RICE UNIVERSITY

Algorithmic Analysis of Regular Repeated Games

by

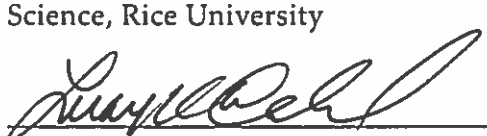
Suguman Bansal

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

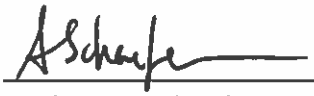
Master of Science

Approved, Thesis Committee:

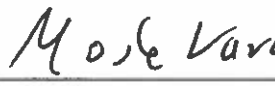
Swarat Chaudhuri, Chair
Associate Professor of Computer
Science, Rice University



Luay Nakhleh
Professor of Computer Science, and of
BioSciences, Rice University



Andrew J. Schaefer
Noah Harding Chair and Professor of
Computational and Applied
Mathematics, Rice University



Moshe Y. Vardi
Karen Ostrum George Distinguished
Service Professor in Computational
Engineering, Rice University

Houston, Texas
August, 2016

ABSTRACT

Algorithmic Analysis of Regular Repeated Games

by

Suguman Bansal

The problem on computing rational behaviors in multi-agent systems with selfish agents (Games) has become paramount for the analysis of such systems. *Nash equilibria* is one of the most central notions of rational behavior. While the problem of computing Nash equilibria in simple games is well understood, the same cannot be said for more complex games. *Repeated games* are one such class of games.

In this thesis, we introduce *regular repeated games* as a model for repeated games with bounded rationality. In regular repeated games, agent strategies are given by weighted (discounted-sum aggregate), non-deterministic Büchi transducers. We design an algorithm ComputeNash to compute all Nash equilibria in a regular repeated game. The crux of the algorithm lies in determining if a strategy profile is in Nash equilibria or not. For this it is necessary to compare the discounted sum on one infinite execution with that one other executions. Such relational reasoning has not been studies in the literature before. To this end, we introduce the concept of an *ω-regular comparators*.

We demonstrate promise of our approach via experimental analysis on case studies: Iterated Prisoner's Dilemma, repeated auctions, and a model of the Bitcoin protocol.

Dedicated to my grandparents,

Srimati Kasturi Devi
and
Late Sri Nidhi Agarwal

To Mummy, my guiding light during her life and after

Acknowledgements

I can no other answer make, but, thanks, and thanks. ~William Shakespeare

First of all, I would like to thank my advisor and mentor, Prof. Swarat Chaudhuri, for guiding me through every step in the completion of this thesis. The raw mathematician in me would never have been evolving into a budding computer scientist had it not been for your constant encouragement to explore unfamiliar territories. Thank you for your unwavering support, confidence and patience with me during failure and moments of despair.

I would like to thank Prof. Moshe Y. Vardi for his astute comments on early drafts of this thesis. I have had, and look forward to continue to have the pleasure to learn from the shadows of your experience. But most importantly, I would like to thank you for your timely mentorship and advice.

I would also like to thank Prof. Luay Nakhleh, and Prof. Andrew J. Schaefer for agreeing to be on my committee, and for evaluating this thesis.

No journey is complete without the camaraderie of amazing friends. And mine has been no exception. Thank you, Dror and Kuldeep, for I cannot recall any celebration, any discussion, any rant, any struggle I may have had in the past two years without the company of you guys. Here it is, raising a toast to many, many more coffee breaks and quick dinners!

Thank you, Milind and Shams, for still responding to my novice PhD student queries. I am greatful for the friendship of Aditya, Afsaneh, Arko, Harsh, Jack, Karthik, Prasanth, Risa, Rohan, Ronnie, Ryan, Sarah, Simba, Sourav, and many more for bringing more colors than the walls of Duncan Hall into my graduate student experience.

Houston would have been so lack-lusture had it not been for the friendship that germinated over the sole intent of finishing *that rice* - Sushma; the (un)fortunate taster of all my culinary expeditions - Sapna; and Rakesh for sharing the love for *Biryani* and desserts.

Dhananjay, Siddharth, Siddhesh, and Visu, thank you for always being just a phone call away since our undergraduate days. Saheli, thank you for continuing to inspire me with your unbeatable spirit since our good old school days.

I owe my deepest gratitude to my parents and brother. I cannot express in words how much comfort I get in knowing that across the oceans and continents lies a home that unconditionally roots for me.

This is a bitter-sweet moment for us. This thesis has been the most difficult and most important reward I have gotten so far. Even the smallest prize I received made you revel for days. Now we can only imagine how proud of me you must be today … We miss you, Mummy.

Contents

Illustrations

List of Algorithms

Chapter 1

Introduction

Most real life interactions can be studied as multi-agent systems in which each agent has its own objectives. Such systems range from simple two-agent interactions such as a game of tic-tac-toe to massive online protocols such as the auction protocols adopted by cyber giants such as Google and eBay. The objective of agents in these systems may differ. In a game of tic-tac-toe, the objective of agents may be to win the game, whereas in an auction it may be to win the auction at the lowest possible bid. In other multi-agent systems such as the market place where firms produce the same goods, agents (firms) interact by assigning a price to their goods. In this case, the agent's objective could be for all firms to assign the same price to the goods, in order to promote mutual co-operation in the market place.

An agent is said to be *rational* if it interacts in a manner to fulfill its objective. The computation of rational behavior of agents enables us to better analyze properties of these systems. These systems and their rational behaviors are modeled and analyzed under the field of *Game Theory* [52]. Modern game theory has proliferated with diverse applications in economics [31, 33, 42, 51, 66], social sciences [18, 45, 46], biology [9, 24, 39, 67], etc. Unsurprisingly, the computation of rational behaviors in multi-agent systems with agents with objectives has garnered a lot of interest in utilitarian research from diverse communities.

One concrete instance of computation of rational behaviors to analyze such systems arises in the context of the *Bitcoin protocol*. Bitcoins are one of the most widely used on-line currency in today's time. The protocol by which Bitcoins are *mined* (minted) is called the Bitcoin protocol. Agents (miners) of the protocol re-

ceive rewards by mining more Bitcoins The objective of agents is to maximize their rewards from the protocol. Earlier it was believed that miners receive the most reward when they mine by abiding by the policies of the protocol. Since the protocol rewarded miners proportional to the number of Bitcoins they mined, it was believed that rational agents would abide by the policies of the protocol. However, a surprising result by Eyal and Sirer disproved this misconception[32]. They proved *via rigorous manual computation and analysis* that there exists a rational behavior in ehich agents receive greater rewards by deviating from the protocol policies. Hence, rational agents would have an incentive to deviate from the protocol policies, which is an undesired trait for the Bitcoin protocol.

The above example illustrates that erroneous system design is common, and can have tremendous harm on all participating agents. This point reinstates the importance of computation of rational behaviors. However, as the number of agents and the complexity of such systems increases, the difficulty of the manual computation of rational behaviors also increases. This necessitates the need for algorithmic development of techniques for computation of rational behaviors in complex multi-agent systems. The problem of computation of rational behaviors has been studied under *algorithmic game theory* [50].

This thesis is a step towards developing a framework and algorithmic techniques for computation of rational behaviors in complex multi-agent systems. Section 1.1 provides a broad overview of the game-theoretic approach for analysis of complex multi-agent systems. Section 1.2 briefly describes the contributions of this thesis. The structure of the thesis is provided in Section 1.3.

1.1 Equilibria computation in repeated games

In game theory, multi-agent systems with objective driven agents are modeled as *games*. A game consists of multiple agents. These agents interact with each other synchronously, and receive rewards from each interaction. Consider the classic

	Cooperate (C)	Defect (D)
Cooperate (C)	(3,3)	(0,4)
Defect (D)	(4,0)	(1,1)

Figure 1.1 : Reward table for Prisoner's Dilemma (PD)

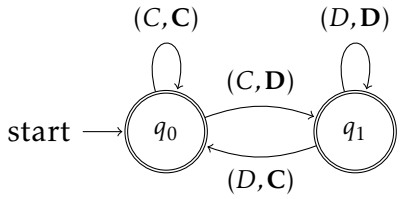


Figure 1.2 : TFT(Tit-For-Tat), See Chapter 3

game of Prisoner's Dilemma [52]. In this system agents interact by either cooperating with the other agent (action C) or by defecting on the other agent (action D). Agents receive rewards according to Table 1.1.

Rational behavior in games are studied as *solution concepts*. Various solution concepts have been studied for games. One such solution concept is that of Nash equilibria [47]. Intuitively, Nash equilibrium corresponds to a global scenario in a system in which no agent can receive a greater reward by *unilaterally* changing its decision. In the Prisoner's Dilemma game as shown in Table 1.1, (D, D) is said to be in Nash equilibria since from here no agent can receive a greater reward by changing its decision. Since its inception in the 1950s, Nash equilibrium has been a fundamental notion of rationality. Nash equilibria has found widespread applications in network analysis, analysis of large economic systems such as auctions and non-cooperative markets, biology etc [38, 56, 65, 67].

Games such as the Prisoner's Dilemma are called *one-shot games*, since agents interact only once. Celebrated results by Chen, Deng and Teng show that computation of a single Nash equilibrium in one-shot games is PPAD-Complete [22, 23]. Other notable works in the computation of Nash equilibria in one-shot games arrived from Conitzer and Sandholm[25], and Daskalakis, Goldberg and Papadimitriou [27]. In short, our understanding of the hardness and algorithms for computing Nash equilibrium in one-shot games has been well established.

However, most real life interactions are rarely one-off. The more typical scenario are ones in which agents interact repeatedly. In this thesis we focus on repeated games with an infinite number of interactions. Agents receive rewards for each round of interaction. The total reward of any agent is computed by accumulating the rewards received in all rounds of interaction using the discounted-sum aggregate function. These are called *infinitely repeated games*. For simplicity, we will refer to infinitely repeated games as *repeated games*. Intuitively, a repeated game occurs when a one-shot game occurs infinitely many times. An agent's behavior in any given round is determined by previous interactions. For all practical purposes, it is justified to assume that agent's behavior will depend on a bounded amount of information from previous interactions. Repeated games with bounded information are called *repeated games with bounded rationality* [38, 59].

Finite state machines have been used to model repeated games with bounded rationality [2, 5, 8, 40, 54, 57]. In this model, states capture intuitions of the bounded amount of information.

As an example, consider a repeated version of the prisoner's dilemma game. For this repeated version, an agent begins the game with co-operating with the other agent, and imitates the other agent's action from the previous round in all subsequent rounds. This strategy is called the *tit-for-tat* strategy. Note that tit-for-tat is a strategy with bounded rationality since an agent's behavior depends on the immediately previous interaction only. The finite state machine denoting this strategy is depicted in Figure 1.2. Each transition denotes one round of interaction in the game. Label $(a, \mathbf{b})$ on transitions denotes that the first and second agents take actions are a and $\mathbf{b}$ respectively. Rewards of agents for each round are as shown in Table 1.1. Furthermore, state q_0 and q_1 denote the states from which the first agent always plays C and D respectively. Note that q_0 and q_1 also denote the states at which the second agent has performed C and D respectively in the previous round. Hence, the states are capturing intuitions about the bounded amount of

information for the tit-for-tat strategy.

Unlike the case of one-shot games, the problem of Nash equilibrium computation in repeated games with bounded rationality has not been fully resolved. Of existing work on the topic of equilibria computation in repeated games with bounded rationality, seminal work by Abreu, Pearce, and Stacchetti [1], and its followups [26], considered the problem of computing all or some equilibrium rewards rather than the equilibria themselves. The problem of efficiently computing a *single* equilibrium strategy in infinitely repeated games has been studied before [4, 35, 37]. To the best of our knowledge, these approaches cannot be extended to compute the set of all Nash equilibria in a repeated game. There are also a few approaches to computing representations of *approximations* to the set of equilibria [11, 12, 17]. Except for work on finding one equilibrium, most other approaches do not guarantee crisp complexity bounds for exactly computing all Nash equilibria in a repeated game. This thesis addresses this caveat: providing algorithmic techniques with crisp complexity bounds for the computation of all Nash equilibria in repeated games with bounded rationality.

1.2 Contributions

This thesis works towards identifying a class of games on which we develop algorithmic techniques to compute Nash equilibria in repeated games. The focus of this thesis is on simultaneous, non-zero sum games, with infinite repetition over bounded rationality.

First, we identify a new class of repeated games, that we call *regular repeated games*, on which the problem of computation of Nash equilibria can be solved. Regular repeated games are a finite-state machine based model for games in which agent strategies are encoded by finite transducers. Agents in regular repeated games have a finite number of strategies, each given by a weighted, non-deterministic Büchi transducer, where the input on each transition are the actions of all other

agents and the output is the agents' action. Weights on transitions denote agent rewards, and the reward from each execution in the strategy transducer is given by the discounted sum of rewards on each transition. In a game execution, each agent picks its' strategy. These strategies compose synchronously on agent actions. This is called a strategy profile, which is a finite state weighted Büchi automata, on which weight tuples on transitions denote the reward of each agent. As earlier, agent reward on executions are computed using the discounted sum.

Intuitively, regular repeated games can be treated as an extension of a well established model for repeated games proposed by Rubinstein in 1986 [57]. The key difference between regular repeated games and previous finite state based models, including the Rubinstein model for repeated games [2, 5, 8, 40, 54, 57], is that regular repeated games are defined such that they can handle non-determinism (as a model of uncertainty) in strategies. Hence, in addition to being amenable to algorithms for Nash equilibria computation, regular repeated games are also more expressible that previous models of repeated games. In fact, in this thesis, we will show that complex systems such as the Bitcoin protocol can be modeled in this framework.

The main contribution of this thesis is an algorithm for the computation of all Nash equilibria in a regular repeated game. As was mentioned in the previous section, earlier approaches pertaining to Nash equilibria computation have dealt with determining the equilibria rewards, computation of a single equilibria, or determining an approximation to the set of equilibria. The algorithm presented in this thesis computes all Nash equilibria, and has a crisp complexity bound: the algorithm is polynomial in the size of the game when all player strategies are deterministic, and is exponential in the size of the game otherwise (See Section 3.1.3).

The crux of the algorithm lies in determining whether an agent strategy is in Nash equilibria or not. For this, it is necessary to compare the rewards received by agents on executions. The technical problem reduces to the following: Given

two weighted Büchi automata, where the aggregate of executions is given by the discounted sum. How does one compare the aggregate on each execution of one automata with executions in the other? To the best of our knowledge, there is no existing approach of this sort of relational reasoning in weighted automata.

This thesis develops a novel automata theoretic approach to enable such relational reasoning: *ω-regular comparator*. An ω-regular comparator is a Büchi automaton that accepts a pair of infinitely long number sequences iff the aggregate (e.g. discounted sum) of the sequences satisfy a given equality/inequality relationship. In other words, let A and B be two infinite number sequence. Then the ω-regular comparator for the discounted sum aggregate function accepts the pair (A, B) iff the discounted sum of A is greater than the discounted sum of B. Hence ω-regular comparators reduce the problem of quantitative comparison into a qualitative problem of acceptance by an automaton. In this thesis, we present an ω-regular comparator for the discounted sum aggregate function, and provide an exploratory discussion on the possibility of an ω-regular comparator for the limit average aggregate function.

To demonstrate the practical utility of our approach, we built a prototype tool of the algorithm, and computed the equilibria set for regular repeated game models of the classical Iterated Prisoner's Dilemma (IPD) [38], repeated auctions and the Bitcoin protocol. The resulting equilibria sets are consistent with classical results on the IPD and repeated auctions. In the case of the Bitcoin protocol, we proved that the protocol is not incentive compatible. This is similar to Eyal and Sirer's result![32] expect that there the result was obtained via manual proofs while ours was obtained via algorithmic analysis.

1.3 Organization

The rest of the thesis is organized as follows:

Chapter 2 discusses the necessary background in automata theory and game

theory required for the thesis.

Chapter 3 provides the formal definitions of regular repeated games, and solution concepts in them.

Chapter 4 describes ComputeNash, our algorithm for computing all Nash equilibria in regular repeated games, and our results from experimental analysis on various repeated games.

Chapter 5 introduces ω-regular comparator in detail.

Lastly, this thesis concludes with a summary and a discussion on some future directions in Chapter 6.

Chapter 2

Preliminaries

This chapter introduces the necessary definitions and notations from game theory and automata theory required for the rest of this thesis.

2.1 Game-theoretic concepts

An interaction between multiple agents are modeled as *games*. Each agent receives a quantitative reward, called *payoff*, from each interaction that depends on the *actions* of all agents. Payoffs capture the gains or losses of an agent in each interaction. Gains and losses correspond to higher and lower payoffs respectively. Agents are said to be *rational* if they act in a manner that *optimizes* their payoff.

Game theory [52] is the mathematical study of games. The central question of game theory pertains to the behavior of rational agents.

2.1.1 Repeated Games

We are interested in games of a specific kind: *infinitely repeated games with bounded rationality* [38]. Infinitely repeated games are simultaneous, non-zero sum games. In these games, agents interact with each other repeatedly for an infinite number of rounds. In each round of interaction, agents take action simultaneously: in each round of interaction, every agent is unaware of the actions that other agents will take. However, agents are aware of a bounded amount of information from actions taken by all agents from earlier rounds of interactions. The governing principle by which an agent decides on what action to play in each round is said to be an agent's *strategy*. Hence, agents choose their actions for any given round based on

its strategy. Agents receive a quantitative reward, called *payoff*, from each round of interaction. The *payoff* of an agent from a game is computed by accumulating the rewards from each round of accumulation using the discounted sum aggregate function. The discounted sum of a sequence of rewards $R = \{r_0, r_1, \dots\}$ with discount factor $d > 1$ is given by $DS(A, d) = r_0 + \frac{r_1}{d} + \frac{r_2}{d^2} + \dots$.

For simplicity, we will refer to infinitely repeated games with bounded rationality as *repeated games* in the rest of this thesis.

The motivation for repeated games arises from real life interactions. As a concrete example, consider a competitive market place consisting of two firms that produce an identical goods. Firms interact with each other by deciding on a price for the goods. Actions for each firms are (1) to weed-out competition, or (2) to mutually co-operate to co-exist in the market place. An example of a firm strategy is to *Tit-for-Tat* i.e to respond by always replicating the other firm's action from the previous round. Note that in each round of an interaction, firms can refer to the action of the other firm from the previous round only, hence agents decide on an action based on a bounded amount on information from previous rounds.

2.1.2 Solution concepts

Rational behavior in games are studied as solution concepts [38, 51]. This thesis discusses two solution concepts: Nash equilibria and best response strategy.

Best response strategy A strategy S for an agent is said to be the best response to a fixed set of strategies taken by all other agents if the agent receives maximum payoff with strategy S as compared to all other strategies of the agent.

The best response strategy is a very local notion of rationality, one that corresponds to the strategy that maximizes a single agent's rewards from a game.

Nash equilibria Nash equilibria is a global notion of rationality that corresponds to strategies taken by all agents.

A tuple of strategies $(S_1, \ldots, S_k)$, S_i corresponds to the strategy of the i-th agent, is said to be in Nash equilibria if no agent receives a greater payoff from the game by unilaterally changing its strategy.

2.2 Automata-theoretic concepts

This section provides definitions and notations for the finite-state machines used in this thesis.

2.2.1 Finite automata

A finite automaton, denoted by $\mathcal{A}$, is a tuple $(S, \Sigma, \delta, Init, \mathcal{F})$ where S is a finite set of states, $Init \subseteq S$ is a non-empty set of initial states, Σ is a finite alphabet, $\delta \subseteq S \times \Sigma \times S$ is the transition relation, and $\mathcal{F} \subseteq S$ is a non-empty set of accepting states.

An automaton is called *deterministic* if for all states s, and all symbols $a \in \Sigma$, $|\{s' | (s, a, s') \in \delta\}| \leq 1$. Otherwise, it is called *non-deterministic*.

Let Σ^* be the set of finite words over Σ. For $w = w_0 w_1 \ldots w_n \in \Sigma^*$, a *run* ρ of w in $\mathcal{A}$ is a sequence of transitions $\tau_0 \tau_1 \ldots \tau_n$ such that there is a sequence of states $s_\rho = s_0 s_1 \ldots s_{n+1}$ satisfying: (1) $s_0 = Init$, and (2) $\tau_i = (s_i, w_i, s_{i+1})$ for all i.

We say w has an *accepting run* in $\mathcal{A}$ if there exists a run ρ of w such that the last state of s_ρ is a final state of $\mathcal{A}$, i.e. $s_{n+1} \in \mathcal{F}$. The automaton $\mathcal{A}$ *accepts* a word w if w has an accepting run in $\mathcal{A}$.

Regular languages are known to be closed under union, intersection, and complementation over Σ^*. Languages accepted by deterministic automata are called *regular languages*.

2.2.2 Büchi automata

A *Büchi automaton* [62] is similar to traditional finite automata, except it operates on infinite words. While automata on finite words accept a word by terminating at an accepting state, a Büchi automaton accepts a word if it visits the set of accepting

states infinitely often while reading the word. Formally, Büchi automata are also defined by $(S, \Sigma, \delta, Init, \mathcal{F})$.

Let Σ^ω be the set of infinite words over Σ (similar notation is used for other alphabets). For $w = w_0 w_1 \cdots \in \Sigma^\omega$, a *run* ρ of w in $\mathcal{A}$ is a sequence of transitions $\tau_0 \tau_1 \ldots$ such that there is a sequence of states $s_\rho = s_0 s_1 \ldots$ satisfying: (1) $s_0 = Init$, and (2) $\tau_i = (s_i, w_i, s_{i+1})$ for all i. Note, each sequence in a Büchi automaton is infinite.

