Nash equilibria in generalised dining philosophers games

Synthesis of collectively acceptable strategy profiles with qualitative and quantitative objectives

by

Johan Pieter van Rooyen

Submitted in fulfilment of the requirements for the degree
Master of Science (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

November 30, 2023

Nash equilibria in generalised dining philosophers games

Synthesis of collectively acceptable strategy profiles with qualitative and quantitative objectives

by

Johan Pieter van Rooyen
E-mail: johan.vanrooyen@tuks.co.za

Abstract

The Generalised Dining Philosophers Game (GDPG) consists of agents which must co-operate (or compete) for shared resources. As there are several cooperating agents, we can think of the GDPG as a multi-agent system. In such a system, there are naturally some qualitative objectives such as fairness and liveness; and quantitative objectives where the agents seek to satisfy their goal as frequently as possible. The GDPG is represented as a concurrent game model and the agents' objectives are represented by $LTL[\mathcal{F}]$ formulas. There are some qualitative objectives which represent the goals of the entire group, and some quantitative objectives which represent the individual agents' and should be optimised. From this point, the $LTL[\mathcal{F}]$ model checking procedure is modified to produce an automaton-based algorithm which identifies a strategy profile which satisfies the qualitative objectives, and also is a Nash Equilibrium with respect to the agent's quantitative objectives. That is, at each configuration of the game, an action must be prescribed to each agent such that the collective objectives of the group are satisfied, and no agent can unilaterally deviate in order to achieve a better outcome.

Keywords: Multi-agent systems, Nash equilibrium, Generalised dining philosophers, rational synthesis.

Supervisor	**:** Dr. N. Timm
External Co-supervisor	**:** Prof. V. Goranko
Department	**:** Department of Computer Science
Degree	**:** Master of Science

Acknowledgements

I would like to whole-heartedly thank the following people for their support and help in the completion of this work. It hasn't been easy, and I am deeply grateful for their guidance, patience and presence:

- Dr Nils Timm and Prof Valentin Goranko. I could not ask for a more excellent supervision team;
- My beautiful girls, Rhiannon and Aurora. You are my everything.

Contents

List of Figures

Chapter 1

Introduction

Multi-agent systems [27] comprise agents taking actions in an environment. An environment is a context in which the agents are acting, and it changes depending on the actions undertaken. Agents take action to satisfy some pre-defined individual objectives. A run of a system entails several discrete rounds where all agents take some action on each time step, and the joint action (called an action profile) affects a deterministic change in the observable state (known as a configuration) of the system. Thus, for a given system, many different runs may be possible. A strategy prescribes an action for each agent for each configuration, and a strategy profile is the mapping of each agent to a strategy. Strategy synthesis [18] is the problem of finding strategy profiles, such that when the agents follow these strategies certain properties such as the achievement of all individual objectives will be satisfied.

To better reason about which runs of the system may occur, it is useful to consider that the agents are playing a *game*. [24] Games are either zero-sum, in which there can only be one winner; or non-zero-sum. In non-zero-sum games, agents' objectives are not mutually exclusive, and it is possible for all agents to satisfy their objectives, or to reach some compromise.

Objectives can be divided into qualitative (Boolean) and quantitative (numeric). Qualitative objectives refer to simple yes/no propositions that are either satisfied or not satisfied. The peculiarity of purely qualitative systems is that if two solutions satisfy a qualitative objective, we can't distinguish whether one solution is better than the other.

Quantitative objectives refer to numeric values (known as payoffs) that are assigned to runs of games, which agents seek to minimise or maximise. Agents may need to compete, or co-operate, to achieve their objectives.

Rational agents are "agents that act to achieve their own objectives" [18]. This dissertation deals with the automated synthesis of collectively acceptable strategies for systems involving rational agents. A collectively acceptable strategy results in a run where all of a set of collective objectives are satisfied and the system is in equilibrium with respect to some individual objectives and a given solution concept - specifically in a Nash equilibrium. The individual objectives are (assumed to be) quantitative and collective objectives are qualitative

Temporal logic formulas over sequences of atomic propositions form the basis of qualitative objectives and functions over the sequences form the basis of quantitative objectives. A labelling function maps configurations of the game to the logical propositions that hold when the game is in that configuration.

Part of the process of strategy synthesis is creating a strategy profile which will lead to satisfaction of the objectives. At each configuration, the agents should be prescribed an optimal action out of a plurality of possible options. A collective optimal play of a game with respect to an objective entails finding the strategy profile which provides the best possible satisfaction of the objectives.

A strategy profile is a Nash Equilibrium [24] if no agent can improve their payoff by unilaterally deviating from the strategy profile. Nash Equilibrium is essential because any strategy profile that is not a Nash Equilibrium will be unstable - rational agents will have an incentive to deviate from the strategy profile. A collectively acceptable strategy profile is a Nash Equilibrium which additionally satisfies all collective qualitative objectives.

The classic problem of the Dining Philosophers, [16] is an example problem involving dynamic, distributed resource allocation. It entails 5 philosophers sitting around a table. In between each pair of philosophers, there is a fork. Each philosopher will think until they become hungry, at which time they will attempt to pick up the two adjacent forks, one after the other. If one of the forks is not available, the philosopher will wait until it becomes available. When they have 2 forks, they will eat until they are no longer hungry at which point they will replace the forks on the table and go back into a thinking state. Solving the problem generally entails finding strategies for the philosophers such that deadlock and starvation are avoided.

The Generalised Dining Philosophers problem [15] is a generalisation of the dining philosophers problem, and extends it to include any number of philosophers, any number of forks, more complicated rules regarding which philosophers can reach each fork, and how many forks are required before a philosopher can eat. Moreover, the nomenclature is changed in that philosophers are referred to as agents; forks as resource units; and eating as the satisfaction of an objective. A play of the GDPG progresses through an infinite number of discrete steps, in which actions, one per agent, occur in one shot. At each time step, the allocation of all resources to agents in the game can be precisely described and is known as a configuration.

GDPG is a useful example of a MAS, used to model resource competition in a distributed system. Moreover, the problem of finding collectively acceptable strategy profiles is not trivial.

The strategies may be subject to some constraints. For instance, agents must always make the same decision given the same configuration, that is, the choice of actions is uniform. Such strategies are known as pure strategies. Another assumption made is about the memory capabilities of agents. If agents have no memory of anything that has happened in the game, they rely only upon the current configuration of the game to make decisions. These are known as positional strategies. If the agent has some memory, finite(or infinite)-memory strategies arise.

The choice of which assumptions are to be made is known as a problem setting, and this dissertation will present a technique for synthesising strategies in pure problem settings. For brevity, in this dissertation, the term strategy profile refers to a pure, positional strategy profile unless otherwise specified.

1.1 Motivation

The study of multi-agent systems facilitates an understanding of how agents, when endowed with certain abilities, will behave when set into a given environment and given some objectives. This is critical for cases where human control over a system is constrained, and agents are not able to receive direct instructions - they need to figure it out for themselves.

Work of this nature is important as real-world systems may be shown to be analogous to the generalised dining philosophers problem, and could therefore use the procedures of this dissertation. For example, a team of autonomous machines may need to collectively explore an alien world. These machines would need to ensure their safety (a qualitative goal) while minimizing battery usage (quantitative), and completing scientific exploration goals (perhaps both). Certain goals of the agents may only be possible to achieve through a collaborative effort.

Moreover, the process of generating the strategy synthesis algorithm is instructive per se. By developing the mathematical models needed to represent the GDP game in the required level of abstraction, one discovers well-developed mathematical analogues for computer concepts, and these techniques can be adapted and refined for other types of problems too. It is hoped that the algorithm presented in this dissertation can be extended to solve other problems in addition to finding the collectively acceptable strategy profiles of GDP games.

1.2 Goals

This dissertation aims to present a procedure for synthesising collectively acceptable strategy profiles for the generalised dining philosophers game using an automaton-based approach. The agents of the game all have the goal of satisfying their demand for resource units and the same individual objective - which is to minimize their maximum waiting time before reaching their goal. Since agents are assumed to be rational, the strategy profile must be an equilibrium with respect to the solution concept of Nash equilibrium. In addition to the individual objectives, there are two collective objectives - the strategy profiles must not result in starvation of agents (starvation-freedom objective), and all agents must receive a payoff that is reasonably close to the others' (fairness objective).

Several technicalities are addressed. Firstly, the temporal logic $LTL[\mathcal{F}]$ **REF** is explained and the objectives of starvation-freedom and fairness, as well as agents' individual objective to minimize maximum waiting time, are expressed in this logic.

$LTL[\mathcal{F}]$ is chosen because it can express qualitative and quantitative, as well as collective and individual objectives. Additionally, a framework for categorising each objective is provided.

This automaton is then combined with automata for fairness and starvation-freedom to produce an automaton which only accepts a strategy word if it corresponds to a collectively acceptable strategy profile of the given GDPG.

Strategy profiles are then reinterpreted as strategy words of ω-regular strategy languages such that specialised Büchi automata (called strategy automata) can act as acceptors of strategy words, and consequently, strategy profiles. $LTL[\mathcal{F}]$ model checking is augmented to produce strategy automata which lead to specified payoff profiles, and strategy automata which represent unilateral deviations (known as deviation automata). These automata are then combined using standard Büchi automaton operations to produce the Nash automaton - an automaton which only accepts a strategy word (strategy profile) if that strategy profile is a Nash equilibrium.

1.3 Contributions

- A new representation of the GDP game is introduced, which has not been examined in the literature. This new representation replaces individual resource units with shared resource bags.
- Strategy profiles are represented as strategy words of a strategy language, thus making them compatible with automaton-based approaches.
- Qualitative and quantitative objectives are combined to represent individual agent objectives as well as collective objectives.
- $LTL[\mathcal{F}]$ is extended to account for multiple objectives where each has a different specified set of satisfaction values which are acceptable.
- Equilibrium synthesis in $LTL[\mathcal{F}]$ model checking is achieved using an automaton-based approach.

- A procedure for the automatic synthesis of collectively acceptable strategies in terms of the GDP game is presented and explained.

1.4 Dissertation Outline

- **Chapter 2** establishes consistent terminology and provides definitions of key terms and concepts and the relevant underlying mathematical concepts are briefly presented.
- **Chapter 3** describes a simple framework from the classification of $LTL[\mathcal{F}]$ objectives as either qualitative or quantitative; and individual or collective.
- **Chapter 4** defines ω-regular strategy languages which are acceptors of strategy words which correspond to strategy profiles. This facilitates constructing automata which function as acceptors of strategic behaviour.
- **Chapter 5** covers the automaton-based construction procedure for collectively acceptable strategy profiles for a given GDPG with specified objectives.
- **Chapter 6** explores work related to the GDP game; the quantification of objectives in similar problems; rational synthesis; and relevant formal logics for qualitative objectives.
- **Chapter 7** provides a summary of the preceding chapters and briefly outlines future work on this topic.

Chapter 2

Preliminaries

This section presents the existing work upon which this dissertation depends. The subjects defined include the Generalised Dining Philosophers Game and its related concepts; some formal models which are used; the temporal logic $LTL[\mathcal{F}]$; and finally concepts which describe the strategic behaviour of agents and Nash Equilibrium.

The Generalised Dining Philosophers Game [15] is a generalisation of the Dining Philosopher's game. The generalisation makes three changes to the classical problem. Firstly, the classical game consists of exactly 5 philosophers sitting around a table, the generalised version deals with an arbitrary number of philosophers. Secondly, in the classical problem the forks which each philosopher can reach are fixed, whereas, in the generalisation version, there are more complex rules which govern which philosophers can reach each fork. Thirdly, in the classical problem, each philosopher must take up two forks to reach their objective, in the generalised version the philosophers require an arbitrary number of forks. There is also a shift in nomenclature, philosophers become agents, forks become resource units, and "eating" becomes the satisfaction of an objective. The definition below is due to [19].

Definition 2.1 (Generalised Dining Philosophers Game (GDPG))**.** A GDPG, $\mathcal{G}$ is a tuple $\langle Agt, Acc, Bags, d\rangle$ where:

- $Agt = \{a_1, \ldots, a_n\}$ is a non-empty set of *agents*.
- $Acc : \mathcal{P}(Agt) \setminus \emptyset \to \mathbb{N}$ is the *accessibility function*, which maps non-empty subsets

of agents to natural numbers. $Acc(A)$ is the number of resources that are shared exclusively by the agents in A.

- $Bags = \{A \in \mathcal{P}(Agt) : Acc(A) \neq 0\}$. A bag is referred to by the set of agents which share exclusive access to the resource unit in that bag.
- $d : Agt \to \mathbb{N}$ is a *demand function* defining the number of resource units that each agent needs to carry out its tasks

Each subset of agents has a shared bag of resources which are available only to agents in the subset. These bags can be empty. Each agent has access to various bags, each of which is shared with some other group of agents.

The above definition differs from the definition provided in [15] by representing the resources differently. The original definition includes resource units as discrete atomic entities and then uses a bipartite accessibility graph to denote which agents could access which resource units. If agent a_1 could access resource unit r_i, then there would be an edge in the accessibility graph connecting these two entities. Definition 2.1 does not deal with resource units, but rather with resource bags. There is a resource bag for every subset of agents, and the resources in this bag are only accessible to those agents. The benefit of this representation is that it treats resources as indistinguishable if they are accessible to the same set of agents.

To illustrate this, consider a simple example of a GDPG with three agents and three resource units with are accessible to all agents. In the original definition, these resource units are all distinct, and there would be actions for each agent taking each resource. In the modified definition, all three resource units would be in the bag shared by all agents, and each agent would only have the action of taking from that shared bag. The particular resource unit which they take is disregarded and resource units accessible to the same group of agents are considered to be interchangeable and fungible.

As the agents take actions and move resource units around, the game moves through different configurations. A configuration is a description of the location of all resource units at a given point in time. For example, after some time, the game configuration might show that x number of resource units from a bag are allocated to the agent a_i, and so forth.

