalX2, CalI2, skJ, e1, JGroup, S;
    S := TransSgp(sk);

    if IsSubsetBlist(x1, x2) then
        for e1 in IdempotentsForSubset(sk, x1) do
            G1 := SchutzenbergerGroup(HClass(S, e1));
```

```

CalX2 := Enumerate(Orb(G1, x2, OnFiniteSet));

# Get all idempotents with images that are members of CalX2
;
CalI2 := Concatenation(List(CalX2, Xt ->
    IdempotentsForSubset(sk, Xt)));

if IsEmpty(CalI2) then
    continue;
fi;

skJ := Skeleton(Semigroup(CalI2));

if x2 in ExtendedImageSet(skJ) then
    JGroup := PermutatorGroup(skJ, x2);

    # The stricter test is PermutatorGroup(sk, x2) = JGroup
    if Size(JGroup) > 1 then
        Assert( 1, PermutatorGroup(sk, x2) = JGroup,
            Concatenation("PermutatorGroup(sk, x2) <> JGroup\nx2
                = ",
                TrueValuePositionsBlistString(x2), "\nx1 = ",
                TrueValuePositionsBlistString(x1), "\n" ) );
        return JGroup;
    fi;
fi;
od;
fi;
return Group(());
end);

InstallGlobalFunction(IdempotentsForSubset,
function(sk, set)
    return Filtered(Idempotents(TransSgp(sk)),
        e -> OnFiniteSet(BaseSet(sk), e) = set);
end);

```

A.2 Lower Bound Algorithm for Krohn-Rhodes Complexity

The following implements the improved chain of sets algorithm. It takes the semigroup skeleton as the input and then finds the chain of sets that constitutes the maximum chain of essential dependency.

- Prints the chain of subsets to the console.
- Returns the length of the maximum chain of essential dependency, i.e. the lower bound.

```
InstallGlobalFunction(MaxChainOfEssentialDependency, function(sk)
  local o, Set2Index, Index2Set, maxChild, chainSizes, DFS,
    classesOnDepth,
    finalMax, idxFinalMax, pos;

  o := ForwardOrbit(sk);
  Set2Index := x -> Position(o, x);
  Index2Set := x -> o[x];

  maxChild := List(o, c -> -1); # Used for tracing the longest
    chain
  chainSizes := List(o, c -> -1);

  # Define the recursion function
  DFS := function(set_idx) # Input class is int
    local children, descendants, trivial, desc_idx, i,
      chainLengths, chainTemp, maxChainTemp;

    # Base Case: first check if the class is already in the list,
      return chain length if it is
    if chainSizes[set_idx] <> -1 then
      return chainSizes[set_idx];
    fi;
```

```

# Use immediate child *sets* below a set with set_idx for
  more memory-efficient DFS
children := Images(InclusionCoverRelation(sk), Index2Set(
  set_idx));
children := Filtered(children, c -> SizeBlist(c) > 1);      #
  Remove singleton sets
children := List(children, c -> Set2Index(c));

# If the class has no nontrivial permutator group, return 0
trivial := false;
if IsTrivial(PermutatorGroup(sk, Index2Set(set_idx))) then
  chainSizes[set_idx] := 0;
  trivial := true;
fi;

# Base Case: if no children, return 1
if IsEmpty(children) then
  if trivial then
    return 0;
  fi;
  chainSizes[set_idx] := 1;
  return 1;
fi;

# If there are children, find the chain length of each child
List(children, c -> DFS(c)); # DFS first

# If trivial, no need to check essential dependency
if trivial then
  return 0;
fi;

# This gives all the set IDs? lower than and equal to itself
descendents := Images(TransitiveClosureBinaryRelation(
  InclusionCoverRelation(sk)), Index2Set(set_idx));
descendents := Filtered(descendents, d -> SizeBlist(d) > 1);
  # Remove singleton sets
descendents := List(descendents, d -> Set2Index(d));

```

```

# If classes in descendent has essential dependency with the
  input class ID,
# add 1 to the chain length
maxChainTemp := 1;
for i in [1..Length(descendents)] do
  desc_idx := descendents[i];

  # Since all descendent classes have been searched, they all
    must have chain lengths  $\geq -1$ .
  # Ignore descendent classes with trivial permutator groups
  if chainSizes[desc_idx] = 0 then
    continue;
  fi;

  # If there is an essential dependency, add 1 to the chain
    length
  if not IsTrivial(EssentialDependencyGroup(sk,
    Index2Set(set_idx),
    Index2Set(desc_idx))) then
    chainTemp := chainSizes[desc_idx] + 1;
    if chainTemp > maxChainTemp then
      maxChainTemp := chainTemp;
      maxChild[set_idx] := desc_idx;
    fi;
  fi;
od;

# Find the max chain length
chainSizes[set_idx] := maxChainTemp;
return chainSizes[set_idx];
end;

# Start the recursion
List([1..Length(o)], i -> DFS(i));

# Find the maximum chain length
finalMax := Maximum(chainSizes);

```

```

idxFinalMax := Position(chainSizes, finalMax);
pos := idxFinalMax;

# Trace the chain, and print the chain of subsets
Print("Maximum Chain Found:");
while maxChild[pos] <> -1 do
    Print( TrueValuePositionsBlistString( Index2Set(pos)), " -> ");
    pos := maxChild[pos];
od;
Print( TrueValuePositionsBlistString( Index2Set(pos)), "\n");

return Maximum(chainSizes);
end);

```

Appendix B

Counterexample Code for Chain of Levels Algorithm

The following is the GAP testing code for the Becks++ semigroup. Note that the testing makes use of the code in Appendix A.1 code, which disproves Assertion 1. Additionally, the last command uses the correct lower bound algorithm in Appendix A.1 and prints the actual maximum chain of essential dependencies.

```
gap> beckspp := [  
> Transformation([1,2,3,1,1,1]),  
> Transformation([4,4,4,5,4,6]),  
> Transformation([4,4,4,5,6,4]),  
> Transformation([4,4,4,4,5,5]),  
>  
> Transformation([4,4,4,1,2,3]),  
> Transformation([2,3,1,4,4,4]),  
> Transformation([2,1,3,4,2,2]),  
> Transformation([1,2,2,2,2,2])];;# makes H({1,2,3}) nontrivial  
gap> BECKSPP := Semigroup(beckspp);;  
gap> skel := Skeleton(BECKSPP);;  
gap> EssentialDependencyGroup(skel, FiniteSet([1,2,3], 6),  
    FiniteSet([1,2], 6));  
Group([ (1,2) ])  
gap> EssentialDependencyGroup(skel, FiniteSet([1,2,3,4], 6),  
    FiniteSet([1,2,3], 6));  
Group(( ))
```

```

gap> EssentialDependencyGroup(skel, FiniteSet([4,5,6], 6),
    FiniteSet([4,5], 6));
Group([ (4,5) ])
gap> EssentialDependencyGroup(skel, FiniteSet([1,2,3,4], 6),
    FiniteSet([4,5], 6));
Group(())
gap> MaxChainOfEssentialDependency(skel);
Maximum Chain Found: {1,2,3} -> {1,2}
2

```