Let $inf(\rho)$ be the set of states that occur infinitely often in s_ρ. We say w has an *accepting run* in $\mathcal{A}$ if there exists a run ρ of w such that $inf(\rho) \cap \mathcal{F} \neq \emptyset$. The automaton $\mathcal{A}$ *accepts* a word w if w has an accepting run in $\mathcal{A}$.

Büchi automata are also known to be closed under set-theoretic union, intersection, and complementation over Σ^ω. Languages accepted by these automata are called *ω-regular languages*.

2.2.3 Weighted Büchi transducer

A *weighted Büchi transducer* is similar to a Büchi automaton, in that its inputs are infinite words. However, on every accepting run, it produces an infinite word as *output*, and receives an infinite stream of *payoffs*. These payoffs are aggregated into an *aggregate payoff*.

Formally, a *(weighted, Büchi) transducer* is a tuple $\mathcal{T} = (S, \Sigma, \Gamma, \delta, Init, \mathcal{F})$ where S is a finite set of states, Σ is an *input alphabet*, Γ is an *output alphabet*, $\delta \subseteq S \times \Sigma \times \Gamma \times \mathbb{Q} \times S$ is a *transition relation*, $Init \subseteq S$ is a set of *initial state*, $\mathcal{F} \subseteq S$ is the set of *accepting states*, and $f_\mathcal{T}$ is the aggregate function.

A transition (s, a, b, m, s') in $\mathcal{T}$ is said to have *input* a, *output* b, and *payoff* m. The transducer is *deterministic* if for all states s, inputs a, outputs b, $|\{(m, s') : (s, a, b, m, s') \in \delta \text{ for some } s'\}| \leq 1$. Otherwise, it is *nondeterministic*.

For a word $w = w_0 w_1 w_2 \cdots \in (\Sigma, \Gamma)^\omega$, a *run* ρ of w in $\mathcal{T}$ is a sequence of transitions $\tau_0 \tau_1 \tau_2 \ldots$ such that there is a (unique) sequence of states $s_0 s_1 s_2 \ldots$ satisfying: (1)

$s_0 = Init$, and (2) $\tau_i = (s_i, w_i, y_i, p_i, s_{i+1})$ for all i. The definitions of accepting runs, accepted words, and languages are as for Büchi automata. The sequences $y_0 y_1 y_2 \ldots$ and $\pi = p_0 p_1 p_2 \ldots$ are known as the *output sequence* and the *payoff sequence* for ρ, respectively.

Let d be a fixed rational number satisfying $d > 1$. The *discounted sum* of any sequence $A = a_0 a_1 a_2 \ldots$ of rationals is defined as $DS(A, d) = \Sigma_{j=0}^{\infty}(a_j / d^j)$. The *aggregate payoff* for the run ρ is the discounted sum $DS(\pi, d)$. Discounted sum is used to compute the payoff of agents in a repeated game.

Note Let $\mathcal{A}$ denote a finite state machine. We write $w \in \mathcal{A}$ to denote that w is a word accepted by $\mathcal{A}$, and $\rho \in \mathcal{A}$ to denote that ρ is an accepting run in $\mathcal{A}$.

Chapter 3

Regular Repeated Games

This chapter introduces our model, regular repeated games, for infinitely repeated games with bounded rationality. It is worthwhile to note that finite-state machine are one of the oldest models for repeated games with bounded rationality. games [2, 5, 8, 10, 40, 48, 49, 54, 57]. Regular repeated games is also a finite state machine based model for repeated games. Section 3.1 expands on regular repeated games in detail. In Section 3.1.1 we will note that regular repeated games are indeed an extension of the finite state based model proposed by Rubinstein in his seminal work on analysis of repeated games [57]. This section will motivate the need for our extension by proving that regular repeated games are strictly more expressive that the Rubinstein model. Later, Section 3.1.2 defines two solution concepts for repeated regular games: Nash equilibria and best response strategy. Lastly, Section 3.1.3 provides a discussion on the size of regular repeated games. The chapter concludes with its summary in Section 3.2.

3.1 Regular repeated games

A regular repeated game consist of a finite fixed finite number of agents, each with a finite set of strategies given by weighted Büchi transducers. The actions of an agent form the outputs of these transducers; the inputs correspond to tuples of actions taken by other agents.

Formally, a *regular repeated game* $\mathcal{G}$ is given by a set of k agents, where the i-th agent $\mathcal{P}_i$ consists of a finite set $Action(i)$ of *actions*, and a finite set $Strategy(i)$ of *strategies*. A strategy $\mathcal{M}$ for $\mathcal{P}_i$ is a weighted Büchi transducer whose output alpha-

bet is $Action(i)$, and whose input alphabet is the Cartesian product $\bigtimes_{j \neq i} Action(j)$ of action sets of other agents. Elements $(a)_{-i}$ of this input alphabet are known as *environment actions* w.r.t agent $\mathcal{P}_i$. The game $\mathcal{G}$ is *deterministic* if all strategies of all agents are deterministic, and *nondeterministic* otherwise.

Intuitively, accepting runs of a strategy $\mathcal{M}$ offer an agent-level view of executions of the game. Each transition denotes one round of a game. Specifically, consider a transition $(q, (a)_{-i}, a_i, p, q')$ that appears at the j-th position of an accepting run of $\mathcal{M}$. This means that there is a *possible* execution of the game in which: (i) the i-th agent is at state q immediately before the j-th round of the game; (ii) in this round, the i-th agent and its environment synchronously perform the actions a_i and $(a)_{-i}$, respectively; (iii) the concurrent action leads agent $\mathcal{P}_i$ to transition to state q' at the end of this round, and receive payoff p.

Recall that we use the discounted sum aggregate function. Intuitively, such a discounted sum follows the standard game-theoretic assumption that the significance of payoffs received in each round of interaction tapers with time.

Nondeterminism in strategies is used to capture the incompleteness of our knowledge about the dynamics of the game. In nondeterministic strategies, a joint action by an agent and its environment can cause the agent to transition one of several distinct states, or receive one of several distinct payoffs. (Examples of each kind of nondeterminism are demonstrated in Section 4.3.) Unlike in probabilistic models of uncertain behavior, we do *not* have a probability distribution on the transitions.

Strategy profiles and executions Now we define the semantics of interactions between agents $\mathcal{P}_1, \ldots, \mathcal{P}_k$ that constitute a game $\mathcal{G}$.

A *strategy profile* $\overline{\mathcal{M}}$ of $\mathcal{G}$ is a tuple of strategies $(\mathcal{M}^1, \ldots, \mathcal{M}^k)$, where $\mathcal{M}^i \in Strategy(i)$ for each $\mathcal{P}_i$. Intuitively, $\overline{\mathcal{M}}$ captures a scenario in which $\mathcal{P}_i$ follows the strategy $\mathcal{M}^i$. A *word* of strategy profile $\overline{\mathcal{M}}$, denoted by $\overline{w} = (w_1, \ldots, w_k) \in$

$(\bigtimes_{i=1}^{k} Action(i))^{\omega}$, is such that for each agent $\mathcal{P}_i$, $\overline{w}_i = ((w_1, \ldots w_{i-1}, w_{i+1}, \ldots w_k), w_i) \in \mathcal{M}^i$. We use $\overline{w} \in \overline{\mathcal{M}}$ to denote $\overline{w}$ is a word of $\overline{\mathcal{M}}$. Each $\overline{w}$ is assigned to *payoff sequences*, $\mathbf{P}(\overline{w})$. Payoff sequence $(p_1, \ldots, p_k) \in \mathbf{P}(\overline{w})$ iff for each $\mathcal{P}_i$, there exists an accepting run of $\overline{w}_i$ with payoff sequence p_i in $\mathcal{M}^i$. A run $\overline{\rho}$ of strategy profile $\overline{\mathcal{M}}$ is defined as the tuple $(\overline{w}, \overline{p})$ where $\overline{w} \in \overline{\mathcal{M}}$ and $\overline{p} \in \mathbf{P}(\overline{w})$. We denote the payoff of $\mathcal{P}_i$ along $\overline{\rho}$ i.e. p_i by $\mathbf{P}^i(\overline{\rho})$. As earlier, we use $\overline{\rho} \in \overline{\mathcal{M}}$ to denote that run $\overline{\rho}$ is present in $\overline{\mathcal{M}}$.

The payoff for agent $\mathcal{P}_i$ on a run ρ is given by the discounted sum of the payoffs received by agent $\mathcal{P}_i$ on the run ρ. Hence, for discount factor $d > 1$ the payoff received by agent $\mathcal{P}_i$ on a run ρ is given by $DS(\mathbf{P}^i(\rho), d)$. The payoff for agent $\mathcal{P}_i$ for a given word $\overline{w}$ is given by $max\{DS(\mathbf{P}^i(\rho), d) | \rho \text{ is a run for word } \overline{w}\}$.

Each strategy profile can be represented by a Büchi automaton $Execs(\overline{\mathcal{M}})$, and a payoff relation $\mathbf{P}$ between transitions of $Execs(\overline{\mathcal{M}})$ and $\mathbb{Q}^k$ s.t. $(\overline{w}, \overline{p}) \in \overline{\mathcal{M}}$ iff $\overline{w} \in Execs(\overline{\mathcal{M}})$ and $\overline{p}$ is a payoff sequence of $\overline{w}$ in $Execs(\overline{\mathcal{M}})$. The construction of a strategy profile automata from its component strategies is shown below:

Strategy profile automata We construct a Büchi automaton $Execs(\overline{\mathcal{M}})$ for strategy profile $\overline{\mathcal{M}} = (\mathcal{M}^1, \ldots, \mathcal{M}^k)$ and a payoff relation $\mathbf{P}$ over transitions in the automaton such word $\overline{w} \in Execs(\overline{\mathcal{M}})$ has payoff sequence $\overline{p}$ iff $(\overline{w}, \overline{p})$ is a run in $\overline{\mathcal{M}}$.

Let $[k]$ denote $\{1, \ldots, k\}$. Suppose $\mathcal{M}^i = (S_i, \Sigma_i, \Gamma_i, \delta_i, Init_i, \mathcal{F}_i, d_i)$ for each i. Then $Execs(\overline{\mathcal{M}})$ is the Büchi automaton $(\overline{S}, \overline{\Gamma}, \overline{\delta}, \overline{Init}, \overline{\mathcal{F}})$ and $\mathbf{P} \subseteq \overline{\delta} \times (\mathbb{Q})^k$ is the payoff relation, where:

- $\overline{S} = S_1 \times \cdots \times S_k \times [k]$
- $\overline{\Gamma} = \bigtimes_{i=1}^{k} \Gamma_i$. Let, $\overline{a} = (a_1, \ldots, a_k)$, and $\overline{a}_{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots a_k)$.
- $\overline{Init} = Init_1 \times \cdots \times Init_k \times \{1\}$
- $\overline{\mathcal{F}} = \mathcal{F}_1 \times S_2 \cdots \times S_k \times \{1\}$

- Transition relation $\overline{\delta}$ and the payoff relation $\mathbf{P} \subseteq \overline{\delta} \times (\mathbb{Q})^k$ are defined as follows:

  - $\tau = ((s_1,\ldots,s_k,m),\overline{a},(t_1,\ldots,t_k,m)) \in \delta$ iff $s_m \notin \mathcal{F}_m$ and for all $i \in [k]$, $\tau_i = (s_i,\overline{a}_{-i},a_i,p_i,t_i) \in \delta_i$.

    All $\overline{p} = (p_1,\ldots,p_k)$ formed from such τ_i-s are contained in $\mathbf{P}(\tau)$.

  - $\tau = ((s_1,\ldots,s_k,m),\overline{a},(t_1,\ldots,t_k,f(m))) \in \overline{\delta}$ iff $s_m \in \mathcal{F}_m$ and for all $i \in [k]$, $(s_i,\overline{a}_{-i},a_i,p_i t_j) \in \delta_j$, where $f(m) = m+1$ if $m < k$, and $f(m) = 1$ otherwise.

    All $\overline{p} = (p_1,\ldots,p_k)$ formed from such τ_i-s are contained in $\mathbf{P}(\tau)$.

The next theorem proves that the automata constructed above is indeed equivalent to the corresponding strategy profile.

Theorem 3.1
Automaton $Execs(\overline{\mathcal{M}})$ *and payoff relation* $\boldsymbol{P} \subseteq \overline{\delta} \times (\mathbb{Q})^k$ *is such that* $\overline{w} \in Execs(\overline{\mathcal{M}})$ *and* $\overline{p} \in \boldsymbol{P}(\overline{w})$ *iff* $(\overline{w},\overline{p})$ *is a run in* $\overline{\mathcal{M}}$.

Proof 3.1 We prove the two directions of the implication separately.

First, we prove that $\overline{w} \in Execs(\overline{\mathcal{M}})$ with $\overline{p} \in \mathbf{P}(\overline{w}) \implies (\overline{w},\overline{p}) \in \overline{\mathcal{M}}$.

Let $\overline{w}$ be a word in $Execs(\overline{\mathcal{M}})$ with payoff sequence $\overline{p}$. By looking at its transition sequence we will reason that $(\overline{w},\overline{p}) \in \overline{\mathcal{M}}$.

Let its transition sequence be given by $\tau_0\tau_1\ldots$ where each $\tau_i = ((s_1^i,\ldots,s_k^i,m_i), \overline{a}, (s_1^{i+1},\ldots,s_k^{i+1},m_i))$. Since this is an accepting run, it must be the case that final states in the automaton are accepted infinitely often. Consider the state sequence for $\overline{w}$, $(s_1^0,\ldots,s_k^0,1)(s_1^1,\ldots,s_k^1,m_1)(s_1^2,\ldots,s_k^2,m_2)\ldots$. Since, the sequence is accepting, $(f^i,\ldots,1)$ is visited infinitely often, where $f^i \in \mathcal{F}_1$. But from construction of the automaton, we know that if $(f_1^i,\ldots,1)$ is visited in the i-th round, then then state sequence in the next round is $(s_1^{i+1},\ldots,2)$. The only way, it can return to a state of the form $(f^i,\ldots,1)$ is by exiting states of the form $(s_0^i,\ldots,2),\ldots,(s_0^i,\ldots,k)$ in that order of m-s (m is the last component in each state). From transition relation we

know that an execution can exit state $(s_0^i, \ldots, m)$ only if $s_m^i \in \mathcal{F}_m$. Therefore to visit $(f_0^i, \ldots, 1)$ infinitely often, it must also visit $(s_0^i, \ldots, m)$ infinitely often where $s_m^i \in \mathcal{F}_m$.

Therefore a state sequence is accepting iff for each $m \in [k]$ $(s_0, \ldots, s_k, m)$ is visited infinitely often where $s_m \in \mathcal{F}_m$.

Let $\overline{w} = (w_1, \ldots w_k)$, and $\overline{w}_{-i} = (w_1, \ldots, w_{i-1}, w_{i+1}, \ldots w_k)$. Then from construction we know that $(\overline{w}_{-i}, w_i)$ has a run in $\mathcal{M}^i$ which follows the state sequence $s_i^0, s_i^1 \ldots$, where s_k^j are as in the transition state sequence. Since from the above argument, we know that $f_i^j \in \mathcal{F}_i$ is visited infinitely often, $(\overline{w}_{-i}, w_i)$ is an accepting word in $\mathcal{M}^i$.

Furthermore, if $\overline{p}$ is a payoff sequence along $\overline{w}$, then by accumulating payoffs along the transition sequence, we get that $\overline{p} = (p_1 \ldots, p_k)$, where p_i is a payoff sequence along $s_i^0, s_i^1 \ldots$.

This completes the proof of the first direction. Next, we prove the other direction: $(\overline{w}, \overline{p}) \in \overline{\mathcal{M}} \implies \overline{w} \in \mathit{Execs}(\overline{\mathcal{M}})$ with $\overline{p} \in \mathbf{P}(\overline{w})$.

Let $\overline{w} = (w_1, \ldots w_k)$, and $\overline{w}_{-i} = (w_1, \ldots, w_{i-1}, w_{i+1}, \ldots w_k)$. Also let, $\overline{p} = (p_1, \ldots, p_k)$.

Since $(\overline{w}, \overline{p}) \in \overline{\mathcal{M}}$, by definition of a run in a strategy profile, we know that for each i, $(\overline{w}_{-i}, w_i) \in \mathcal{M}^i$ and p_i is a payoff sequence of $(\overline{w}_{-i}, w_i)$ the i-th agent in $\mathcal{M}^i$. Let $\overline{s}_i = s_i^0 s_i^1 s_2^1 \ldots$ denote the state sequence corresponding to $(\overline{w}_{-i}, w_i)$ and payoff sequence p_i.

Construct the state tuple sequence $\overline{s} = (s_1^0, \ldots s_k^0)(s_1^1, \ldots s_k^1)(s_1^2, \ldots s_k^2) \ldots$. Let $(s_1^i, \ldots s_k^i)$ be the first state where $s_i^i \in \mathcal{F}_1$. Append all state tuples till the i-th state tuple with 1. Let j be the first index after i, where $s_2^j \in \mathcal{F}_2$. Append all state tuples after i until j (including j) with 2. After appending k to state sequences, we go back to appending 1, and so on. One can show that the resulting state sequence is accepted by $\mathit{Execs}(\overline{\mathcal{M}})$. Furthermore, $\overline{p} \in \mathbf{P}(\overline{w})$.

Example We illustrate the construction of a strategy profile from its component strategies through an example. Recall that the Prisoner's Dilemma [52] is a classi-

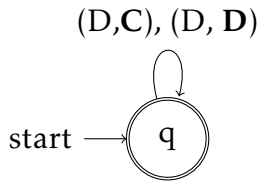


Figure 3.1 : IPD: Only Defect. For payoff, see Table 3.4

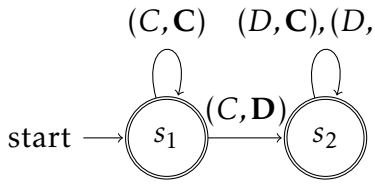


Figure 3.2 : IPD: Grim Trigger. For payoff, see Table 3.4

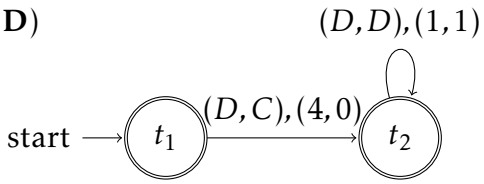


Figure 3.3 : Strategy profile when $\mathcal{P}_1$ plays Only Defect, and $\mathcal{P}_2$ plays Grim Trigger

Note: Transitions of strategies are labeled by $(a, \mathbf{b})$, where a is output, $\mathbf{b}$ is input. We omit agent payoff on transitions for clarity of figure.

cal synchronous game between two agents; each can either defect (D) or co-operate (C) with the other player. The payoff of each prisoner received from this game are given in Table 3.4. The infinitely repeated variant of PD is called the Iterated Prisoner's Dilemma (IPD) [38].

Figure 3.1 and Figure 3.2 depict two strategies for agents in IPD. Figure 3.1 depicts the strategy in which the agent always defects. Figure 3.2 depicts the Grim Trigger Strategy, in which the agent co-operates until the environment defects, following which the agent always defects. Figure 3.3 illustrates the strategy profile generated when $\mathcal{P}_1$ performs only defect strategy, and $\mathcal{P}_2$ performs the Grim Trigger strategy. In the figure transitions are labeled with $\overline{a}, \overline{p}$: action profile and payoff tuple along the transition respectively.

3.1.1 Comparison with the Rubinstein model

In the Rubinstein model, every agent strategy is given by a weighted, deterministic Moore machine. On the other hand, strategies in regular repeated games are given by weighted, non-deterministic transducers. While, every Rubinstein strategy can

	Cooperate (C)	**Defect (D)**
Cooperate (C)	3	0
Defect (D)	4	1

Figure 3.4 : Payoff table for Iterated Prisoner's Dilemma Strategies. **Row Agent**: Agent, **Column Agent**: Environment

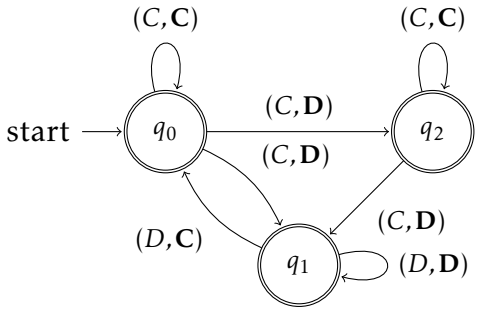


Figure 3.5 : TF2T(Tit-for-Tat-with-Forgiveness)

trivially be encoded as a regular repeated game strategy, the reverse is not true. This point is illustrated through an example and a formal proof in this section.

Consider the tit-for-tat-with-forgiveness strategy for the IPD illustrated in Figure 3.5. This strategy is a spin on the tit-for-tat strategy in which an agent may or may not perform D if the other agent takes action **D** once, but will certainly respond with a D if the other agent performs **D** twice. In Figure 3.5, q_1 denotes the state from which the agent forgives the other agent for performing **D** once, while state q_2 is the state at which the agent retaliates with D. Therefore, from the start state q_0 when the other agent performs **D** for the first time, transition on $(C, \mathbf{D})$ can either lead to state q_1 or q_2 depending on whether the agent forgives or not. Hence non-determinism is inherently present in this strategy.