Definition 2.2 (GDPG configuration)**.** Given a GDP game $\mathcal{G}$, a configuration is a function c, which maps each resource bag to a set of ordered pairs. The first element of each pair is in $Agt \cup \{null\}$, and the second is in $\mathbb{N}$. The reason for the inclusion of $null$ is to include a "dummy" agent. Any resource unit which is not allocated to a real agent is represented as being allocated to the "dummy" agent. $Conf$ is the set of all configurations of the GDP game.

$$c : (\{Agt \cup null\} \times Bags) \rightarrow \mathbb{N}$$

For example, consider the configuration of a GDP game with three agents and one shared bag, β which is accessible to all agents:

$$\{((a_1, \beta) \rightarrow 0), ((a_2, \beta) \rightarrow 2), ((a_3, \beta) \rightarrow 1), ((null, \beta) \rightarrow 4)\}$$

In the resource bag which is shared by all three agents, zero resource units are allocated to agent a_1, two are allocated to a_2, one to a_3, and four remain unallocated (allocated to $null$).

With this notation is it simple to check some crucial properties of the game, for example:

- An agent a has reached its demand in c iff $\sum_{\beta \in Bags} c(a, \beta) = d(a)$;
- A bag β is empty if $\sum_{a \in Agt} c(a, \beta) = |\beta|$;
- A bag β is not empty if $\sum_{a \in Agt} c(a, \beta) < |\beta|$;

As the agents play the game, they must each take actions at each time period. The change of configuration that occurs is determined by the joint action of all the actions, known as the action profile.

Definition 2.3 (Actions)**.** Agents have four types of actions. Given agent a and resource bag β

- $Take^a_\beta$ describes an agent a taking a free resource from bag β.

- $Release^a_\beta$ describes an agent a putting a single resource back in resource bag β.
- $Release^a_{all}$ describes an agent a replacing all their resources in their respective bags.
- $Idle^a$ describes an agent, a, which does nothing.

The set of all possible actions is:

$$\begin{aligned} Act = &\{Take^a_\beta : a \in \beta\} \\ &\cup \{Release^a_\beta : a \in \beta\} \\ &\cup \{Release^a_{all}\} \\ &\cup \{Idle^a\} \end{aligned}$$

Definition 2.4 (Action Profile)**.** An action profile is a mapping of agents to actions. It describes the action taken by each agent at any particular point in the game.

$$ap : Agt \to Act$$

ACT is the set of all action profiles.

When the agents detect a certain configuration, they must decide which action is to be taken. This decision is based on a strategy profile which, in general, maps configurations to action profiles. The strategy profiles dealt with in this dissertation are all *pure*, ie. they are all deterministic, thus whenever the term strategy profile is used, a pure strategy profile is meant. Additionally, strategy profiles can be either *positional*, *finite-memory*, or *infinite-memory*. In a positional strategy, the decision is based on only the current detected configuration. In finite- and infinite-memory strategy profiles the choice depends upon other information which is retained in a finite (or infinite) memory store.

Definition 2.5 (Positional Strategy Profile)**.** A positional strategy profile provides a mapping of each configuration to exactly one action profile.

$$sp_{pos} : Conf \to ACT$$

Memory-based strategy profiles can be represented in one of two ways. Either based on a transducer or based upon histories. With a transducer-based strategy profile, each agent has an internal transition system which retains some information about the previous configurations of the game. The decision about which action profile to take depends on both the detected configuration and the internal memory state. The exact details of how these transducers work can be found in [13].

Definition 2.6 (Memory-Based Strategy Profile with Transducer)**.** Given a set of configurations $Conf$, a set of action profiles, ACT, and a set of internal memory states T, a memory-based strategy profile with transducer is:

$$sp_{mem} : (Conf \times T) \to (ACT \times T)$$

The strategy profile is finite-memory if $|T|$ is finite, otherwise it is infinite-memory.

Alternatively, memory can manifest in strategy profiles by having a strategy profile provide a mapping from a sequence of the last few configurations (known as a history) to an action profile.

Definition 2.7 (Memory-Based Strategy Profile with Histories)**.** Given a set of configurations $Conf$, and a set of action profiles, ACT, a memory-based strategy profile with Histories is:

$$sp_{mem} : (c \in Conf)^k \to ACT$$

The strategy profile is finite-memory if $k \in \mathbb{N}$ and it is infinite memory if $k \in \infty$.

It is taken for granted that these two mechanisms of using memory are equally expressive and any strategy profile described by one can be converted to the other type. It is also noted that a positional, finite-memory, and infinite-memory strategy profile might produce the same behaviour. Instead of dealing with these different types of strategy profiles, they are dealt with more generally using *strategy words* and *strategy languages*, which are detailed in Chapter 4.

To facilitate the quantification of objectives, it is necessary to depart from logic which deals exclusively in Boolean values. This is because, in GDPG, agents may have qualitative objectives, such as satisfaction of an objective, but also quantitative objectives, such as maximising the frequency of satisfaction. For this purpose, the logic $LTL[\mathcal{F}]$ is chosen, and the following Definitions 2.8 - 2.11 are due to [2]. A detailed description of the objectives of agents is given in Chapter 2.

Linear Temporal Logic (LTL) [26] is a modal logic which facilitates reasoning about how the truth value of logical propositions can change over time in a given system. $LTL[\mathcal{F}]$ extends LTL by replacing the boolean operators of LTL with arbitrary functions over $[0, 1]$. $LTL[\mathcal{F}]$ is a family of logics, each parameterised by a set of functions, $\mathcal{F}$. Definitions for the syntax and semantics of $LTL[\mathcal{F}]$ are provided.

Definition 2.8 ($LTL[\mathcal{F}]$ syntax)**.** Given a set of atomic propositons, AP and $\mathcal{F} \subseteq \{f : [0, 1]^k \rightarrow [0, 1] | k \in \mathbb{N}\}$, and set of functions over $[0, 1]$. An $LTL[\mathcal{F}]$ formula is:

- $True$, $False$, or p, for $p \in AP$;
- $f(\varphi_1, \dots \varphi_k)$, $X\varphi_1$, or $\varphi_1 U \varphi_2$ for $LTL[\mathcal{F}]$ formulas $\varphi_1, \dots, \varphi_k$, and a function $f \in \mathcal{F}$.

$LTL[\mathcal{F}]$ formulas deal with truth values of atomic propositions that change over time. For this reason, they are evaluated over infinite sequences of sets of logical propositions.

The GDPG does not explicitly include logical propositions to indicate the reaching of certain configurations, but one can imagine that each configuration has some attached set of logical propositions, and thus a computation is generated as the game progresses. Concurrent Game Models (see Definition 2.16) do include these logical propositions, and for this reason (amongst others), the GDPG will first be converted to an equivalent CGM.

Definition 2.9 (Computations and suffixes)**.** Given a set of atomic propositions AP, a computation is a word $\pi = \pi_0, \pi_1, \dots \in (2^{AP})^\omega$. That is a sequence of sets of atomic propositions. The suffix $\pi_i, \pi_{i+1}, \dots$ is denoted π^i.

Definition 2.10 ($LTL[\mathcal{F}]$ semantics)**.** $LTL[\mathcal{F}]$ semantics are defined over infinite computations. And are defined thus:

Formula	Satisfaction value
$[\![\pi, True]\!]$	1
$[\![\pi, False]\!]$	0
$[\![\pi, p]\!]$	1 if $p \in \pi_0$, otherwise 0
$[\![\pi, f(\varphi_1, \ldots, \varphi_k)]\!]$	$f([\![\pi, \varphi_1]\!], \ldots, [\![\pi, \varphi_k]\!])$
$[\![\pi, X\varphi_1]\!]$	$[\![\pi^1, \varphi_1]\!]$
$[\![\pi, \varphi_1 U \varphi_2]\!]$	$max_{i\geq 0}\{min\{[\![\pi^i, \varphi_2]\!], min_{0\leq j\leq i}[\![\pi^j, \varphi_1]\!]\}\}$

A difference between LTL and $LTL[\mathcal{F}]$ is that the latter treats logical formulas as functions, which is not immediately compatible with the logical operators of LTL. This is addressed by re-defining the standard logical operators of $\wedge, \vee$, and $\neg$, as well as the LTL operators X, U, F, and G, as functions in $LTL[\mathcal{F}]$ and showing how these functions retain their intuitive meaning.

Definition 2.11 (LTL operators as $LTL[\mathcal{F}]$ functions)**.** Standard logical operators are defined as functions in $LTL[\mathcal{F}]$ and used as a shorthand for the corresponding functions. They are defined as:

- $\neg x = 1 - x$;
- $x \vee y = max\{x, y\}$;
- $x \wedge y = min\{x, y\}$;
- $x \rightarrow y = max\{1 - x, y\}$.

Furthermore, the inclusion of the U operator from LTL suggests that it is possible to derive F and G operators. These are defined as:

- $[\![\pi, F\varphi_1]\!] = max_{i\geq 0}\{[\![\pi^i, \varphi_1]\!]\}$;
- $[\![\pi, G\varphi_1]\!] = min_{i\geq 0}\{[\![\pi^i, \varphi_1]\!]\}$;

Note that $LTL[\mathcal{F}]$ for $\mathcal{F} = \{\neg, \vee, \wedge\}$, corresponds directly to LTL.

In LTL, the possible evaluations of a formula over a computation are intuitively, the set $\{true, false\}$, however in $LTL[\mathcal{F}]$ there is no single intuitive set of possible evaluations. Instead, the set of possible evaluations must be defined and these values are known as *satisfaction values*.

Definition 2.12 (Satisfaction Values)**.** Given an $LTL[\mathcal{F}]$ formula φ and a computation (or suffix) π, the satisfaction value of φ in π is the value in $[0, 1]$ which represents the evaluation of φ given the computation π. It is denoted as $[\![\pi, \varphi]\!]$. The set of all possible satisfaction values of an $LTL[\mathcal{F}]$ formula, φ is denoted as $V(\varphi)$.

Later in this dissertation, it will be necessary to reason about $LTL[\mathcal{F}]$ formulas and their characteristics. This is aided by examination of the subformulas which make up the formula. This set of subformulas is also used in the algorithm presented in Chapter 5.

Definition 2.13 (Closure of an $LTL[\mathcal{F}]$ formula)**.** Given an $LTL[\mathcal{F}]$ formula φ, the closure of $cl(\varphi)$, is the set of subformulas of φ.

Definition 2.14 (Consistent functions of an $LTL[\mathcal{F}]$ closure)**.** Given an $LTL[\mathcal{F}]$ formula φ with closure of $cl(\varphi)$ and set of possible satisfaction values $V(\varphi)$, a function g maps subformulas to satisfaction values $g : cl(\varphi) \to V(\varphi)$. g is *consistent* if no two mappings defined by the function are contradictory.

For example, if $(\phi, 1) \in g$ and $(\neg\phi, \frac{1}{3}) \in g$ then g is not consistent.

Another critical difference between LTL and $LTL[\mathcal{F}]$ is that an LTL has an intuitive outcome, that is, it is generally expected that an LTL formula evaluates to *true*. This intuition does not directly apply to $LTL[\mathcal{F}]$ formulas because they produce one of many satisfaction values from the interval $[0, 1]$. Thus, it is necessary to define a predicate which indicates which satisfaction values can be considered "acceptable".

Definition 2.15 ($LTL[\mathcal{F}]$ predicates)**.** A predicate $P \subseteq [0, 1]$ is defined as the set of satisfaction values for a $LTL[\mathcal{F}]$ which are deemed to be acceptable.

Two fundamental models are used for representing the multi-agent systems. First, the Concurrent Game Model, which is used for modelling how the configuration of a multi-agent system changes as the agents take collective actions; and second, the Non-deterministic Generalised Büchi Automaton which is used for $LTL[\mathcal{F}]$ model checking and synthesis (further elaborated in Chapter 5).

A Concurrent Game Model(CGM) [5] is a graph-like structure consisting of states and transitions which connect these states. Each state represents a configuration of the game; at a given configuration, the agents take a joint action which affects a change in the configuration. For each possible joint action, there is a transition which leads to the configuration which would result if that joint action was taken.

Definition 2.16 (Concurrent Game Model (CGM))**.** A CGM, $\mathcal{M}$, is a tuple $\langle \mathcal{A}, St, Act, av, out, Prop, L\rangle$ where:

- $\mathcal{A}$ is a finite set of agents (players);
- St is a set of states;
- Act is the set of all possible actions. An action profile is a function, $ap : Agt \to Act$, which maps each agent to an action. AP is the set of all action profiles;
- $av : \mathcal{A} \times St \to \mathcal{P}(Act)$ is a mapping which assigns each pair of agent and state to a set of available actions. These are the actions which the agent can undertake while in the given state;
- $out : St \times Act^{\mathcal{A}} \to St$ is a mapping of states and available action profiles for that state to a new successor state. Hence, out is a function which determines what state to move to, given a certain state and action profile.
- $Prop$ is a set of atomic propositions
- $L : St \to \mathcal{P}(Prop)$ is a labelling function which maps some atomic propositions to each state.

Büchi Automata are automata which function as acceptors of ω-regular languages. ω-regular languages generalise regular languages to infinite-length words. Büchi automata are used extensively for model checking [17], and the model checking procedure for $LTL[\mathcal{F}]$ is the basis for the synthesis algorithm developed in Chapter 5. The Non-deterministic, Generalised Büchi Automaton (NGBA) [2] [28] is a generalisation of Büchi automata which allows for non-deterministic transition functions, and has modified word acceptance criteria, which allows languages to be expressed more succinctly.

Definition 2.17 (Non-deterministic Generalised Büchi Automaton (NGBA))**.** Given a set of atomic propositions AP, an NGBA, $\mathcal{A}$ is an tuple $\langle 2^{AP}, Q, \delta, Q_0, \alpha \rangle$, where:

- 2^{AP} is the alphabet;
- Q is a set of states;
- $\delta : (Q \times 2^{AP}) \rightarrow 2^Q$; a *transition function* where each state and set of atomic propositions maps to a set of possible new states;
- $Q_0 \subseteq Q$ is the set of initial states;
- $\alpha \subseteq 2^Q$ is the set of *sets of accepting states* - for a run to be accepted, it must visit a member of each set in α infinitely many times.