Since the Rubinstein model does not permit non-determinism, TF2T as illustrated in Figure 3.5 cannot be encoded in the Rubinstein model, but can be encoded in regular repeated games. This motivates the need of a model with non-determinism. Furthermore, we can prove that regular repeated games permits a strictly richer class of strategies. For this it is sufficient to prove that not every non-deterministic strategy can be encoded as a deterministic strategy (see Lemma 3.1).

Lemma 3.1

Not all non-deterministic strategies can be determinized.

Proof 3.2 (Proof sketch) Consider a non-deterministic game. In this game, a non-deterministic strategy for the agent can also be viewed as a non-deterministic discounted sum automaton for infinite length words over the alphabet of $\bigtimes_{i=1}^{k} Action(i)$. Suppose it were possible to determinize a discounted sum automaton over infinite length words, then we could also determinize discounted sum automaton over finite length words. But determinization of all discounted sum automata over finite words is not possible, as proved in a result by Böker and Henzinger [13]. This proves that it is not possible to determinize every non-deterministic strategy.

Hence we have proved that regular repeated games permit a strictly larger class of strategies as compared to the Rubinstein model. Hence, it is justified to say the regular repeated games comprise of a strictly richer class of repeated games as compared to the Rubinstein model.

Another interesting point of difference between the Rubinstein model and regular repeated games arises in the structure of strategy profiles in them. In the Rubinstein model strategy profiles can have at most a single run, while strategy profiles in regular repeated games may have more than one run. As a result, in regular repeated games agents may receive multiple rewards from a single strategy profile. Hence, we are required to re-define the various solution concepts in regular repeated games (See Section 3.1.2).

3.1.2 Solution concepts

In this section, we define the solution concepts of Nash equilibria and best response strategy for repeated regular games.

But first, we introduce some necessary notation. For each agent $\mathcal{P}_i$ and strategy $\mathcal{M}'$ in $Strategy(i)$, we define a strategy profile $\overline{\mathcal{M}}[i := \mathcal{M}']$ obtained by starting with $\overline{\mathcal{M}}$, and then switching the strategy of agent $\mathcal{P}_i$ to $\mathcal{M}'$. Precisely, $\overline{\mathcal{M}}[i := \mathcal{M}']$ is defined as the tuple $(\mathcal{M}^1, \ldots, \mathcal{M}^{i-1}, \mathcal{M}', \mathcal{M}^{i+1}, \ldots, \mathcal{M}^k)$.

Nash equilibria We define Nash equilibria for repeated regular games.

Definition 3.1 A strategy profile $\overline{\mathcal{M}}$ is a *Nash equilibrium* if for each agent $\mathcal{P}_i$, for each strategy $\mathcal{M}^i_* \in \mathit{Strategy}(i)$, there exists a run $\overline{\rho} \in \overline{\mathcal{M}}$ such that for all other runs $\overline{\rho'} \in \overline{\mathcal{M}}[i := \mathcal{M}^i_*]$, $\mathbf{P}^i(\overline{\rho}) \geq \mathbf{P}^i(\overline{\rho'})$.

A run $\overline{\rho} \in \overline{\mathcal{M}}$ is a *non-Nash run* if there exists another strategy profile $\overline{\mathcal{M}}'$ s.t. the strategy profiles differ in strategy of exactly one agent, say $\mathcal{P}_i$, and there exists a run $\overline{\rho'} \in \overline{\mathcal{M}}'$, s.t. $\mathbf{P}^i(\overline{\rho}) < \mathbf{P}^i(\overline{\rho'})$. A run is a *Nash run* otherwise. In cases when $\overline{\rho}$ is a non-Nash run, we call run $\overline{\rho'}$ the *witness* of $\overline{\rho}$s non-Nashness. A strategy profile is called Nash if it is in Nash equilibria, and non-Nash otherwise. A strategy profile $\overline{\mathcal{M}_*}$ that demonstrates profile $\overline{\mathcal{M}}$ non-Nash is called a *witness* of $\overline{\mathcal{M}}$.

One can show that strategy profiles in deterministic games have a unique run. Therefore, in deterministic games, the above definition of Nash equilibria coincides with the standard definition of Nash equilibrium. The definition is less standard in nondeterministic games, where a strategy profile can have multiple (in fact an unbounded number of) runs. Here, a strategy profile is a Nash equilibrium if it has *at least one* Nash run. Intuitively, this is a conservative definition, guided by the goal of not ruling out rational behaviors. Since our knowledge of runs is incomplete, the way to achieve this goal is to call a strategy profile Nash so long as the agents have *some* way of realizing a Nash run by following it.

Best response strategy We define *best response strategy* for regular repeated games.

Definition 3.2 A strategy $\mathcal{M}^i_*$ is said to be a *best response* to the environment action w.r.t. $\mathcal{P}_i$, $(\mathcal{M}^1, \ldots, \mathcal{M}^{i-1}, \mathcal{M}^{i+1}, \ldots, \mathcal{M}^k)$, if for all other strategies $\mathcal{M}^i \in \mathit{Strategy}(i)$, there exists a run $\overline{\rho} \in (\mathcal{M}^1, \ldots, \mathcal{M}^{i-1}, \mathcal{M}^i_*, \mathcal{M}^{i+1}, \ldots, \mathcal{M}^k)$ s.t. for all runs $\overline{\rho'} \in (\mathcal{M}^1, \ldots, \mathcal{M}^{i-1}, \mathcal{M}^i, \mathcal{M}^{i+1}, \ldots, \mathcal{M}^k)$, $\mathbf{P}^i(\rho) \geq \mathbf{P}^i(\rho')$.

As earlier, this definition of best response strategies coincides with the standard

definition in deterministic games. The reason behind choosing this conservative definition is same as that in the case of Nash equilibria.

Relationship between Nash equilibria and Best response strategy We make the following interesting observation between Nash equilibria and Best response strategy in repeated regular games.

Lemma 3.2

Let $\mathcal{G}$ be a 2-agent game. Suppose $\mathcal{P}_2$ receives a constant payoff in each strategy. Then, $(\mathcal{M}^1, \mathcal{M}^2)$ is in Nash equilibria in $\mathcal{G}$ iff $\mathcal{M}^1$ is a best response to the environment w.r.t $\mathcal{P}_1$.

Proof 3.3 (Proof Sketch) We provide an intuitive understanding behind the claim. The formal proof directly falls from this idea.

We already know that the first agent receives maximum payoff in the Nash equilibria strategy profiles. Hence if (S_1, S_2) is in Nash equilibria, then S_1 is also a best response strategy w.r.t. S_2 for the first agent.

The second agent does not gain or loose payoff by unilaterally changing its strategy profile since it gets constant payoff on all. Therefore, only the first agent can receive greater or lower payoff from the game. Next, if S_1 is a best response strategy to S_2 for the first agent, since the second agent is agnostic to the strategy it plays, (S_1, S_2) is also in Nash equilibrium.

3.1.3 Size of repeated regular games

Definition 3.1 and Definition 3.2 are both defined for/on strategy profiles. Therefore, in all analysis that follows, we consider strategy profiles, *not* strategies, to be the fundamental unit for analysis. Hence we denote the size of a game w.r.t. the number and size of strategy profiles in the game.

Let SP denote the set of all strategy profiles in a given game $\mathcal{G}$. We define $|\mathcal{G}| = \Sigma_{S \in SP}|S|$, where strategy profiles are represented in their automaton form.

From the construction of strategy profile automata(See Section 3.1), we observe that the size of a strategy profile is proportional to the product of the size of each component strategy. Let $S_i \in Strategy(i)$ denote the largest strategy for $\mathcal{P}_i$. Then the size of each strategy profile S, $|S|$, is given by $\mathcal{O}(\Pi_{i=1}^{k}|S_i|)$. Also the number of strategy profiles in game $|\mathcal{G}|$ is given by $\mathcal{O}(\Pi_{i=1}^{k}|Strategy(i)|)$. Together this implies $|\mathcal{G}| = \Sigma_{S\in SP}|S| \implies |\mathcal{G}| = \mathcal{O}(\Pi_{i=1}^{k}|Strategy(i)|) \cdot \mathcal{O}(\Pi_{i=1}^{k}|S_i|)$.

3.2 Summary

In summary, this chapter formally defines a regular repeated game, a finite state machine based model for games. Here every agent has a finite number of strategies, which are given by finite-state, weighted, non-deterministic, transducers, where the inputs are actions taken by other agents, and outputs are the agent's actions. Weights on transitions denote the reward received by the agent. Agent strategies synchronize on their actions in a game, and the reward on an executions is given by the discounted sum of rewards on transitions.

The chapter defines Nash equilibria (best response strategies) in regular repeated games conservatively: if the strategy profile (strategy) consists of at least one run that satisfies the criteria of the solution concept for deterministic games, the strategy profile (strategy), then the strategy profile (strategy) is a Nash equilibrium (best response strategy). Our motivation behind this choice of definition arises from our goal of analyzing properties of rational behaviors (solution concepts). Hence, we do not to rule out any possible rational behavior. These definitions can be altered depending on the motivation.

Lastly, the chapter discusses the parameter for the size of a game. We compute the size of a game in terms of the size of all strategy profiles in it.

Chapter 4

Computing Equilibria

This chapter presents ComputeNash: an algorithm to compute all Nash equilibria in a repeated regular game.

Section 4.1 gives an overview of relevant prior work with respect to equilibria computation in repeated games. Our algorithm ComputeNash is discussed in detail in Section 4.2. The algorithm is described in Section 4.2.1, and proofs of its correctness and complexity are presented in Section 4.2.2. We also discuss an adaption of ComputeNash to compute best response strategies for an agent in a regular repeated game. A detailed discussion on the methodology and results from experiments on a prototype implementation of ComputeNash is presented in Section 4.3. We summarize the chapter in Section 4.4.

4.1 Prior work

There is a large literature on repeated games, and infinitely repeated games with discounted payoffs as the model for aggregation of payoff [38]. Equilibrium computation in repeated games has also been studied. Of existing work on this topic, one thread focuses on computing the set of equilibrium payoffs in repeated games [1, 26]. The computation of strategy profiles that form equilibria is not considered. Also, the algorithms here are fixpoint computations over reals, and do not come with complexity guarantees.

A different thread of prior work studies the computation of a *single* equilibrium strategy in repeated games [4, 35, 37]. There are also a few approaches to computing representations of *approximations* to the set of equilibria [11, 17]. The tech-

niques here involve iterative computations in continuous spaces, and their complexity increases with precision. In contrast, we give algorithms that compute sets of equilibrium strategies exactly and have crisp complexity bounds. The price that we pay is in expressiveness — rationality in our model is bounded.

None of the works mentioned so far consider finite-state machine based models for repeated games. ComputeNash considers the problem of equilibria computation on a finite-state machine based model for repeated games, namely regular repeated games.

The literature on equilibria of finite state machine games has primarily focused on two questions: properties of equilibria under different kinds of structural assumptions on strategy machines [2, 8, 10, 49, 53, 54, 57], and the ability of bounded rationality to support cooperative outcomes [48]. In contrast, the *computation* of equilibria has not received much attention in this setting. Our high-level contribution is to show that automata-theoretic decision procedures offer a promising set of tools for this problem.

While finite state machine based models [57] are among the oldest models for games with bounded rationality [59], Turing machine based models for strategies have also been considered. In some of these approaches, strategy machines have restrictions on time, space usage or number of states [28, 41, 64]; in a recent approach of this sort, agents have to pay an explicit cost for computation while executing their strategies [34]. We are not aware of any work on equilibrium computation in any of these contexts.

There is an emerging literature, motivated by applications in formal methods, on equilibria in games given as ω-automata [15, 16, 20]. Many of these approaches focus on turn-based and/or zero-sum games; most of them aim to compute a single equilibrium. We look at the problem of equilibria computation on regular repeated games, which are a form of simultaneous and non-zero sum games. Another paper by Klimoš, Larsen, Štefaňák and Thaarup [36], which computes

an automata representation of Nash equilibria on deterministic concurrent graph games. Graph games are again different from regular repeated games.

4.2 Computing Nash equilibria

We present ComputeNash, an algorithm to compute all Nash equilibria in a regular repeated game $\mathcal{G}$. Before delving into the details of ComputeNash, we explain the key ideas of the algorithm. This is followed by a theoretical analysis of the correctness and complexity of the algorithm.

4.2.1 ComputeNash algorithm description

The crux of the algorithm ComputeNash is to determine whether a given strategy profile is a Nash equilibrium or not. This is based on the following key ideas:

1. To determine whether a strategy profile $\overline{\mathcal{M}}$ is not in Nash equilibria, we search for a witness by iterating over all possible unilaterally deviated strategy profiles from $\overline{\mathcal{M}}$. (Algorithm 1).

2. $\overline{\mathcal{M}_*}$ is a witness of $\overline{\mathcal{M}}$, say they differ on $\mathcal{P}_i$ only, iff each run $\overline{\rho} \in \overline{\mathcal{M}}$ there is a run $\overline{\rho'} \in \overline{\mathcal{M}_*}$ s.t. $\mathbf{P}^i(\overline{\rho}) < \mathbf{P}^i(\overline{\rho'})$ (Algorithm 2).

3. Finally, to compare payoff of two runs, $\overline{\rho}$ and $\overline{\rho'}$ we construct a finite state ω-regular comparator automaton for the discounted sum aggregate function. We discuss the details of this construction in detail in Section 5.2. For now, it is sufficient to know that the discounted sum comparator for discount factor $d > 1$ is a Büchi automaton, denoted by $\mathcal{A}_{>_{DS(d)}}$, that accepts a pair of infinite length integer sequence (A, B) iff $DS(A, d) < DS(B, d)$.

In this section, we abuse notation and use $\overline{\mathcal{M}}$ to denote both the strategy profile and its corresponding Büchi automaton. In the following, we explain steps of the algorithm with a running example. For the running example we take S to be the strategy profile in Figure 3.3, and T to be as in Figure 4.1.

ALGORITHM 1: ComputeNash($d, Strategy(1), Strategy(2), \ldots, Strategy(k)$)

Input: Discount factor $d > 1$, set of strategies $Strategy(i)$ for each agent $\mathcal{P}_i$

Output: Set of all Nash equilibria from the game generated by all agents $\mathcal{P}_i$

1 $\gamma \leftarrow$ MaxPayoffValue($Strategy(1), Strategy(2), \ldots, Strategy(k)$)

2 $\mathcal{A}_{\succ_{DS(d)}} \leftarrow$ ComparatorAut(γ, d)

3 $Nash \leftarrow \{$StrategyProfile($\mathcal{M}^1, \mathcal{M}^2, \ldots, \mathcal{M}^n$)$|\mathcal{M}^1 \in Strategy(1), \mathcal{M}^2 \in Strategy(2), \ldots, \mathcal{M}^k \in Strategy(k)\}$

4 **forall** StrategyProfile($\mathcal{M}^1, \mathcal{M}^2, \ldots, \mathcal{M}^n$) $\in Nash$ **do**

5 **forall** $i \in \{1, 2, \ldots k\}$ **do**

6 **forall** $\mathcal{M}^i_* \in Strategy(i)$ **do**

7 $isWitness \leftarrow$ Witness(i,StrategyProfile($\mathcal{M}^1, \mathcal{M}^2, \ldots, \mathcal{M}^k$), StrategyProfile($\mathcal{M}^1, \ldots, \mathcal{M}^{i-1}, \mathcal{M}^i_*, \mathcal{M}^{i+1}, \ldots, \mathcal{M}^k$))

8 **if** $isWitness = True$ **then**

9 Remove StrategyProfile($\mathcal{M}^1, \mathcal{M}^2, \ldots, \mathcal{M}^k$) from $Nash$

10 **return** $Nash$

Details of ComputeNash The pseudo code for ComputeNash is presented in Algorithm 1. ComputeNash accepts as inputs a set of strategies, $Strategy(i)$, for each agent $\mathcal{P}_i$ and a rational valued discount factor $d > 1$. Recall that these parameters, the set of strategies for all agents, and the discount factor $d > 1$, completely specify a game. The output is the set of all Nash equilibria $Nash$ in the game $\mathcal{G}$ described by interaction of the set of agents of the input game.

As sketched earlier, the algorithm ComputeNash checks for the existence of a witness for each strategy profile. This involves a mechanism to compare the payoff, computed as the discounted sum of the payoff sequence with discount factor $d > 1$, of two number sequences. We construct the ω-regular comparator automaton for the discounted sum aggregate function with maximal bound γ equal to the maximum payoff in the game $\mathcal{G}$ (obtained in Line 1), and discount factor $d > 1$

(Line 2). The details of the construction of the ω-regular comparator automaton for discount factor $d > 1$ is given in Section 5.2. For now it is sufficient to know that this automaton, denoted by $\mathcal{A}_{>_{DS(d)}}$, accepts pair of number sequences (A, B) iff $f(A) < f(B)$. We assume that sequences A and B are bounded by γ.

We initialize *Nash* to the set of all strategy profiles in $\mathcal{G}$ (Line 3). StrategyProfile returns the Büchi automaton representation and the payoff relation for the strategy profile. Algorithm ComputeNash iterates over all strategy profiles present in *Nash* (Line 4), and removes a strategy profile $\overline{\mathcal{M}}$ from *Nash* if there exists a witness to $\overline{\mathcal{M}}$ (Line 7- 9). This way, at termination *Nash* consists of only those strategy profiles for which no witness exists. In other words, on termination of ComputeNash, *Nash* consists of the set of all Nash equilibria in $\mathcal{G}$.

The crux of ComputeNash lies in the subroutine Witness (Algorithm 2), which determines if a given strategy profile is a witness of another given strategy profile. Procedure Witness accepts as inputs an agent identity number i, and and two strategy profiles $\overline{\mathcal{M}}_1$ and $\overline{\mathcal{M}}_2$ that are unilaterally deviated from each other. In fact, $\overline{\mathcal{M}^1}$ and $\overline{\mathcal{M}^1}$ are different in the strategy of agent $\mathcal{P}_i$ only (Algorithm 1, Line 5-7). Procedure Witness returns *True* if $\overline{\mathcal{M}}_2$ is a witness of $\overline{\mathcal{M}}_1$, and *False* otherwise. Details of Witness are given below; its pseudo code is given in Algorithm 2.

Details of Witness Different algorithms for Witness under deterministic games and non-deterministic games are presented (Algorithm 2, Deterministic games: Line 1- 8; Non-deterministic games: Line 9- 18). However, the underlying ideas are similar. The key ideas of Witness are:

- Compare payoffs of run(s) on one strategy profile with run(s) in the other strategy profile. Suppose, the strategy profiles differ in the strategy of agent $\mathcal{P}_i$ only, then comparison of the payoffs of agent $\mathcal{P}_i$ are performed. The discounted sum comparator automaton is used for this comparison.

- Strategy profile $\overline{\mathcal{M}^2}$ is a witness of $\overline{\mathcal{M}^1}$ (assuming the profiles differ in the

strategy of agent $\mathcal{P}_i$ only) if for every run in $\overline{\mathcal{M}^1}$, there exists a run in $\overline{\mathcal{M}^2}$ along which agent $\mathcal{P}_i$ receives a strictly greater payoff.

The difference between the algorithm of Witness for deterministic games and non-deterministic games arises in ensuring this, since there is at most one run in strategy profiles for deterministic games, but there may be infinitely many such runs in strategy profiles for non-deterministic games.

Details of Witness for each case are described in detail below:

Witness **for deterministic games** To determine whether a strategy profile is non-Nash, we reason at the level of runs in the strategy profiles. Note that in deterministic games, each strategy profile can have at most a single run.

Let the inputs to Witness be the agent identity index i, and strategy profiles S and T. Recall that strategy profiles S and T differ in the strategy of agent $\mathcal{P}_i$ only (Algorithm 1, Line 5- 7). The objective is to determine if T is a witness of S. First, automaton S and T are transformed to automaton $\hat{S}$ and $\hat{T}$ respectively with the procedure AugmentWt (Line 2- 3). Procedure AugmentWt augments the payoff-tuples on transitions to the corresponding transition alphabet. Concretely, each transition of the form $\tau : (s, \overline{a}, t)$ and with payoff tuple $\overline{p} \in \mathbf{P}(\tau)$ is transformed to $\hat{\tau} : (s, (\overline{a}, \overline{p}), t)$ for every payoff tuple $\overline{p}$. This transformation leads to a one-to-one correspondence between accepting runs in the strategy profile automaton and the *augmented strategy profile automaton*. Furthermore, this transformation ensures that an accepting run in the strategy profile automaton is non-Nash iff its corresponding augmented run is non-Nash in the augmented strategy profile automaton.