A run of $\mathcal{A}$ is $\rho \in Q^\omega$ such that for all pairs of adjacent entries q_i, q_{i+1} in ρ, $q_{i+1} \in \delta(q_i, \sigma)$ for some $\sigma \in 2^{AP}$; and $q_0 \in Q_0$. The sequence of values for σ is a computation $\pi \in (2^{AP})^\omega$. Intuitively, a run is the series of states of an NGBA which are all connected by transitions, and a computation is the series of sets of atomic propositions which are associated with each of the transitions. A computation is accepted if there exists a run which generates it.

Now that the necessary elements for the evaluation of computations have been defined, we can proceed with the definition of Nash Equilibrium. Firstly, it is necessary to connect the realm of configurations and actions to the realm of computations. On a high level, as agents detect different configurations, and memory states change, they make decisions about actions. As they move through configurations, a computation is generated. Each strategy profile generates exactly one computation, however, the same computation can be generated by various strategy profiles.

Definition 2.18 (Computation corresponding to a strategy profile)**.** The computation generated by the execution of a strategy profile, sp, is denoted as π_{sp}. Conversely, the set of strategy profiles which generate the computation π is denoted as sp_π.

A strategy profile generates a computation, which is then evaluated according to $LTL[\mathcal{F}]$ formulas, one for each agent. Given an $LTL[\mathcal{F}]$ formula, there is an ordering of the relative value of each strategy profile - some are "better" than others - based on the evaluation of the $LTL[\mathcal{F}]$ formula over the computation generated by the strategy profiles. Since each agent has an associated $LTL[\mathcal{F}]$ formula, we can think of this ordering as describing the "preferences" of individual agents.

Definition 2.19 (Profitable deviation)**.** Given strategy profiles, sp and sp', and an agent a_i with $LTL[\mathcal{F}]$ objective, φ_i. $sp >^{a_i} sp'$ if and only if $[\![\pi_{sp}, \varphi_i]\!] > [\![\pi_{sp'}, \varphi_i]\!]$.

A unilateral deviation is a change from one strategy profile to another by altering the strategy of only a single agent.

Definition 2.20 (Unilateral deviation)**.** Given strategy profiles, sp and sp', and a set of agents $a_1, \ldots, a_n$, sp is a unilateral deviation of sp' for agent a_i if and only if, for all configurations, c, if $sp(c) \neq sp(c)$, then for all agents, $a_k \in (Agt \setminus \{a_i\})$, $sp(c)(a_k) = sp'(c)(a_k)$. That is, assignments of configurations to actions are the same for all agents except a_i. Unilateral deviation by agent a_i is denoted as $sp \Yup^{a_i} sp'$ and is symmetric.

A strategy profile is a Nash Equilibrium if there does not exist a profitable unilateral deviation for any agent.

Definition 2.21 (Nash Equilibrium)**.** Given a GDPG with n agents, each with a corresponding $LTL[\mathcal{F}]$ objective φ_i, a strategy profile sp is a Nash Equilibrium if there does not exist some other strategy profile sp' such that for an agent a_i:

- $sp' \Yup^{a_i} sp$;
- $sp' >^{\varphi_i} sp$.

That is, there does not exist an alternative strategy profile for which there is an agent that can unilaterally deviate to that strategy profile *and* the strategy profile provides an improved evaluation to that agent.

Identifying Nash Equilibria is one of the major goals of this dissertation, but is not the only one. In the next chapter, the objectives of agents are developed in greater detail.

Chapter 3

Objectives

This dissertation deals with different types of objectives. Particularly, quantitative and qualitative objectives; and collective and individual objectives. This chapter describes how an $LTL[\mathcal{F}]$ formula φ can be examined and categorised as either qualitative or quantitative, and as either individual or collective (or constructed to be of the desired type). The algorithms presented in later chapters describe the procedure by which all types of objectives can be accounted for. We begin with a high-level understanding of each of these types of objectives before seeing how they are represented in $LTL[\mathcal{F}]$. Thereafter, concepts such as fairness, starvation and acceptability are defined clearly in terms of the $LTL[\mathcal{F}]$.

3.1 Categories of Objectives

A qualitative objective is a temporal logic condition which is either $true$ or $false$. For example, an LTL formula when evaluated over a computation π will result in either $true$, or $false$, and nothing else. In the context of strategy profile synthesis, the expectation is to construct a strategy profile which results in the qualitative objective evaluating to $true$. In contrast to this is a quantitative objective, which can evaluate to numerical values. For example, if an agent receives some points for reaching certain goals, it may be desirable to maximise the number of points which the agent is awarded. Unlike qualitative objectives, which just need to be $true$, quantitative objectives may be expected to be "as much as possible" and we are no longer just concerned with satisfaction of objectives, but optimisation. An individual objective is an objective which only concerns a single agent. Collective objectives concern all agents.

We will illustrate how an $LTL[\mathcal{F}]$ formula can fit into any of these categories with a simple example.

A concurrent game model is modelling a muli-agent system comprising n agents. The system has distinct states and at each state, the agents take actions which affect a change in the state of the system. Each state is also associated with a set of logical propositions which hold while the system is in that state. For each agent, there is an associated atomic logical proposition which is used in $LTL[\mathcal{F}]$ formulas - p_1 for agent a_1 and so forth. When p_1 holds, agent a_1 has satisfied their demand.

An $LTL[\mathcal{F}]$ formula, φ, can now be examined for being individual, coalition-based[1], or collective; and for being qualitative or quantitative. Firstly, the closure of φ can be constructed and examined. If a proposition p_i is in $cl(\varphi)$, then the formula φ describes a goal of agent a_i. We can say that φ *depends upon* p_i. Thus we can categorize φ as:

- *Collective* if and only if φ depends upon the propositions associated with all agents;
- *Coalition-based* if and only if φ depends upon the propositions associated with more than one, but not all, agents;
- *Individual* if and only if φ depends upon the propositions associated with only one agent.

Next, an $LTL[\mathcal{F}]$ formula can be evaluated for quantitative- or qualitative-ness. As explained in Definition 2.12, $LTL[\mathcal{F}]$ formulas map to a value in $[0, 1]$, known as the satisfaction value. By induction over an $LTL[\mathcal{F}]$ formula, it is possible to construct the set of all possible satisfaction values. As was shown with LTL in Definition 2.11, an $LTL[\mathcal{F}]$ formula, φ, may have $V(\varphi) = \{0, 1\}$, which correspond to the boolean values $false$ and $true$. Thus we can categorise an $LTL[\mathcal{F}]$ formula, φ as:

- Qualitative if $V(\varphi) = \{0, 1\}$;

[1]This dissertation does not deal with coalition-based objectives, but they are included above for completeness

Quantitative Individual $Gf(p_1)$	Qualitative Individual Fp_2
Quantitative Collective $\bigvee_{p_i \in P} Gf(p_i)$	Qualitative Collective $\bigvee_{p_i \in P} Fp_i$

Table 3.1: Table showing the categorisation of $LTL[\mathcal{F}]$ formulas

- Quantitative otherwise.

Each type of objective can be expressed by placing it into the correct quadrant of Table 3.1. Each quadrant also contains an example $LTL[\mathcal{F}]$ formula, where p_1, p_2, and p_3 are the agent objectives and f is an arbitrary function over $\{0, 0.5, 1\}$.

3.2 Quantifying Starvation and Fairness

Starvation freedom and fairness are typical requirements for cooperating agents in a multi-agent system. These concepts are conventionally thought of as qualitative things - a system is either fair or not; agents starve or they do not. By re-expressing these concepts using quantitative formulas, it is possible to have a more nuanced representation of fairness and starvation. We can assert not just that a system is fair/starvation-free, but provide a metric for exactly how fair/starvation-free it is.

3.2.1 Quantitative Starvation

In qualitative settings, starvation occurs when an agent is never able to reach their goal. Intuitively, this is insufficient because, in actual scenarios, the timeliness of each satisfaction event is important. If a process can only access a needed resource once every 100 years, this would be unacceptable; similarly, if a philosopher was only able to eat after waiting for 100 years, they would starve before their turn arrives.

Only one quantitative operator is required for expressing quantitative starvation - the scaling operator ∇_λ.

Definition 3.1 (Scaling Operator)**.** Let $x \in [0,1]$ be a satisfaction value of an $LTL[\mathcal{F}]$ formula, and $\lambda \in [0,1]$ be a scaling parameter, then $\nabla_\lambda x = \lambda \times x$. In the context of evaluating an $LTL[\mathcal{F}]$ formula over a computation π:

$$[\![\pi, \nabla_\lambda \varphi]\!] = \lambda \times [\![\pi, \varphi]\!]$$

which has the effect of multiplying the satisfaction value of φ by some fixed value $\lambda \in [0,1]$.

Next we borrow exponentiation syntax for use in LTL[$\mathcal{F}$] (where p is an atomic proposition and X is the *Next* operator):

$$X^n p = \overbrace{X \cdots X}^{n \text{ times}} p$$

$$X^0 p = p$$

Thereafter we define the operator τ_n:

$$\tau_n p = X^n p \wedge \bigwedge_{i=0}^{n-1} \neg X^i p$$

Which is 1 when p is satisfied after waiting for exactly n turns. Finally, τ is combined with the scaling operator ∇_λ to create the *discounted future value operator*

Definition 3.2 (Discounted Future Value Operator)**.** Given an atomic proposition p and a threshold value $m \in \mathbb{N}$, the discounted future value operator $\mathcal{D}_m p$ is:

$$\mathcal{D}_m p = \nabla_1 p \vee \bigvee_{i=1}^{m} \nabla_{\frac{m-i}{m}} \tau_i p$$

Note that the usage of $\bigvee_{i=1}^{m} \nabla_{\frac{m-i}{m}}$ is a specific discounting function. But in general, this approach will work with any monotonically decreasing function over $[0,1]$.

$\mathcal{D}_m$ is a family of operators, parameterised by the *threshold value m*. The satisfaction value of $\mathcal{D}_m$ is 1 if p is satisfied immediately along a computation and $\frac{m-i}{m}$ if p is satisfied by the suffix π^i - that is if there is a waiting time of i turns. If the agent must wait more than m (the threshold value) turns, the satisfaction value of $\mathcal{D}_m$ is 0. In the context of $LTL[\mathcal{F}]$ the operator G corresponds to *minimum*. Thus, if the atomic proposition p

	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{p_1, p_2\}$	$\{p_1\}$	$\emptyset$	$\{p_2\}$	$\emptyset$
p_1	0	0	0	0	1	1	0	0	0
p_2	0	0	0	0	1	0	0	1	0
$\mathcal{D}_5 p_1$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{3}{5}$	$\frac{4}{5}$	1	1	0	$\frac{1}{5}$	$\frac{2}{5}$
$\mathcal{D}_5 p_2$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{3}{5}$	$\frac{4}{5}$	1	$\frac{3}{5}$	$\frac{4}{5}$	1	$\frac{2}{5}$
$G\mathcal{D}_5 p_1$	0	0	0	0	0	0	0	0	0
$G\mathcal{D}_5 p_2$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$

Figure 3.1: Table showing the evaluation of individual, quantitative objectives

holds when the agent achieves their objective, then $G\mathcal{D}_m p = \frac{m-n}{m}$ when n represents their maximum waiting time. The computation, π, that maximises the satisfaction value of the formula $[\![\pi, G\mathcal{D}_m p]\!]$, will minimize the maximum waiting time for the agent concerned with proposition p.

For example, consider the computation $\pi = \emptyset, \emptyset, (\emptyset, \emptyset, \{p_1, p_2\}, \{p_1\}, \emptyset, \{p_2\}, \emptyset)^\omega$. There are two agents, a_1 and a_2 with logical propositions p_1 and p_2 which represent when the agents have achieved their respective goals. The individual quantitative objective of each agent is $\varphi_i = G\mathcal{D}_m p_i$, where $m = 5$. The table in Figure 3.1 notes the evaluations of $p_1, p_2, \mathcal{D}_5 p_1, \mathcal{D}_5 p_2, G\mathcal{D}_5 p_1$, and $G\mathcal{D}_5 p_2$.

The rows for $\mathcal{D}_5 p_1$ in Figure 3.1 are easiest understood when reading from right to left. When a cell of the table has the value 1, it means that in the state, the goal is satisfied. In the cell to the left, representing the moment before, the satisfaction value of $\mathcal{D}_5 p_1$ is $\frac{4}{5}$ - indicating a waiting time of $5 - 4 = 1$ and a threshold value of 5. We can also observe that both agents reach their goal twice during the loop portion of the computation. By the standards of LTL, the computation is starvation free, because the condition GFp_i holds for both agents. But, with the added nuance of quantitative starvation, we can see that agent a_1 reaches its goal twice consecutively and then must wait longer than the threshold value of 5 before the next occasion of meeting the goal. Hence, for agent a_1 the computation is unacceptable.

3.2.2 Quantitative Fairness

In a qualitative setting, fairness may refer to computations where all agents can satisfy their goals infinitely often, however, it does not consider the relative amount of times that agents can reach their goals. For example, if one agent satisfies their goals every 5 turns, but another must wait for 500 turns, this is intuitively not fair. Quantitative fairness facilitates this kind of reasoning by using a function to evaluate how fair a computation is, and a predicate to decide if this computation is fair enough.

The functions $\mathcal{F}$ which parameterise $LTL[\mathcal{F}]$ logic are defined as $\mathcal{F} \subseteq \{f : [0,1]^k \rightarrow [0,1] | k \in \mathbb{N}\}$ (as shown in Definition 2.8). Any function which maps a vector of values in $[0,1]$ to a value in $[0,1]$, can be included in $\mathcal{F}$. Given a set of agents, Agt, each with an associated logical proposition φ_i, fairness is represented by the dispersion function, with signature, $disp : V(\varphi_i)^{|Agt|} \rightarrow [0,1]$. The input of a dispersion function is a vector of the possible satisfaction values of each agent's individual objective. Higher values of the function indicate a higher level of dispersion and hence less fairness. As the vector of satisfaction values $V(\varphi_i)^{|Agt|}$ is in $[0,1]^{|Agt|}$, and dispersion functions maps to $[0,1]$, any dispersion function can be included in $\mathcal{F}$.