The next step determines if accepting run $\rho' = (\overline{w}', \mathbf{P}(\overline{w}'))$ in $\hat{T}$ is witness of accepting run $\rho = (\overline{w}, \mathbf{P}(\overline{w}))$ in $\hat{S}$. Note that since every strategy profile in deterministic games has a unique accepting run, this is the same as determining if $\hat{T}$ is a witness of $\hat{S}$ (Line 4-Line 8). The first step here takes the product of $\hat{S}$ with $\hat{T}$

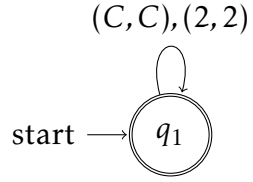


Figure 4.1 : Strategy Profile T

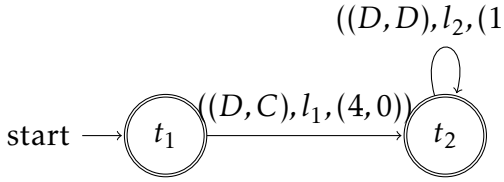


Figure 4.2 : $\hat{S}$ = AugmentWtAndLabel(S)

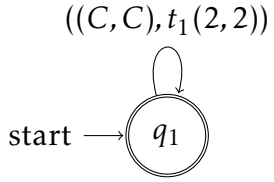


Figure 4.3 : $\hat{T}$ = AugmentWtAndLabel(T)

to obtain $\hat{\mathcal{A}}^{prod}$ (Line 4). Note that $\hat{\mathcal{A}}^{prod}$ accepts a single run only. More specifically, $\hat{\mathcal{A}}^{prod}$ accepts the run (ρ, ρ') only. Next, we *intersect* the discounted sum comparator automaton $\mathcal{A}_{\succ_{DS(d)}}$ with $\hat{\mathcal{A}}^{prod}$ over the i-th component of payoff-tuple sequences in $\hat{\mathcal{A}}^{prod}$ (Line 5). We denote this intersection by $\cap_i^*$ in the pseudo code Intuitively, the payoffs, computed as the discounted sum with discount factor d, received by agent $\mathcal{P}_i$ along ρ and ρ' are compared in this step. More concretely, $\hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}$ accepts (ρ, ρ') iff $\mathbf{P}^i(\rho) < \mathbf{P}^i(\rho')$. In other words, $\hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}$ accepts (ρ, ρ') iff run ρ' is a witness of run ρ. Recall that $\hat{\mathcal{A}}^{prod}$ accepts a single run which is (ρ, ρ'). Hence, $\hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}$ is non-empty iff ρ is a witness of ρ'. In other words, $\hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}$ is non-empty iff $\hat{T}$ is a witness of $\hat{S}$. Lines 5- 8 ensure that Witness returns *True* if $\hat{T}$ is witness of $\hat{S}$, and *False* otherwise. The transformation performed by AugmentWt in Line 2- 3 ensures that Witness returns *True* if T is a witness of S, and returns *False* otherwise.

Witness **for non-deterministic games** Recall, it is necessary to reason at the level of runs in the strategy profiles to determine whether a strategy profile is non-Nash. Unlike in deterministic games, where all strategy profiles accept a at one run, strategy profiles in non-deterministic games may accept more than one run. Therefore, to reason at the level of runs in strategy profiles in non-deterministic games, we disambiguate between the different runs in a strategy profile. To this

end, we assign a *unique label* to each transition in the strategy profile automaton . Procedure AugmentWtAndLabel is similar to procedure AugmentWt, except that AugmentWtAndLabel also augments the unique label for each transition in addition to the payoff-tuples to every transition. Hence, AugmentWtAndLabel transforms strategy profile automaton S and T into *augmented strategy profile automaton* $\hat{S}$ and $\hat{T}$ respectively, by converting each transition $\tau : (s_1, \overline{a}, s_2)$ to $\hat{\tau} : (s_1, \hat{a}, s_2)$ for $\hat{a} = (\overline{a}, label(\tau), \overline{p})$ where $label(\tau)$ is the unique label assigned to τ, and $\overline{p} \in \mathbf{P}(\tau)$ for every $\overline{p}$. (Line 10- 11). The result employing AugmentWtAndLabel to the running examples S (Figure 3.3) and T (Figure 4.1) is shown in Figure 4.2 and Figure 4.3 respectively. There is a one-to-one correspondence between accepting runs in S and T and in $\hat{S}$ and $\hat{T}$ respectively. This transformation ensures that an accepting run is non-Nash in S or T iff it is non-Nash in $\hat{S}$ or $\hat{T}$.

As in the deterministic case, the next step is to determine if each run in $\hat{S}$ has a witness in $\hat{T}$. For this, as earlier, the product automaton $\hat{\mathcal{A}}^{prod}$ of $\hat{S}$ and $\hat{T}$ is constructed (Line 12, Figure 4.4), and the product automaton $\hat{\mathcal{A}}^{prod}$ is intersected with the discounted sum comparator automaton $\mathcal{A}_{\succ_{DS(d)}}$ over the i-th component of the payoff-tuples. The automaton representing this intersection is named *Witness* (Line 13). For the running example, *Witness* automaton is constructed w.r.t. the second agent, and with discount factor $d = 2$ (Figure 4.5). Let ρ and ρ' be runs in $\hat{S}$ and $\hat{T}$ respectively. Then ρ and ρ' are of the form $(\overline{w}, \overline{p})$ for $\overline{p} \in \mathbf{P}_{\hat{S}}(\overline{w})$, and $(\overline{w}', \overline{p}')$ for $\overline{p}' \in \mathbf{P}_{\hat{T}}(\overline{w}')$ respectively. As in the case of deterministic games, *Witness* accepts (ρ, ρ') iff $DS(\mathbf{P}^i(\overline{p}), d) < DS(\mathbf{P}^i(\overline{p}'), d)$. Hence, *Witness* accepts (ρ, ρ') iff ρ' is a witness of ρ, hence ρ is a non-Nash run.

Unlike in deterministic games, it it not sufficient to check if any single run is non-Nash in a strategy profile to prove that the profile is non-Nash. It is necessary to prove that every run in the strategy profile is non-Nash. Hence, for $\hat{S}$ to be non-Nash, each of its accepting runs should have a witness. For this, we collect all non-Nash accepting runs of $\hat{S}$ in $\overline{Nash}$ by taking the projection of *Witness* along

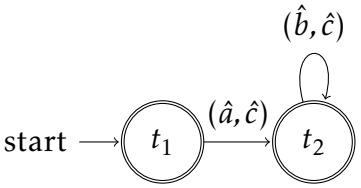


Figure 4.4 : $\hat{\mathcal{A}}^{prod} = \hat{S} \times \hat{T}$

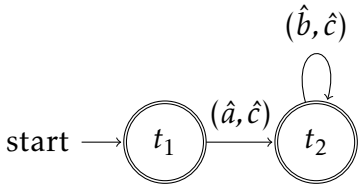


Figure 4.5 : *Witness* = $\hat{\mathcal{A}}^{prod} \cap_2^* \mathcal{A}_{\succ_{DS(2)}}$

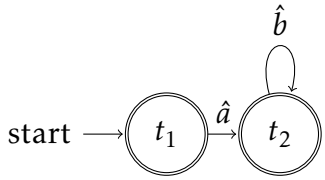


Figure 4.6 : $\overline{Nash}$

$$\hat{a} = ((D,C), l_1, (4,0)),\ \hat{b} = ((D,D), l_2, (1,1)),\ \hat{c} = ((C,C), t_1(2,2))$$

the first component (Line 14, Figure 4.6). ProjectFirst converts each transition $(s_1,(a,b),s_2)$ to (s_1,a,s_2).

Note that, *Witness* will be equal to $\hat{S}$ iff each accepting run in $\hat{S}$ has a witness in $\hat{T}$. In other words, *Witness* will be equal to $\hat{S}$ iff $\hat{T}$ is a witness of $\hat{S}$. Therefore, Lines 15- 18 ensure that Witness returns *True* if $\hat{T}$ is a witness of $\hat{S}$, and *False* otherwise. In the running example, $\hat{S} = \overline{Nash}$ and therefore, the subroutine Witness returns *True*.

4.2.2 **Analysis of** ComputeNash

In this section, we analyze the correctness and worst-case complexity of ComputeNash.

Theorem 4.1

ComputeNash *returns the set of all Nash equilibria in* $\mathcal{G}$.

Proof 4.1 The correctness of ComputeNash hinges on the correctness of Witness. The outer skeletal of ComputeNash ensures that every non-Nash strategy profile is removed from the set *Nash*. ComputeNash relies on Witness for detecting in w.r.t. a given agent $\mathcal{P}_i$, strategy profile T is a witness of strategy profile S.

The correctness has been sketched in the paper. There are two crucial steps in the proof. First, $\hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}$ accepts run (ρ,ρ') iff ρ' is a witness of ρ. Second, non-emptiness check in case of deterministic games. Third, for the non-

deterministic case, $\overline{Nash} = \hat{S}$ is a valid ultimate test. We expand on the above two point here.

For the first, this is immediate from the correctness of $\mathcal{A}_{\succ_{DS(d)}}$ (See Chapter 5). Note that from our construction in the paper, $\mathcal{A}_{\succ_{DS(d)}}$ accepts $(\mathbf{P}^i(\rho), \mathbf{P}^i(\rho)')$ iff $f(\mathbf{P}^i(\rho)) < f(\mathbf{P}^i(\rho'))$ iff $\mathbf{P}^i(\rho) < \mathbf{P}^i(\rho')$. Hence, $(\rho, \rho') \in \hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}$ iff ρ' is a witness of ρ.

We present the correctness of the procedure for non-emptiness check in Witness for deterministic games.

Uniqueness of runs in deterministic games ensures that if $(\rho, \rho') \in \hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}$ then all runs in strategy profile S are non-Nash. Therefore, S cannot be in Nash equilibria. Hence, S is non-Nash if $\hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}} \neq \emptyset$. Therefore, Witness returns *True* in this case.

Witness returns False if T is not a witness of S.

Note that since in ComputeNash we iterate over all possible unilaterally deviated strategy profiles, if there exists a witness, then ComputeNash and Witness will be able to find it. Non-existence of any witness proves Nash-ness of S.

We present the correctness of the procedure for equivalence check in Witness for non-deterministic games.

We know that $(\rho, \rho') \in \mathit{Witness}$ iff ρ' is a witness of ρ. In other words, $(\rho, \rho') \in \mathit{Witness}$ iff ρ is a non-Nash run in $\hat{S}$. Therefore, this ensures that $\rho \in \hat{S} \in \overline{Nash}$ iff ρ is non-Nash. From construction, it is clear that $\overline{Nash} \subseteq \hat{S}$. If every ρ in $\hat{S}$, is also present in $\overline{Nash}$, then $\hat{S} = \overline{Nash}$. Furthermore, if every ρ in $\hat{S}$, is also present in $\overline{Nash}$ then no ρ in $\hat{S}$ is in Nash. Therefore, this means $\hat{S} = \overline{Nash}$ iff no ρ in $\hat{S}$ is a Nash run. Therefore, $\hat{S}$ cannot be in Nash equilibria, and $\hat{T}$ is a witness of $\hat{S}$.

Therefore, if ever $\hat{S} = \overline{Nash}$, Witness returns *True* since T is a witness of S. Otherwise, Witness returns *False*.

Lastly, if there exists a witness to S, ComputeNash and Witness will certainly find it, since in ComputeNash we iterate over all unilaterally deviated strategy

profiles from S.

Lemma 4.1

An upper bound on the complexity of Witness *in deterministic games is* $\mathcal{O}(|S|\cdot|T|\cdot|\mathcal{A}_{\succ_{DS(d)}}|)$ *where* S *and* T *are the input strategy profiles and* $\mathcal{A}_{\succ_{DS(d)}}$ *is the discounted-sum comparator automata with discount factor* $d > 1$.

Proof 4.2 In the deterministic case, Witness effectively reduces to checking for non-emptiness of the Büchi automaton $\hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}$. We know that non-emptiness checking is linear in the size of the automaton, therefore complexity of Witness is of the order of $\mathcal{O}(|\hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}|) \implies \mathcal{O}(|\hat{\mathcal{A}}^{prod}|\cdot|\mathcal{A}_{\succ_{DS(d)}}|) \implies \mathcal{O}(|\hat{S}|\cdot|\hat{T}|\cdot|\mathcal{A}_{\succ_{DS(d)}}|)$. AugmentWt does not change the size of the strategy profiles, therefore Witness is of the order $\mathcal{O}(|\hat{\mathcal{A}}^{prod}|\cdot|\mathcal{A}_{\succ_{DS(d)}}|) \implies \mathcal{O}(|S|\cdot|T|\cdot|\mathcal{A}_{\succ_{DS(d)}}|)$ where S and T are the two input strategy profile automata.

Therefore, complexity of Witness in deterministic games is $\mathcal{O}(|S|\cdot|T|\cdot|\mathcal{A}_{\succ_{DS(d)}}|)$ where S and T are the input strategy profiles and $\mathcal{A}_{\succ_{DS(d)}}$ is the constantω-regular comparator automata for f.

Lemma 4.2

An upper bound on the complexity of Witness *in non-deterministic games is* $\mathcal{O}(|S|\cdot 2^{(|S|\cdot|T|\cdot|\mathcal{A}_{\succ_{DS(d)}})|})$ *where* S *and* T *are the input strategy profiles and* $\mathcal{A}_{\succ_{DS(d)}}$ *is the discounted-sum comparator automata with discount factor* $d > 1$.

Proof 4.3 Given input strategy profiles S and T, we want to determine if T is a witness of S. In the first few steps (Line 10- 13), we are constructing an automaton $\overline{Nash}$. The complexity of constructing witness is $\mathcal{O}(|\overline{Nash}|) = \mathcal{O}(|Witness|)$ since ProjectFirst does not change the size of the automaton. $\mathcal{O}(Witness) = |\hat{\mathcal{A}}^{prod}|\cdot|\mathcal{A}_{\succ_{DS(d)}}| \implies \mathcal{O}(\overline{Nash}) = |\hat{S}|\cdot|\hat{T}|\cdot|\mathcal{A}_{\succ_{DS(d)}}|$. Since AugmentWtAndLabel does not change the size of the automaton, $\mathcal{O}(\overline{Nash}) = |S|\cdot|T|\cdot|\mathcal{A}_{\succ_{DS(d)}}|$.

Line 15 is where we check for equivalence of two automaton, $\overline{Nash}$ and $\hat{S}$. Here $\hat{S}$ is deterministic but $\overline{Nash}$ is not. So, in the worst case, this step takes $\mathcal{O}(2^{|\overline{Nash}|}\cdot|\hat{S}|)$.

Therefore, Witness is of the order of $\mathcal{O}(2^{|\overline{Nash}|} \cdot |S|)$ where $\mathcal{O}(\overline{Nash}) = |S| \cdot |T| \cdot |\mathcal{A}_{\succ_{DS(d)}}|$.

Therefore, complexity of Witness in non-deterministic games is $\mathcal{O}(|S| \cdot 2^{(|S| \cdot |T| \cdot |\mathcal{A}_{\succ_{DS(d)}})|})$ where S and T are the input strategy profiles when we are checking if T a the witness of S and $\mathcal{A}_{\succ_{DS(d)}}$ is the constant ω-regular comparator automaton for f.

Theorem 4.2

Time complexity of ComputeNash *for deterministic game $\mathcal{G}$ is given by $\mathcal{O}(|\mathcal{A}_{\succ_{DS(d)}}| \cdot |\mathcal{G}|^2)$, where $\mathcal{A}_{\succ_{DS(d)}}$ is the discounted sum comparator with discount factor $d > 1$.*

Theorem 4.3

Time complexity of ComputeNash *for non-deterministic game $\mathcal{G}$ is given by $\mathcal{O}(|\mathcal{G}| \cdot (2^{|\mathcal{A}_{\succ_{DS(d)}}| \cdot |\mathcal{G}|^3}))$, where $\mathcal{A}_{\succ_{DS(d)}}$ is the discounted sum comparator with discount factor $d > 1$.*

We combine the proofs of Theorem 4.2 and Theorem 4.3 below.

Proof 4.4 For each strategy profile S, to find its witness we need to compare against all unilaterally deviated potential strategy profiles. There are as many potential strategy profiles as there are strategies in the game (each profile generated by changing one strategy)

Let S be the profile for which we want to check for witness. Let $T = \{T_1, \ldots T_q\}$ be the multi-set of all strategies in the game. We take the multiset since the same strategy may appear in different agents. Let $S[T_k]$ denote the strategy profile in which one component of S is changed to T_k. Then $|S[T_k]| = \mathcal{O}(|S||T_k|)$.

In case of deterministic games, checking witness against all potential profiles is $\Sigma_{T_i \in T}\mathsf{Witness}(S, (S[T_i])) = \Sigma_{T_i \in T}|S| \cdot |\mathcal{A}_{\succ_{DS(d)}}| \cdot |S[T_i]| = \Sigma_{T_i \in T}|S|^2 \cdot |\mathcal{A}_{\succ_{DS(d)}}| \cdot |T_i|$. Therefore for a fixed S, it takes $|S|^2 \cdot |\mathcal{A}_{\succ_{DS(d)}}| \cdot \Sigma_{i \in T_i}|T_i|$ to search for a witness. Let us call this $T_d(S)$.

In the non-deterministic case, checking for a witness of S takes time of the order $|S| \cdot \Sigma_{T_i \in T} 2^{|S|^2 \cdot |\mathcal{A}_{\succ_{DS(d)}}| \cdot |T_i|}$. Let us call this $T_n(S)$.

Let SP denote the set of all strategy profiles.

Therefore, complexity of this for all strategy profiles in deterministic games is $C_d = \Sigma_{S \in SP} T_d(S) = (\Sigma_{T_i \in T} |T_i| \cdot |\mathcal{A}_{\succ_{DS(d)}}|) \cdot (\Sigma_{S \in SP} |SP|) = |\mathcal{A}_{\succ_{DS(d)}}| \cdot (\Sigma_{T_i \in T} |T_i| \cdot) \cdot (\Sigma_{S \in SP} |SP|)$.

In the non-deterministic case, checking for witness in all strategy profiles is given by $C_n \Sigma_{S \in SP} T_n(S) \leq (\Sigma_{S \in SP} |S|) \cdot (\Sigma_{S \in SP} (\Sigma_{T_i \in T} 2^{|S|^2 \cdot |\mathcal{A}_{\succ_{DS(d)}}| \cdot |T_i|})) \leq (\Sigma_{S \in SP} |S|) \cdot (\Sigma_{S \in SP} (2^{|S|^2 \cdot |\mathcal{A}_{\succ_{DS(d)}}| \cdot \Sigma_{T_i \in T} |T_i|})) \leq (\Sigma_{S \in SP} |S|) \cdot (2^{(\Sigma_{S \in SP} (|S|^2) \cdot |\mathcal{A}_{\succ_{DS(d)}}| \cdot \Sigma_{T_i \in T} |T_i|)})$.

Now, $\Sigma_{T_i \in T} |T_i| \leq \Sigma_{S \in SP} |S|$. Therefore, $C_d \leq |\mathcal{A}_{\succ_{DS(d)}}| \cdot (\Sigma_{S \in SP} |SP|)^2$. And $C_n \leq (\Sigma_{S \in SP} |S|) \cdot (2^{(\Sigma_{S \in SP} (|S|^2) \cdot |\mathcal{A}_{\succ_{DS(d)}}| \cdot \Sigma_{S \in SP} |S|)}) \leq (\Sigma_{S \in SP} |S|) \cdot (2^{|\mathcal{A}_{\succ_{DS(d)}}| \cdot (\Sigma_{S \in SP} |S|)^3})$ since $\Sigma_{i_i}^n a_i^2 \leq (\Sigma_{i=1}^n a_n)^2$.

Since $|\mathcal{G}| = \Sigma_{S \in SP} |S|$,

$$\text{Complexity of ComputeNash in det. games } = \mathcal{O}(|\mathcal{A}_{\succ_{DS(d)}}| \cdot |\mathcal{G}|^2)$$

$$\text{Complexity of ComputeNash in non-det. games } = \mathcal{O}(|\mathcal{G}| \cdot (2^{|\mathcal{A}_{\succ_{DS(d)}}| \cdot |\mathcal{G}|^3}))$$

Note The exponential complexity in non-deterministic games arises due to a Büchi equivalence testing in Line 15 of Algorithm 2. Fortunately, there has been significant progress on tools for Büchi automaton equivalence in recent years [63]. The implementations of our algorithm leverages some of these tools.

4.2.3 Best response strategy

The algorithm for ComputeNash can be adopted to compute all best response strategies for agent $\mathcal{P}_i$ in a given game. For this we modify the input game G into $\hat{G}$ where the payoff on every transition of every strategy for all agents other than agent $\mathcal{P}_i$ is assigned to 0. Now, the strategies of $\mathcal{P}_i$ in strategy profiles returned from running ComputeNash on $\hat{G}$ will exactly be all the best response strategies for agent $\mathcal{P}_i$ in game G.

The correctness ComputeNash and Lemma 3.2 ensure that the resultant is indeed the set of all best response strategy for $\mathcal{P}_i$.

The worst case complexity of this algorithm is also the same as that for ComputeNash in the deterministic and non-deterministic case.

4.3 Experimental results

To demonstrate the practical utility of ComputeNash, we implemented a prototype of ComputeNash in Python. We employed the existing tools, GOAL [60] and RABIT – Reduce [61], for operations over Büchi automata. We implemented ComputeNash for both deterministic and non-deterministic games. In our implementation, we optimized on time and space by reducing the size of the automata generated at intermediate levels of the algorithm via minimization.

For our experiments, we performed case studies on the iterated prisoner's dilemma (IPD), the Bitcoin protocol, and repeated auctions. The experimental methodology for each case study was two-step process (1) modeling these systems as regular repeated games, (2) running ComputeNash on the system model, and tally our observations from the obtained results with previously known results.

The primary objectives of our experiments were:

1. to validate our model of regular repeated games and our definitions of solution concepts, by demonstrating that our analysis of the on case studies corroborate with previously known results on them.

2. to demonstrate that regular repeated games can model complex systems such as the Bitcoin protocol. Infact, we do not know of any earlier analysis in which the Bitcoin protocol has been viewed as a repeated game.

3. to compute Nash equilibria (or best response strategies) in complex game systems and analyze properties on them.

Our experiments fulfilled all objectives, hence demonstrated promise of the approach. In particular, ComputeNash allows us to rediscover several known

facts about various models of Iterated Prisoner's Dilemma (IPD) and repeated auctions. In contrast to the automated nature of ComputeNash, previously known results were obtained via manual analysis, which can be error-prone. Furthermore, the set of all Nash equilibria computed by ComputeNash leads us to infer new observations about well studied Iterated Prisoner's Dilemma (IPD). Finally, we demonstrate that Bitcoin protocol does not ensure fairness – a result also obtained through manual analysis by Eyal and Sirer.

Each experiment was conducted on a single core of a 12 core 2.2GHz Intel Xeon processor with 64 GB RAM.

4.3.1 Iterated Prisoner's Dilemma (IPD)

The iterated prisoner's dilemma (IPD) [38] is the classical example of a repeated game. We performed a number of experiments with variants of the IPD games.

We discuss ours observations from the results obtained from ComputeNash for infinitely and finitely repeated Prisoner's Dilemma. In the following, we use C, D, E to refer to the actions *co-operate*, *defect*, and *end of game* respectively. The discount factor is set to 2 for all experiments; Table 3.4 lists the payoffs of agents.

Non-deterministic Infinitely Repeated Iterated Prisoner's Dilemma Figure 4.7 depicts various strategies for agents considered in our modeling of Infinitely repeated IPD. In particular, All corresponds to the strategy where agent non-deterministically chooses one of the actions. Furthermore, AlwaysC and AlwaysD (Figure 3.1) correspond to the strategies where the agent always takes the action C, i.e. cooperate, and D, i.e. defect, respectively. Under the GT (Grim Trigger) strategy (Figure 3.2), the agent cooperates until its opponent defects and then always takes action D (i.e. defects) regardless of its opponent's actions. On the other hand, under TFT(Tit-For-Tat) strategy, the agent begins with C (i.e. cooperates) and simply mimics its opponent's actions. Finally, under TF2T (Tit-for-Tat-with-forgiveness) the agent,

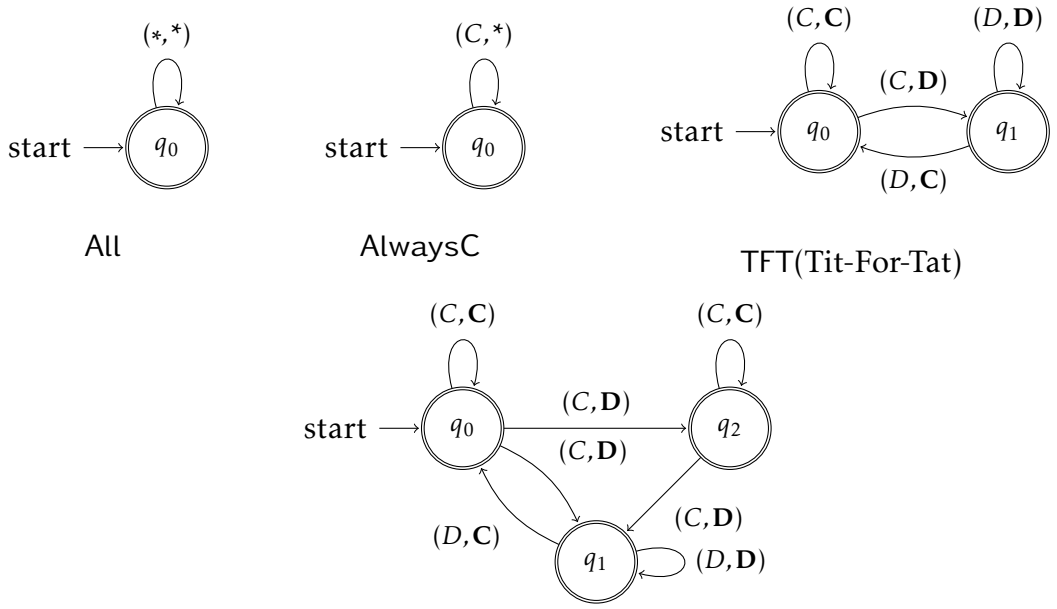


Figure 4.7 : Strategies for agents under IPD. Gradient $*$:C or D; *: **C** or **D**. For payoff, see Table 3.4

similar to TFT, mimics its opponent's actions until its opponent defects. Unlike TFT, the agent might immediately retaliate by defecting, or it might wait for its opponent to defect once more before it retaliates by defecting.

Our experimental evaluation consisted of equilibria computation for games resulting from various combination of strategies for both agents. For lack of space, we present only a subset of results below:

1. $Strategy(1) = \{\mathsf{TFT}\}$, $Strategy(2) = \{\mathsf{TFT}, \mathsf{All}, \mathsf{AlwaysD}, \mathsf{AlwaysC}\}$: The set of Nash equilibria for this game is $\{(TFT, \mathsf{All}), (\mathsf{TFT}, \mathsf{TFT}), (\mathsf{TFT}, \mathsf{AlwaysC})\}$. In other words, when $\mathcal{P}_1$ adopts TFT, $\mathcal{P}_2$ should not choose AlwaysD. A closer look at the strategy profile reveals that when $\mathcal{P}_1$ adopts TFT with C as its first action, it is in the interest of $\mathcal{P}_2$ to cooperate.

2. $Strategy(1) = \{GT\}$, $Strategy(2) = \{\mathsf{GT}, \mathsf{All}, \mathsf{AlwaysD}, \mathsf{AlwaysC}\}$. The set of Nash equilibria is $\{(\mathsf{GT}, \mathsf{GT}), (\mathsf{GT}, \mathsf{All}), (\mathsf{AlwaysC})\}$. Similar to above, the closer look at the strategy profile reveals that when $\mathcal{P}_1$ begins with co-operating, it is best for $\mathcal{P}_2$ to always co-operate.

3. $Strategy(1) = \{\mathsf{All}\}, Strategy(2) = \{\mathsf{All}, \mathsf{AlwaysC}, \mathsf{AlwaysD}\}$. In this case, the set of Nash equilibria is $\{(\mathsf{All}, \mathsf{AlwaysD}), (\mathsf{All}, \mathsf{All})\}$. An interesting observation is that in absence of assurance of cooperation from $\mathcal{P}_1$, $\mathcal{P}_2$ can pursue the strategy of always defecting.

While the above set of mentioned results were known before [38], our experimental study also provides **new** exciting results that, to the best of our knowledge, have not been known before. We summarize one such observation below:

- $Strategy(1) = \{\mathsf{TFT}, \mathsf{TF2T}\}, Strategy(2) = \{\mathsf{All}\}$. The set of Nash equilibria in this case is: $\{(\mathsf{TFT}, \mathsf{All}), (\mathsf{TF2T}, \mathsf{All})\}$. The presence of $(\mathsf{TFT}, \mathsf{All})$ indicates that if if $\mathcal{P}_1$ has to choose between being forgiving by nature (and hence TF2T) and Tit-for-Tat, it does not hurt to choose TF2T.

All experiments mentioned above, except for Experiment (3) mentioned above completed in less than 2.5 mins. Experiment 3 completed in ∼5 mins.

Deterministic Infinitely Repeated PD In this case, we consider games under the set of deterministic strategies, represented by deterministic transducers, shown in Figure 4.7. In particular, we consider AlwaysC, AlwaysD, GT and TFT. ComputeNash is invoked to compute the set of Nash equilibria formed from the above strategies. Similar to previously known results, we observed that when $\mathcal{P}_1$ acts according to Grim Trigger or Tit-for-Tat, Nash equilibrium is attained when $\mathcal{P}_2$ always co-operates.

Most experiments completed in less than 30 sec. The experiment in which $Strategy(1) = Strategy(2) = \{\mathsf{AlwaysC}, \mathsf{AlwaysD}, \mathsf{GT}, \mathsf{TFT}\}$ took ∼4 mins to complete.

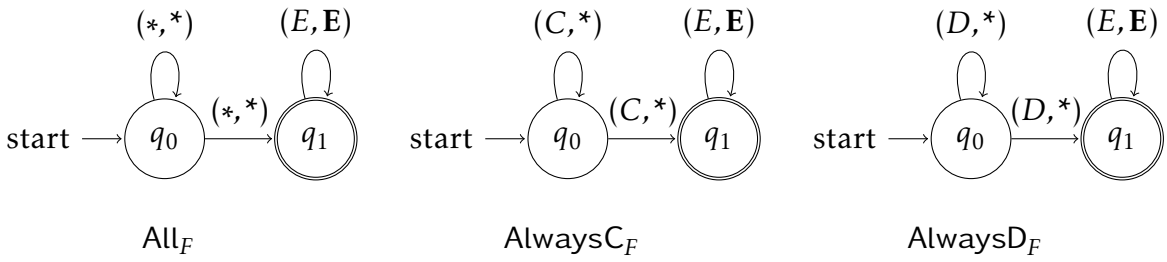


Figure 4.8 : Strategies for agents under finitely repeated Prisoner's Dilemma. Gradient $*$:C or D; *: **C** or **D**. Payoff $(q_1, E, E, q_1) = 0$, for rest, see Table 3.4

The same experiment using the non-deterministic procedure Witness took ∼8.5 mins to complete. The blow up in time is due to the Büchi complementation steps involved in the non-deterministic procedure for Witness.

Finitely Repeated Iterated Prisoner's Dilemma Unlike infinitely repeated IPD, the game in finitely repeated IPD ends after a finite but arbitrary number of interactions. In order to use powerful model of infinitely repeated IPD, we model finite repeated IPD as an instance of infinitely repeated IPD. To this end, we modify transducers proposed above to handle finitely many interactions. In particular, we add a special end-of-game action E that agents take after a finite number of steps. Once action E is taken, agents are forced to repeatedly take action E. We take the payoff on all transitions emerging from the E-state to be 0, indicating that no agent receives any further payoff. Note the initial and accepting states in Figure 4.8.

We let $\mathit{Strategy}(1) = \mathit{Strategy}(2) = \{\mathsf{All}_F, \mathsf{AlwaysC}_F, \mathsf{AlwaysD}_F\}$, where X_F indicates the finite repeated version of X as illustrated in Figure 4.8. The set of Nash equilibria is: $\{(\mathsf{AlwaysD}_F, \mathsf{AlwaysD}_F), (\mathsf{All}_F, \mathsf{AlwaysD}_F), (\mathsf{AlwaysD}_F, \mathsf{All}_F)\}$.

$(\mathsf{AlwaysD}_F, \mathsf{AlwaysD}_F)$ indicates that Nash equilibria is attained when both agents defect at all times. The other two strategy profiles are in set of all Nash equilibria since $(F - \mathsf{AlwaysD}, F - \mathsf{AlwaysD})$ is contained in them. This validates a known result for this game obtained via backward induction [38]: the Nash equilibrium in

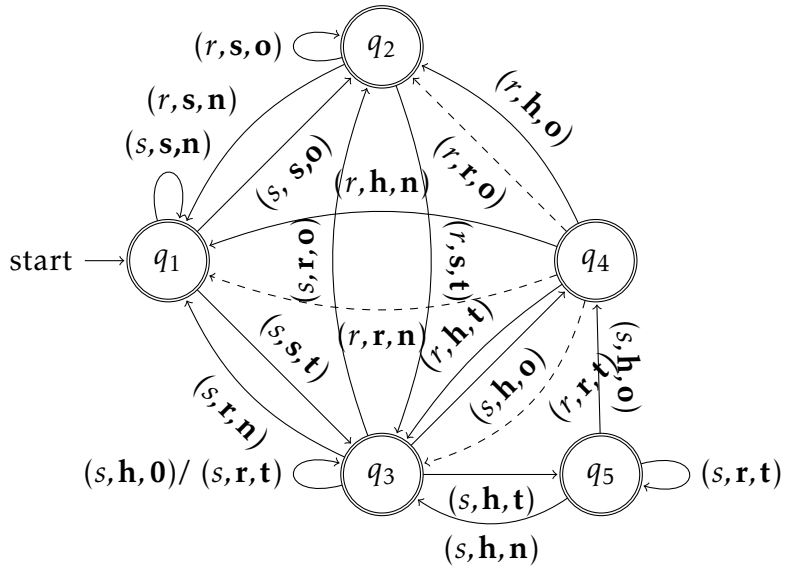


Figure 4.9 : Honest strategy in Bitcoin protocol. Payoffs explained in text.

finitely repeated Prisoner's Dilemma is attained when both agents always defect.

This experiment completed in ∼6 mins.

4.3.2 Bitcoin Protocol

Bitcoins are one of the most commonly used online-currencies in today's times. The protocol by which Bitcoins are *mined* is called the Bitcoin protocol. The Bitcoin protocol consists of agents that *mine* for Bitcoins, called *miners*, and a *scheduler* that assigns Bitcoins to miners. Bitcoins are present on a linear blockchain. All miners constantly mine at the tip of the block-chain. The scheduler assigns the Bitcoin at the tip to one of the miners. On being assigned the bitcoin, this miner informs all other miners that it has received the Bitcoin present at the tip, this Bitcoin is *popped* from the blockchain, exposing a new tip for all miners to mine on. The miner receives the Bitcoin only after it is popped from the tip. The scheduler can keep assigning the Bitcoin at the tip until it is not popped from the blockchain. *Honest miners* inform other miners that they have been assigned the Bitcoin immediately

after receiving it, while *dishonest miners* may delay releasing the information. By delaying this information, it dupes all other miners into continuing to mine at the older tip, while itself mining on the Bitcoin present next to the tip. In the instance when the scheduler assigns the same Bitcoin to another miner, it leads to a situation where two miners have been assigned the same Bitcoin. In this case, if both miners release information of the Bitcoin, then only one of the miners receives it. Depending on whether the dishonest agent received the Bitcoin or not, it either looses a Bitcoin that it could have easily received, or it manages to receive Bitcoin while forcing other miners to consume resources on an already assigned Bitcoin, and while establishing a good lead on the next Bitcoin in the blockchain.

The Bitcoin protocol is said to be incentive compatible if all agents play honestly in order to receive greater rewards from the game. In other words, the Bitcoin protocol is incentive compatible if in no solution concept an agent decides to play according to a dishonest strategy. Earlier, Eyal and Sirer proved via rigorous manual analysis that the Bitcoin protocol is not incentive compatible [32]. We show that same result, however via the automated model checking approach discussed in Section 4.

Game Model To the best of our knowledge, the Bitcoin protocol has not been viewed has a repeated game before. In our repeated game model for the Bitcoin protocol, we assume three agents in the protocol: one honest miner, one dishonest miner, and one scheduler. The miners either mine for Bitcoins, denoted by s (for search), or they release information of a Bitcoins' assignment, denoted by r (for release). The dishonest miner may additionally hide a mined Bitcoin, denoted by h (for hide). The scheduler either assigns a Bitcoin to no miner, to the first miner or to the second miner, denoted by n, o, s. Each agent takes an action in each round. For simplicity, we assume that the dishonest miner cannot hide information for more that one Bitcoin.

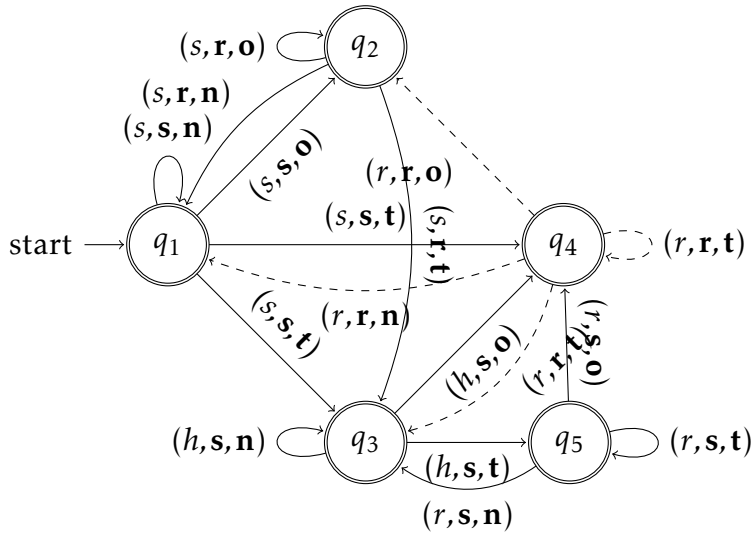


Figure 4.10 : Dishonest strategy in Bitcoin protocol. Payoffs explained in text.

Miners can play according to two strategies: Honest and Dishonest as shown in Figure 4.9 and Figure 4.10 respectively. In both these figures, state q_1 denotes the state in which no agent has been assigned a Bitcoin by the scheduler, q_2 and q_3 denote the state in which the honest and dishonest agent have been assigned a Bitcoin respectively. q_5 denotes the state in which the dishonest agent delays informing the other agents. Note how the dishonest agent is forced to release information about the earlier assignment of Bitcoin if it is assigned another one from this state. q_4 the state at which both agents have been assigned the same Bitcoin.

Also note that the miner plays h on outgoing transitions from q_3. These are the transitions along which the dishonest agent "cheats".

Under the Honest strategy, a miner performs action s and r only, while the other miner and scheduler choose between performing s, r, h and n, o, t respectively (Figure 4.9). As shown in the figure, both miners begin with searching for a mine in state q_1. Therefore, state q_1 denotes the state where neither agent has received a mine. When the honest miner receives a mine, control shifts to state q_2 if the

dishonest agent is not hoarding a mine, and to state q_4 otherwise. State q_3 denotes the state where control shifts to when the dishonest agent receives it's first mine. From this state, the dishonest agent may choose to release the mine immediately or hoard it. Since we have assumed that the dishonest agent has a limited hoarding capacity of one, control shifts to state q_5 if it receives more mine, from where it is forced to release all but one mine. If the honest agent receives a reward while control is present in state q_3 or q_5 (where the dishonest agent is hoarding a mine) then control shifts to state q_4. At q_4, either both miners release the mine and compete it or the dishonest agent continues to hoard it, thereby losing the mine to the honest agent. Similarly, under the Dishonest strategy, the miner can perform actions s, r, and h.

Rewards Since we introduce scheduler to model non-deterministic in reward function, the schedule does not receive any reward. To this end, we assume that it always receives award of 0. On the other hand, a miner receives reward of 1 if it releases a block in a state other than q_4 and 2 for releasing a block from state q_4. The higher reward of 2 indicates the success of a miner in wasting other miners resources – a tactic that a dishonest miner would pursue. For all other actions, the reward is 0 for miners. Note that the dotted transitions result in the game being non-deterministic due to different utilities.

Experiment and Result We let $Strategy(1)$ = {Honest}, and $Strategy(2)$ = {Honest, Dishonest}. The Nash equilibria in this game were {(Honest, Dishonest)}. This denotes that the second agent has an incentive to play dishonestly in the Bitcoin Protocol. This experiment completed within 30 sec. We suspect that this is because of fewer strategies for each agent, low values of γ, and fewer iterations for each agent.

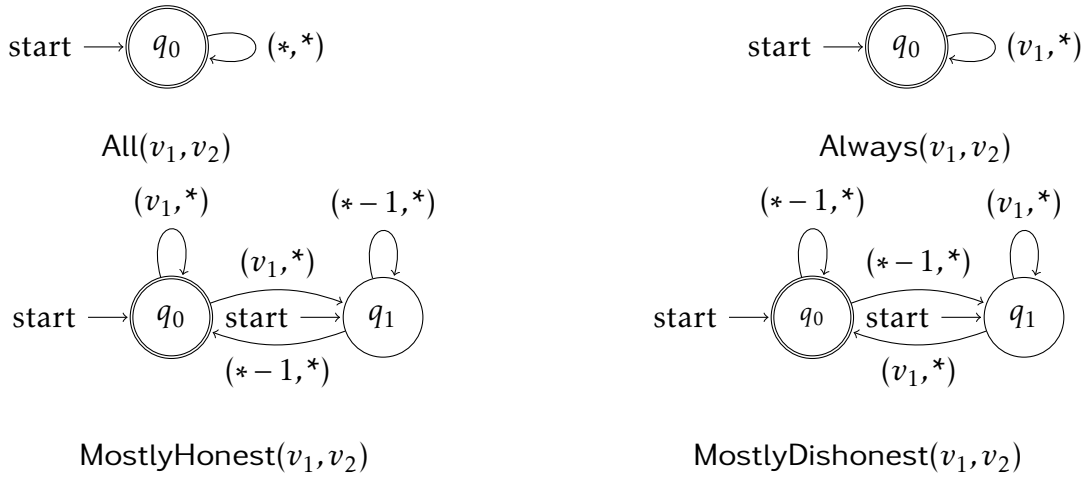


Figure 4.11 : Auction Strategies for $\mathcal{P}_1$ in 2-agent auction. True valuation of $\mathcal{P}_1 = v_1$, True valuation of $\mathcal{P}_2 = v_2$. Gradient: $* : \{0, 1 \dots v_1\}$, $* - 1 : \{0, 1 \dots v_1 - 1\}$, $* : \{\mathbf{0}, \mathbf{1} \dots \mathbf{v}_2\}$. For payoff, see the corresponding auction rule.

Rewards The scheduler does not receive any reward from this protocol, so on each transition it receives a reward of 0.

Miners do not receive any reward on action s. Transitions along which the miner plays r in Figure 4.9 and Figure 4.10 are either represented by dashed lines or by undashed lines. Note that the dashed transitions denote situations in which both miners release information about the same Bitcoin being assigned to them, and hence are competing with each other to receive it. In both strategies, the miner receives 1 unit reward along the undashed lines on action r, denoting that the miner has received the Bitcoin. In the Honest strategy (Figure 4.9), the miner receives either 0 or 1 unit reward for taking action r on the dashed lines, depending on whether it receives the Bitcoin. However, in the Dishonest strategy, the miner either receives 0 or 2 unit reward. It receives 0 if it looses the Bitcoin, but receives 2 if it receives it since it also manages to waste resource of the other miner.