Consider a computation, π, of a system with n agents, each with a corresponding individual, quantitative $LTL[\mathcal{F}]$ objective, φ_i and an arbitrary dispersion function $disp$. Then if we have an $LTL[\mathcal{F}]$ formula Φ such that $[\![\pi, \Phi]\!] = disp([\![\pi, \varphi_1]\!], \dots)$. Φ is a collective, quantitative objective. A fairness objective would then be satisfied by a strategy profile which leads to a computation π, such that $[\![\pi, \Phi]\!] \in P$, where $P \subseteq [0,1]$ is a fairness predicate.

For example, consider the dispersion function $range(S) = max_{s \in S} - min_{s \in S}$, where S is a vector of satisfaction values; *range* is the difference between the largest and smallest of the input values. A high value indicates a large gap between the highest and lowest satisfaction values, which is a less fair outcome. Setting the fairness predicate $P = \{0\}$ would indicate a requirement that all agent's satisfaction values must be the same.

3.3 Collective Acceptability

The preceding sections laid the groundwork for defining appropriate $LTL[\mathcal{F}]$ formulas for synthesising strategy profiles in GDPG. Here, the objectives are defined more specifically. A strategy profile which leads to a computation which meets the requirements of Definition 3.3 is called *collectively acceptable.*

Definition 3.3 (Collectively Acceptablity)**.** Given a GDPG, $\mathcal{G}$, with n agents, each with a corresponding atomic logic proposition, p_i and the following objectives:

1. For each agent, $\varphi_i = GD_{m_i}p_i$, a quantitative, individual objective which represents the agent's maximum waiting time. Lower satisfaction values of φ_i correspond to higher maximum waiting time, so the satisfaction values should be maximised to minimize the maximum waiting time.

2. A collective objective $\Phi = min_{0\leq i\leq n}\{\varphi_i\}$, which represents the lowest satisfaction value (and hence, highest maximum waiting time) that any agent achieves. A predicate $P_s = (0, 1]$ is used to ensure that no agent receives a satisfaction value of zero (ensures that no agent starves);

3. A collective fairness objective $\Psi = range([\varphi_1, \ldots, \varphi_n])$. A predicate $P_f = [0, \frac{i}{n}]$ ensures that all agents' satisfaction values are within $\frac{i}{n}$ of one another for some arbitrary value i and consequently, the maximum waiting times are within i of one another.

A strategy profile is collectively acceptable if and only if it leads to a computation π, where Objectives 2 and 3, above, evaluate to values within their respective predicates; and is a Nash Equilibrium with respect to Objective 1 (which means that no agent can unilaterally deviate to achieve a shorter maximum waiting time).

Thus, we can see that $LTL[\mathcal{F}]$ is sufficiently expressive to include formulas for both qualitative and quantitative objectives, and for individual and collective objectives. By applying the framework introduced in this chapter, it is possible to create objectives that suit the requirements of Nash Equilibrium, fairness and starvation-freedom. In the next chapter, strategy profiles are examined in greater depth and the concept of strategy languages is introduced to facilitate automaton-based approaches to strategy profile synthesis.

Chapter 4

Strategy Language

The purpose of the algorithm developed in this dissertation is to generate collectively acceptable strategy profiles for CGMs, as defined in Definition 3.3 using an automaton-based approach. However, there is no intuitive way to represent strategy profiles in an automaton-based algorithm. The algorithm of the subsequent Chapter 5 constructs a Büchi Automaton which acts as an acceptor of strategy words and will only accept if the strategy word if it corresponds to a strategy profile which produces a collectively acceptable computation. For this reason, it is necessary to describe the strategy profiles of the agents as an ω-regular language, defined here as a *strategy language*. This chapter briefly explains some of the features of strategy languages and their relationship to various representations of strategy profiles.

A strategy profile, when enacted by a group of agents, will produce an infinite sequence of states and action profiles. At the beginning of the game run, the agents find themselves in a state. Based on the state the strategy profile prescribes an action profile. The agents then collectively perform the action profile and find themselves in some resultant state. This continues indefinitely.

Definition 4.1 (Strategy Language)**.** Given a CGM, $\mathcal{M}$, with states St and action profiles AP, the strategy language of $\mathcal{M}$ is $\mathcal{L}_{\mathcal{M}} \subseteq (St \times AP)^{\omega}$. A strategy word is a word of a strategy language, and the alphabet of a strategy language is all pairs (s, ap) where s is a state and ap is in action profile. A strategy profile, when enacted by the agents produces the strategy word such that for all adjacent pairs (s, ap), (s', ap'), such

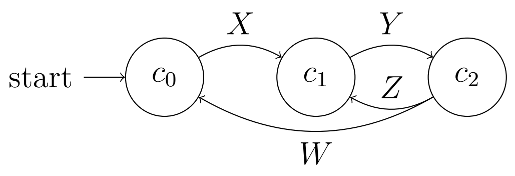

Figure 4.1: Simple CGM where the agents have multiple choices

that $Out(s, ap) = s'$, and $ap \in av(s)$, $ap' \in av(s')$ for all agents. s' is the outcome of enacting action profile ap in state s, and ap and ap' are available action profiles for their respectively states.

Note that any sequence of pairs St, AP can be a strategy word, but most strategy words will not produce a collectively acceptable outcome. In subsequent chapters, strategy automata will be constructed such that only strategy words that produce a precise outcome are accepted.

Generally, there will be several strategy words which represent an acceptable run of the CGM with respect to a given agent's individual objectives, see Figure 4.1. In this example, which assumes 2 agents, it is assumed that both agents reach their goal at configuration c_2. From c_0, they must progress to c_1 via action profile X, and similarly from c_1, they must progress to c_2 via action profile Y. However, from c_2, they can progress to either c_0 or c_1 via action profiles W and Z respectively.

It is clear that no matter which strategy profile is followed, the agents will always reach their goal with a maximum waiting time of 2 turns. The strategy language which describes runs of this model is provided by the ω-regular expression $\mathcal{L} = (c_0, X)(c_1, Y)((c_2, Z)(c_1, Y)|(c_2, W)(c_0, X)(c_1, Y))^\omega$. The agents must begin with (c_0, X), followed by (c_1, Y). Thereafter, however, they loop forever and for each iteration of the loop, they must choose between $(c_2, Z)(c_1, Y)$ and $(c_2, W)(c_0, X)(c_1, Y)$. There is an infinite number of strategy words in this language, and similarly, an infinite number of strategy profiles which can generate those words.

It is trivial to create a positional strategy profile which produces a word $w \in \mathcal{L}$. $\{(c_0 \to X), (c_1 \to Y), (c_2 \to Z)\}$ is such an example. A finite-memory strategy profile can be created by defining a 2-state memory. Presuming that there is some transition system which defines transitions between the two memory states, in state 1, the agents will choose Z when c_2 is detected and in state 2, the agents will choose W when c_2 is detected.

Finally, we will show an infinite memory strategy profile that will produce strategy words in $\mathcal{L}$. Firstly, for convenience, we can represent the decisions made by the agents in a more compact form. If they choose $(c_2, Z)(c_1, Y)$ it is represented as 0 and if they choose $(c_2, W)(c_0, X)(c_1, Y)$ it is represented as 1. Hence we can represent the sequence of decisions made with a base-2 number. Next, consider some irrational number in base-2 which represents the sequence of decisions made, that is, it can be seen as representing a strategy profile. Consequently, we see that an infinite amount of memory would be required to store this strategy profile; and, there are infinitely many such strategy profiles.

Due to the potential intractability of dealing with infinitely large sets, any brute force algorithm will need bouned memory usage at some point and can only give a weak assurance of collective acceptability. Instead, we will reason about strategy languages. Sets of perhaps infinitely many strategy words which share particular objective-satisfaction characteristics will be described with Büchi automata, and instead of directly creating a strategy profile which is collectively acceptable, a corresponding strategy language will be produced instead. From the strategy language, strategy profiles can be distilled.

In the next chapter, the strategy automaton and deviation automaton are developed as acceptors of strategy languages.

Chapter 5

Construction of Automata for Collectively Acceptable Strategy Profiles

Here we begin with the procedure for generating collectively acceptable strategy profiles. Given a GDPG with n agents, a set of n quantitative individual objectives (one for each agent), as well as a collective fairness objective and predicate, a collective starvation-freedom objective and predicate, and a threshold value for starvation, an automaton is generated which only accepts a strategy word if that word corresponds to a strategy profile which is collectively acceptable. Thereafter, a finite-memory strategy profile is distilled from the automaton.

Section 5.1 provides the conversion procedures which are necessary to bring the realm of GDPG to $LTL[\mathcal{F}]$ model checking. Thereafter $LTL[\mathcal{F}]$ model checking is outlined in Section 5.2, and some necessary augmentations are explained to account for multiple agents that have their own objectives, additionally, heuristics for fast construction of automata in the special case of quantitative starvation are examined. Payoff profiles are introduced in Section 5.3, and they are subsequently used to guide the construction of strategy automata and deviation automata, 5.4, and Nash automata 5.5. Finally, the various threads are pulled together in Section 5.6 to produce an automaton that only accepts strategy words that correspond to collectively acceptable strategy profiles.

5.1 Translation Procedures

Two conversions are defined in this section. The first translates a GDPG into a corresponding CGM, and the second translates a CGM into a corresponding NGBA appropriate for $LTL[\mathcal{F}]$ model checking. While it is possible to translate the GDPG directly to an NGBA, the intermediary step provides an added measure of generality which will support future work where it may be possible to apply this procedure to other CGMs.

Definition 5.1 (Translation of GDPG to CGM). Let $\mathcal{G} = \langle Agt, Bags, d, Acc\rangle$ be a GDPG with $Conf$ as the set of all configurations in $\mathcal{G}$. Then $\mathcal{G}$ can be translated into a CGM $\mathcal{M} = \langle \mathcal{A}, St, Act, av, out, Prop, L\rangle$ where:

- $\mathcal{A} := Agt$
- $St := \{c : c \in Conf\}$
- $$\begin{aligned} Act = &\{Take^a_\beta : a \in \beta \wedge \beta \in Bags\} \\ &\cup \{Release^a_\beta : a \in \beta \wedge \beta \in Bags\} \\ &\cup \{Release^a_{all} : a \in Agt\} \\ &\cup \{Idle^a : a \in Agt\} \end{aligned}$$
- For all pairs of $c \in St$ and $a \in \mathcal{A}$,
  $$\begin{aligned} av(c,a) = &\{Take^a_\beta : a \in \beta \wedge \beta \in Bags \wedge c(\beta)(null) > 0 \wedge card(c,a) < d(a)\} \\ &\cup \{Release^a_\beta : a \in \beta \wedge \beta \in Bags \wedge c(\beta)(a) > 0 \wedge card(c,a) < d(a)\} \\ &\cup \{Release^a_{all} : card(c,a) = d(a)\} \\ &\cup \{Idle^a : card(c,a) \neq d(a)\} \end{aligned}$$
- Given a configuration c, and an allowed action profile ap, the successor c' is the new configuration which arises when the agents follow ap while in c. The successor configuration is defined piecewise. Let $out(c, ap) = c'$, then for all resource bags, β:

- $\star$ $c'(\beta)(null) = c(\beta)(null)$
  $- |\{a \in Agt : ap(a) = Take^a_\beta \wedge |\{a' \in Agt : ap(a') = Take^{a'}_\beta\}| < c(\beta)(null)\}|$
  $+ |\{a \in Agt : ap(a) = Release^a_\beta\}|$
  $+ \sum_{a \in Agt : ap(a) = Release^a_{all}} c(\beta)(a)$
- $\star$ And, for all $a \in Agt$:

$$c'(\beta)(a) = \begin{cases} c(\beta)(a) + 1 & \text{iff } Take^a_\beta = ap(a) \wedge \\ & |\{a' \in Agt : ap(a') = Take^{a'}_\beta\}| < c(\beta)(null) \\ c(\beta)(a) - 1 & \text{iff } Release^a_\beta = ap(a) \\ 0 & \text{iff } Release^a_{all} = ap(a) \end{cases}$$

Each resource bag allocation to *null* is incremented for all successful *Release* and $Release_{all}$ actions, and decremented for all successful *Take* actions and each resource bag allocation to an agent is decremented for all successful *Release* and $Release_{all}$ actions by that agent and incremented for all successful *Take* actions by that agent;

- $Prop := \{p_a : a \in \mathcal{A}\}$, where each p_a is a new atomic proposition introduced for the CGM;
- For all $c \in St$, $L(c) := \{p_a : a \in \mathcal{A}, card(a, c) = d(a)\}$

Definition 5.2 (Translation of a CGM to an NGBA)**.** Given a CGM $\mathcal{M} = \langle \mathcal{A}, St, Act, av, out, Prop, L\rangle$, then $\mathcal{M}$ can be translated into a NGBA $\mathcal{N} = \langle 2^{AP}, Q, \delta, Q_0, \alpha\rangle$ where:

- $AP := Prop$
- $Q := St$
- For all $q \in Q$ and $\sigma \in 2^{AP}$, $\delta(q, \sigma) := \{q' : (\exists ap : out(q, ap) = q') \wedge L(q') = \sigma\}$
- $Q_0 := Q$
- $\alpha := \{Q\}$

The first step of the procedure is to convert the given GDPG into the corresponding NGBA by the translations defined in Definitions 5.1 and 5.2.

5.2 LTL[F] Model Checking for Multiple Agents

The $LTL[\mathcal{F}]$ model checking procedure takes as input a model and a specification (both represented as NGBAs) and then produces an output (also an NGBA) which only accepts a computation if it is a valid computation of the model and it satisfies the specification. We may wish to ask the questions, "Do all computations satisfy the specification?", or "Does there exist a computation which satisfies the specification?". Strategy profile synthesis differs in that it asks the question, "What is the strategic behaviour required to produce a computation which satisfies the specification?".

In our case, the question is more complicated because the Boolean satisfiability problem is replaced with the optimisation of some quantitative satisfaction values. Moreover, we are concerned with the identification of a Nash equilibrium, which cannot be expressed in the logic $LTL[\mathcal{F}]$. Because Nash equilibrium cannot be expressed with the logic, some extra steps are needed to identify this property. Naturally, $LTL[\mathcal{F}]$ model checking can be of use, but some augmentations are necessary.