Figure 4.11 illustrates various strategies for agent $\mathcal{P}_1$ in a 2-agent auction (Similar strategies exist for $\mathcal{P}_2$ as well). All(v_1, v_2) denotes the strategy in which $\mathcal{P}_1$ bids

at all values from 0 to v_1. $\mathsf{Always}(v_1, v_2)$ denotes the strategy in which $\mathcal{P}_1$ bids at v_1 only. $\mathsf{MostlyDishonest}(v_1, v_2)$ and $\mathsf{MostlyHonest}(v_1, v_2)$ denote strategies in which $\mathcal{P}_1$ may not always bid at v_1. In $\mathsf{MostlyDishonest}(v_1, v_2)$, $\mathcal{P}_i$ bids at values lesser than v_1 infinitely often. In $\mathsf{MostlyHonest}(v_1, v_2)$, $\mathcal{P}_i$ bids at v_1 infinitely often. In all of these strategies $\mathcal{P}_2$ can bid at all values ranging from 0 to v_2. Finally, we consider $\mathit{Strategy}(1) = \{\, \mathsf{All}(v_1, v_2), \mathsf{All}(v_1 - 1, v_2), \mathsf{Always}(v_1, v_2), \mathsf{MostlyDishonest}(v_1, v_2), \mathsf{MostlyHonest}(v_1, v_2) \,\}$, and $\mathit{Strategy}(2) = \{\mathsf{All}(v_2, v_1)\}$. Both our auction experiments completed in ~ 3 mins.

Repeated First-Price Auctions The auction rule for $\mathcal{P}_1$, when $\mathcal{P}_1$ and $\mathcal{P}_2$ bid at u_1 and u_2 respectively, is given as follows:

$$\mathit{FirstPriceAuction}(u_1, u_2) = \begin{cases} 1 & \text{if } u_1 > u_2 \\ 0 & \text{if } u_1 < u_2 \\ 0 \text{ or } 1 & \text{if } u_1 == u_2 \end{cases}$$

The set of Nash equilibria computed by ComputeNash is $\{(\mathsf{All}(v_1, v_2),\ \mathsf{All}(v_2, v_1)),$ $(\mathsf{All}(v_1 - 1, v_2), \mathsf{All}(v_2, v_1)), (\mathsf{Always}(v_1, v_2), \mathsf{All}(v_2, v_1)), (\mathsf{MostlyDishonest}(v_1, v_2), \mathsf{All}(v_2, v_1)),$ $(\mathsf{MostlyHonest}(v_1, v_2), \mathsf{All}(v_2, v_1))\}$. The presence of $(\mathsf{All}(v_1 - 1, v_2)$ and $(\mathsf{MostlyDishonest}(v_1, v_2)$ in Nash equilibria indicates that $\mathcal{P}_1$ has a dominant strategy even she always bids at values lower than her true valuation, i.e. $(\mathsf{All}(v_1 - 1, v_2))$, or when she does not always bid truthfully, i.e. $(\mathsf{MostlyDishonest}(v_1, v_2))$. This validates the well established result that the fist-price auction system is not incentive compatible, i.e. the agents may have an incentive to not bid truthfully in the auction.

Repeated Vickery Auctions The auction rule for $\mathcal{P}_1$, when $\mathcal{P}_1$ and $\mathcal{P}_2$ bid at u_1 and u_2 respectively, is given as follows:

$$\mathit{VickeryAuction}(u_1, u_2) = \begin{cases} u_1 - u_2 & \text{if } u_1 \geq u_2 \\ 0 & \text{otherwise} \end{cases}$$

The set of Nash equilibria computed by ComputeNash is {(Always(v_1, v_2),All(v_2, v_1)), (All(v_1, v_2), All(v_2, v_1))}. Therefore, $\mathcal{P}_1$ does not have a dominant strategy if she does not always bid at v_1. Therefore, unlike repeated First-Price auctions, repeated Vickery auctions are incentive compatible.

4.4 Summary

This chapter presents the key technical contribution of this work — an algorithm for computing Nash equilibria in a regular repeated game: ComputeNash. Algorithm ComputeNash executes by searching for a witness strategy profile for every strategy profile in the input game. The algorithm for determining whether a profile has a witness differs for deterministic and non-deterministic games, since the number of runs in strategy profiles of both games differ. As a result, we observe an increase by one exponential factor in the size of the input game in the worst case analysis (an upper bound) of ComputeNash on deterministic games, and non-deterministic games. It runs in polynomial and exponential time in the size of the input game respectively.

The case studies on the iterated prisoner's dilemma, repeated auctions, and the Bitcoin protocol demonstrate the utility of our automated approach, and provide a proof of concept for validation of regular repeated games as an appropriate model for repeated games. In our experiments, we first modeled the games/systems we considered as regular repeated games, and then tallied our observations with previously known results. Our observations on the Nash equilibria and best response strategy in our case studies corroborate with previously known results on these games/systems.

ALGORITHM 2: Witness(i, StrategyProfile(1), StrategyProfile(2))

Input: Agent idendity index i, StrategyProfile(1) and StrategyProfile(2) in automaton form

Output: *True* if StrategyProfile(2) is witness of StrategyProfile(1) w.r.t $\mathcal{P}_i$, *False otherwise*

Deterministic Case:

$\hat{S} \leftarrow$ AugmentWt(StrategyProfile1)

$\hat{T} \leftarrow$ AugmentWt(StrategyProfile2)

$\hat{\mathcal{A}}^{prod} \leftarrow$ MakeProduct($\hat{S}, \hat{T}$)

if $\hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}} \neq \emptyset$ **then**

  return True

else

  return False

Non-deterministic Case:

$\hat{S} \leftarrow$ AugmentWtAndLabel(StrategyProfile(1))

$\hat{T} \leftarrow$ AugmentWtAndLabel(StrategyProfile(2))

$\hat{\mathcal{A}}^{prod} \leftarrow$ MakeProduct($\hat{S}, \hat{T}$)

$Witness \leftarrow \hat{\mathcal{A}}^{prod} \cap_i^* \mathcal{A}_{\succ_{DS(d)}}$

$\overline{Nash} \leftarrow$ ProjectFirst($Witness$)

if $\overline{Nash} = \hat{S}$ **then**

  return True

else

  return False

Chapter 5

ω-Regular Comparator

In Chapter 4, we noted that we require a mechanism to compare the aggregate payoff along payoff sequences under various aggregate functions. However, computation of the aggregate along infinite sequences may pose several obstacles, including (1). One cannot perform explicit aggregation of an infinite length sequence to compute its actual value – to calculate the discounted sum of an infinite sequence, (2). the aggregate value of the payoff sequence may not exists – limit average of a sequence does not necessarily converge, etc.

However, our objective is simpler than computation – it is comparison. For our purpose it is sufficient to compare the aggregate of the sequences *without* performing their explicit computation, if possible. To this end we introduce *ω-regular comparators*. The ω-regular comparator for aggregate function f is used to determine whether $f(A) < f(B)$ for number sequences A and B. More concretely, an ω-regular comparator for aggregate function f is a Büchi automaton that accepts a pair of sequences (A, B) iff $f(A) < f(B)$. An ω-regular comparator exploits the properties of the aggregate function f to determine the relationship between $f(A)$ and $f(B)$ by *simply reading* the sequences A and B simultaneously. Note that the existence of a comparator automaton for $f(A) < f(B)$, automatically implies the existence of a comparator for $f(B) < f(A)$, their respective negations i.e. $f(A) \geq f(B)$ and $f(B) \geq f(A)$, and equality. Therefore, the existence of a comparator automaton for any one inequality ensures the existence of a comparator for all other inequality and equality relationships.

In this chapter, we look at two aggregate functions – discounted sum and limit

average. Let A be an infinite number sequence. The discounted sum for discount factor $d > 1$ is $DS(A, d) = \Sigma_{i=0}^{\infty} a_i/d^i$. The limit average of a sequence A is $LA(A) = \lim_{T\to\infty} \frac{1}{T} \cdot \Sigma_{i=0}^{T} a_i$. We show a complete construction of an ω-regular comparator for the discounted sum aggregate function (Section 5.2), and present partial results for the same with the limit average comparator in Section 5.3.

Since we draw upon number sequences that arise from repeated regular games or similar finite state machines, we confine our interest to *ω-regular number sequences* i.e. number sequences that are derived from a Büchi automaton over a finite alphabet of rational numbers.

In the sequel, we assume all payoffs to be non-negative integers. This assumption is justified as our objective is to *compare* discounted sum/limit average of rational number sequences over a finite set of rational number alphabet. Rational-valued weights can be normalized to non-negative integers using a linear transformation that does not affect the result of the above comparisons.

The rest of this chapter is organized as follows: Section 5.1 discusses previous work on discounted sum and limit average with finite state machines. The comparator automaton for discounted sum and limit average are given in Section 5.2 and Section 5.3 respectively.

5.1 Prior work

Finite state machines with weights on accepting words have been studied previously as weighted automata [29, 30], quantitative languages [19] etc. Words with weights find diverse applications. Some of them include text, speech and image processing [43], in probabilistic system [6] and in probabilistic automata [55], and in verification of quantitative systems. Questions arising from these domains have lead to conceptual and algorithmic development in quantitative variants of emptiness, universality, equivalence, and language-inclusion problems in the context of (finite and infinite) words with weights over various aggregate functions [3, 19,

44].

The game-theoretic setting gives rise to a distinct but important problem: the ability to compare aggregate along two infinite sequences *regularly* i.e. with an automaton. To the best of our knowledge, the problem of comparison of the aggregate value along two sequences has not been explored by the regular quantitative analysis community. ω-regular comparators work towards filling this gap in the analysis of quantitative words.

First we present the discounted sum comparator: comparator when sequence aggregate is computed using the discounted sum function. Discounted sum automata have been considered previously in various other contexts [13**?** , 14]. The key idea for comparison under discounted sum aggregate function with discount factor d is to treat the values as a number in base d. This is akin to the treatment of real numbers in regular real analysis [21].

5.2 Discounted sum comparator

We now describe the construction of an automaton that enables us to compare discounted sum of two number sequences. We introduce some useful notation. For an infinite sequence A and a rational $d = \frac{p}{q} > 1$, the discounted sum of A with respect to discount factor d, denoted by $DS(A, d)$, is defined as $\Sigma_{i=0}^{\infty} A[i]/d^i$, where $A[i]$ denotes i-th element of A. Let $\mathcal{A}_{>_{DS(d)}}$ denote the automaton that accepts (A, B) iff $DS(A, d) < DS(B, d)$. Before diving into the details of construction, we first discuss the key ideas behind our construction.

Key ideas We construct an automaton $\mathcal{A}_{>_{DS(d)}}$ for a given rational $d = \frac{p}{q} > 1$ that accepts a pair of positive number sequences (A, B) iff $DS(A, d) < DS(B, d)$. The core idea behind this construction is to enable $\mathcal{A}_{>_{DS(d)}}$ to non-deterministically guess a sequence C s.t $DS(B, d) = DS(A, d) + DS(C, d)$ s.t. $DS(C, d) > 0$. Note that such a sequence C exists if and only if $DS(B, d) > DS(A, d)$. Note that every element of A

and B has to be bounded but *may not necessarily* be to smaller than d.

Now consider the equation: $DS(B,d) = DS(A,d) + DS(C,d)$. From arithmetic of numbers, we can rewrite it as $B = A + C$, where the operators + and = are defined in base d. If one were to perform addition of A and C in base d, one would also obtain sequence of carry elements, denoted by X henceforth. It is easy to see that following relationship between sequences A, B, C, and X.

Lemma 5.1

Let A, B, C, X be the number sequences, $d = \frac{p}{q} > 1$ be a positive rational such that following invariant holds true:

1. *When $i = 0$, $A[0] + C[0] + X[0] = B[0]$*
2. *When $i \geq 1$, $A[i] + C[i] + X[i] = B[i] + d \cdot X[i-1]$*

Then $DS(B,d) = DS(A,d) + DS(C,d)$.

Proof 5.1 $DS(A,d) + DS(C,d) = \Sigma_{i=0}^{\infty} A[i]\frac{1}{d^i} + \Sigma_{i=0}^{\infty} C[i]\frac{1}{d^i} = \Sigma_{i=0}^{\infty}(A[i] + C[i])\frac{1}{d^i}$. On substituting each $A[i] + C[i]$ based on Equations 1 and 2, we obtain $DS(A,d) + DS(B,d) = (B[0] - X[0]) + \Sigma_{i=1}^{\infty}\ (B[i] + d \cdot X[i-1] - X[i])\frac{1}{d^i}$. On rearranging the elements this is equal to $\Sigma_{i=0}^{\infty} B[i] \cdot \frac{1}{d^i} - \Sigma_{i=0}^{\infty} X[i] \cdot \frac{1}{d^i} + \Sigma_{i=1}^{\infty} d \cdot X[i-1] \cdot \frac{1}{d^i} = \Sigma_{i=0}^{\infty} B[i] \cdot \frac{1}{d^i} = DS(B,d)$. Hence, we have proved that under the conditions imposed by Equation 1 and 2, $DS(A,d) + DS(C,d) = DS(B,d)$.

Hence, an alternate way of determining the difference between B and A is to guess sequences C and X which satisfy the above equations. It is worth noting that computation of i-th element of C and X depends only on i and $(i-1)$th elements of A and B. In addition, since A and B are bounded integer sequences, the difference of $DS(B,d)$ and $DS(A,d)$ is also bounded. Therefore, C and X are also bounded. Furthermore, we can prove that C and X can both be constructed from a fixed finite set of rational numbers as long as A and B are both bounded integer sequences (We defer the formal proof to the supplemental material). Together, these three observation suggest that we can construct a Büchi automaton $\mathcal{A}_{>_{DS(d)}}$ in which (i). state

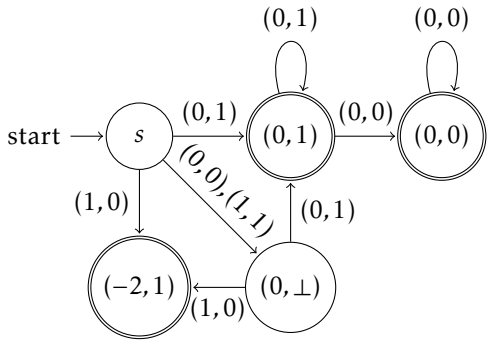


Figure 5.1 : Snippet of $\mathcal{A}_{\succ_{DS(2)}}$.

are represented by (x,c) where x and c range over all possible values (which are finite) of elements of X and C, (ii) the start state is a special state s, (iii) transitions from the start state $s \xrightarrow{(a,b)} (x,c)$ satisfy Equation 1, $a+c+x=b$, (iv) all other transitions $(x_1,c_1) \xrightarrow{(a,b)} (x_2,c_2)$ satisfies Equation 2, $a+c_2+x_2 = b+d\cdot x_1$, and (v) word rejection by termination. From Lemma 5.2, we can see that automaton $\mathcal{A}_{\succ_{DS(d)}}$ will accept (A,B) iff $DS(B,d) = DS(A,d)+DS(C,d)$, where C is obtained from the state sequence of (A,B).

However, there is caveat: Merely satisfying $DS(B,d) = DS(A,d)+DS(C,d)$ does not guarantee $DS(C,d) > 0$, which is required to ensure $DS(B,d) > DS(A,d)$. To handle this corner case, we add additional states represented by $(x,\perp)$, where x ranges over all possible (finite) values of X. We explain the reason for this at the end of the construction. The formal construction of automaton $\mathcal{A}_{\succ_{DS(d)}}$ is given below:

Construction We provide a construction for $\mathcal{A}_{\succ_{DS(d)}}$.

Take $maxC = \max\cdot\frac{d}{d-1}$ and $maxX = 1+\frac{\max}{d-1}$. $\mathcal{A}_{\succ_{DS(d)}} = (S,\Sigma,\delta_d,Init,\mathcal{F})$ where

- $S = Init \cup S_{acc} \cup S_{\perp}$ where

  $Init = \{s\}$,

  $S_{acc} = \{(x,c) \mid |x| \le maxX, 0 \le c \le maxC\}$, and

$S_\perp = \{(x, \perp) || x| \leq maxX\}$
where x and c are of the form $\frac{m}{q}$ for integral values of m.

- $\Sigma = \{(a, b) : 0 \leq a, b \leq \max\}$ where a and b are integers.
- δ_d is defined as follows:
  1. Transitions from start state s:
     i $(s, (a, b), (x, c))$ for all $(x, c) \in S_{acc}$ s.t. $a + x + c = b$ and $c \neq 0$.
     ii $(s, (a, b), (x, \perp))$ for all $(x, \perp) \in S_\perp$ s.t. $a + x = b$
  2. Transitions within $S_\perp$: $((x, \perp), (a, b), (x', \perp))$ for all $(x, \perp), (x', \perp) \in S_\perp$, if $a + x' = b + d \cdot x$
  3. Transitions within S_{acc}: $((x, c), (a, b), (x', c'))$ for all $(x, c), (x', c') \in S_{acc}$ where $c' < d$, if $a + x' + c' = b + d \cdot x$
  4. Transition between $S_\perp$ and S_{acc}: $((x, \perp), (a, b), (x', c'))$ for all $(x, \perp) \in S_\perp$, $(x', c') \in S_{acc}$ where $0 < c' < d$, if $a + x' + c' = b + d \cdot x$
- $Init = \{s\}$
- $\mathcal{F} = S_{acc}$

Note that c only takes non-negative values. Therefore the requirement for $DS(C, d) > 0$ is to ensure that the resulting sequence C consists of at least one non-zero value. In the construction above, transitions into and out of states $(x, \perp)$ satisfy Equation 1 or 2 (depending on whether transition is from start state s) where $\perp$ is treated as $c = 0$. Any valid execution of (A, B) can enter the accepting states S_{acc} from the start state s and states in $S_\perp$ only if the execution witnesses a non-zero value of c that satisfies Equation 1 and 2 respectively. This ensures that (A, B) is accepted only if $DS(C, d) > 0$.

Analysis We now prove that the construction given above results in an automaton that can compare the discounted sum of two bounded integer number sequences for a given discount factor $d > 0$.

First, we show that each accepting run for word (A, B) in $\mathcal{A}_{>_{DS(d)}}$ satisfies the condition that $DS(B, d) > DS(A, d)$. We prove this by showing that for each accepting run of (A,B) one can construct a number sequence C s.t. $DS(B, d) = DS(A, d) + DS(C, d)$ and $DS(C, d) > 0$.

We make the following crucial observation about the structure of number sequences A, B and C.

Lemma 5.2
Let A, B, C, X be the number sequences, $d = \frac{p}{q} > 1$ be a positive rational such that following invariant holds true:

1. *When $i = 0$, $B[0] = A[0] + X[0] + C[0]$*
2. *When $i \geq 1$, $B[i] + d \cdot X[i-1] = A[i] + X[i] + C[i]$.*

Then $DS(B, d) = DS(A, d) + DS(C, d)$.

Proof 5.2 $DS(A,d)+DS(C,d) = \Sigma_{i=0}^{\infty} A[i]\frac{1}{d^i} + \Sigma_{i=0}^{\infty} C[i]\frac{1}{d^i} = \Sigma_{i=0}^{\infty}(A[i]+C[i])\frac{1}{d^i} = (B[0]-X[0]) + \Sigma_{i=1}^{\infty}(B[i]+d\cdot X[i-1]-X[i])\frac{1}{d^i} = (B[0]-X[0]) + \Sigma_{i=1}^{\infty}(B[i]+d\cdot X[i-1]-X[i])\frac{1}{d^i} = \Sigma_{i=0}^{\infty} B[i]\cdot\frac{1}{d^i} - \Sigma_{i=0}^{\infty} X[i] + \Sigma_{i=0}^{\infty} X[i] = \Sigma_{i=0}^{\infty} B[i]\cdot\frac{1}{d^i} = DS(B,d)$

In Lemma 5.3, we show that we can derive sequences C and X that satisfy the above mentioned constraints from the values of c and x from state (x, c) present in an accepting state sequence of word (A, B) in the constructed automaton $\mathcal{A}_{>_{DS(d)}}$. In Lemma 5.4, we show that sequence C generated by collecting c from the state sequence of the accepting run is such that $DS(C, d) > 0$. We bind the above two observations in Corollary 5.1 to show that (A, B) is accepted by $\mathcal{A}_{>_{DS(d)}}$ only if $DS(B, d) > DS(A, d)$.

Lemma 5.3

For every infinite run of (A, B) *in* $\mathcal{A}_{\succ_{DS(d)}}$*, there exists a sequence* C*, s.t* $DS(B, d) = DS(A, d) + DS(C, d)$.

Proof 5.3 Define functions $XVal$ and $CVal : S \mapsto \mathbb{Z}$ where S is the set of all states in $\mathcal{A}_{\succ_{DS(d)}}$ as follows:

$$CVal(state) = \begin{cases} 0 & \text{if } state = s \in Init \\ c & \text{if } state = (x, c) \in S_{acc} \\ 0 & \text{if } state = (x, \perp) \in S_0 \end{cases}$$

$$XVal(state) = \begin{cases} 0 & \text{if } state = s \in Init \\ x & \text{if } state = (x, c) \in S_{acc} \\ x & \text{if } state = (x, \perp) \in S_0 \end{cases}$$

Consider an infinite run of word (A, B) in $\mathcal{A}_{\succ_{DS(d)}}$. Let $\{state_i\}_i$ be the sequence of states visited by this run. Construct sequences C and X s.t. $C[i] = CVal(state_{i+1})$ and $X[i] = XVal(state_{i+1})$.