In the classic case of $LTL[\mathcal{F}]$ model checking, a model and a $LTL[\mathcal{F}]$ are provided and converted into corresponding NGBAs. One NGBA only accepts computations that can arise from a valid run of the model, and the other only accepts computations that provide a satisfaction value for an $LTL[\mathcal{F}]$ formula that is within a given interval. A necessary modification of the $LTL[\mathcal{F}]$ model checking procedure, for the procedure presented in this dissertation, is to have several $LTL[\mathcal{F}]$ objectives, each associated with a distinctive interval $P \in [0, 1]$ - known as a predicate. The output will then be an NGBA which only accepts computations for which the satisfaction values of each $LTL[\mathcal{F}]$ objective are within the respective predicate. In LTL model checking, this step is not needed because the objectives of multiple agents can simply be combined to $p_1 \wedge p_2 \wedge p_3 \wedge \dots$, however in conventional $LTL[\mathcal{F}]$ model checking, only one predicate is provided. We begin by defining a helper function and then proceed with the modified model-checking procedure as described above.

Definition 5.3 (Predecessor of Satisfation Value)**.** Given an $LTL[\mathcal{F}]$ formula φ, and its corresponding set of satisfaction values $V(\varphi)$, for all $v \in V(\varphi)$, the *predecessor* of v is $Pred(v) = \{v' \in V(\varphi) : v' > v \wedge \forall v'' \in V(\varphi) \setminus \{v, v'\} : v'' > v' \rightarrow v'' > v \wedge v'' < v' \rightarrow$

$v'' < v\}$.

The values of $V(\varphi)$ are generated by applications of the $\nabla_{\frac{n-i}{n}}$ operator for increasing integer values of i and a fixed threshold value n, thus the set of satisfaction values form a finite sequence $\frac{n}{n}, \frac{n-1}{n}, \ldots, \frac{1}{n}, 0$. A predecessor function is defined such that $Pred(v)$ will contain the satisfaction value that comes directly before v in the sequence. There is also the special case where v is the first item in the sequence, in which case $Pred(v) = \emptyset$.

Definition 5.4 (Translation of an $LTL[\mathcal{F}]$ formula to an NGBA)**.** An $LTL[\mathcal{F}]$ formula, φ of the form $G\mathcal{D}_m p$, where m is a threshold value, and predicate P can be translated into an NGBA $\mathcal{N}_\varphi^P = \langle 2^{AP}, Q, \delta, Q_0, \alpha \rangle$, where:

- AP is the set of atomic propositions that occur in φ;
- $Q = C_\varphi$, the set of consistent functions with regard to φ;
- $Q_0 = \{q \in Q : q(\varphi) \in P\}$;
- For all pairs, $q, q' \in Q$ and propositions, $p \in AP$, $\delta(q, r) = q'$, where:
  - ⋆ $q(\mathcal{D}_m p) = \frac{1}{2} \leftrightarrow p = r$
  - ⋆ $q(\mathcal{D}_m p) \notin \{\frac{1}{2}, 1\} \leftrightarrow p \neq r$
  - ⋆ $q(\mathcal{D}_m p) \notin \{0, 1\} \leftrightarrow q'(\mathcal{D}_m p) \in Pred(q(\mathcal{D}_m p))$
  - ⋆ $q(\mathcal{D}_m p) = 0 \leftrightarrow q'(\mathcal{D}_m p) = 0 \vee q'(\mathcal{D}_m p) \in Pred(0)$
  - ⋆ $q'(G\mathcal{D}_m p) \leq q(G\mathcal{D}_m p)$
- $\alpha = \{q \in Q : \{p \in AP : q(\mathcal{D}_m p = 0)\}\}$. We must always reach all states where some p are satisfied. In such states, the waiting time is zero.

Note that the predicate, P, is only used to determine the initial states of the NGBA. For this reason, an NGBA can be generated irrespective of the particular predicate, and then the initial states are modified to account for different predicates. Thus, generating these NGBAs for a reasonable number of different predicates does not impose a significant computational penalty.

5.2.1 Construction of NGBAs for Objectives of Individual Agents

The construction of NGBAs for the special purpose of synthesising strategies which avoid quantitative starvation can be simplified. Firstly, it is noted that agent objectives are individual objectives (see Section 3.1). Given an agent of a GDPG with atomic proposition p which is satisfied when the agent meets their goal, and a threshold value $m \in \mathbb{N}$ and a value, $t \in \mathbb{N}$, where, $0 \leq t \leq m$, then a new type of NGBA, $\mathcal{N}_t^p$ can be constructed which will only accept a computation if the agent can always meet their goal with a maximum waiting time of t. That is, $G\mathcal{D}_m p >= \frac{m-t}{m}$. The construction of this NGBA is sketched.

The automata $\mathcal{N}_t^p$ are defined inductively. First, we consider an NGBA which only accepts words where the waiting time for atomic proposition p to be true is zero - the agent whose objective is represented by p never waits. $\mathcal{N}_0^p$ is represented pictorially in Figure 5.1 (note that all states of these NGBAs are possible initial states, but to avoid cluttering the automaton the arrows indicated initial states are omitted). The automaton only has one state, which has a self-loop labelled with p, and it is defined as:

- $2^{AP} = \{\{p\}, \emptyset\}$;
- $Q = \{q : q(G\mathcal{D}_m p) = 1\}$;
- Transitions δ follow the same restrictions as the construction of the NGBA in Definition 5.4.
- $Q_0 = Q$;
- $\alpha = \{\{q \in Q : q(\mathcal{D}_m p) = 1\}\}$.

In Definition 5.4 it was noted that there is an element of α for each atomic proposition. For this reason, all of these single-agent NGBAs will have α be a set containing only a single set.

Next, we see an NGBA with 2 states, a computation will be accepted by this NGBA if the maximum number of the symbol $\emptyset$ encountered in a sequence is one, ie. the maximum waiting time is 1, and $G\mathcal{D}_2 p = \frac{1}{2}$. This NGBA is represented pictorially in Figure 5.2 and is formally defined as:

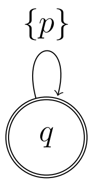

Figure 5.1: The single-agent NGBA, $\mathcal{N}_0^p$, where $G\mathcal{D}_m p = \frac{m-0}{m} = 1$

- $2^{AP} = \{\{p\}, \emptyset\}$;
- $Q = \{q : q(G\mathcal{D}_m p) = \frac{m-1}{m}\}$;
- Transitions δ follow the same restrictions as the construction of the NGBA in Definition 5.4;
- $Q_0 = Q$;
- $\alpha = \{\{q \in Q : q(\mathcal{D}_m p) = \frac{m-1}{m}\}\}$.

Finally, by induction, we can define the NGBA for the general case. $\mathcal{N}_t^p$ is represented in Figure 5.3 and is formally defined:

- $2^{AP} = \{\{p\}, \emptyset\}$;
- $Q = \{q : q(G\mathcal{D}_m p) = \frac{m-t}{m}\}$;
- Transitions δ follow the same restrictions as the construction of the NGBA in Definition 5.4
- $Q_0 = Q$;
- $\alpha = \{\{q \in Q : q(\mathcal{D}_m p) = \frac{m-t}{m}\}\}$.

Since agent objectives are orthogonal, it is possible to construct an NGBA for each agent objective and a desired maximum waiting time independently. These NGBAs can then be combined to form a new one which accepts a word iff that word is accepting on all constituent NGBAs and consequently provides a specified satisfaction value for the $LTL[\mathcal{F}]$ objective for each agent.

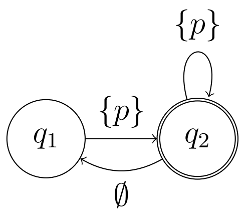

Figure 5.2: The single-agent NGBA, $\mathcal{N}_1^p$ where $G\mathcal{D}_m = \frac{m-1}{m}$

5.2.2 Product of NGBAs

As we defined an NGBA for each agent and their specified waiting time, these NGBAs need to be combined into a single NGBA such that the NBGA only accepts a computation, π, if π results in the specified waiting time for all agents. The NGBA product is formally defined as:

Definition 5.5 (NGBA product)**.** Given an NGBA $\mathcal{N} = \langle 2^{AP}, Q, \delta, Q_0, \alpha \rangle$ and $\mathcal{N}' = \langle 2^{AP'}, Q', \delta', Q'_0, \alpha' \rangle$. The product of $\mathcal{N}$ and $\mathcal{N}'$ is $\mathcal{N}'' = \langle 2^{AP''}, Q'', \delta'', Q''_0, \alpha'' \rangle$, where:

- $AP'' = AP \cup AP'$;
- $Q'' = Q \times Q'$;
- $Q''_0 = \{(q \times q') \in Q'' : q \in Q_0 \wedge q' \in Q'_0\}$;
- $\delta'' = \{((q \times q'), l, (q_2 \times q'_2)) : q \in Q \wedge q' \in Q' \wedge (q, b, q_2) \in \delta \wedge (q', b', q'_2) \in \delta' \wedge l = b \cup b'\}$;
- $\alpha'' = \bigcup_{s \in \alpha} \{(q \times q') \in Q'' : q \in s\} \cap \bigcup_{s' \in \alpha'} \{(q \times q') \in Q'' : q' \in s'\}$

The successive application of this definition allows the construction of the product of arbitrarily many NGBAs.

Using the above, it is possible to construct the NGBAs suitable for the $LTL[\mathcal{F}]$ model checking procedure. Put simply, if there is an agent a with proposition p_a, and the requirement that their maximum waiting time is t_a, then we can construct $\mathcal{N}_{t_a}^{p_a}$. This is done for all agents and the result is an NGBA for each agent. The product of these NGBAs can then be taken by application of Definition 5.5 to produce an NGBA that will only accept a computation π if all agent's maximum waiting times are adhered to.

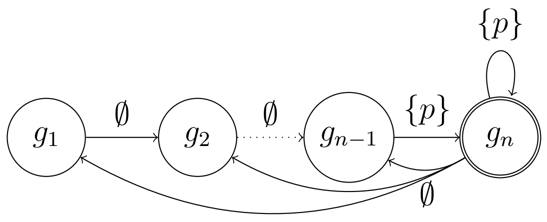

Figure 5.3: The single-agent NGBA, $\mathcal{N}_n^p$ where $G\mathcal{D}_m = \frac{n-N}{n}$

The main benefit of this process is the "cherry-picking" of exact maximum waiting times for each agent, which is elaborated in the following section.

5.3 Payoff profiles

At this point, it is necessary to address an ambiguity that has been introduced. In the opening paragraph of Chapter 5, it was mentioned that for each $LTL[\mathcal{F}]$ objective, there must be an associated predicate for $LTL[\mathcal{F}]$ to be done, but in Section 5.2, maximum waiting times were used to construct NGBAs for model checking. These concepts can, however, be easily linked. Consider for example an agent, with associated atomic proposition p, and the objective $\varphi = G\mathcal{D}_m p$ for a threshold $m \in \mathbb{N}$. If there is a computation π where the maximum waiting time is t, then $[\![\pi, \varphi]\!] = \frac{m-t}{m}$. There is a satisfaction value which corresponds directly to the maximum waiting time.

Furthermore, a predicate can be easily defined which includes only the required satisfaction value. That is, let $P = \{\frac{m-t}{m}\}$, then $[\![\pi, \varphi]\!] = \frac{m-t}{m} \leftrightarrow [\![\pi, \varphi]\!] \in P$.

Hence, it is possible to create a predicate for each value in $V(\varphi)$ which facilitates the selection of exactly one satisfaction value (and exactly one maximum waiting time). Instead of constructing NGBAs with particular predicates, we construct NGBAs for the corresponding waiting times. Since this is done for all agents, it is possible to create an NGBA product which only accepts a computation which results in a specific payoff to each agent. In other words, there is an NGBA for each possible *payoff profile*.

Definition 5.6 (Payoff Profile)**.** Given the GDGP with agents Agt and possible satisfaction values for their objectives, $V(\varphi)$, a payoff profile, PP, is a mapping of agents to satisfaction values $PP : Agt \rightarrow V(\varphi)$.

For example, if there are n agents and the set of possible satisfaction values of the $LTL[\mathcal{F}]$ objectives is $\{v_1, v_2, \dots\}$, then a payoff profile PP is defined as $PP(a_1) = v_i$, $PP(a_2) = v_k$, etc.

Definition 5.7 (NGBA Corresponding to a Payoff Profile)**.** Consider a game, $\mathcal{G}$, with n agents, each with a corresponding individual $LTL[\mathcal{F}]$ objective, φ_i and a payoff profile PP. The game is represented as a CGM, $\mathcal{M}$, and the NGBA corresponding to the CGM is $\mathcal{N}_{\mathcal{M}}$. For each agent's individual objective we construct the NGBA $\mathcal{N}_{\varphi_i}^{P_i}$ where $P_i = \{PP(a_i)\}$ is the predicate. A satisfaction value is only in P_i if it is exactly the value assigned to the agent by the payoff profile. Hence, the NGBA corresponding to $\mathcal{G}$ resulting in the payoff profile PP is $\mathcal{N}_{\mathcal{G}}^{PP} = \mathcal{N}_{\mathcal{M}} \times \mathcal{N}_{\varphi_1}^{P_1} \times \cdots \times \mathcal{N}_{\varphi_n}^{P_n}$

A benefit of payoff profiles is that for some collective objectives, there is an intuitive way to look at payoff profiles and assess if computations resulting in the payoff profile produce satisfaction values within the predicate. For example, consider a scenario where *range* (see Section 3.2.2) is chosen as the dispersion function and the predicate is $P_f = \{0\}$ - that is, the satisfaction value for each agent's objective must be the same. Then it is straightforward to assess a payoff profile for conformance with the fairness objective and predicate. PP_1 would not be fair because $range(\{0, 1\}) = 1$, but PP_2 would be fair because $range(\{\frac{1}{2}, \frac{1}{2}) = 0$. Note that P_f is a predicate for the fairness objective and is defined *in addition to* the individual agent predicates.

Given a computation π, a collective $LTL[\mathcal{F}]$ objective Φ, where $\Phi = f([\![\pi\varphi_1]\!], \dots, [\![\pi, \varphi_n]\!])$ and f is an $LTL[\mathcal{F}]$ function and each φ_i is an individual objective of an agent, then it is possible to evaluate a given payoff profile directly and decide if a computation leading to that payoff profile would satisfy Φ. For example, consider a collective objective $\Phi = min([\![\pi\varphi_1]\!], \dots, [\![\pi, \varphi_n]\!])$ - the satisfaction value of Φ over a computation is equal to the lowest individual agent satisfaction value. If this objective has the predicate $P = [\frac{1}{2}, 1]$ then we can say that if a payoff profile has in its domain a value less than $\frac{1}{2}$, then any computation which results in this payoff profile will not satisfy Φ.

Hence, for a payoff profile PP, if $f(PP(a_1), \dots, PP(a_n)) \in P$ (Φ is satisfied) then all computations, π resulting in PP will have $[\![\pi, \Phi]\!] \in P$.

The main rationale behind using payoff profiles in this way is that the synthesis procedure presented in this chapter iterates over the set of payoff profiles and then synthesises strategy profiles which result in each of these payoff profiles. If a given payoff profile does not satisfy one of the collective objectives, then we can simply avoid synthesising strategy profiles for it. If there is a Nash Equilibrium strategy profile which results in an unsatisfactory payoff profile, we can also disregard it.

At this stage, we can construct the NGBA, $\mathcal{N}_{\mathcal{G}}^{PP}$ for each payoff profile. These NGBAs will be used in the following chapters for the synthesis of Nash equilibrium strategy profiles.

5.4 Strategy and Deviation Automaton Construction

As mentioned in Chapter 4, a strategy language can be constructed. Here, we will generate the strategy language which comprises all words which represent strategy profiles that result in a computation that produces a specific payoff profile. It is straightforward to identify if some payoff profile is an improvement for a given agent over some over payoff profile, therefore, if strategy language $\mathcal{L}(S_1)$ contains only strategy words resulting in payoff profile PP_1, and $\mathcal{L}(S_2)$ contains only strategy words resulting in payoff profile PP_2, then a rational agent following a strategy from S_1 will have the incentive to deviate to a strategy from S_2 if the associated payoff profile yields a better satisfaction value.

For example, given the payoff profile $PP_1 = \{(a_1, 0), (a_2, 1)\}$ and $PP_2 = \{(a_1, \frac{1}{2}), (a_2, \frac{1}{2})\}$, it is clear that for agent a_1 PP_2 yields a higher satisfaction value than PP_1, the opposite is true for a_2. Thus, if there are automata $\mathcal{S}_1$ and $\mathcal{S}_2$ which accepts strategy words which result in payoff profile PP_1 and PP_2 respectively (the construction of such automata, called strategy automata, will be formally defined in Section 5.4.1), then for agent a_1, any strategy word from $\mathcal{S}_2$ will yield a higher satisfaction value than any strategy word from $\mathcal{S}_1$. More formally, for all computations π_1 and π_2 which correspond to a strategy word in $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively, we have $[\![\pi_2, \varphi_1]\!] > [\![\pi_1, \varphi_1]\!]$ and $[\![\pi_1, \varphi_2]\!] > [\![\pi_2, \varphi_2]\!]$, where φ_1 and φ_2 are the objectives of agents a_1 and a_2, respectively. Hence for all strategy profiles derived from $\mathcal{S}_1$, sp_1, and $\mathcal{S}_2$, sp_2 the following properties, $sp_2 >^{\varphi_1} sp_1$ and $sp_1 >^{\varphi_2} sp_2$ hold.

A strategy profile, when enacted by a group of agents, generates a computation

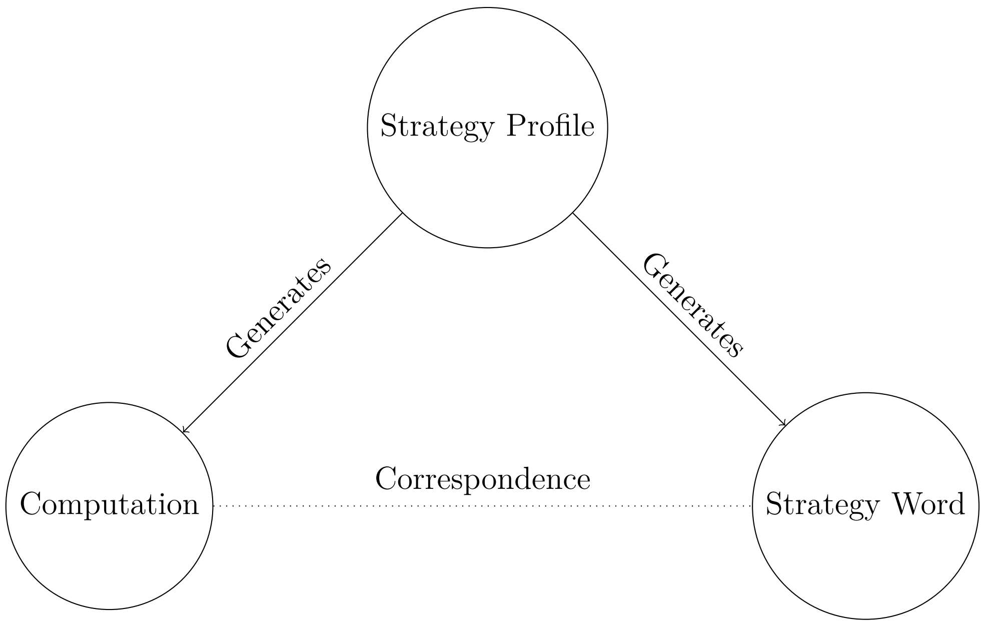

Figure 5.4: Relationship between Strategy Words, Computations and Strategy Profiles

which can be represented as a strategy word. Thus we can say that a computation, π, *corresponds* to a strategy word, w, (and vice versa) if they can be derived from the same strategy profile. This is notated as $\pi \widehat{=} w$ Each strategy profile generates exactly one strategy word and computation, but, a strategy word or computation can be generated by more than one different strategy profile, as depicted in Figure 5.4.

5.4.1 Strategy Automata

Because the model-checking automaton is a product of several Buchi automata, each of the states is a tuple consisting of a state from each of the original automata in the product composition. One automaton represents the game model, and thus, the first item in the model checking automaton state tuple is a configuration of the GDPG, the rest are states of the Buchi automaton representations of the $LTL[\mathcal{F}]$ formulas. Transitions are labelled by a set of atomic propositions. Hence each transition of the model checking automaton has the form $(c_i, n_1, \dots), \lambda, (c_j, m_1, \dots)$ where c_i, c_j are configurations, n_1, m_1 are states of the NGBA representing the $LTL[\mathcal{F}]$ formula and $\lambda \in 2^{AP}$.

A helper function is first given, and thereafter the translation procedure from the

$LTL[\mathcal{F}]$ model checking automaton to a strategy automaton is given.

Definition 5.8 (Implied action profiles function)**.** Given a transition of the model checking automaton, $d = (c_i, n_1, \ldots), \lambda, (c_j, m_1, \ldots)$, and the CGM underlying the model checking automaton, $\mathcal{M}$, the implied action profile function maps d to all the action profiles which would lead from the origin configuration of d, to the destination configuration of d.

$$imp(d) = \{ap : out(c_i, ap) = c_j\}$$

Definition 5.9 (Translation of the model checking automaton to a Strategy Automaton)**.** Given a GDPG with set of configurations $Conf$, the model checking automaton, $\mathcal{N}_{\mathcal{G}}^{PP} = \langle 2^{AP}, Q, \delta, Q_0, \alpha\rangle$, can be translated into a strategy Automaton $\mathcal{S}_{\mathcal{F}}^{PP} = \langle \Lambda, Q', \delta', Q_0', \alpha'\rangle$, where:

- $\Lambda = Conf \times ACT$, the set of all pairs of configurations and action profiles;
- $Q' = Q$;
- $\delta' = \bigcup_{d \in \delta}\{(q, a, q') : (q, r, q') = d \wedge a = imp(d) \wedge r \in 2^{AP}\}$;
- $Q_0' = Q_0$;
- $\alpha' = \alpha$.

The strategy automaton is the bridge between speaking of acceptable computations and strategy profiles. If we begin with the construction of the automaton corresponding to a specified payoff profile; and apply the above translations, a strategy automaton is generated whose language characterises strategy profiles which, when enacted by a group of agents, will result in acceptable computations, and consequently, yield the underlying payoff profile. It is thus possible at this stage to derive strategy profiles which result in a specified payoff profile.

From the strategy automata, we can derive strategy profiles which result in specified payoff profiles, but our ultimate goal is to synthesise Nash equilibrium strategy profiles. Nash equilibrium is concerned with unilateral deviations by agents, so next we construct an automaton that will accept any strategy word which can be produced if a given rational agent can take any action, and not just those that result in the underlying payoff profile.

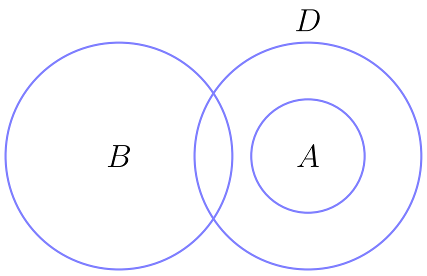

Figure 5.5: Venn Diagram showing the languages of 2 strategy automata, one with its associated deviation automaton

5.4.2 Deviation Automaton

In Definitions 2.19, 2.20, for profitable and unilateral deviation, deviation was defined as a relation between two strategy profiles. However, since a strategy profile has a corresponding strategy word, these definitions can be extended naturally to strategy words. Given an agent a_i with objective φ_i, and strategy profiles sp and sp' with corresponding strategy words w and w', w is a profitable deviation of w' for agent a_i if $sp >^{\varphi_i} sp'$; and w is a unilateral deviation of w' for agent a_i if $sp \curlyvee^{a_i} sp'$.

Note that per the unilateral deviation definition in 2.20, every strategy profile (and consequently strategy word) is a deviation of itself for all agents. Thus, given a GDPG, $\mathcal{G}$ with an agent a_i, if we have an arbitrary set of strategy profiles, A and the set of unilateral deviations, $D = \{sp : \exists sp' : sp \curlyvee^{a_i} sp'\}$, then $A \subseteq D$.

In the Venn diagram in Figure 5.7, we can see 3 sets, A, D and B. Sets A and D are the sets described in the previous paragraph, while B is some other arbitrary set of strategy profiles. Note that $A \cap B = \emptyset$, but $B \cap D \neq \emptyset$.

Let us assume that there are two payoff profiles PP_1 and PP_2. These are the only possible payoff profiles. The set A contains all strategy profiles which result in PP_1 and B results in all payoff profiles which result in PP_2. D would therefore contain all strategy words which correspond to a strategy profile sp, such that $sp \curlyvee^{a_i} sp'$ for an agent a_i and sp' corresponds to a strategy word that is accepted by A. That is, given a strategy word in D, agent a_i will either be receiving the payoff of PP_1, or can receive that payoff if they unilaterally deviate.

Consequently, $D \backslash A$ will accept a strategy word if that word corresponds to a strategy profile which does not result in PP_1, but agent a_i can deviate to achieve PP_1, if it has an incentive to.

$(D \setminus A) \cap B$ is all the strategy words which correspond to strategy profiles which result in payoff profile PP_2, but a_i can unilaterally deviate to achieve a payoff profile of PP_1.

Moreover we can presume the existence of strategy automata $\mathcal{S}_{\mathcal{G}}^{PP1}$ and $\mathcal{S}_{\mathcal{G}}^{PP2}$, such that $\mathcal{L}(\mathcal{S}_{\mathcal{G}}^{PP1}) = A$ and $\mathcal{L}(\mathcal{S}_{\mathcal{G}}^{PP2}) = B$. A deviation automaton $\mathcal{S}_{PP_1}^{\Delta a_i}$ will be defined such that $\mathcal{L}(\mathcal{S}_{PP_1}^{\Delta a_i}) = D$.

Finally, let $PP_1(a_i) > PP_2(a_i)$, that is, agent a_i receives a better payoff from strategy words in A than in B. We can make the following assertions about the deviation behaviour of an arbitrary strategy word w''.

- if $w'' \in A$, then a_i receives their better payoff when following the strategy profile corresponding to w'' and will not deviate.
- if $w'' \in (D \setminus A) \cap B$, then a_i is not receiving their best payoff and has an incentive to deviate to A, moreover, it is possible to unilaterally from the strategy profile corresponding to w'' to a strategy profile which corresponds to a word in A.

Now that the formalities of the sets have been addressed, we can proceed to the construction of the unilateral deviation function and deviation automaton.

Definition 5.10 (Unilateral deviation function)**.** The unilateral deviation function for agent a_i, dev^{a_i}, maps each transition of a strategy automaton to the set of transitions where a_i is possibly deviating, but not such that a new payoff profile arises after the deviation. $dev^{a_i}(c, ap, c') = \{(c, ap', c'') : \forall a_k \in (Agt \setminus \{a_i\}) ap(a_k) = ap'(a_k) \wedge ap' \in av(c, a_i) \wedge out(c, ap') = c''\}$

Definition 5.11 (Construction of Deviation Automaton for an agent corresponding to a Strategy Automaton)**.** Given a strategy automaton for a payoff profile, $\mathcal{S}_{\mathcal{G}}^{PP} = \langle 2^{AP}, Q, \delta, Q_0, \alpha \rangle$. The deviation automaton for agent a_i is a tuple: $\mathcal{S}_{PP}^{\Delta a_i} = \langle 2^{AP'}, Q', \delta', Q'_0, \alpha' \rangle$, where:

- $AP' = AP$;

- $Q' = Q$;
- $\delta' = \bigcup_{d \in \delta} dev^{a_i}(\delta)$;
- $Q'_0 = Q_0$;
- $\alpha' = \alpha$.

At this stage, we have addressed profitable and unilateral deviation in the case of only one deviating agent and two payoff profiles. To reason about Nash equilibria, it is necessary to extend profitable and unilateral deviation to arbitrary numbers of agents and payoff profiles.