When $i = 0$, then either $X[0]$ and $C[0]$ derive their values from either $(x, c) \in S_{acc}$, or from $(x, \perp) \in S_0$. We consider the cases separately below:

1. From $(x, c) \in S_{acc}$: In this case, the 0-th transition is $s \xrightarrow{(a,b)} (x, c)$. From construction, we know that $a + x + c = b$. Since, $A[0] = a$, $B[0] = b$, $C[0] = c$, and $X[0] = x$, we have $A[0] + X[0] + C[0] = B[0]$.

2. From $(x, \perp) \in S_0$: In this case, the 0-th transition is $s \xrightarrow{(a,b)} (x, \perp)$. From construction, we know that $a + x = b$. Since, $A[0] = a$, $B[0] = b$, $C[0] = 0$, and $X[0] = x$, we have $A[0] + X[0] + C[0] = B[0]$.

Therefore, in either case, the first condition on Lemma 5.2 is satisfied.

For $i \geq 1$, transitions are either within S_0, or within S_{acc}, or from S_0 to S_{acc}. We will consider each of these cases separately:

1. Within S_0: Transitions are of the form $(x, \perp) \xrightarrow{(a,b)} (x', \perp)$ where $a + x' = b + d \cdot x$. Suppose this is the i-th transition in a run (for $i \geq 1$). In this case, $A[i] = a$, $B[i] = b$, $C[i] = CVal(x') = 0$, $X[i] = XVal(x') = x'$, and $X[i-1] = XVal(x) = x$. So, $B[i] + d \cdot X[i-1] = A[i] + X[i] + C[i]$.

2. Within S_{acc}: Transitions are of the form of $(x, c) \xrightarrow{(a,b)} (x', c')$. Suppose this is the ith transition in a run (for $i \geq 1$). From construction we know that $a + x' + c' = b + d \cdot x$. Here $A[i] = a$, $B[i] = b$, $C[i] = c$, $X[i] = x$, and $X[i-1] = x'$. Therefore, $B[i] + d \cdot X[i-1] = A[i] + X[i] + C[i]$.

3. From S_0 to S_{acc}: Transitions are of the form $(x, \perp) \xrightarrow{(a,b)} (x', c')$ s.t. $a + x' + c' = b + d \cdot x$. Suppose this is the i-th transition, then $A[i] = a$, $B[i] = b$, $X[i-1] = x$, $X[i] = x'$, $C[i] = c'$. Therefore, $B[i] + d \cdot X[i-1] = A[i] + X[i] + C[i]$.

In each of these cases, when $i \geq 1$, the second condition in Lemma 5.2 is satisfied.

Together we see that, for any infinite run of (A, B), we can construct sequences C and X, s.t. using Lemma 5.2 we can prove that $DS(B, d) = DS(A, d) + DS(C, d)$.

Lemma 5.4

For every accepting run of (A, B) *in* $\mathcal{A}_{>_{DS(d)}}$, $DS(C, d) > 0$.

Proof 5.4 For a run to be accepting in $\mathcal{A}_{>_{DS(d)}}$, it must visit at least one state in S_{acc} infinitely often. Note that once a run visits a state in S_{acc}, it only visits states in S_{acc} since there are no transitions out of S_{acc}. Since there are finitely many S_{acc}, the sufficient condition for a state to be accepting is that a run must enter S_{acc}. There are two ways of entering S_{acc}, either via $s \xrightarrow{(a,b)} (x, c)$ or via $(x, \perp) \xrightarrow{(a,b)} (x', c')$. Suppose this is the i-th transition in the accepting run, then $C[i] \neq 0$. Also, we know that for all j, $C[j] \geq 0$. Hence, $DS(C, d) \geq C[i] > 0$.

Corollary 5.1

(A, B) *in* $\implies DS(B, d) > DS(A, d)$.

Proof 5.5 From Lemma 5.3, and Lemma 5.4, it is clear that for all accepting runs of (A,B) in $\mathcal{A}_{\succ_{DS(d)}}$, $DS(B,d) > DS(A,d)$.

So far we have shown that every accepting word (A,B) of $\mathcal{A}_{\succ_{DS(d)}}$ satisfies the condition that $DS(B,d) > DS(A,d)$. We want to show the other direction: for every pair of non-negative sequences A,B if $DS(B,d) > DS(A,d)$ then (A,B) is accepted by automaton $\mathcal{A}_{\succ_{DS(d)}}$.

For this part of the proof we assume that sequences A, B are bounded by μ. We also introduce some useful notation. Let $DS^{-}(B,A,d,i) = \Sigma_{j=0}^{i}(B[j]-A[j])\cdot\frac{1}{d^j}$. Also, let $DS^{-}(B,A,d,\cdot) = \Sigma_{j=0}^{\infty}(B[j]-A[j])\cdot\frac{1}{d^j} = DS(B,d) - DS(A,d)$. Define $\mu_C = \mu\cdot\frac{d}{d-1}$, $\mu_X = 1+\frac{\mu}{d-1}$. We define the *residual function* $Res : \mathbb{N}\cup\{0\}\mapsto\mathbb{R}$ as follows:

$$Res(i) = \begin{cases} DS^{-}(B,A,d,\cdot) - \lfloor DS^{-}(B,A,d,\cdot)\rfloor & \text{if } i = 0 \\ Res(i-1) - \lfloor Res(i-1)\cdot d^i\cdot q\rfloor\cdot\frac{1}{d^i\cdot q} & \text{otherwise} \end{cases}$$

Accordingly, we define function $C : \mathbb{N}\cup\{0\}\mapsto\mathbb{Z}$ as follows:

$$C(i) = \begin{cases} \lfloor DS^{-}(B,A,d,\cdot)\rfloor & \text{if } i = 0 \\ \lfloor Res(i-1)\cdot d^i\cdot q\rfloor/q & \text{otherwise} \end{cases}$$

Intuitively, $C(i)$ is computed by *stripping off* the value of the i-th digit in a representation of $DS^{-}(B,A,d,\cdot)$ in base d. $C(i)$ denotes the numerical value of the i-th position of the difference between B and A. The residual function denotes the numerical value of the difference remaining after assigning the value of $C(i)$ until that i. Furthermore, we define $CSum(i) = \Sigma_{j=0}^{i}C(i)\cdot\frac{1}{d^j}$. and $X : \mathbb{N}\cup\{0\}\mapsto\mathbb{R}$ s.t $X(i) = (DS^{-}(B,A,d,i) - CSum(i))\cdot d^i$.

Lemma 5.5

For all $i \geq 0$, $Res(i) = DS^{-}(B,A,d,\cdot) - CSum(i)$.

Proof 5.6 Proof by simple induction on the definitions of functions *Res* and *C*.

Lemma 5.6

When $DS^{-}(B,A,d,\cdot) \geq 0$, for all $i \geq 0$, $0 \leq Res(i) < \frac{1}{d^i}$.

Proof 5.7 Since, $DS^{-}(B,A,d,\cdot) \geq 0$, $Res(0) = DS^{-}(B,A,d,\cdot) - \lfloor DS^{-}(B,A,d,\cdot) \rfloor \geq 0$ and $Res(0) = DS^{-}(B,A,d,\cdot) - \lfloor DS^{-}(B,A,d,\cdot) \rfloor < 1$. Specifically, $0 \leq Res(0) < 1$.

Suppose for all $i \leq k$, $0 \leq Res(i) < \frac{1}{d^i}$. We show this is true even for $k+1$.

Since $Res(k) \geq 0$, $Res(k) \cdot d^{k+1} \cdot q \geq 0$. Let $Res(k) \cdot d^{k+1} \cdot q = x + f$, for integral $x \geq 0$, and fractional $0 \leq f < 1$. Then $Res(k+1) = \frac{x+f}{d^{k+1} \cdot q} - \frac{x}{d^{k+1} \cdot q} \implies Res(k+1) < \frac{1}{d^{k+1} \cdot q}$. Since $q \geq 1$, $Res(k+1) < \frac{1}{d^{k+1}}$.

Also, $Res(k+1) \geq 0$ since $a - \lfloor a \cdot b \rfloor / b \geq 0$ for all positive values of a and b.

Lemma 5.7

When $DS^{-}(B,A,d,\cdot) \geq 0$, for $i = 0$, $0 \leq C(0) \leq \mu_C$, and for $i \geq 1$, $0 \leq C(i) < d$.

Proof 5.8 Since both A and B are non-negative bounded number sequences, maximum value of $DS^{-}(B,A,d,\cdot)$ is when $B = \{\mu\}_i$ and $A = \{0\}_i$. In this case $DS^{-}(B,A,d,\cdot) = \mu_C$. Therefore, $0 \leq C(0) \leq \mu_C$.

From Lemma 5.6, we know that for all i, $0 \leq Res(i) < \frac{1}{d^i}$. Alternately, when $i \geq 1$, $0 \leq Res(i-1) < \frac{1}{d^{i-1}} \implies 0 \leq Res(i-1) \cdot d^i < \frac{1}{d^{i-1}} \cdot d^i \implies 0 \leq Res(i-1) \cdot d^i < d \implies 0 \leq \lfloor Res(i-1) \cdot d^i \rfloor < d \implies 0 \leq C(i) < d$.

Corollary 5.2

When $DS^{-}(B,A,d,\cdot) \geq 0$, $Range(C)$ is finite.

Proof 5.9 From definition of C, we know that $C(0)$ takes integral values only. Further, from Lemma 5.7, we know that $C(0)$ is bounded. Hence, $C(0)$ takes finitely many values.

For $i > 0$, from definition of C it is clear that $C(i)$ is of the form $\frac{m}{q}$ for integral values of m. Lemma 5.7 shows that each C(i) is bounded by a fixed constant. Hence there are finitely many values of $C(i)$ for all other values of i.

Together, the above two show that $Range(C)$ is finite.

Lemma 5.8

When $DS^{-}(B,A,d,\cdot) \geq 0$, then for all $i \geq 0$, $|X(i)| \leq \mu_X$.

Proof 5.10 From definition of X, we know that $X(i) = (DS^-(B,A,d,i) - CSum(i)) \cdot d^i \implies X(i) \cdot \frac{1}{d^i} = DS^-(B,A,d,i) - CSum(i)$. From Lemma 5.5 we get $X(i) \cdot \frac{1}{d^i} = DS^-(B,A,d,i) - (DS^-(B,A,d,\cdot) - Res(i)) \implies X(i) \cdot \frac{1}{d^i} = Res(i) - (DS^-(B,A,d,\cdot) - DS^-(B,A,d,i)) \implies X(i) \cdot \frac{1}{d^i} = Res(i) - (\Sigma_{j=i+1}^{\infty}(B[j] - A[j]) \cdot \frac{1}{d^j}) \implies |X(i) \cdot \frac{1}{d^i}| \leq |Res(i)| + |(\Sigma_{j=i+1}^{\infty}(B[j] - A[j]) \cdot \frac{1}{d^j})| \implies |X(i) \cdot \frac{1}{d^i}| \leq |Res(i)| + \frac{1}{d^{i+1}} \cdot |(\Sigma_{j=0}^{\infty}(B[j+i+1] - A[j+i+1]) \cdot \frac{1}{d^j})| \implies |X(i) \cdot \frac{1}{d^i}| \leq |Res(i)| + \frac{1}{d^{i+1}} \cdot |\mu_C|$. From Lemma 5.6, this implies $|X(i) \cdot \frac{1}{d^i}| \leq \frac{1}{d^i} + \frac{1}{d^{i+1}} \cdot |\mu_C| \implies |X(i)| \leq 1 + \frac{1}{d} \cdot |\mu_C| \implies |X(i)| \leq 1 + \frac{\mu}{d-1} \implies |X(i)| \leq \mu_X$

Corollary 5.3

When $DS^-(B,A,d,\cdot) \geq 0$, $Range(X)$ is finite.

Proof 5.11 From expanding $X(i)$, we see that each $X(i)$ is of the form $\frac{m}{q}$ for integral values of m. Since $X(i)$ is bounded (see Lemma 5.8), this proves that $Range(X)$ is finite.

We overload notation, and define number sequences C and X s.t. for all $i \geq 0$, $C[i] = C(i)$ and $X[i] = X(i)$.

Lemma 5.9

When $DS^-(B,A,d,\cdot) \geq 0$, then A, B, C and X satisfy the following invariant

1. $B[0] = A[0] + C[0] + X[0]$
2. *For $i \geq 1$,* $B[i] + d \cdot X[i-1] = A[i] + C[i] + X[i]$

Proof 5.12 We prove this by induction on i using definition of function X.

When $i = 0$, then $X[0] = X(0) = DS^-(B,A,d,0) - CSum(0) \implies X[0] = B[0] - A[0] - C[0] \implies B[0] = A[0] + C[0] + X[0]$.

When $i = 1$, then $X[1] = X(1) = (DS^-(B,A,d,1) - CSum(1)) \cdot d = (B[0] + B[1] \cdot \frac{1}{d} - (A[0] + A[1] \cdot \frac{1}{d}) - (C[0] + C[1] \cdot \frac{1}{d})) \cdot d \implies X[1] = B[0] \cdot d + B[1] - (A[0] \cdot d + A[1]) - (C[0] \cdot d + C[1])$. From the above we obtain $X[1] = d \cdot X[0] + B[1] - A[1] - C[1] \implies B[1] + d \cdot X[0] = A[1] + C[1] + X[1]$.

Suppose the invariant holds true for all $i \leq n$, we show that it is true for $n+1$. $X[n+1] = (DS^{-}(B,A,d,n+1) - CSum(n+1)) \cdot d^{n+1} \implies X[n+1] = (DS^{-}(B,A,d,n) - CSum(n)) \cdot d^{n+1} + (B[n+1] - A[n+1] - C[n+1]) \implies X[n+1] = X[n] \cdot d + B[n+1] - A[n+1] - C[n+1] \implies B[n+1] + X[n] \cdot d = A[n+1] + C[n+1] + X[n+1]$.

We construct state sequence $S = \{s\}_i$ as follows: Suppose $j \geq 0$ is the first instance where $C[j] > 0$

$$s_i = \begin{cases} s & \text{if } i = 0 \\ (X[i-1], \perp) & \text{if } 0 < i \leq j \\ (X[i-1], C[i-1]) & \text{if } i > j \end{cases}$$

Lemma 5.10

When $DS^{-}(B,A,d,\cdot) \geq 0$, then S is a valid run of (A,B) in $\mathcal{A}_{\succ_{DS(d)}}$.

Proof 5.13 This is straight forward from Lemma 5.9 and construction of $\mathcal{A}_{\succ_{DS(d)}}$. Note that the construction makes use of finiteness of $Range(C)$ (See Corollary 5.2) and $Range(X)$ (See Corollary 5.3)

Corollary 5.4

Let A and B be two non-zero bounded (by μ) integer sequences. Suppose $DS(B,d) > DS(A,d)$, then (A,B) has an accepting run in $\mathcal{A}_{\succ_{DS(d)}}$.

Proof 5.14 For the given sequences A and B, generate sequences C and X as defined by functions C and X respectively. Then from Lemma 5.2 and Lemma 5.9, we know that $DS(C,d) > 0$. Since $0 \leq C[i]$, there exists at least on i where $C[i] \neq 0$. Let j be the first such index. Then from S we see that the state run moves into states in S_{acc} in the j-th transition, and remains in S_{acc} thereon. Therefore this run an accepting run for (A,B) in $\mathcal{A}_{\succ_{DS(d)}}$.

Theorem 5.1
Let $d = \frac{p}{q} > 1$ be a positive rational, and $\mu \in \mathbb{Z}^+$. Suppose A and B are non-negative integer sequences bounded by μ. Then Büchi automaton $\mathcal{A}_{>_{DS(d)}}$ accepts (A, B) if and only if $DS(A, d) < DS(B, d)$.

Proof 5.15 Immediate from Corollary 5.1 and Corollary 5.4.

5.3 Limit average comparator

The limit average of a sequence A is defined as follows: $\lim_{T\to\infty} \frac{1}{T} \cdot \Sigma_{i=0}^{T} a_i$. The limit average saga is much more complex that the discounted sum case. Some reasons are (1). the limit average is not defined for all sequences[58] (2). we have not been able to identify any *regular* property i.e. any property that will enable the construction of an automaton with finitely many states. In the case of discounted sum aggregate function, we were able to identify properties such as a finite number set for carry-bits and the difference sequence bits (sequences X and C respectively) etc.

We have had partial success in designing a comparator for the limit average aggregate function. We have been able to identify a special class of language (for number sequence) for which it is possible to determine the value of the limit average of sequences in it without performing the computation explicitly.

We begin with some useful definitions. A language L over infinite words is said to be a *bounded language* (BL) if all its words are derived by concatenation of finite words present in a set B. B is said to be the *building block* of language L, and L is denoted by B^ω. A building block B is said to be a *same average building block* (SABB) if all words in B have the same average. A bounded language L is said to be a *same limit-average bounded language* (SABL) if the limit average is defined for all words in L, and all words have the same limit average.

We will show a correspondence between the limit average of words in bounded language L and the average of words in its building block B, where $L = AB^\omega$ and

A is a regular language. The key idea is that in infinite words the limit average is determined by finite sequences that appear infinitely often i.e. in B and not by the words in A. Hence, we can shift focus only on the relationship between the average in B and the limit average in $L = AB^\omega$. We observe that

Lemma 5.11
Let $L = B^\omega$ s.t. each word in L has a well-defined limit average. Then each word in B must have the same average.

Proof 5.16 (Proof sketch) Suppose not, then there exists two words w_1 and w_2 in B s.t. their averages are given by a_1 and a_2 respectively where $a_1 \neq a_2$, and their lengths are given by l_1 and l_2. WLOG, let $a_1 < a_2$. Then we can construct a word such that the average of its prefixes will keep fluctuating between $\frac{3a_1+a_2}{4}$ and $\frac{a_1+a_2}{2}$. The idea is to construct a word in $(w_1^* \cdot w_2^*)^\omega$ as follows:

1. Begin with the smallest $x^1 = w_1^{l_2} w_2^{l_1}$. The average of this word is $\frac{a_1+a_2}{2}$.
2. Append $x^2 \in (w_1^* \cdot w_2^*)$ to x^1 s.t. the number of w_1 and w_2 in x^1x^2 (n and m respectively) satisfy the following constraint: $\frac{m}{n} \approx \frac{3l_2}{l_1}$. The average of x^1x^2 is $\approxeq \frac{3a_1+a_2}{4}$.
3. Append $x^3 \in (w_1^* \cdot w_2^*)$ to x^1x^2 s.t. the number of w_1 and w_2 in $x^1x^2x^3$ (n and m respectively) satisfy the following constraint: $\frac{m}{n} \approx \frac{l_2}{l_1}$. The average of $x^1x^2x^3$ is $\approxeq \frac{a_1+a_2}{2}$.

Keep appending $x^1 \dots x^{2k+1}$ with x^{2k+2} and x^{2k+3} s.t. the conditions in Step 2 and Step 3 are satisfied. In the resulting word $x = x^1x^2\dots$, the average of prefixes will keep fluctuating between $\frac{3a_1+a_2}{4}$ and $\frac{a_1+a_2}{2}$. Hence the limit average of x will not be well defined. But, this is contradictory to our assumption, that the limit average of each word in L is well defined.

Hence, we have shown via contradiction that all words in B must have the same average.

Corollary 5.5

Let $L = AB^\omega$ s.t. each word in L has a well-defined limit average. Then each word in B must have the same average.

Proof 5.17 (Proof sketch) Since words in A are of finite length, the limit average of each word is really determined by the word segment present in B. The rest follows directly from Lemma 5.11.

However, for a language $L = B^\omega$ to be a SABL it is not sufficient for the building block B to be an SABB. As an example, Let $B \subseteq 0^*1^*$ s.t each building block $b \in B$ has average of $0 < k < 1.$, then $L = B = \{0,1\}^*$. We will construct a word in B^ω for which the limit average doesn't exists. Take a long enough $w \in B^*$. Clearly$Avg(w[n]) = k$. Let its length be l_n. Take a string of l_n 0s. Since l_n is large enough, and $0 < k < 1$, we can find l_m number of 1s s.t. $0^{l_m}1^{l_n} \in B$ has an average of k. Then $Avg(w[n]0^{l_m}) = \frac{k}{2}$, but $Avg(w[n]0^{l_m}1^{l_n}) = k$. Now extend $w[n]0^{l_m}1^{l_n}$ the way we extended w. We will notice that the average will keep fluctuating between k and $\frac{k}{2}$. So, the limit average of this word will not exist.

In what follows we show some restrictions on the SABL B under which the resulting bounded language $L = B^\omega$ is an SABL.

Lemma 5.12

Let B be an SABB. Let all words of B be of bounded length. Then the corresponding bounded language $L = B^\omega$ is an SABL.

Furthermore, limit average of each word in L is the same as the average of each word in B.

Proof 5.18 Let B be a set of words where the length of the word w_i is given by l_i, and the average of all words is a. Let l be the maximum length of all words in B.

Let $w = a_0a_1\cdots \in B^\omega$. Let $w[n]$ denote the n-length prefix of w. Since $w \in B^\omega$, each $w[n]$ is represented by $w_0\ldots\ w_k a_{n-p}a_{n-p+1}\ldots a_n$ for some $p < l_m$ and no word $w_i \in B$ is a prefix of $a_{n-p+1}\ldots a_n$.