5.5 Nash Automata

To begin with, given a GDPG, $\mathcal{G}$, an arbitrary payoff profile PP_1 is selected as the starting point in the search for Nash equilibrium. PP_1 has an associated strategy automaton, $\mathcal{S}_{\mathcal{G}}^{PP_1}$ which only accepts strategy words corresponding to strategy profiles which result in PP_1. There are three steps, firstly, the sets of alternative payoff profiles from PP_1 are constructed, secondly, the alternatives are used to construct the *Nash automaton for payoff profile* PP_1. Finally, we take a step back and define the Nash automaton for the entire game - an automaton which only accepts a strategy word if that word is a Nash equilibrium with respect to the individual objectives of each agent.

With the assumption of PP_1, each agent in $\mathcal{G}$ will have zero or more alternatives. An alternative payoff profile for an agent is the set of all payoff profiles which result in a better payoff to that agent.

Definition 5.12 (Alternative Payoff Profiles For An Agent)**.** Given a GDPG with agent a_i, a set of payoff profiles $\mathcal{P}$, and a payoff profile $PP_i \in \mathcal{P}$, $ALT(a_i, PP_1) = \{PP_k \in \mathcal{P} : PP_k(a_i) > PP_1(a_i)\}$.

If $\mathcal{S}_{PP_1}^{\Delta a_i}$ is the deviation automaton for agent a_i from strategy profiles resulting in PP_1, then we can construct the automaton X, such that $\mathcal{L}(X) = \bigcup_{PP \in ALT(a_i, PP_1)} \mathcal{L}(\mathcal{S}_{PP}^{\Delta a_i})$ which accepts all strategy words which correspond to a strategy profile from which a_i

can deviate to *any* of its alternatives which provide a better payoff than PP_1. Next, we can consider all of the agents, and construct the set of all strategy words which correspond to strategy profiles from which *any* agent can deviate to *any* of its alternatives which provide a better payoff than PP_1. This is, an automaton X such that $\mathcal{L}(X) = \bigcup_{a \in Agt} \bigcup_{PP \in ALT(a,PP_1)} \mathcal{L}(\mathcal{S}_{PP}^{\Delta a})$. Consequently, we can define the Nash automaton for the payoff profile PP_1.

Definition 5.13 (Nash Automaton for Payoff Profile)**.** Given a set of agents Agt from a GDPG, $\mathcal{G}$, and a payoff profile PP_1, the Nash automaton for PP_1 is such that $\mathcal{L}(\mathcal{S}_{nash}^{PP_1}) = \mathcal{L}(\mathcal{S}_{\mathcal{G}}^{PP_1}) \setminus \bigcup_{a_i \in Agt} \bigcup_{PP \in ALT(a_i,PP_1)} \mathcal{L}(\mathcal{S}_{PP}^{\Delta a_i})$. This is the set of all strategy words which correspond to a strategy profile which results in payoff profile PP_1 and for which no agent can deviate to any alternative payoff profile for that agent. Since no agent can unilaterally and profitably deviate, these strategy words correspond to Nash equilibrium strategy profiles.

If the Nash automaton for a payoff profile does not contain any words, that does not mean that there is no Nash equilibrium. Indeed, we must consider all possible payoff profiles as potential starting points. If we do this, we can define the general Nash automaton.

Definition 5.14 (Nash Automaton)**.** Given a set of agents Agt from a GDPG, $\mathcal{G}$, and a set of all possible payoff profiles $\mathcal{P}$, the Nash automaton is constructed such that $\mathcal{L}(\mathcal{S}_{nash}) = \bigcup_{PP_i \in \mathcal{P}} \mathcal{L}(\mathcal{S}_{nash}^{PP_i})$.

The Nash automaton acts as an acceptor of strategy words which correspond to strategy profiles that are Nash equilibrium, but Nash equilibrium is not sufficient for collectively acceptable strategy profiles. The next section combines $\mathcal{S}_{nash}$ with automata for fairness and starvation-freedom to construct an automaton which only accepts a strategy word if it corresponds to a collectively acceptable strategy profile.

5.6 Pulling it All Together

With all the heavy lifting attended to, the final step is to simply combine the Nash Automaton with automata which guarantee satisfaction of the collective objectives to

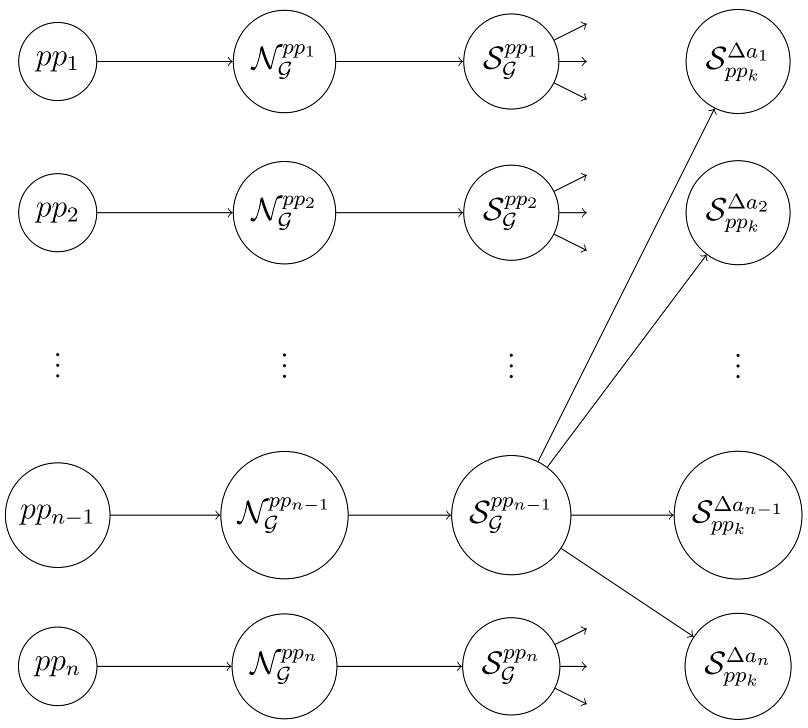
pp_1
$\mathcal{N}_{\mathcal{G}}^{pp_1}$
$\mathcal{S}_{\mathcal{G}}^{pp_1}$
$\mathcal{S}_{pp_k}^{\Delta a_1}$
pp_2
$\mathcal{N}_{\mathcal{G}}^{pp_2}$
$\mathcal{S}_{\mathcal{G}}^{pp_2}$
$\mathcal{S}_{pp_k}^{\Delta a_2}$
pp_{n-1}
$\mathcal{N}_{\mathcal{G}}^{pp_{n-1}}$
$\mathcal{S}_{\mathcal{G}}^{pp_{n-1}}$
$\mathcal{S}_{pp_k}^{\Delta a_{n-1}}$
pp_n
$\mathcal{N}_{\mathcal{G}}^{pp_n}$
$\mathcal{S}_{\mathcal{G}}^{pp_n}$
$\mathcal{S}_{pp_k}^{\Delta a_n}$

Figure 5.6: Relationship between payoff profiles, NGBAs, strategy automata and deviation automata.

produce an automaton which only accepts collectively acceptable strategy words; and thereafter, to produce a corresponding strategy profile.

5.6.1 Algorithm

Given a GDPG with n agents, each with an associated objective φ_i, the overall Nash automaton is constructed by combining the various strategy and deviation automata.

In Figure 5.6, we can see that there are finitely many payoff profiles. More specifically, there are $|Agt|^{|V(\varphi)|}$ distinct payoff profiles. For each payoff profile, an NGBA $\mathcal{N}$ can be constructed which only accepts a computation if that computation yields exactly the associated payoff profile. Then, for each of these NGBAs, $\mathcal{N}$ there is a strategy automaton which accepts a strategy word if that strategy word corresponds to a computation accepted by the associated NGBA. Finally, for each strategy automaton $\mathcal{S}$, we can construct n deviation automata. Altogether there are $|Agt|^{|V(\varphi)|} \times |Agt|$ deviation

Algorithm 1 Algorithm for construction of Nash automaton

```
Input: A GDPG $\mathcal{G}$, a set of all payoff profiles $\mathcal{P}$, the set of agents from the GDPG, $Agt$
Output: The Nash automaton for $\mathcal{G}$
procedure ConstructNashAutomaton($\mathcal{G}, \mathcal{P}, Agt$)    ▷ Constructs the Nash automaton for a given GDPG
    $S_{Nash} = \emptyset$
    for all $PP_1 \in \mathcal{P}$ do
        $\mathcal{N}_{\mathcal{G}}^{PP_1} := ModelChecking(\mathcal{G}, PP_1)$    ▷ As per Definition 5.7
        $\mathcal{S}_{\mathcal{G}}^{PP_1} := MC2SA(\mathcal{N}_{\mathcal{G}}^{PP_1})$    ▷ As per Definition 5.9
        $D := \emptyset$
        for all $a_i \in Agt$ do
            $ALT(PP_1, a_i) := Alternatives(PP_1, a_i)$    ▷ As per Definition 5.12
            $D_{a_i} := \emptyset$
            for all $PP \in ALT(PP_1, a_i)$ do
                $\mathcal{N}_{\mathcal{G}}^{PP} := ModelChecking(\mathcal{G}, PP)$
                $\mathcal{S}_{\mathcal{G}}^{PP} := MC2SA(\mathcal{N}_{\mathcal{G}}^{PP})$
                $\mathcal{S}_{PP}^{\Delta a_i} := Deviation(\mathcal{S}_{\mathcal{G}}^{PP})$    ▷ As per Definition 5.11
                $D_{a_i} := D_{a_i} \cup \mathcal{S}_{PP}^{\delta a_i}$
            end for
            $D := D \cup D_{a_i}$
        end for
        $S_{Nash} := S_{Nash} \cup (\mathcal{S}_{\mathcal{G}}^{PP_1} \setminus D)$
    end for
    return $S_{Nash}$
end procedure
```

automata. The algorithm for the construction of the Nash automaton is Algorithm 1.

The set construction for a Nash automaton for a particular payoff profile can also be visualised by means of the Venn diagram in Figure **.

In Figure 5.7, we see a large central representing the language of $\mathcal{S}_{\mathcal{G}}^{PP_1}$, which contains

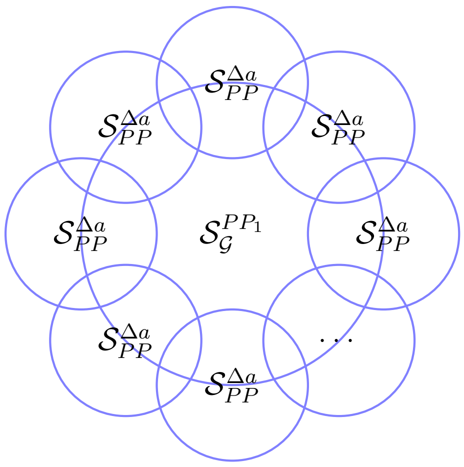
$\mathcal{S}^{\Delta a}_{PP}$
$\mathcal{S}^{\Delta a}_{PP}$
$\mathcal{S}^{\Delta a}_{PP}$
$\mathcal{S}^{\Delta a}_{PP}$
$\mathcal{S}^{PP_1}_{\mathcal{G}}$
$\mathcal{S}^{\Delta a}_{PP}$
$\mathcal{S}^{\Delta a}_{PP}$
$\dots$
$\mathcal{S}^{\Delta a}_{PP}$

Figure 5.7: Venn Diagram showing the language of a strategy automaton for a given payoff profile and the languages of deviation automata for all alternatives for all agents.

all strategy words which correspond to a strategy profile which results in payoff PP_1. Each of the surrounding circles shows the language of the deviation automaton for an alternative for an agent. If a strategy word falls in the overlap between the central strategy automaton and a deviation automaton, it means that given that strategy word, there exists a profitable and unilateral deviation for that agent. If a strategy is in the central circle but none of the overlapping regions, then there does not exist a profitable and unilateral deviation for any agent.

5.6.2 Collectively Acceptable Strategy Language

To recall, there were 3 requirements - 2 collective objectives which must be satisfied, and a set of individual objectives which must result in a Nash Equilibrium.

For a game with n agents, a fairness predicate P_f and dispersion function f, and a set of all payoff profiles $\mathcal{P}$, we can construct the set of all fair payoff profiles. Recall that a computation π is fair if $[\![\pi, \Phi]\!] \in P_f$. Thus we define the set $FAIR = \{PP \in \mathcal{P} : f(PP(a_1), \dots, PP(a_n)) \in P_f\}$. If $PP \in FAIR$, then for all computations π which result in PP, we know that $f([\![\pi, \varphi_1,]\!], \dots, [\![\pi, \varphi_n]\!]) \in P_f$. The collective starvation freedom objective works in much the same way, with its own predicate P_s and function g. Thus we define the set $FREE = \{PP \in \mathcal{P} : g(PP(a_1), \dots, PP(a_n)) \in P_s\}$.

If the set FAIR contains all payoff profiles which are considered fair, and FREE contains all payoff profiles which are starvation-free. Hence, we can produce $\mathcal{S}_{fair}$ such that $\mathcal{L}(\mathcal{S}_{fair}) = \bigcup_{PP \in FAIR} \mathcal{L}(\mathcal{S}_G^{PP})$ - a strategy automaton which accepts all strategy words corresponding to a strategy profile that results in a fair payoff profile, and $\mathcal{S}_{free}$, such that $\mathcal{L}(\mathcal{S}_{free}) = \bigcup_{PP \in FREE} \mathcal{L}(\mathcal{S}_G^{PP})$ - a strategy automaton which accepts all strategy words which correspond to strategy profiles resulting in a starvation free payoff profile. We also have the Nash automaton as defined in the previous section, $\mathcal{S}_{nash}$. Finally, an automaton which only accepts collectively acceptable strategy words can be expressed simply as $\mathcal{S}$ such that $\mathcal{L}(\mathcal{S}) = \mathcal{L}(\mathcal{S}_{free}) \cap \mathcal{L}(\mathcal{S}_{fair}) \cap \mathcal{L}(\mathcal{S}_{nash})$.