Now $LA(w) = \lim_{n\to\infty} Avg(w[n])$. Since, $Avg(w[n]) = \frac{m\cdot\Sigma_{j=0}^{k} l_k + \Sigma_{j=n-p+1}^{n} val(a_j)}{\Sigma_{j=0}^{k} l_k + p}$. Now, $f_k^l = \frac{m\cdot\Sigma_{j=0}^{k} l_k}{\Sigma_{j=0}^{k} l_k + l} \leq \frac{m\cdot\Sigma_{j=0}^{k} l_k + \Sigma_{j=n-p+1}^{n} val(a_j)}{\Sigma_{j=0}^{k} l_k + p} \leq \frac{m\cdot\Sigma_{j=0}^{k} l_k + m\cdot l}{\Sigma_{j=0}^{k} l_k} = f_k^u$.

Since l is fixed, as $k \to \infty$, $l \ll \Sigma_{j=0}^{k} l_k$. Therefore, $f_k^l, f_k^u \to \frac{m\cdot\Sigma_{j=0}^{k} l_k}{\Sigma_{j=0}^{k} l_k} = m$.

Therefore, $Avg(w[n])$ converges to m [58].

Let B be a finite state automata with weight on all transitions. The weight on any finite path in B is given by the sum weights along transitions along the path. Average of weight along a path is calculated as the ratio of weight the of path and length of the path. If the average weight of all simple cycles in B is the same, we call B a *regular same average building block* (RSABB). The corresponding bounded language $L = B^\omega$ consists of words that are present in B when B is treated like a Büchi automaton. Note that we have overload the notation of B^ω. However, the definition of L is clear from whether B is an SABB or a RSABB. Hence, all words in L are words of the form $w_1(Cycle)^\omega$ where w_1 is a finite accepting word in finite automaton B, and *Cycle* denotes the language of complex cycles in B. In Lemma 5.13 we will show that when B is a RSABB, the limit average of all words in bounded language L exists, and that the limit average of all words in L is the same . We call bounded language $L = B^\omega$ a *regular same average bounded language* (RSABL).

Lemma 5.13

Let B be a RSABB. Then $L = B^\omega$ is a RSABL.

Proof 5.19 (Proof sketch) Each word w in L is of the form $w_0(Cycle)^\omega$ where w_0 is a finite accepting word in finite automaton B, and *Cycle* denotes the language of complex cycles i.e. of simple cycles in B, possibly nested within one another.

We will argue via induction on the depth of nesting of complex cycles in *Cycle*, that $w \in w_0(Cycle)^\omega$ has limit average a, where a is the average of simple cycles in B. At nested depth 0, $w \in w_0(c_1 + \cdots + c_m)^\omega$ where c_i is a simple cycle in B. Since

w_0 is of finite length, we can ignore its weight in the limit average computation of w. Furthermore, since each c_i is of bounded length and has the same average of a, from Lemma 5.12 we know that the limit average of w exists, and it is equal to a.

Suppose, we know that limit average is true for words with nested depth k. We give the intuition behind the proof for word with nested depth $k+1$. Let word $w \in w_0(Cycle_{k+1})^\omega$, where $Cycle_{k+1}$ is the the set of cycles of maximum nested depth $k+1$. Let C_{k+1} be a cycle of depth $k+1$. We look at the simplest possible such cycle. $C_{k+1} = c_0^f \dots c_k^f c_{k+1} c_k^b \dots c_0^b$ where $c_i^f c_i^b$ (for all i) and c_{k+1} are simple cycles. Not that the average of each $c_i^f \dots c_i^b$ is also a i.e. sub-sequences of cycle C_{k+1} have average an of a. When the word length is sufficiently long, the weight of word $c_0^f \dots c_{i-1}^f$ can be ignored. With this additional piece of information, we can argue as we did in Lemma 5.12 that all words w with nested cycles such as C_{k+1} also have limit average of a. Similarly, it holds true for more complicated cycles of nested depth $k+1$.

Corollary 5.6

Let B be a RSABB, and A be a weighted automaton. Then $L = AB^\omega$ is an RSABL.

Furthermore, the limit average of each word in L is the same as the average of each word in B

Proof 5.20 (Proof sketch) Since A is a set of finite words, the limit average is determined by words in B. From Lemma 5.13, we know that B^ω is a SABL with limit average equal to the average of words in B.

Theorem 5.2

Let B be a building block weighted automaton, and A be any weighted automaton. Then $L = AB^\omega$ is a RSABL iff B is an RSABB.

Furthermore, the limit average of words in L is the same as the average of words in B.

Proof 5.21 This is immediate from Corollary 5.5 and Corollary 5.6.

Theorem 5.2 proves a necessary and sufficient condition for when a bounded language L is an RSABL. Therefore, to check if L is a RSABL, we only need to determine if B is an RSABB. Furthermore, there is an easy algorithm to test if B is an RSABB. The algorithm is based on enumerating all simple cycles in B, and checking if their average is the same.

Extension to general Büchi automaton We extend the ideas developed for SABL to design an incomplete procedure to compute and compare the limit average of ω-number sequences present in a a general Büchi automaton.

Let L be the language of a Büchi automaton $\mathcal{A}_L$. We know that each ω-regular language L can be written as $A_1B_1^\omega + \cdots + A_nB_n^\omega$ [7]. Our goal is to decompose $\mathcal{A}_L$ in a manner that identifies building blocks B_i^ω which are SABBs.

We utilize standard graph theoretic and automata-theoretic procedures to decompose a given Büchi automaton $\mathcal{A}_L$ into its $\Sigma_{i=1}^n A_iB_i^\omega$ form. For this we identify all distinct strongly connected components in the given automaton $\mathcal{A}_L$. We discard all those SCCs which do not contain even a single accepting state of $\mathcal{A}_L$. Suppose S is an SCC with at least one accepting state. We check if all simple cycles present in S have the same average. If not, we discard it, else we keep it. It is sufficient to check for simple cycles in an SCC because every path in an SCC is present on a cycle. Note that the set of SCC is disjoint.

For each SCC make as many copies of it as many accepting states it contains. This ensures that there is a one-one correspondence between SCCs and the accepting state present in it. Hence, the SCC containing accepting state q_i is denotes by B_i.

For each such q_i, construct the automaton that accepts all finite words in $\mathcal{A}_L$ that end in state q_i. Denote this by $\mathcal{A}_i$.

Consider the language $L' = \Sigma_{i=1}^n A_iB_i^\omega$. Every word ending in SCC B_i has a well defined limit average and it is equal to the average of simple cycles present in B_i.

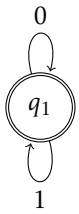


Figure 5.2 : Discarded SCC

Therefore, we can compare an implicitly compute the limit average of all words present in L'.

To compare the limit average of these words, we take the product of L' with itself, identify each q_i with the average of simple cycles passing through it, and assigning state (q_i, q_j) to an accepting state iff average of simple cycles passing through q_i is lesser than that of simple cycles passing through q_j.

Clearly, this procedure is incomplete. Whenever it successfully compares the limit average of two words, it returns the correct result, however, it may also fail to compare limit average of words even though they are well defined. As an example consider the SCC demonstrated in Figure 5.2. The SCC is discarded since the average along simple cycles is different: it is 0 for one, and 1 for the other.

5.4 Summary

This chapter introduces the concept of ω-regular comparator for aggregate functions over infinite length number sequences. An ω-regular comparator for the aggregate function is a Büchi automaton that accepts a pair of sequences (A, B) iff $f(A) < f(B)$. Our motivation for investigating this problem arises from the game-theoretic setting in which comparison of rewards is crucial.

We have succeeding in constructing an ω-regular comparator for the discounted sum aggregate function. We also present partial results, and core insights gained while attempting to do the same for the limit-average comparator.

Chapter 6

Concluding remarks

This chapter summarizes the main contributions of this thesis. The chapter concludes with a brief discussion of some directions to pursue for future research.

6.1 Summary

Multi-agent systems consisting of agents with self-objectives can model various real life interactions ranging from simplistic two-agent systems like tic-tac-toe, to massive online protocols like online auctions etc. An agent in these systems is said to be *rational* if it interacts in a manner to fulfill its objectives. Computation of rational behavior of agents is paramount to analyze properties of these systems. As these systems become more elaborate and the number of agents increase, analysis of these systems becomes more and more difficult. This necessitates the need for the development of algorithmic techniques for computation of rational behavior in systems.

In game-theoretic terms, these systems are called games. This thesis focuses on *repeated games*. These are simultaneous, non-zero sum games, with infinite repetition over bounded rationality. Agent objective in repeated games is to maximize the quantitative reward it receives from a game. Rational agents are said to be in Nash equilibrium if no agent can increase its reward from the game by changing its own behavior only. This thesis focuses on the computation of Nash equilibria in repeated games.

The first contribution of this thesis is to identify a class of repeated games which permit the computation of Nash equilibria: *regular repeated games*. In sum-

mary, a regular repeated game in a finite-state machine based model for repeated games. Here every agent has a finite number of strategies, which are given by finite-state, weighted, non-deterministic, transducers, where the inputs are actions taken by other agents, and outputs are the agent's actions. Weights on transitions denote the reward received by the agent. Agent strategies synchronize on their actions in a game, and the reward on an executions is given by the discounted sum of rewards on transitions.

The main contribution of this thesis is an algorithm ComputeNash that computes all Nash equilibria in a regular repeated game. For deterministic games, it runs in time polynomial in the size of the game, while it runs in exponential time in size of the game for non-deterministic games.

The crux of the algorithm lies in determining whether a strategy profile is in Nash equilibria or not. For this, the crucial technical step is to compare rewards earned by agents from infinite executions of the game. Technically, this problem reduces to the following: Given two weighted Büchi automata, where the aggregate is given by the discounted sum. How do you compare the weight of every executions on one automata with those on the other.

We address this relational reasoning problem via a novel automata-theoretic construction: ω-regular comparator. An ω-regular comparator is a Büchi automaton that accepts a pair of number sequences (A, B) iff the discounted sum on A is lesser than that on B. While various problems related to quantitative languages have been explored earlier, the problem of comparing two quantitative words has not been explored yet. In this thesis, we construct an comparator automaton for the discounted sum aggregate function, and provide an exploratory discussion on one for the limit-average comparator. The most interesting outcome of ω-regular comparators is that they reduce a quantitative problem of comparing the aggregate on two sequences into a qualitative problem of determining Büchi acceptance condition for a language over infinite words.

To demonstrate the practical utility of our approach, we compute Nash equilibria in various repeated games: versions of the iterated prisoner's dilemma, repeated auctions, and the Bitcoin protocol. Our computations corroborate with previously known results on these games. Furthermore, we were able to perform automated analysis of more complex games such as the Bitcoin protocol, by modeling it as a regular repeated game. Together, these case studies exhibit the promise of our approach for automated algorithmic analysis of repeated games.

6.2 Future directions

We discuss directions for future research below:

6.2.1 Exploration of ω-regular comparator

The problem of comparing aggregate of sequences has not been explored within the field of quantitative languages and weighted automata. From a practical point of view, ω-regular comparators reduce the quantitative problem of comparing the aggregate on two sequences into a qualitative problem of determining Büchi acceptance condition for a language over infinite words. We believe, this indicates that there may exist a relationship between quantitative and qualitative reasoning. Hence, indicating potential for applications in quantitative verification. We believe exploring this relationship can be an interesting direction to purse in itself.

From a more theoretical point of view, the concrete question here is to determine for which aggregate functions it is possible to construct an ω-regular comparator. Some commonly used aggregate function are limit average, $\liminf$, $\limsup$, sup, inf, etc. A construction of the comparator when it exists, and a proof of impossibility/undecidability when it doesn't will improve our understanding of the aggregate functions and the boundaries of ω-regular comparators.

6.2.2 Extension of regular repeated games

This thesis considers the analysis of Nash equilibria in repeated games with finitely many agents. The regular repeated games framework can be enriched in numerous ways. One such extension is by allowing an agent to have infinitely many strategies. Such agents may exists in games in which agents *dynamically build* new strategies from previously existing ones. While each strategy may still be represented as a finite state machine, it is not clear how to obtain a succinct representation for all infinitely many strategies of the agent. One proposal could be to shift from the machine-view of strategies to a tree-view of strategies. Under this representation, the set of all strategies could be represented as a language of strategy trees.

In the current model for regular repeated games, uncertainty is modeled with non-determinism. Another way to model uncertainly is via probability. A probabilistic model of a strategy extends the proposed model by assigning a probability distribution to transitions from each state in the strategy machine. This distribution denotes the probability with which an agent chooses a transition from each state.

These proposed extensions of regular repeated games may have implications on algorithms for the analysis. Therefore, the design of algorithms for Nash equilibria, dominant strategy, best response strategy, subgame-perfect equilibria etc offers immense scope for research in this direction.

6.2.3 Frameworks for other games

Evolutionary games [31] appear in the context of evolutionary biology, gene analysis etc. Evolutionary games differ from repeated games in two major ways: (1). The number of agents in an evolutionary game change with time (2). In evolutionary games agents do not explicitly make decisions. We illustrate this point through an example. Consider an initial population of small beetles and large beetles. Here the strategies are either to be a large beetle, or to a small beetle. Note that the

size of a beetle is a genetic behavior, hence beetles *can not make* this decision. The notion of rationality in evolutionary games is *evolutionary stability*. Intuitively, it corresponds to those strategies that thrive even after a long period of time. These differences in the modeling and analysis of such games indicate that they might not fall under the framework of regular repeated games.

The modeling and analysis of evolutionary games pose an interesting challenge for future research. One proposal to model evolutionary games could be by using a weighted version of *Petri nets*. A population protocol can be thought of as a vector addition system (or alternatively, a Petri net) defined over non-negative integers with transitions between vectors. Every state in a pert-net will correspond to the number of species adhering to a strategy. For example, number of large beetles, and the number of small beetles. Weights correspond to the quantitative measure of the fitness of each species in a state.

Bibliography

[1] Dilip Abreu, David Pearce, and Ennio Stacchetti. Toward a theory of discounted repeated games with imperfect monitoring. *Econometrica*, pages 1041–1063, 1990.

[2] Dilip Abreu and Ariel Rubinstein. The structure of nash equilibrium in repeated games with finite automata. *Econometrica*, pages 1259–1281, 1988.

[3] Shaull Almagor, Udi Boker, and Orna Kupferman. What?s decidable about weighted automata? In *Automated Technology for Verification and Analysis*, pages 482–491. Springer, 2011.

[4] Garrett Andersen and Vincent Conitzer. Fast equilibrium computation for infinitely repeated games. In *Proc. of AAAI*, pages 53–59, 2013.

[5] Robert J. Aumann. Survey of repeated games. *Essays in game theory and mathematical economics : in honor of Oskar Morgenstern*, pages 11–42, 1981.

[6] Christel Baier, Nathalie Bertrand, and Marcus Grösser. Probabilistic automata over infinite words: Expressiveness, efficiency, and decidability.

[7] Christel Baier, Joost-Pieter Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[8] Jeffrey S Banks and Rangarajan K Sundaram. Repeated games, finite automata, and complexity. *Games and Economic Behavior*, 2(2):97–117, 1990.

[9] David Basanta, Haralambos Hatzikirou, and Andreas Deutsch. Studying the

emergence of invasiveness in tumours using game theory. *The European Physical Journal B*, 63(3):393–397, 2008.

[10] Elchanan Ben-Porath. Repeated games with finite automata. *Journal of Economic Theory*, 59(1):17–32, 1993.

[11] Kimmo Berg and Mitri Kitti. Computing equilibria in discounted 2× 2 supergames. *Computational Economics*, 41(1):71–88, 2013.

[12] Kimmo Berg, Mitri Kitti, et al. Equilibrium paths in discounted supergames. Technical report, Working paper, 2012.

[13] Udi Boker and Thomas A. Henzinger. Exact and approximate determinization of discounted-sum automata. *Logical Methods in Computer Science*, 10(1), 2014.

[14] Udi Boker, Thomas A. Henzinger, and Jan Otop. The target discounted-sum problem. In *Proc. of LICS*, pages 750–761, 2015.

[15] Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Nash equilibria in concurrent games with büchi objectives. In *Proc. of FSTTCS*, pages 375–386, 2011.

[16] Thomas Brihaye, Véronique Bruyère, and Julie De Pril. On equilibria in quantitative games with reachability/safety objectives. *Theory Comput. Syst.*, 54(2):150–189, 2014.

[17] Andriy Burkov and Brahim Chaib-draa. An approximate subgame-perfect equilibrium computation technique for repeated games. In *Proc. of AAAI*, pages 729–736, 2010.

[18] Colin Camerer. *Behavioral game theory*. New Age International, 2010.

[19] Krishnendu Chatterjee, Laurent Doyen, and Thomas A Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4):23, 2010.

[20] Krishnendu Chatterjee and Thomas A Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 78(2):394–413, 2012.

[21] Swarat Chaudhuri, Sriram Sankaranarayanan, and Moshe Y. Vardi. Regular real analysis. In *Proc. of LICS*, pages 509–518, 2013.

[22] Xi Chen and Xiaotie Deng. 3-nash is PPAD-complete. In *Electronic Colloquium on Computational Complexity*, volume 134, 2005.

[23] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM*, 56(3):14:1–14:57, 2009.

[24] Andrew M Colman. *Game theory and its applications: in the social and biological sciences*. Psychology Press, 2013.

[25] Vincent Conitzer and Tuomas Sandholm. New complexity results about nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008.

[26] Mark B Cronshaw. Algorithms for finding repeated game equilibria. *Computational Economics*, 10(2):139–168, 1997.

[27] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[28] Yevgeniy Dodis, Shai Halevi, and Tal Rabin. A cryptographic solution to a game theoretic problem. In *Advances in Cryptology*, pages 112–130, 2000.

[29] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1):69–86, 2007.

[30] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer, 2009.

[31] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[32] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[33] James W Friedman. *Game theory with applications to economics*. Oxford University Press New York, 1990.

[34] Joseph Y Halpern and Rafael Pass. Algorithmic rationality: Game theory with costly computation. *Journal of Economic Theory*, 156:246–268, 2015.

[35] Kenneth L Judd, Sevin Yeltekin, and James Conklin. Computing supergame equilibria. *Econometrica*, pages 1239–1254, 2003.

[36] Miroslav Klimoš, Kim G Larsen, Filip Štefaňák, and Jeppe Thaarup. Nash equilibria in concurrent priced games. In *Proc. of LATA*, pages 363–376. Springer, 2012.

[37] Michael L Littman and Peter Stone. A polynomial-time nash equilibrium algorithm for repeated games. *Decision Support Systems*, 39(1):55–66, 2005.

[38] George J. Mailath and Larry Samuelson. *Repeated games and reputations: Long-running relationships*. Oxford University Press, 2006.

[39] Matthieu Manceny, A Lackmy, C Chettaoui, F Delaplace, and Cours Monseigneur Romero. Application of game theory to gene networks analysis. 2005.

[40] Robert E. Marks. Repeated games and finite automata. *Recent Developments in Game Thoery*, pages 43–64, 1992.

[41] Nimrod Megiddo and Avi Wigderson. On play by means of computing machines. In *Theoretical aspects of reasoning about knowledge*, pages 259–274, 1986.

[42] Paul R Milgrom and Robert J Weber. A theory of auctions and competitive bidding. *Econometrica*, pages 1089–1122, 1982.

[43] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311, 1997.

[44] Mehryar Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer, 2009.

[45] James D Morrow. *Game theory for political scientists*. Princeton University Press Princeton, NJ, 1994.

[46] Herve Moulin. *Game theory for the social sciences*. NYU press, 1986.

[47] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[48] Abraham Neyman. Bounded complexity justifies cooperation in the finitely repeated prisoners' dilemma. *Economics letters*, 19(3):227–229, 1985.

[49] Abraham Neyman. Finitely repeated games with finite automata. *Mathematics of Operations Research*, 23(3):513–552, 1998.

[50] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay Vazirani. *Algorithmic game theory*. Cambridge University Press, 2007.

[51] Martin J Osborne and Ariel Rubinstein. *Bargaining and markets*. Academic press, 1990.

[52] Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.

[53] Christos H Papadimitriou and Mihalis Yannakakis. On complexity as bounded rationality. In *Proc. of STOC*, pages 726–733, 1994.

[54] Michele Piccione and Ariel Rubinstein. Finite automata play a repeated extensive game. *Journal of Economic Theory*, 61(1):160–168, 1993.

[55] Michael O Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

[56] Robert W Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[57] Ariel Rubinstein. Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory*, 39(1):83–96, 1986.

[58] Walter Rudin. *Principles of mathematical analysis*, volume 3. McGraw-Hill New York, 1964.

[59] Herbert A Simon. A behavioral model of rational choice. *The quarterly journal of economics*, pages 99–118, 1955.

[60] Goal. GOAL. `http://goal.im.ntu.edu.tw/wiki/`.

[61] Rabit-Reduce. Rabit-reduce. `http://www.languageinclusion.org/`.

[62] Wolfgang Thomas, Thomas Wilke, et al. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

[63] Ming-Hsien Tsai, Seth Fogarty, Moshe Y Vardi, and Yih-Kuen Tsay. State of büchi complementation. In *Proc. of CIAA*, volume 10, pages 261–271. Springer, 2010.

[64] Amparo Urbano and Jose E Vila. Computational complexity and communication: Coordination in two–player games. *Econometrica*, 70(5):1893–1927, 2002.

[65] Adrian Vetta. Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 416–425. IEEE, 2002.

[66] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton university press, 2007.

[67] Jörgen W Weibull. *Evolutionary game theory*. MIT press, 1997.