Chapter 6

Overview of Related Work

The GDPG has been examined from a model-checking perspective, [15] but thus far there is no existing work that examines the problem of automated synthesis with special reference to the GDPG. Furthermore, the combination of qualitative and quantitative goals has not been adapted to the GDPG.

Existing work in synthesis has addressed either the problem of rational synthesis [18][23], the problem of quantifying what was previously, purely boolean, logical systems [1][2][4], or the identification of Nash equilibria in automated synthesis problems [8][3] [23][9]. Much work has also been done on heuristics for strategy synthesis given intractable problem spaces (for example, chess playing systems) [27]. This dissertation differs by considering settings that combine automated synthesis, quantification, and Nash equilibria in the specific context of the generalised dining philosopher's problem.

Rational synthesis is defined as the automatic construction of a system from a specification such that the system satisfies the specification [18]. In contrast to synthesis, rational synthesis models the environment as composed of rational agents, agents acting to achieve their objectives.

Linear-Time Temporal Logic (LTL) [26][25], is an extension of propositional logic, which facilitates reasoning about how logical propositions vary over time. LTL objectives are used to specify qualitative objectives. Computation Tree Logic (CTL) [17] and ATL [5] extend LTL by considering events that *might* occur based on actions of agents. In assessing the satisfaction of objectives, it is necessary to consider how strategic behaviour amongst coalitions of agents might result in different objectives becoming satisfied.

The combination of quantitative and qualitative objectives is addressed in various ways. One approach taken is to provide a mechanism for mapping qualitative boolean objectives to quantitative values. For example, LTL conditions may be satisfied sooner, rather than later, and a discounting function maps qualitative objectives to numerical values based on how it takes for these objectives to become satisfied [1]. In addition to discounting operators, weights and priorities can be assigned to different combinations of LTL formulas [2].

The mapping of qualitative values to quantitative values can be included directly into the semantics of the logic used, as in Objective LTL (OLTL) [8], $LTL[\mathcal{F}]$ [3], and $SL[\mathcal{F}]$ [10]. The logic $LTL[\mathcal{F}]$ is chosen for this dissertation.

Quantification of objectives in the synthesis of strategies can also be achieved using a mean-payoff game. In a mean-payoff game, each game configuration allocates some number of points to each agent, the mean of the points gained by agents in a play can be computed, and agents act to maximise it [4].

When agents pursue quantitative objectives, Nash Equilibrium is a useful tool for examining and anticipating the actions of agents [8], especially in mean-payoff games [9]. It is also possible to consider cases where the mean payoff is kept strictly within some pre-defined bound [11], and when agents seek to maximise mean payoff[7]. $LTL[\mathcal{F}]$ lacks the expressiveness for mean payoff, so instead this dissertation has agents which seek to minimize maximum waiting time.

Qualitative and quantitative objectives can also be combined as in [21]. This dissertation differs in that qualitative objectives are used to specify collective objectives, while quantitative objectives are used to specify individual objectives.

Nash Equilibrium requires some assumptions to be made about whether or not agents will deviate in cases when their evaluation is not strictly better. For example, agents can either be controllable or not [23], or they may deviate even if it results in a slightly smaller evaluation [20]. This dissertation is restricted to cases where agents receive strictly better evaluations, as this is the Nash Equilibrium.

The implications of partial observability [12] and external schedulers [6] are explored, and this dissertation deals with cases of full observability and no external interference with the decision of agents. Synthesis can also be flipped by using mechanism design, where the game is adjusted to produce Nash Equilibria in the game [22].

The complexity of rational synthesis is 2EXPTIME for qualitative LTL objectives [14]. It is expected that considering quantitative objectives and Nash Equilibrium will affect the complexity.

Chapter 7

Conclusions

In this dissertation, a procedure for the automated synthesis of collectively acceptable strategy profiles for the generalised dining philosopher's game using an automaton-based approach was developed. A key feature of this procedure is that there is a combination of quantitative and qualitative objectives, as well as collective and individual objectives.

7.1 Summary of Conclusions

In the developed procedure, reasoning about quantitative and qualitative objectives is achieved by observing that the set of LTL formulas is a subset of $LTL[\mathcal{F}]$ if the right functions $\mathcal{F}$ are chosen. Moreover, any $LTL[\mathcal{F}]$ formula which can only map to the satisfaction values $\{0, 1\}$ can be interpreted as qualitative. Or, put differently, any qualitative formula is implicitly also quantitative, but only has two possible satisfaction values to map to. Additionally, if the goal of each agent is represented by an atomic logical proposition, then $LTL[\mathcal{F}]$ (and consequently also, LTL) objectives can be either individual or collective. If the formula representing the objective depends upon all atomic propositions representing the agents' goals then it is a collective objective, and if it depends upon only one proposition, then it is an individual objective.

The agents are given an individual objective which is a quantitative reinterpretation of the prevention of starvation. Each agent must repeatedly reach their goal before a waiting time threshold is reached, and the longer their longest waiting time, the worse their payoff. This objective is defined in $LTL[\mathcal{F}]$. Furthermore, collective objectives for fairness and starvation are defined. These objectives are qualitative because they can only take on satisfaction values of 0 or 1.

The synthesis of strategy profiles does not lend itself immediately to automaton-based approaches and it is necessary to construct a ω-regular language, where each word is analogous to a strategy profile. This then allows automata to function as acceptors of strategy profiles. Strategy languages are defined to facilitate this, and consequently the $LTL[\mathcal{F}]$ model checking procedure is modified to produce strategy automata which only accept a strategy word if the strategy profile which corresponds to that strategy word produces a specified payoff to each agent.

By the construction of the automata described in Chapter 5, it is clear that collectively acceptable strategy profiles for the GDPG can be synthesized by extending the $LTL[\mathcal{F}]$ model checking procedure. The automaton that results from running the $LTL[\mathcal{F}]$ model checking procedure is used to produce a strategy automaton - an automaton that accepts all strategy words which match a given payoff profile. Because there is a one-to-one correspondence between strategy profiles and the words accepted by a strategy automaton, we can synthesize these strategy languages using automata instead of thinking directly about strategy profiles.

Using the modified $LTL[\mathcal{F}]$ model checking procedure, we can produce an NGBA for each possible payoff profile. These NGBAs only accept a computation of the given GDPG if that computation yields the specified payoff profile. For each of these, we construct the corresponding strategy automaton for each payoff profile. The strategy automaton only accepts a strategy word if that strategy word corresponds to a computation which yields the associated payoff profile.

A further modification to the strategy automaton is done to produce a deviation automaton for each agent. Unilateral deviation is a relation between two strategy profiles, however, it can be extended to strategy words. If the strategy automaton accepts a strategy word that is associated with a strategy profile sp, then a corresponding deviation automaton for a given agent can be constructed to accept every strategy word that corresponds to a strategy profile that is a deviation from sp for that particular agent. If, given a strategy word, w, that is accepted by a deviation automaton for a given agent, then that agent could potentially deviate to the strategy profile corresponding to w to achieve the payoff associated with the deviation automaton. If that payoff is better for the deviating agent than the payoff offered by sp, then a rational agent will have an incentive to deviate.

To recap, each payoff profile has an associated strategy automaton, which has an associated deviation automaton for each agent. We iterate over each of these payoff profiles. Given a payoff profile, a strategy automaton is constructed; moreover, for each agent, there is a set of payoff profiles which are improvements for the agent. For each of these improved payoff profiles, a deviation automaton is constructed. If a strategy word is accepted by the strategy automaton, but not the deviation automaton, then the agent cannot deviate. All words which are accepted by the strategy automaton, but none of the deviation automata are Nash equilibria resulting in the given payoff profile. If this is done for all payoff profiles, then the union of all Nash equilibria for each payoff profile will be the set of all Nash equilibrium strategy words - and consequently, the set of all Nash equilibrium strategy profiles can be deduced.

Hence, the problem of strategy synthesis with respect to the generalised dining philosopher's problem with the individual objectives of minimizing maximum waiting time, and collective objectives of fairness and starvation-freedom is solved algorithmically. The procedure presented leans heavily upon the established procedure of $LTL[\mathcal{F}]$ model checking, and consequently shares some of its drawbacks. For example, the usage of arbitrary evaluation functions is not permitted and only those that can be expressed using $LTL[\mathcal{F}]$ can be used. This dissertation only dealt with minimizing maximum waiting time, and more generality would be desirable. The procedure developed here has the strength of being Buchi automaton-based, so existing software implementations and algorithms for processing these automata can be used - making the procedure herein much easier to implement in software.

7.2 Future Work

As this work unfolded, various new ideas and questions arose, that were beyond the intended scope of this dissertation. These are explained briefly now:

Formal Proof of Correctness

The procedure proposed in this dissertation is defined but no theorems or proofs are provided. Rigorous correctness proof of the procedures is necessary.

Iterative Methods

The procedure presented is a complete algorithm which will produce all Nash Equilibria that exist. However, there may be very many such Nash Equilibria and it may only be of interest to find one. Some iterative methods could be explored which will generate a candidate strategy profile, and then through iterative refinement tend towards a Nash Equilibrium strategy profile.

Heuristics

The algorithm presented requires the $LTL[\mathcal{F}]$ model checking procedure to be completed several times. Some heuristics may be developed that would prevent the time expense of repeated runs of $LTL[\mathcal{F}]$ model checking. For example, some NGBAs constructed in previous steps may be reused or simply adapted in later steps.

Mixed Strategies

The construction procedure presented herein was restricted to the synthesis of pure strategy profiles, that is, strategies in which the action chosen for a given configuration (or configuration/memory state) is deterministic. The proposed approach could be extended to the synthesis of mixed strategies. In a mixed strategy, the decision is stochastic. Instead of a single action being prescribed, a variety of options are given as well as the probabilities of each action being chosen. In this case, a solution would involve calculating what the different probabilities should be to ensure the game satisfies the specified objectives with the desired probability.

Applicability To Other Problems

The GDPG is represented as a CGM, which is a common model used for representing multi-agent systems. Moreover, $LTL[\mathcal{F}]$ is a flexible logic that is well-developed in the literature. Thus, it may be possible to express other multi-agent system problems using these formalisms and apply the procedure of this dissertation. For example, processes in a computer system need to compete for shared resources, and these processes must be completed within tightly specified time constraints. It is possible that these processes can autonomously develop resource allocation strategies which are stable and guarantee maximum waiting times, timeout prevention, and so forth.

Analytic Approaches

It may be possible that there are necessary or sufficient conditions that can be applied to GDP games that will demonstrate the existence of solutions without the need to explicitly find these Nash Equilibria strategy profiles.

Bibliography

[1] Shaull Almagor, Udi Boker, and Orna Kupferman. Discounting in ltl. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 424–439. Springer, 2014.

[2] Shaull Almagor, Udi Boker, and Orna Kupferman. Formally reasoning about quality. *Journal of the ACM (JACM)*, 63(3):1–56, 2016.

[3] Shaull Almagor, Orna Kupferman, and Giuseppe Perelli. Synthesis of controllable nash equilibria in quantitative objective game. In *IJCAI*, volume 18, pages 35–41, 2018.

[4] Shaull Almagor, Orna Kupferman, and Yaron Velner. Minimizing expected cost under hard boolean constraints, with applications to quantitative synthesis. *arXiv preprint arXiv:1604.07064*, 2016.

[5] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, sep 2002.

[6] Guy Avni, Thomas A. Henzinger, and Orna Kupferman. Dynamic resource allocation games. *Theoretical Computer Science*, 807:42–55, 2020. In memory of Maurice Nivat, a founding father of Theoretical Computer Science - Part II.

[7] Aaron Bohy, Véronique Bruyere, Emmanuel Filiot, and Jean-François Raskin. Synthesis from ltl specifications with mean-payoff objectives. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 169–184. Springer, 2013.

[8] Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Nash equilibria in concurrent games with büchi objectives. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

[9] Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Concurrent games with ordered objectives. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures*, pages 301–315, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[10] Patricia Bouyer, Orna Kupferman, Nicolas Markey, Bastien Maubert, Aniello Murano, and Giuseppe Perelli. Reasoning about quality and fuzziness of strategic behaviours. *arXiv preprint arXiv:1905.11537*, 2019.

[11] Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games (full version), 2016.

[12] Patricia Bouyer, Nicolas Markey, and Steen Vester. Nash equilibria in symmetric graph games with partial observation. *Information and Computation*, 254:238–258, 2017.

[13] Daniel IA Cohen and Daniel IA Cohen. *Introduction to computer theory*, volume 2. Wiley New York, 1991.

[14] Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016.

[15] Riccardo De Masellis, Valentin Goranko, Stefan Gruner, and Nils Timm. Generalising the dining philosophers problem: competitive dynamic resource allocation in multi-agent systems. In *European Conference on Multi-Agent Systems*, pages 30–47. Springer, 2018.

[16] Edsger W Dijkstra. Operating systems techniques. In *Chapter Hierarchical Ordering of Sequential Processes*. Academic Press, 1972.

[17] E Allen Emerson and Edmund M Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer programming*, 2(3):241–266, 1982.

[18] Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 190–204, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[19] Valentin Goranko and Nils Timm. Unpublished notes on gdp games.

[20] Julian Gutierrez, Aniello Murano, Giuseppe Perelli, Sasha Rubin, Thomas Steeples, and Michael Wooldridge. Equilibria for games with combined qualitative and quantitative objectives. *Acta Informatica*, 58(6):585–610, 2021.

[21] Julian Gutierrez, Aniello Murano, Giuseppe Perelli, Sasha Rubin, and Michael Wooldridge. Nash equilibria in concurrent games with lexicographic preferences. 2017.

[22] Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael Wooldridge. Equilibrium design for concurrent games. *arXiv preprint arXiv:2106.10192*, 2021.

[23] Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Annals of Mathematics and Artificial Intelligence*, 78(1):3–20, jun 2016.

[24] Kevin Leyton-Brown and Yoav Shoham. Essentials of game theory: A concise multidisciplinary introduction. *Synthesis lectures on artificial intelligence and machine learning*, 2(1):1–88, 2008.

[25] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. Springer Science & Business Media, 2012.

[26] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

[27] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[28] Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *1st Symposium in Logic in Computer Science (LICS)*. IEEE Computer Society, 1986.